МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
имени Н.Э. БАУМАНА

Учебное пособие

Методические указания
по выполнению РПЗ к курсовой работе
по единому комплексному заданию по блоку дисциплины

«Системотехника электронных средств, комплексы и сети»

МГТУ имени Н.Э. Баумана

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
имени Н.Э. БАУМАНА

Методические указания
по выполнению РПЗ к курсовой работе
по единому комплексному заданию по блоку дисциплины

«Системотехника электронных средств, комплексы и сети»

Москва
МГТУ имени Н.Э. Баумана

2012

УДК 681.3.06(075.8)
ББК 32.973-018
И201

Методические указания по выполнению РПЗ к курсовой работе по единому комплексному заданию по блоку дисциплины «Системотехника электронных средств, комплексы и сети» / Коллектив авторов –
М.: МГТУ им. Н.Э. Баумана, 2012. – 78 с.: ил.

В методических указаниях рассмотрены основные этапы, их последовательность и содержание по выполнению РПЗ к курсовой работе по единому комплексному заданию по блоку дисциплины «Системотехника электронных средств, комплексы и сети».

Ил. 39. Табл. 5. Библиогр. 7 назв.

УДК 681.3.06(075.8)

СОДЕРЖАНИЕ

Введение

В настоящее время активно развиваются различные области микроэлектроники, появляется множество небольших проектов с ограниченными финансовыми ресурсами. Им требуются недорогие решения в области осциллографов, предоставляющие базовые функции осциллографов с возможностью расширения их функционала за счет добавления модулей или расширения программного обеспечения. Таким недорогим и функциональным решением может стать разрабатываемый двухканальный цифровой осциллограф.

Основными особенностями разрабатываемого осциллографа являются набор функций, сравнимый с существующими промышленными аналогами, невысокая стоимость, возможность съема сигналов по двум каналам, ручная настройка развертки по обеим осям, возможность остановить в какой-либо момент изображение на дисплее, при этом сбор данных в память продолжается.

1. Обзор существующих осциллографов

Осциллограф — прибор, предназначенный для исследования (наблюдения, записи; также измерения) амплитудных и временных параметров электрического сигнала, подаваемого на его вход, либо непосредственно на экране, либо записываемого на фотоленте.

Современные осциллографы позволяют исследовать сигналы гигагерцовых частот. Для разворачивания более высокочастотных сигналов можно использовать электронно-оптические камеры.

1.1. Обзор аналогов двухканального цифрового осциллографа

В настоящее время существует огромное количество промышленных аналогов разработанного двухканального цифрового осциллографа. В качестве аналогов рассматриваются цифровые осциллографы Vellman HPS-40 и APS-230 и LCD Scope 40MSPS [17].

Для начала рассмотрим характеристики цифрового осциллографа Vellman HPS-40. Общий вид осциллографа представлен на рисунке 1.

Рисунок 1 – Общий вид цифрового осциллографа Vellman HPS-40

Дискретизация: 40 МГц (12 МГц для единичных импульсов)

Полоса пропускания усилителя: 5 МГц (при развертке 5 м.в/дел) до 12 МГц (при развертке 50 мВ-1 В-20 В/дел.)

Входной импеданс: 1 МОм/20 пФ (стандартный щуп осциллографа)

Жидкокристаллический дисплей: 112 х 192 точек

Скорость развертки: 50 нс – 1 час/дел., 32 шага

Напряжение питания: 9 В/300 мА постоянного тока (нерегулируемое)

Батареи: тип АА или аккумулятор NiCd/NiMn 90 мА (6 шт.)

Время автономной работы: 20 часов

Размеры: 105 x 220 x 35 мм

Вес: 450 г без батарей.

Данный осциллограф управляется посредством 12 клавишной клавиатуры и 4 кнопок направлений. 4 клавиши отвечают за настройку яркости, контрастности, автоподстройки развертки и выставления щупа осциллографа на ноль. 6 клавиш отвечаю за дополнительные функции: задание триггера, выбор меток на графике, изменение развертки по выбранной оси, смена оси. 2 клавиши отвечают за вызов осциллограмм из памяти и смену множителя щупа. Управление представлено на рисунке 2.

Рисунок 2 – Панель управления цифровым осциллографом Vellman HPS-40

Основным преимуществом данного осциллографа является автоматическая настройка вертикальной и горизонтальной развертки входящего сигнала. Следует отметить, что данный осциллограф является одноканальным. Есть возможность вывода истинных среднеквадратичных и амплитудных значений, измерение мощности аудиосигнала. Особенностью данного осциллографа является возможность работы в режиме самописца, т.е. одна выборка в 25 часов. Так же данный осциллограф производится в больших количествах.

Из недостатков данного осциллографа следует отметить то, что он является одноканальным, а так же его высокая стоимость и вес прибора.

Рассмотрим следующий цифровой осциллограф - LCD Scope 40MSPS, его общий вид представлен на рисунке 3.

Рисунок 3 – Общий вид двухканального цифрового осциллографа LCD Scope 40MSPS

Данный осциллограф является одноканальным. Рассмотрим его характеристики:

Входной импеданс: 10 КОм

Частота выборки: 40 Мегавыборок/с на канал

Максимальный входной сигнал: 5 МГц

Жидкокристаллический дисплей: 240 х 128 точек

Скорость развертки: 500 нс – 1 сек/дел.

Диапазон чувствительности на входе: 40 мВ/дел.

Напряжение питания: 8 В/1 А регул.

Данный осциллограф управляется при помощи кнопок на передней панели прибора. Этот осциллограф разработан не для промышленного производства, а для индивидуального.

Входная цепь данного осциллографа использует операционный усилитель OPA2652 фирмы Texas Instruments совместно с фильтрами с пропусканием частот до 20 МГц. Сигнал далее проходим через ШИМ, виртуально запрограммированный на микроконтроллере, для возможности развертки сигнала во времени и по амплитуде. В качестве АЦП используется микросхема ADS830 – это восьми-битный АЦП фирмы Texas Instruments. Чтобы справиться с высокой пропускной способностью АЦП, используется высокоскоростная память FIFO IDT7201. Для окончательной обработки сигнала и вывода его на дисплей LMG6402PFLR от Hitachi используется микроконтроллер ATmega162 от Atmel[14].

Основные преимущества данного осциллографа это ЖКИ дисплей большого размера, низкий порог чувствительности, доступность всех компонентов, использованных при создании данного осциллографа, а так же возможность динамического исследования сигнала.

Главными недостатками рассматриваемого осциллографа являются его потребность в питании, вес по сравнению с разработанным двухканальным осциллографом, его габариты и один канал для исследования сигнала.

Рассмотрим следующий цифровой осциллограф - Vellman APS-230, его общий вид представлен на рисунке 4.

Рисунок 4 – Общий вид двухканального
цифрового осциллографа Vellman APS-230

Данный осциллограф является двухканальным. Рассмотрим его характеристики:

Входной импеданс: 1 МОм/20 пФ (стандартный щуп осциллографа)

Дискретизация: 2 x 30 МГц

Частота выборки: 240 Мегавыборок/с на канал

Максимальный входной сигнал: 100 В (AC+DC), 200 В (AC)

Жидкокристаллический дисплей: 128 x 192 точек

Скорость развертки: 25 нс – 1 час/дел.

Диапазон чувствительности на входе: 14 диап. от 1 мВ/дел. до 20 В/дел. (10 мВ – 200 В/дел. с пробн. x10)

Напряжение питания: 9 В/350 мА регул.

Батареи: аккумулятор NiMH 6 В/1800 мА

Размеры: 230 x 150 x 50 мм

Вес: 0.85 кг.

Данный осциллограф управляется аналогично рассмотренному ранее, за исключением того, что кнопки управления сигналом продублированы для второго канала. Управление данным осциллографом представлено на рисунке 5.

Рисунок 5 – Панель управления двухканальным цифровым осциллографом Vellman APS-230

Основные преимущества данного осциллографа это измерение значительного входного сигнала, большой диапазон временной развертки, полностью автоматическая настройка вертикальной и горизонтальной развертки, режим самописца с возможностью выбора времени проведения

выборок, измерительные щупы с делителями x1 и x10, сохранение 2047 выборок в режиме самописца, регулируемый триггер запуска съема сигнала.

Главными недостатками рассматриваемого осциллографа являются его значительный вес по сравнению с разработанным двухканальным осциллографом и его стоимость.

1.2. Изучение методов обработки поступающих сигналов

Любой непрерывный (аналоговый) сигнал s(t) может быть подвергнут дискретизации по времени и квантованию по уровню (оцифровке), то есть представлен в цифровой форме. Если частота дискретизации сигнала Fd не меньше, чем удвоенная наивысшая частота в спектре сигнала Fmax (то есть), то полученный дискретный сигнал s(k) эквивалентен сигналу s(t) (см. теорему Котельникова). При помощи математических алгоритмов s(k) преобразуется в некоторый другой сигнал s1(k) имеющий требуемые свойства. Процесс преобразования сигналов называется фильтрацией, а устройство, выполняющее фильтрацию называется фильтр. Поскольку отсчёты сигналов поступают с постоянной скоростью Fd, фильтр должен успевать обрабатывать текущий отсчет до поступления следующего (чаще - до поступления следующих n отсчётов, где n - задержка фильтра), то есть обрабатывать сигнал в реальном времени. Для обработки сигналов (фильтрации) в реальном времени применяют специальные вычислительные устройства — цифровые сигнальные процессоры[1,4].

Всё это полностью применимо не только к непрерывным сигналам, но и к прерывистым, а также к сигналам, записанным на запоминающие устройства. В последнем случае скорость обработки непринципиальна, так как при медленной обработке данные не будут потеряны.

При разработке двухканального цифрового осциллографа для преобразования сигнала применяется теорема Котельникова.

Теоре́ма Коте́льникова (в англоязычной литературе — теорема Найквиста — Шеннона или теорема отсчётов) гласит, что, если аналоговый сигнал *x(t)* имеет ограниченный спектр, то он может быть восстановлен однозначно и без потерь по своим дискретным отсчётам, взятым с частотой строго большей удвоенной максимальной частоты спектра Ω:

$$f > \frac{\Omega}{\pi}, \qquad (1)$$

где Ω — верхняя частота в спектре, или (формулируя по-другому) по отсчётам, взятым с периодом Δ, чаще полупериода максимальной частоты спектра Ω:

$$\Delta < \frac{\pi}{\Omega}, \tag{2}$$

Такая трактовка рассматривает идеальный случай, когда сигнал начался бесконечно давно и никогда не закончится, а также не имеет во временно́й характеристике точек разрыва. Именно это подразумевает понятие «спектр, ограниченный частотой Ω».

Разумеется, реальные сигналы (например, звук на цифровом носителе) не обладают такими свойствами, так как они конечны по времени и, обычно, имеют во временной характеристике разрывы. Соответственно, их спектр бесконечен. В таком случае полное восстановление сигнала невозможно и из теоремы Котельникова вытекают 2 следствия:

1. Любой аналоговый сигнал может быть восстановлен с какой угодно точностью по своим дискретным отсчётам, взятым с частотой $f > 2\Omega$, где Ω — максимальная частота, которой ограничен спектр реального сигнала.
2. Если максимальная частота в сигнале превышает половину частоты дискретизации, то способа восстановить сигнал из дискретного в аналоговый без искажений не существует.

Говоря шире, теорема Котельникова утверждает, что непрерывный сигнал можно представить в виде интерполяционного ряда

$$x(t) = \sum_{k=-\infty}^{\infty} x(k\Delta) sinc\left[\frac{\pi}{\Delta}(t - k\Delta)\right], \tag{3}$$

где $sinc(x) = sin(x)/x$ – функция *sinc*. Интервал дискретизации удовлетворяет ограничениям $0 < \Delta < \pi/\Omega$. Мгновенные значения данного ряда есть дискретные отсчёты сигнала $x(k\Delta)$[1].

Выводы

В данной главе были рассмотрены существующие промышленные аналоги цифровых осциллографов. Рассмотрены одноканальный и двухканальный цифровые осциллографы. Они обладают сходными функциями, равными характеристиками, за исключением количества входных каналов и длительности работы в качестве самописца. Главными недостатками являются их стоимость и вес.

Следует отметить, что цифровой двухканальный осциллограф APS-230 обладает отдельными клавишами настройки для каждого канала. За счет введения клавиши выбора канала для настройки и исключения второго набора клавиш для настройки канала можно уменьшить габариты прибора, но уменьшится удобство пользования прибором.

В разрабатываемом двухканальном цифровом осциллографе используется панель управления схожая с панелью управления цифрового осциллографа LCD Scope 40MSPS, с учетом того, что управление двумя каналами будет осуществляться одним набором кнопок, а переключение между каналами будет осуществляться при помощи отдельной кнопки.

Теорема Котельникова позволяет представлять мгновенные значения сигнала в виде дискретных отсчетов. Данная теорема позволит нам обрабатывать входной сигнал и передавать на дисплей.

2. Разработка двухканального цифрового осциллографа

Разработка любой, достаточно сложной, аппаратуры состоит из нескольких этапов. На первом этапе разрабатывается алгоритм работы устройства и алгоритм работы программного обеспечения устройства, затем пишется программное обеспечение устройства (прошивки для микроконтроллеров, ПЛИС). На следующем этапе производится описание структуры разрабатываемого устройства при помощи языка формального описания VHDL. После чего разрабатываются структурная, функциональная и принципиальная схемы. Далее для сборки устройства производится выбор элементной базы.

2.1. Разработка алгоритма работы двухканального цифрового осциллографа

Алгоритм работы двухканального цифрового осциллографа представлен на рисунке 6.

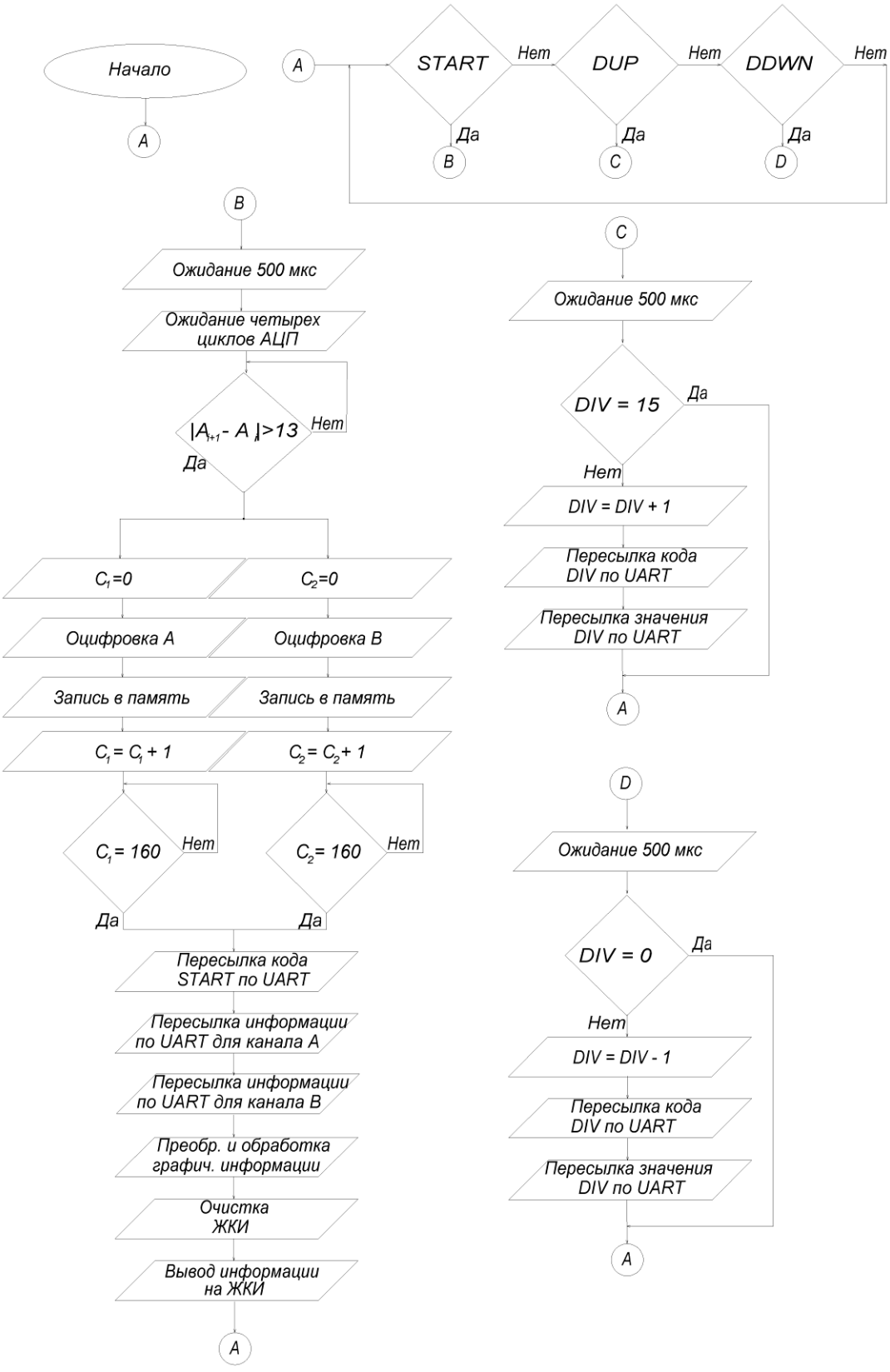

Рисунок 6 – Алгоритм работы ДЦО

После подключения питания осциллограф ждет нажатия одной из трех кнопок: START, DUP, DDWN – начала ожидания фронта сигнал, увеличения коэффициента деления частоты дискретизации (уменьшение частоты дискретизации) и уменьшения коэффициента деления частоты дискретизации (увеличение частоты дискретизации).

В зависимости от нажатой кнопки: по START начинает основная рабочая часть ПЛИС и затем микроконтроллера после развертывания сигнала. Если же нажаты кнопки снижения или увеличения коэффициента деления, то соответственно вначале проводится проверка на достижения максимального или минимального его значения, после принимается решение выполнять или нет заданное действие. Далее информация передается по UART в микроконтроллер. После нажатия кнопки выполняется ожидания для удаления дребезга контактов[5,6].

Основная программа заключается в следующем:

1. Считывание четырех «пустых» отсчетов с каждого из АЦП, по требованию документации на АЦП.
2. Для основного канала А, считывается сигнал, который затем сохраняется в буфер.
3. После оцифровывания следующего отсчета, проводится сравнение между текущим и предыдущим значение, если разность больше значения порядка 0,5В, то выполняется п.4, иначе алгоритм ожидания фронта сигнала повторяется.
4. С обоих каналов проводится оцифровка запись в встроенную ОЗУ ПЛИС, всего записывается 160х2 отсчетов.
5. После заполнения всех 160 отсчетов, подается сигнала на передатчик UART, который последовательно передает информацию с памяти канала А и информацию с памяти канала В.
6. Микроконтроллер обрабатывает принятую информацию, масштабирует ее в соответствии с типом ЖКИ и выдает на экран.

2.2. Разработка алгоритма работы и исходного кода программы, реализующей алгоритм работы микроконтроллера

Программа микроконтроллера разработана на языке программирования Си в соответствии с требованиями алгоритма, приведенного в предыдущем разделе. Микропрограмма захватывает такие действия как прием данных по UART, графическая обработка данных, их масштабирование, драйвер для управления ЖКИ, вывод графического интерфейса на ЖКИ.

На лист. 1 приведена основная микропрограмма, задачей которой является инициализация всех используемых устройств, таких как ЖКИ и встроенный в контроллер приемник UART[9].

Листинг 1 – Основная микропрограмма

```
int main(void)
{
   u08 lcd_buffer[30];
   u08 cur_pos_x, cur_pos_y;

   lcd_init();
   lcd_graphics_init();
   lcd_graphics_clear();

   uartInit();
   u32 baudrate = 19200;
   u16 bauddiv = uartSetBaudRate(baudrate);

   cur_pos_x = 0;
   cur_pos_y = LCD_HEIGHT-11*FONT_HEIGHT;
   sprintf(lcd_buffer,"Now I'm initialized!");
   g_draw_string(0,cur_pos_y,lcd_buffer);
   cur_pos_y += FONT_HEIGHT + 1;

   sprintf(lcd_buffer,"Baud
        rate: %li/%i",baudrate,bauddiv);
   g_draw_string(0,cur_pos_y,lcd_buffer);
   cur_pos_y += FONT_HEIGHT + 1;

   for(ever) {RunUART();}

   return 0;
}
```

И в конце программы запускает бесконечный цикл, в котором ожидается поступление информации по UART, проводится постоянная проверка буфера, до тех пор, пока в нем не появятся данные.

Для отладки приемника UART использовалась функция, приведенная на лист. 2, которая позволила, начиная с заданного в качестве входных параметров значения смещения пройти по всему объему данных принятому по UART.

Листинг 2 – Микропрограмма отладки передачи данных по UART

```
void DebugRecieveProc(u08* uartData, u08 start)
{
   char lcd_buffer[30];
   u08 i = 0;
   u08 j = 0;
   u08 cur_pos_x = 0;
   u08 cur_pos_y = LCD_HEIGHT-11*FONT_HEIGHT;

   lcd_graphics_clear();

   u08 k = start;
   for (i = 0; i < 9; i++)
   {
        for (j = 0; j < 8; j++)
        {
             sprintf(lcd_buffer,"%2X",uartData[k]);
             g_draw_string(cur_pos_x,cur_pos_y,
                  lcd_buffer);
             cur_pos_x += 2*FONT_WIDTH+10;
             k++;
        }
        cur_pos_x = 0;
        cur_pos_y += FONT_HEIGHT + 1;
   }
}
```

С помощью данной функции было определенно, что прием по UART микроконтроллером ведется не с номинальной частотой, а отличной от нее. Анализ документации микроконтроллера показал, что скорость передачи определяется коэффициентом деления системной тактовой частоты, и вычисляется по следующему выражению:

$$BAUD = \frac{f_{osc}}{16 \cdot (UBRR + 1)}$$

Тогда при системной тактовой частоте, на которую рассчитан кварцевый резонатор, с помощью которого происходит тактирование микроконтроллера равную 16МГц для передачи данных по UART на скорости 19200 бод, необходимо, чтобы коэффициент деления составлял:

$$UBRR = \frac{f_{osc}}{16 \cdot BAUD} - 1 = \frac{16MHz}{16 \cdot 19200} - 1 \approx 51{,}0833$$

Однако получить такой коэффициент деления в микроконтроллере не возможно, вместо этого в нем используется коэффициент ближайший к степени двойки, а именно 51, но тогда ошибка частоты UART равна порядка 0,2%, чего оказалось достаточным для неверного приема данных. Частота, на которой не происходило ошибки передачи, равна 19608 бод.

Функция, выполняющаяся в бесконечном цикле основной программы, приведена на лист. 3. Задача функции определить, что в буфере приемника UART находятся принятые данные, после чего принять еще один байт данных. Дальше проводится анализ: либо был принят код START (0xBC и 0x1F), который означает, что дальше будет передаваться пакет данных снятых с АЦП или код DIV (0xFF и 0x0A), который означает, что дальше будет передан еще один байт – коэффициент деления максимальной частоты дискретизации данных.

Листинг 3 – Микропрограмма приема по UART данных от ПЛИС

```
void RunUART(void)
{
   u08 div = 0;
   u08 k = 0;

   u08 adcData0[SAMPLENUM];
   u08 adcData1[SAMPLENUM];

   uartFlushReceiveBuffer();

   for (k = 0; k < 2; k++)
        while (!uartReceiveByte(adcData0+k));

   if ((adcData0[0] == 0xBC) &&
        (adcData0[1] == 0x1F))
   {
        for (k = 0; k < SAMPLENUM; k++)
             while (!uartReceiveByte(adcData0+k));
```

```
            for (k = 0; k < SAMPLENUM; k++)
                 while (!uartReceiveByte(adcData1+k));

            for (k = 0; k < SAMPLENUM; k++)
            {
                 adcData0[k] = 255 - adcData0[k];
                 adcData1[k] = 255 - adcData1[k];
            }

            //DebugRecieveProc(adcData0,0);

            lcd_graphics_clear();
            PrintInterface(div);
            PrintRecieveData2ch(adcData0,adcData1,SAMPLENUM);
            _delay_ms(500);
        }
        else if ((adcData0[0] == 0xFF) &&
             (adcData0[1] == 0x0A))
        {
             uartReceiveByte(&div);
             PrintInterface(div);
        }
    }
```

Как только завершен прием данных проводится их инвертирование так как, на входе АЦП находится инвертирующий усилитель, поэтому для восстановления сигнала необходимо это действие.

После того, как принят новый коэффициент деления экран ЖКИ очищается на экране заново отрисовывается интерфейс ДЦО, на котором находятся временная ось с делениями через 16 отсчетов, всего 10 делений. Функция выполняющая эту задачу приведена на лист. 4.

Также 7 пикселей по высоте занимает вывод надписи определяющий текущую частоту дискретизации ДЦО и временной интервал приходящийся на 16 отсчетов.

Листинг 4 – Вывод интерфейса на ЖКИ

```
  void PrintInterface(u08 DivCode)
  {
     u08 k = 0;
     u08 LinePosY = LCD_HEIGHT-11;
     g_draw_horizontal_line(0,LinePosY,LCD_WIDTH-1);
     g_draw_string_p(0,LCD_HEIGHT-
  FONT_HEIGHT,pgm_read_word(divtable+DivCode));
     u08 slen = strlen_P(pgm_read_word(fsmptable+DivCode));
```

```
    g_draw_string_p(LCD_WIDTH-slen*(FONT_WIDTH+1),LCD_HEIGHT-
FONT_HEIGHT,pgm_read_word(fsmptable+DivCode));
    for (k = 1; k < 10; k++)
        g_draw_vertical_line(16*k,LinePosY-2,5);
}
```

Для вывода информации в удобном виде перед тем, как включать или выключать пиксель ЖКИ, необходим пересчет значений принятых отсчетов в координаты экрана. На лист. 5 приведен код такой функции для одного отсчета.

С этой целью вначале задается размер, т.е. число пикселей соответствующее максимальной амплитуде сигнала, после чего происходит пересчет координат[9,10].

Листинг 5 – Вывод одного отсчета снятого сигнала

```
u08 DrawSample(u08 CurY, u08 PrevY,
        u08 YScale, u08 YOffset, u08 k)
{
    u08 y0 = PrevY;
    u08 y1 = CurY*YScale >> 8;
    u08 ylen = abs(y1-y0);

    if (y1 > y0)
    {
        u08 ScrY = LCD_HEIGHT - (y1 + YOffset);
        g_draw_vertical_line(k,ScrY,ylen);
    }
    else if (y1 == y0)
    {
        u08 ScrY = LCD_HEIGHT - (y1 + YOffset);
        lcd_graphics_plot_pixel(k, ScrY, PIXEL_ON);
        lcd_graphics_plot_pixel(k, ScrY-1, PIXEL_ON);
        if (k != LCD_WIDTH-1)
        {
            lcd_graphics_plot_pixel(k+1, ScrY, PIXEL_ON);
            lcd_graphics_plot_pixel(k+1, ScrY-1, PIXEL_ON);
        }
    }
    else if (y1 < y0)
    {
        u08 ScrY = LCD_HEIGHT - (y0 + YOffset);
        g_draw_vertical_line(k,ScrY,ylen);
        g_draw_vertical_line(k+1,ScrY,ylen);
    }

    return y1;
}
```

Число пикселей соответствующее максимальной амплитуде задается перед компиляцией проекта, поэтому в рабочее варианте ДЦО масштаб сигнала менять нельзя.

Для того, чтобы просчитать экранные координаты для всех отсчетов обоих каналов была разработана функция, приведенная на лист. 6.

Листинг 6 – Вывод данных с двух каналов

```
void PrintRecieveData2ch(u08* ch0, u08* ch1, u08 nums)
{
     u08 k = 0;
     u08 YScale = 30;
     u08 YOffCh0 = 16;
     u08 YOffCh1 = YOffCh0+YScale+2;

     u08 ych0 = ch0[0]*YScale >> 8;
     u08 ych1 = ch1[1]*YScale >> 8;

     g_draw_horizontal_line(0,LCD_HEIGHT-((0*YScale >> 8) +
YOffCh0),LCD_WIDTH-1);
     g_draw_horizontal_line(0,LCD_HEIGHT-((255*YScale >> 8)
+ YOffCh0),LCD_WIDTH-1);
     g_draw_horizontal_line(0,LCD_HEIGHT-((0*YScale >> 8) +
YOffCh1),LCD_WIDTH-1);
     g_draw_horizontal_line(0,LCD_HEIGHT-((255*YScale >> 8)
+ YOffCh1),LCD_WIDTH-1);

     for (k = 0; k < nums; k++)
     {
          ych0 = DrawSample(ch0[k],ych0,YScale,YOffCh0,k);
          ych1 = DrawSample(ch1[k],ych1,YScale,YOffCh1,k);
     }
}
```

Функции, реализующие в себе драйвер с ЖКИ и UART, являются частью библиотеки AVRLib, распространяемой по лицензии GNU. Код использованных функций приведен приложение данной работы.

2.3. Разработка структурной схемы логического проекта ПЛИС и VHDL-описания блоков ПЛИС

Структурная схема логического проекта ПЛИС приведена на рисунке 7.

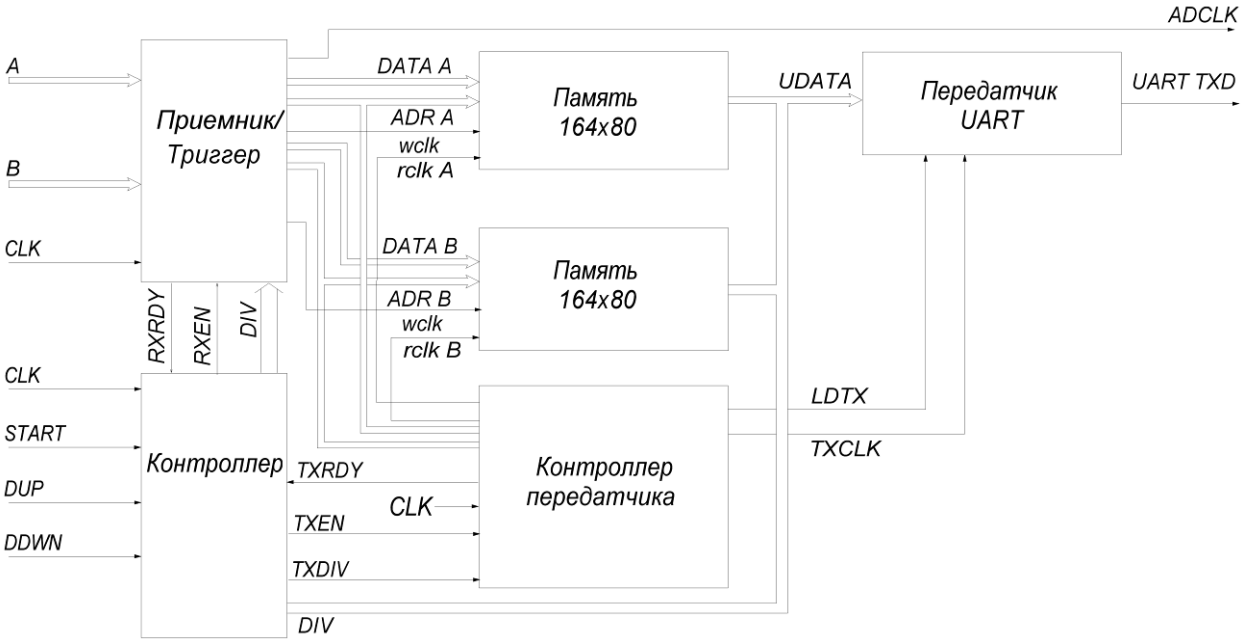

Рисунок 7 – Структурная схема логического проекта ПЛИС

Программируемая логическая интегральная схема фирмы Xilinx содержит блок управления алгоритмом работы схемы осциллографа. Контроллер Controller ожидает нажатие кнопки начала работы, либо кнопок увеличения или снижения частоты дискретизации. После чего проводится обработка нажатия с целью устранения паразитного эффекта множественного срабатывания – дребезга контактов. Далее если нажата, кнопка «Start», то активизируется триггер ADCReciever, который начинает тактировать АЦП и считывать с него информацию.

Перед получение рабочих данных проводится считывание 4 «пустых» данных необходимых для корректного преобразования в АЦП (4 пустых отчета принимаются также и в канале B). Затем принятые данные обрабатываются с целью фильтрации шумов, для этого триггер ждет изменения сигнала +/-6, что при уровне сигнала +/-5В соответствует 5/127*6 = 0,25В, то есть триггер настроен на срабатывание по уровню сигнала порядка 0,25В.

При срабатывании триггера, он начинает тактировать также и блоки памяти для каналов A и B, также начинает тактироваться и АЦП канала B, при этом происходит запись в память канала A и B. После сохранения 160 отчетов подается сигнал на блок uartA, которой вначале по интерфейсу UART передает данные в микроконтроллер из памяти канала A, затем блок uartB передает данные из памяти канала B [2].

Скорость передачи составляет 19200 бод. В схеме ADCReciver находится делитель частота деления, которого задается сигналом div. При нажатии на кнопки DUP и DWN, происходит изменение сигнала div, увеличение его и снижение соответственно. Вывод на микроконтроллер осуществляется через uartDiv.

На рисунке 8 приведена структурная схема логического проекта ПЛИС ДЦО для САПР ISE WebPack 11.0.

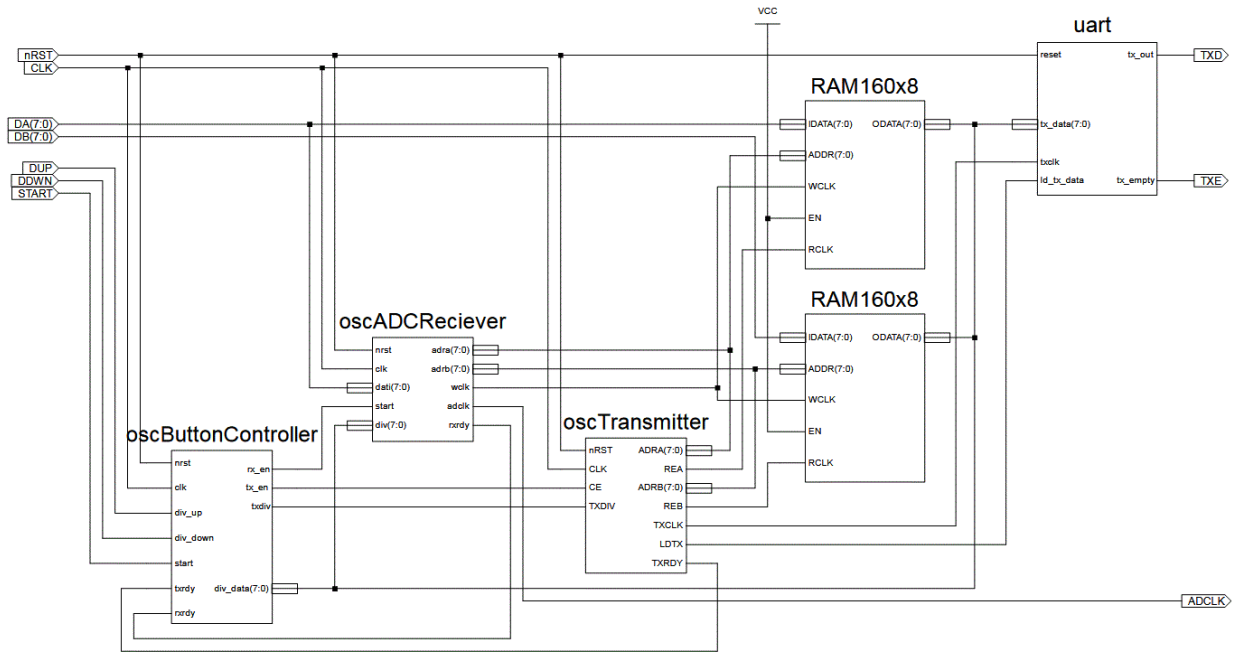

Рисунок 8 – Структурная схема логического проекта ПЛИС для САПР ISE WebPack 11.0

Для данной схемы блок памяти RAM160x8 реализован в виде соединения стандартных блоков памяти поставляемых фирмой Xilinx для выбранной ПЛИС: RAM32X8S. Адресация блоков памяти осуществляется посредством, также стандартного блока дешифратора 8 в 3: D3_8E. На выходе каждого блока памяти располагается восьми битный буфер, с третьим

Z-состоянием позволяющий подключать к выходной шине только один блок памяти в каждый момент времени. Также на выходе блока памяти расположен еще один Z-буфер для возможности подключения блоков памяти обоих каналов к одной шине для передачи данных по UART.

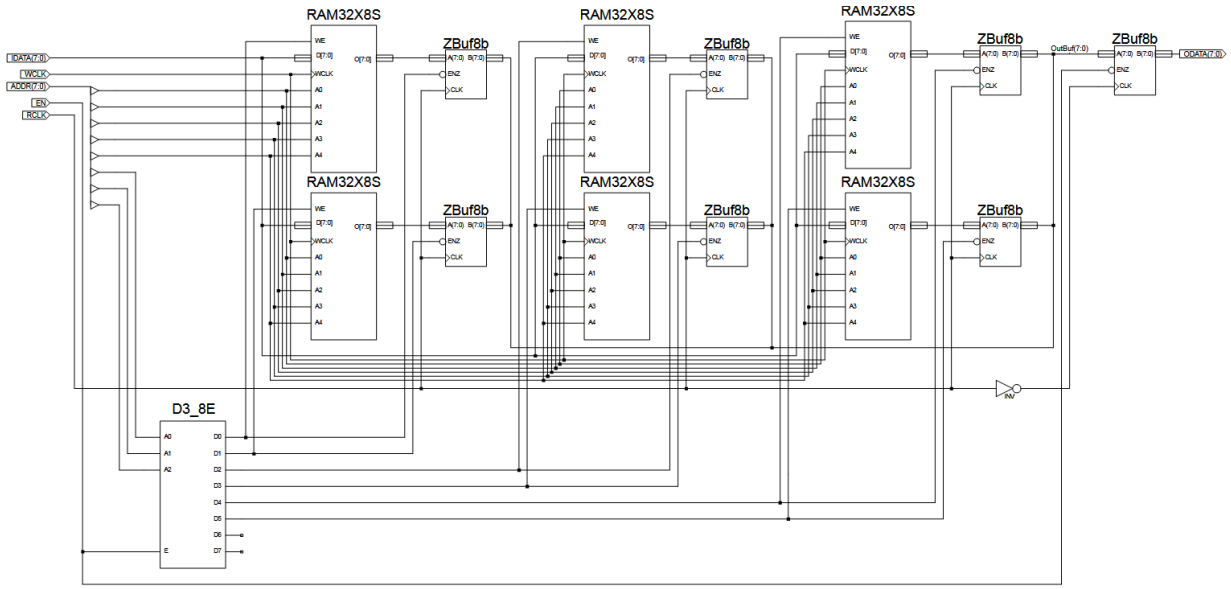

Рисунок 9 – Структурная схема блока RAM160x8 логического проекта ПЛИС для САПР ISE WebPack 11.0

Блок буфера с третьим Z-состоянием ZBuf8b реализован с помощью VHDL-кода, который представлен на лист. 7.

Листинг 7 – Реализация на языке VHDL буфера с третьим Z-состоянием

```
entity ZBuf8b is
    Port ( A : in  STD_LOGIC_VECTOR (7 downto 0);
           B : out  STD_LOGIC_VECTOR (7 downto 0);
                 CLK : in STD_LOGIC;
           ENZ : in  STD_LOGIC);
end ZBuf8b;

architecture Behavioral of ZBuf8b is
     SIGNAL S_B : STD_LOGIC_VECTOR (7 DOWNTO 0);
begin
     PROCESS(CLK,ENZ)
     BEGIN
          IF ENZ = '1' THEN
               IF RISING_EDGE(CLK) THEN
                    S_B <= A;
               END IF;
          ELSE
               S_B <= (others => 'Z');
          END IF;
     END PROCESS;
```

```
     B <= S_B;
end Behavioral;
```

Остальные блоки логического проекта ПЛИС, реализованы также с помощью языка VHDL, листинги которых приведены ниже.

Листинг 8 – Реализация на языке VHDL передатчика UART

```
entity uart is
    port (
        reset       :in  std_logic;
        txclk       :in  std_logic;
        ld_tx_data  :in  std_logic;
        tx_data     :in  std_logic_vector (7 downto 0);
        tx_out      :out std_logic;
        tx_empty    :out std_logic
    );
end entity;
architecture rtl of uart is
    -- Internal Variables
    signal tx_reg         :std_logic_vector (7 downto 0);
    signal tx_over_run    :std_logic;
    signal tx_cnt         :std_logic_vector (3 downto 0);
    signal rx_reg         :std_logic_vector (7 downto 0);
    signal tx_is_empty    :std_logic;
      signal tx_enable      :std_logic;
begin
     tx_enable <= '1';
    -- UART TX Logic
    process (txclk, reset) begin
        if (reset = '1') then
            tx_reg        <= (others=>'0');
            tx_is_empty   <= '1';
            tx_over_run   <= '0';
            tx_out        <= '1';
            tx_cnt        <= (others=>'0');
        elsif (rising_edge(txclk)) then

            if (ld_tx_data = '1') then
                if (tx_is_empty = '0') then
                    tx_over_run <= '0';
                else
                    tx_reg   <= tx_data;
                    tx_is_empty <= '0';
                end if;
            end if;
            if (tx_enable = '1' and tx_is_empty = '0') then
                tx_cnt <= tx_cnt + 1;
                if (tx_cnt = 0) then
                    tx_out <= '0';
                end if;
                if (tx_cnt > 0 and tx_cnt < 9) then
                    tx_out <= tx_reg(conv_integer(tx_cnt) -1);
```

```
                end if;
                if (tx_cnt = 9) then
                    tx_out <= '1';
                    tx_cnt <= X"0";
                    tx_is_empty <= '1';
                end if;
            end if;
            if (tx_enable = '0') then
                tx_cnt <= X"0";
            end if;
        end if;
    end process;
    tx_empty <= tx_is_empty;

end architecture;
```

Листинг 9 – Реализация на языке VHDL контроллера передатчика UART

```
entity oscTransmitter is
   generic (
     FCLOCK : integer := 100; -- MHz
     FUART : integer := 19608 -- Baudrate
     --FUART : integer := 20000
     --FUART : integer := 500000
   );
   port (
     CLK : in  std_logic;
     CE : in std_logic;
     nRST : in std_logic;
     TXCLK : out std_logic;
     TXRDY : out std_logic;
     LDTX : out  std_logic;
     REA : out std_logic;
     REB : out std_logic;
     ADRA : out  std_logic_vector (7 downto 0);
     ADRB : out  std_logic_vector (7 downto 0);
     TXDIV : in std_logic
   );
end oscTransmitter;

architecture Behavioral of oscTransmitter is
   signal STXCLK : std_logic;
   signal SLDTX : std_logic;
   signal SREA : std_logic;
   signal SREB : std_logic;
   signal SBTX : std_logic;
   signal SADRA : std_logic_vector(7 downto 0);
   signal SADRB : std_logic_vector(7 downto 0);
   signal STXRDY : std_logic;
begin
   process (CLK,nRST,STXRDY)
     constant TX_PERIOD : integer := (FCLOCK*1000000)/FUART;
     variable COUNT_TXCLK : integer := 0;
```

```
    variable COUNT_READY : integer := 0;
    variable BITNUM : integer := 0;
  begin
    if (nRST = '0') then
         COUNT_TXCLK := 0;
         COUNT_READY := 0;
         BITNUM := 0;

         STXRDY <= '1';
         SREA <= '0';
         SREB <= '0';
         SBTX <= '0';
         STXCLK <= '0';
         SLDTX <= '0';
         SADRA <= x"00";
         SADRB <= x"00";
    else
         if rising_edge(CLK) then
              if (CE = '1') then
                   if (STXRDY = '1') then
                        STXRDY <= '0';
                   end if;
              end if;

              if STXRDY = '0' then
                   if BITNUM = 0 then
                        if COUNT_TXCLK <= 1 then
                             SLDTX <= '1';
                        elsif COUNT_TXCLK = TX_PERIOD/8 then
                             if SBTX = '1' then
                                  SREB <= '1';
                             else
                                  SREA <= '1';
                             end if;
                        elsif COUNT_TXCLK = TX_PERIOD/4 then
                             if SBTX = '0' then
                                  SREB <= '1';
                             else
                                  SREA <= '1';
                             end if;
                        elsif COUNT_TXCLK = TX_PERIOD/2 then
                             STXCLK <= '1';
                        elsif COUNT_TXCLK = TX_PERIOD then
                             SLDTX <= '0';
                             STXCLK <= '0';
                             COUNT_TXCLK := 0;
                             BITNUM := BITNUM + 1;
                        end if;
                   elsif BITNUM = 14 then
                        BITNUM := 0;
                        if SADRB = x"A2" then
                             COUNT_TXCLK := 0;
                             SADRB <= x"00";
                             SBTX <= '0';
                             STXRDY <= '1';
```

```
                        elsif SADRA = x"A2" then
                             COUNT_TXCLK := -1;
                             SADRA <= x"00";
                             SBTX <= '1';
                        else
                             COUNT_TXCLK := -1;
                             if SBTX = '1' then
                                  SADRB <= SADRB + 1;
                             else
                                  SADRA <= SADRA + 1;
                             end if;
                        end if;
                   else
                        if COUNT_TXCLK = TX_PERIOD/2 then
                             STXCLK <= '1';
                        elsif COUNT_TXCLK = TX_PERIOD then
                             STXCLK <= '0';
                             COUNT_TXCLK := 0;
                             BITNUM := BITNUM + 1;
                        end if;
                   end if;
                   COUNT_TXCLK := COUNT_TXCLK + 1;
              end if;
         end if;
     end if;
   end process;

   TXRDY <= STXRDY;
   TXCLK <= STXCLK;
   LDTX <= SLDTX;
   REA <= SREA;
   REB <= SREB;
   ADRA <= SADRA when (STXRDY = '0') else (others => 'Z');
   ADRB <= SADRB when (STXRDY = '0') else (others => 'Z');
   --ADRA <= SADRA;

end Behavioral;
```

Листинг 10 – Реализация на языке VHDL управляющего контроллера

```
entity oscButtonController is
     port (
          div_up : in std_logic;
          div_down : in std_logic;
          start : in std_logic;
          nrst : in std_logic;
          clk : in std_logic;
          rxrdy : in std_logic;
          txrdy : in std_logic;
          rx_en : out std_logic;
          tx_en : out std_logic;
          div_data : out std_logic_vector(7 downto 0);
          txdiv : out std_logic
     );
```

```
end oscButtonController;

architecture Behavioral of oscButtonController is
     signal divup_sync : std_logic := '0';
     signal divup_sync_nms : std_logic := '0';
     signal divup_sync_nms_del : std_logic := '0';
     signal divup_pulse : std_logic := '0';

     signal divdwn_sync : std_logic := '0';
     signal divdwn_sync_nms : std_logic := '0';
     signal divdwn_sync_nms_del : std_logic := '0';
     signal divdwn_pulse : std_logic := '0';

     signal start_sync : std_logic := '0';
     signal start_sync_nms : std_logic := '0';
     signal start_sync_nms_del : std_logic := '0';
     signal start_pulse : std_logic := '0';

     signal div_cnt : integer range 0 to 255 := 15;

     signal rx_en_buf : std_logic;
     signal tx_en_buf : std_logic;

     signal txdiv_buf : std_logic;
begin

     process(clk)
     begin
          if rising_edge(clk) then
               divup_sync <= div_up;
               divup_sync_nms <= divup_sync;
               divup_sync_nms_del <= divup_sync_nms;

               divdwn_sync <= div_down;
               divdwn_sync_nms <= divdwn_sync;
               divdwn_sync_nms_del <= divdwn_sync_nms;

               start_sync <= start;
               start_sync_nms <= start_sync;
               start_sync_nms_del <= start_sync_nms;
          end if;
     end process;

     divup_pulse <= divup_sync_nms AND NOT divup_sync_nms_del;
     divdwn_pulse <= divdwn_sync_nms AND NOT divdwn_sync_nms_del;
     start_pulse <= start_sync_nms AND NOT start_sync_nms_del;

     process(nrst,div_cnt,clk)
          variable cnt_seg : integer := 0;
     begin
          if nrst = '0' then
               div_cnt <= 15;
               txdiv_buf <= '0';
               cnt_seg := 0;
          else
```

```
            if rising_edge(clk) then
                if txdiv_buf = '1' then
                    if cnt_seg = 25E6 then
                        txdiv_buf <= '0';
                    else
                        cnt_seg := cnt_seg + 1;
                    end if;
                end if;
                if   (txrdy = '1') and
                 (rxrdy = '1') and (txdiv_buf = '0') then
                    if (divup_pulse = '1') and
                     (div_cnt < 15) then
                        div_cnt <= div_cnt + 1;
                        txdiv_buf <= '1';
                        cnt_seg := 0;
                    elsif (divdwn_pulse = '1')
                              and (div_cnt > 0) then
                        div_cnt <= div_cnt - 1;
                        txdiv_buf <= '1';
                        cnt_seg := 0;
                    end if;
                end if;
            end if;
        end if;
    end process;

    txdiv <= txdiv_buf;
    div_data <= conv_std_logic_vector(div_cnt,8);

    process(nrst,clk)
        variable bstart : boolean := false;
        variable cnt_tx : integer := 0;
        variable cnt_rx : integer := 0;
    begin
        if nrst = '0' then
            bstart := false;
            cnt_rx := 0;
            cnt_tx := 0;
            rx_en_buf <= '0';
            tx_en_buf <= '0';
        else
            if rising_edge(clk) then

                if (tx_en_buf = '1') then
                    if (cnt_tx = 10) then
                        tx_en_buf <= '0';
                        cnt_tx := 0;
                    else
                        cnt_tx := cnt_tx + 1;
                    end if;
                end if;

                if (rx_en_buf = '1') then
                    if (cnt_rx = 10) then
                        rx_en_buf <= '0';
```

```
                              cnt_rx := 0;
                         else
                              cnt_rx := cnt_rx + 1;
                         end if;
                    end if;

                    if (start_pulse = '1') then
                         bstart := true;
                         rx_en_buf <= '1';
                    elsif (bstart = true) then
                         if (rxrdy = '1')
                              and (rx_en_buf = '0') then
                              tx_en_buf <= '1';
                              bstart := false;
                         end if;
                    end if;

               end if;
          end if;
     end process;

     rx_en <= rx_en_buf;
     tx_en <= tx_en_buf;

end Behavioral;
```

Листинг 11 – Реализация на языке VHDL приемника данных с АЦП

```
entity oscADCReciever is
  Port (
    rxrdy : out  std_logic;
    wclk : out  std_logic;
    adclk : out std_logic;
    adra : out  std_logic_vector(7 downto 0);
    adrb : out  std_logic_vector(7 downto 0);
    dati : in std_logic_vector(7 downto 0);
    div : in std_logic_vector(7 downto 0);
    start : in  std_logic;
    clk : in std_logic;
    nrst : in std_logic
  );
end oscADCReciever;

architecture Behavioral of oscADCReciever is
  signal swclk : std_logic;
  signal sadclk : std_logic;
  signal srxrdy : std_logic;
  signal sget : std_logic;
  signal sadra: std_logic_vector(7 downto 0);
  signal sadrb: std_logic_vector(7 downto 0);
begin

  process (clk,nrst,srxrdy)
    variable count_clk : integer := 0;
    variable count_div : integer := 2;
```

```
    variable adcwait : integer := 0;
    variable vdat : integer := 0;
    constant dummy_sample : integer := 4;
  begin
    if (nrst = '0') then
      count_clk := 0;
      adcwait := 0;

      srxrdy <= '1';
      swclk <= '0';
      sadclk <= '0';
      sadra <= x"01";
      sadrb <= x"01";
      sget <= '0';
    else
      if rising_edge(CLK) then
        if (start = '1') then
          if (srxrdy = '1') then
            adcwait := 0;
            srxrdy <= '0';
            swclk <= '0';
            sadclk <= '0';
            sget <= '0';
            if div = x"00" then count_div := 2;
            elsif div = x"01" then count_div := 4;
            elsif div = x"02" then count_div := 10;
            elsif div = x"03" then count_div := 20;
            elsif div = x"04" then count_div := 40;
            elsif div = x"05" then count_div := 100;
            elsif div = x"06" then count_div := 200;
            elsif div = x"07" then count_div := 400;
            elsif div = x"08" then count_div := 1000;
            elsif div = x"09" then count_div := 2000;
            elsif div = x"0A" then count_div := 4000;
            elsif div = x"0B" then count_div := 10000;
            elsif div = x"0C" then count_div := 20000;
            elsif div = x"0D" then count_div := 100000;
            elsif div = x"0E" then count_div := 200000;
            elsif div = x"0F" then count_div := 400000;
            end if;
          end if;
        end if;

        if srxrdy = '0' then
          if adcwait = 2*dummy_sample then
            if sadclk = '0' then
               if count_clk = count_div/2 then
                    vdat := conv_integer(signed(dati));
                    if sget = '0' then
                         if (vdat >= 6 or vdat <= -6) then
                              sget <= '1';
                         end if;
                    end if;

                    if sget = '1' then
```

```
                        sadra <= sadra + 1;
                        sadrb <= sadrb + 1;
                    end if;

                end if;
              elsif sadclk = '1' then
                if count_clk = count_div/2 then
                     swclk <= '1';
                elsif count_clk = count_div then
                     swclk <= '0';
                end if;
              end if;
            end if;

            if count_clk = count_div then
              count_clk := 0;
              sadclk <= not sadclk;
              if adcwait /= 2*dummy_sample then
                adcwait := adcwait + 1;
              elsif sadra = x"A0" then
                count_clk := 0;
                adcwait := 0;

                srxrdy <= '1';
                swclk <= '0';
                sadclk <= '0';
                sadra <= x"01";
                sadrb <= x"01";
              end if;
            else
              count_clk := count_clk + 1;
            end if;
          end if;

        end if;
      end if;
    end process;

    rxrdy <= srxrdy;
    wclk <= swclk;
    adclk <= sadclk;
    adra <= sadra when (srxrdy = '0') else (others => 'Z');
    adrb <= sadrb when (srxrdy = '0') else (others => 'Z');

end Behavioral;
```

С помощью разработанных блоков на языке VHDL стало возможным управление быстродействующим АЦП и дальнейшая передача данных в микроконтроллер для последующей обработки.

Для тестирования макет разрабатываемого устройства использовалась структурная схема логического проекта ПЛИС приведенная на рисунке 10.

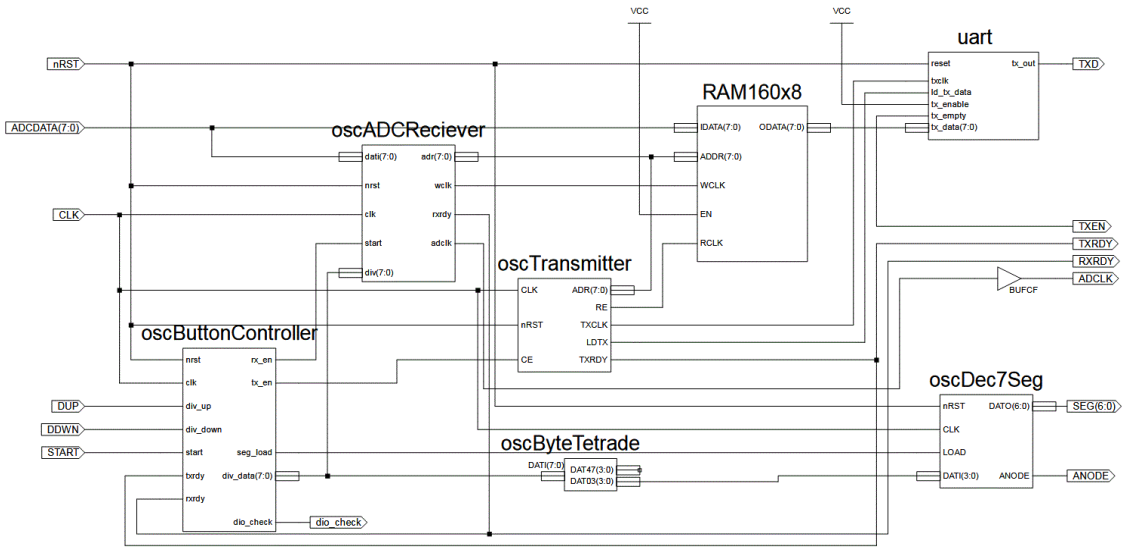

Рисунок 10 – Структурная схема логического проекта макета разрабатываемого устройства для ПЛИС САПР ISE WebPack 11.0

Было также проведено моделирование разработанной структурной схемы логического проекта ПЛИС, а именно моделирование передачи данных по UART, результат которого представлен на рисунке 11, и моделирование снятие данных с АЦП, результат приведен на рисунке 12.

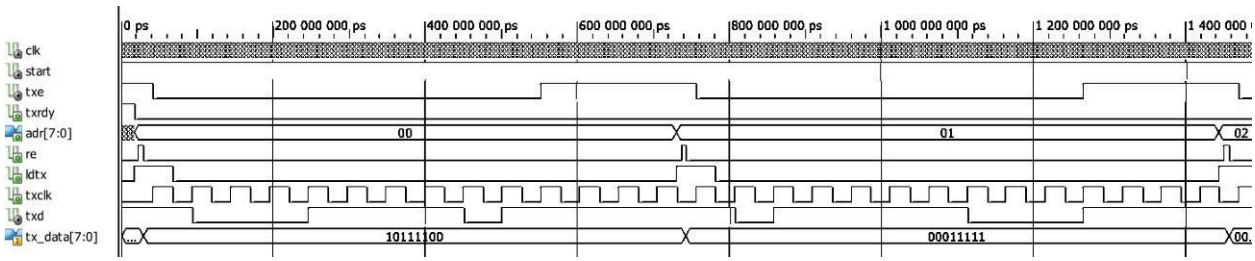

Рисунок 11 – Временные диаграммы, полученные в результате моделирования передачи данных по UART

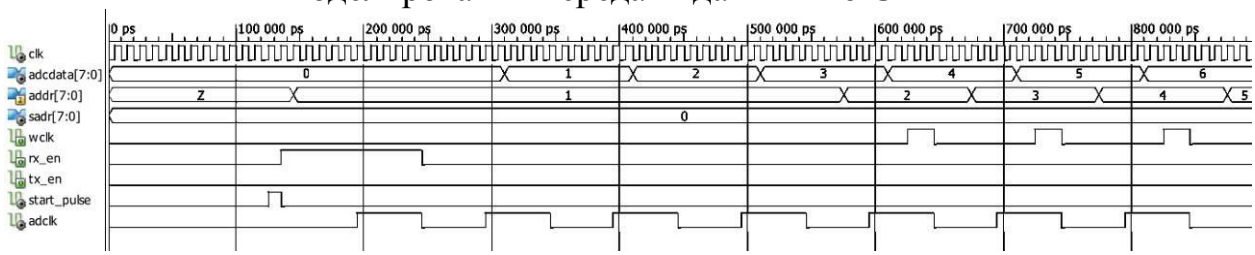

Рисунок 12 – Временные диаграммы, полученные в результате моделирования приема данных с АЦП

В результате моделирования видно, что прием данных по АЦП и передача данных по UART осуществляется в соответствии с необходимыми для них требованиями.

2.4. Разработка структурной схемы двухканального цифрового осциллографа

Структурная схема устройства двухканального цифрового осциллографа приведена на рисунке 13.

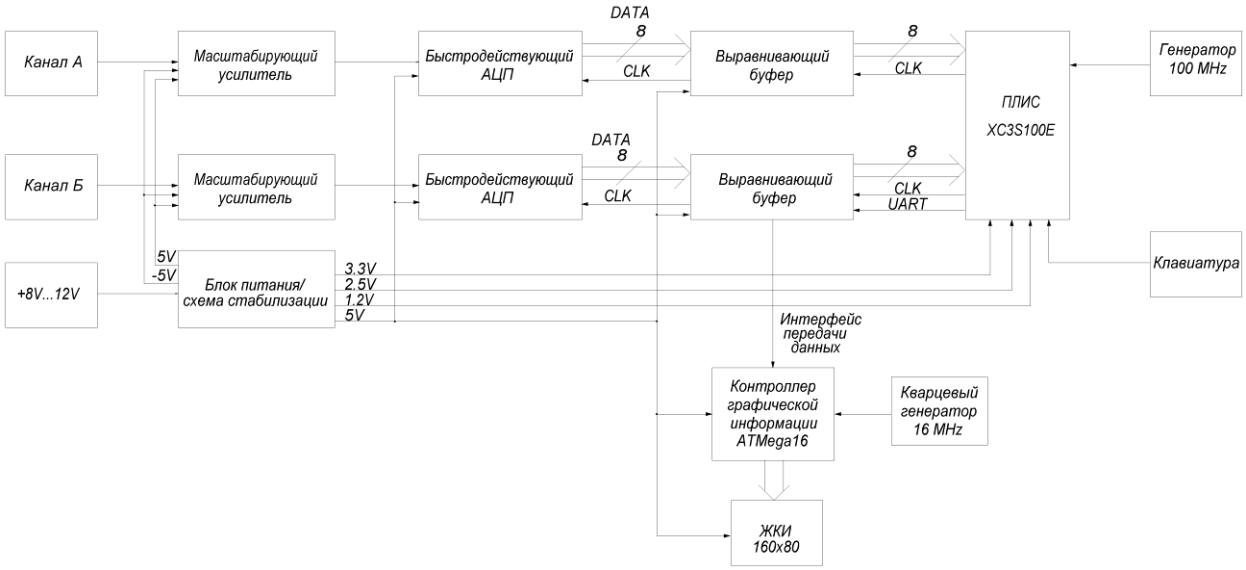

Рисунок 13 – Структурная схема ДЦО

Схема содержит следующие основные блоки: схему блока стабилизации, схему блока входного преобразователя сигнала, АЦП, буфера, схему сбора данных, обработки данных и отображения данных. Блок входного преобразования сигнала – масштабирующий усилитель состоит из схемы высокочастотного операционного усилителя. Задача данного блока – это масштабирование входного сигнала с амплитудой +/-5В в сигнал от +0.6В…+2.6В. Это необходимо для правильной работы АЦП. Уровни сигнала приняты таковыми из-за, того, что АЦП имеет встроенный генератор опорного напряжения с нижним 0.6В и верхним 2.6В, что является для него оптимальными значениями опорного напряжения[1,3].

Быстродействующий АЦП преобразует входной аналоговый сигнал в цифровые коды. Для тактирования АЦП и используется схема, которая прошита в ПЛИС – блок сбора данных. Задачей ПЛИС является задание тактовой частоты АЦП, определения момента начала сбора данных, временное хранение собранных данных и далее передачи их в микроконтроллер – блок обработки данных.

Для того, чтобы согласовать по уровням напряжения микроконтроллер и АЦП (ТТЛ-уровни +5В) с ПЛИС (уровни +3.3В) используется выравнивающий буфер, который на выходе всегда имеет высокий уровень напряжения +3В, а низкий 0В. Причем на вход может быть подано как 3В, так и 5В. Передача данных осуществляется в микроконтроллер по асинхронному интерфейсу приемо-передачи UART, имеющий достаточную для используемого объема данных (160 байт на канал) пропускную способность при скорости 19200 бод. Данный интерфейс при этом прост в реализации, как на микроконтроллере, так и на ПЛИС. Причем, хотя контроллер и имеет встроенный приемо-передатчик UART для этого был можно использовать один из отдельных портов контроллера с программной реализацией интерфейса.

Встроенный приемо-передатчик UART может быть зарезервирован для использования при связи с персональным компьютером через COM порт (для этого необходима расширение, например в виде схемы MAX232) или USB (необходима схема FTDI232). Блок обработки данных принимает из ПЛИС собранную информацию масштабирует ее в соответствии с требованиями вывода информации на дисплей, формирует интерфейс и выводит текущее значение частоты дискретизации. В контроллер зашит драйвер для работы с выбранным ЖКИ, тактирование контроллера происходит от отдельного кварцевого генератора 16МГц.

2.5. Разработка функциональной и принципиальной схемы двухканального цифрового осциллографа

Функциональная схема устройства двухканального цифрового осциллографа приведена на рисунке 14, принципиальная схему устройства приведена на рисунке 15.

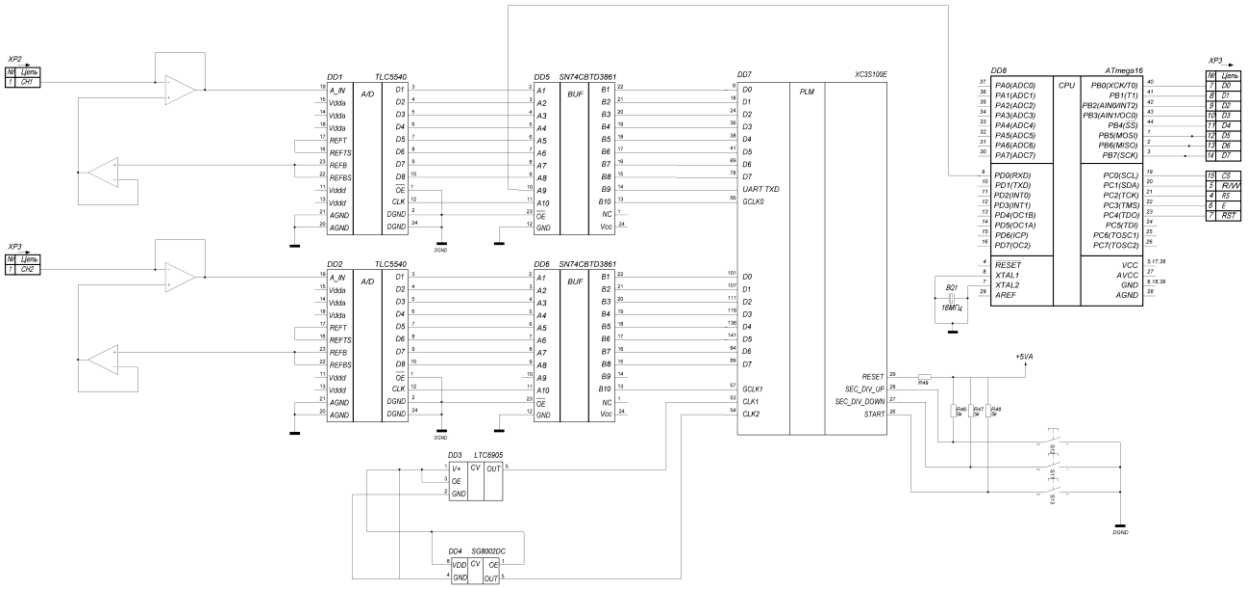

Рисунок 14 – Функциональная схема ДЦО

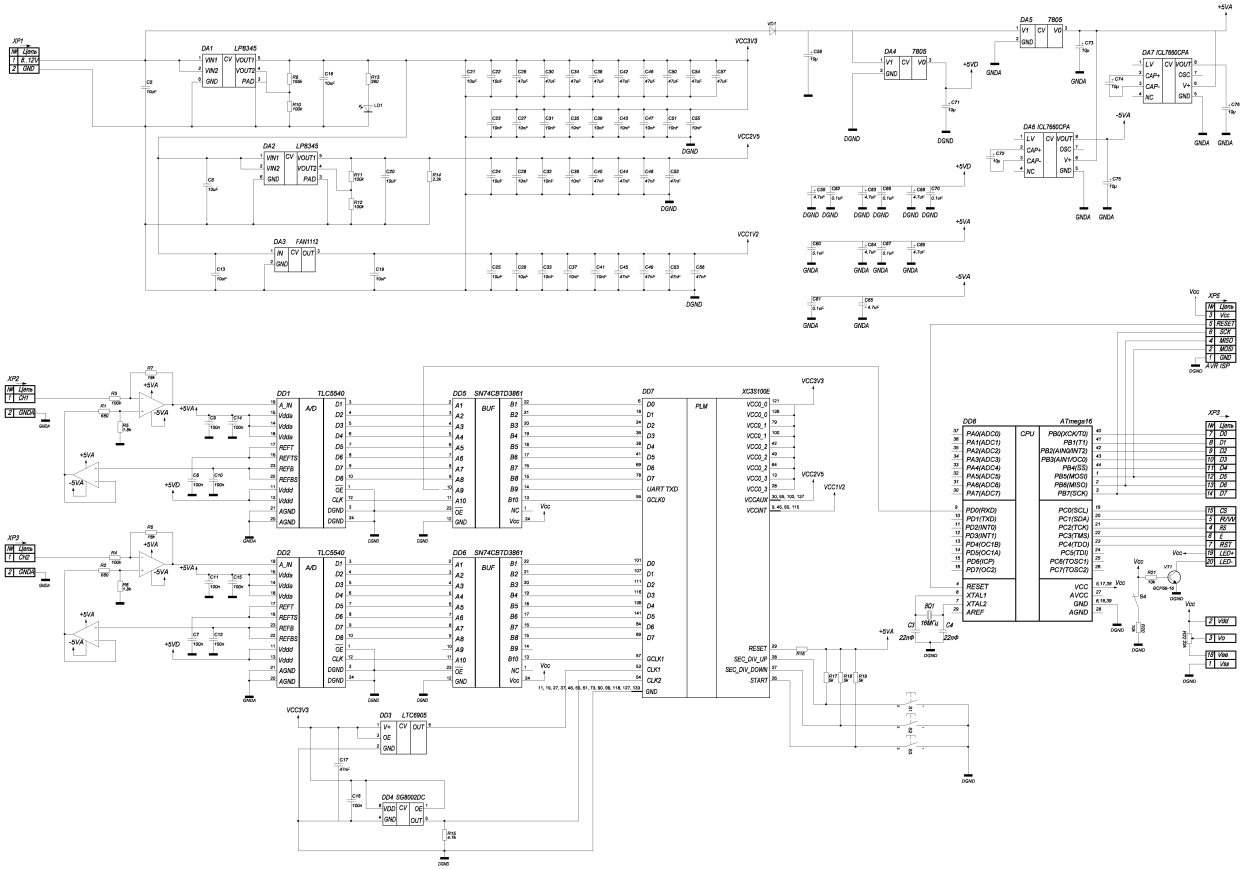

Рисунок 15 – Принципиальная схема ДЦО

Интервал напряжений который можно подать на АЦП с опорными напряжениями получаемыми от самого АЦП составляет от 0,6В до 2,6В, а значения амплитуды входного сигнала составляет от -5В до +5В. Для масштабирования сигнала на входе использован входной масштабирующий усилитель, модель которого приведена на рисунке 16[3].

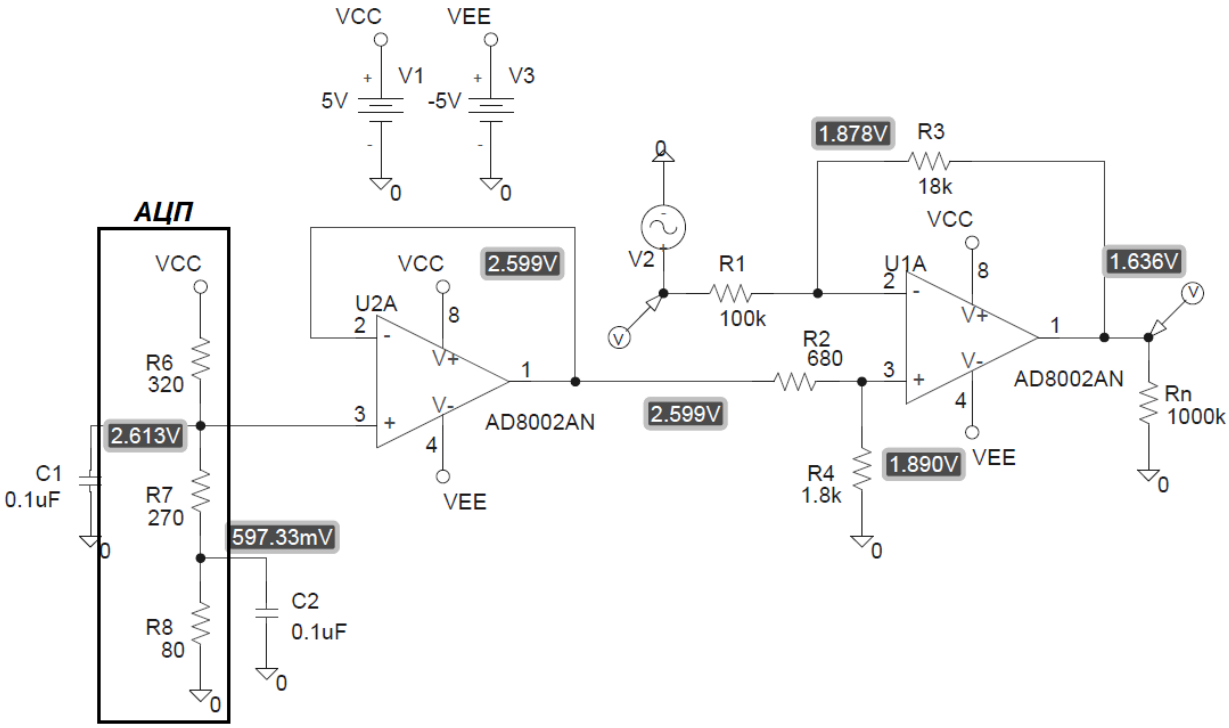

Рисунок 16 – Модель входного каскада ДЦО

Входной масштабирующий усилитель должен обеспечивать передаточную характеристику, которая показан на рисунке 17[1,4].

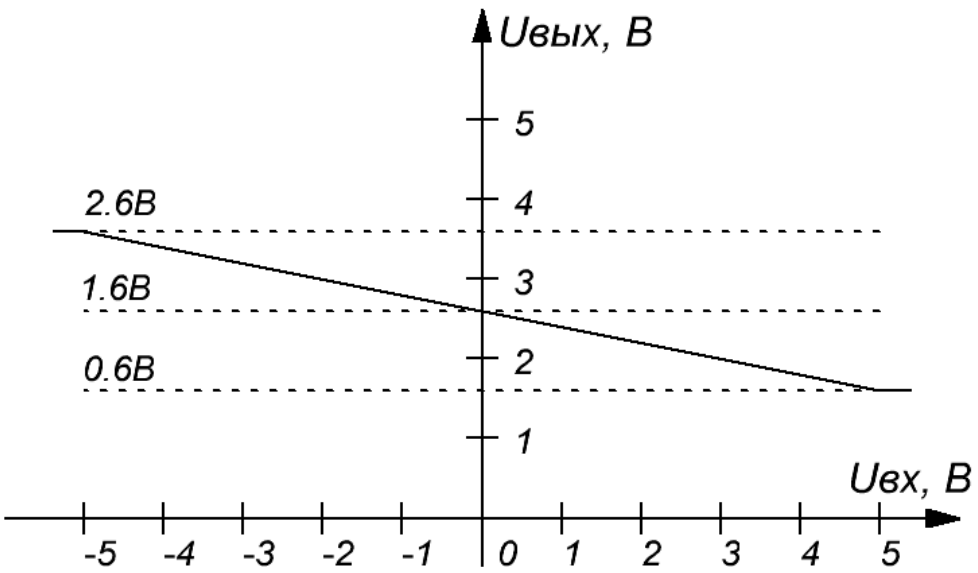

Рисунок 17 – Передаточная характеристика входного масштабирующего усилителя ДЦО

Для расчета значений сопротивлений входного масштабирующего усилителя воспользуемся следующими выражениями (обозначения схемы см. рисунок 16):

Коэффициент усиления схемы входного каскада рассчитывается по формуле (передаточный коэффициент усиления по переменному току):

$$K_{\sim} = -\frac{R_3}{R_1}$$

Сдвиг усилителя можно рассчитать, используя выражение (передаточный коэффициент усиления по постоянному току):

$$K_{-} = 1 + \frac{R_3}{R_1}$$

Напряжение смещения вычисляется, как:

$$U_0 = U_{off} \cdot \frac{R_4}{R_2 + R_4}$$

Для выбора значений сопротивлений резисторов R_3 и R_1 следует выбирать большие значения, для того чтобы входное сопротивление ДЦО было максимальным, так как входное сопротивления инвертирующего усилителя определяется как $Z_{in} = R_{вх} = R_1$. Также примем, чтобы максимальный размах выходного напряжения был не 2,0В, а несколько меньше 1,8В, это необходимо, чтобы из-за неточности номиналов сопротивлений и других факторов, значение выходного напряжения не входило в зону насыщения. Поэтому выберем для $R_1 = 100k$, а для $R_3 = 18k$ с допуском 1%. Соответственно рассчитаем значение коэффициента усиления по постоянному току:

$$K_{-} = 1 + \frac{R_3}{R_1} = 1 + \frac{18k}{100k} = 1.18$$

Воспользуемся для смещения напряжением, генерируем встроенным в АЦП делителем напряжения, при этом предварительно стабилизировав его с помощью повторителя. Напряжение делителя АЦП составляет 2,6В. Тогда:

$$\frac{U_0}{U_{off}} = 1 + \frac{R_4}{R_2} \Rightarrow \frac{R_4}{R_2} = \frac{U_{off}}{U_0} - 1 = \frac{2.6}{1.6} - 1 = 0.625$$

Для такого соотношение сопротивлений выберем $R_4 = 1.8k$, а для $R_2 = 680$ с допуском 1%. Большие значения этих сопротивлений могут привести к тому, что входное сопротивление усилителя станет меньше, чем выходное сопротивление делителя[1].

На рисунке 18, показан результат моделирования входного масштабирующего каскада по переменному току.

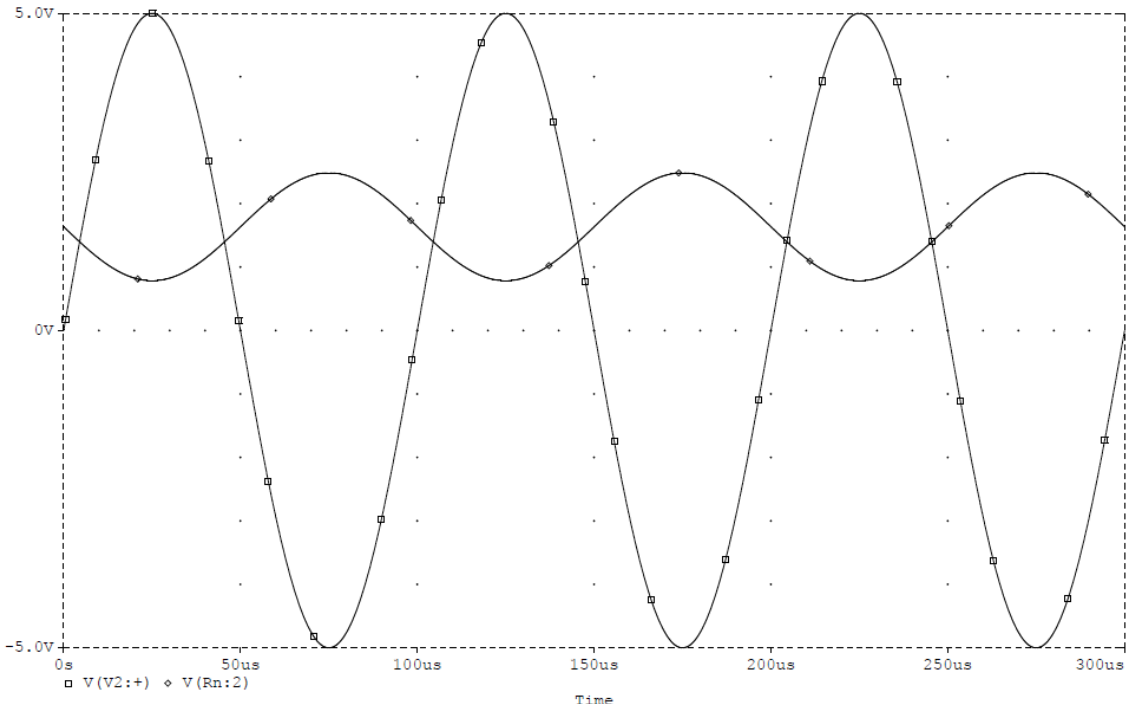

Рисунок 18 – Моделирование входного каскада ДЦО

Из рисунка видно, что усилитель удовлетворяет заданным требованиям и преобразует входной напряжение с амплитудой с 10В и смещением 0В, в выходное напряжение с амплитудой в 1,8В и смещение 1,6В. Также на рисунке 16 приведены напряжение при моделировании схемы на постоянном токе.

2.6. Выбор элементной базы

Для реализации электронной схемы двухканального цифрового осциллографа на базе микроконтроллера и ПЛИС были выбраны следующие элементы: микроконтроллер семейства ATmega16, АЦП TLC5540, буфер SN74CBTD3861, микросхема с двухкаскадными операционными усилителями AD8032 и дисплей для отображения графической информации WG16080A-YYH-V.

Семейство микроконтроллеров ATmega, производимых фирмой Atmel, включает 4х-разрядные, 8-ми разрядные, 16-ти разрядные и 32х-разрядные модели. Для схемы двухканального цифрового осциллографа была выбрана 16ти-разрядная модель семейства (относящаяся к разряду базовых или классических моделей) ATmega16. Основными критериями при выборе этой модели были цена в сочетании с функциональными возможностями микросхемы, а также учитывалось доступность средств программирования и отладки микроконтроллера, наличие технической документации и т.д.

Для того, чтобы получить наибольшую экономическую выгоду от реализации ДЦО, необходимо сочетание минимальных затрат на производство прибора (а в них входит и стоимость элементной базы), максимальных показателей производительности (функциональности) и качества.

Микросхема ATmega16 является одной из наиболее простых в семействе, что обусловило ее низкую стоимость по сравнению с более старшими моделями, при этом она обладает всеми функциями и параметрами, необходимыми для реализации ДЦО. А именно: высокая производительность, достаточное количество выводов (а, следовательно, их хватает для подключения необходимого количества элементов и устройств– ПЛИС, буфер и ЖКИ), условия нормальной работы микроконтроллера не противоречат условиям, в которых будет работать ДЦО (влажность и температура окружающей среды, устойчивость к ударам и вибрациям, которые будут возникать при пользовании ДЦО), а также совместимость с другими элементами (совместимость последовательных интерфейсов

микроконтроллера и микроконтроллера управления ЖКИ), наличие UART. Микросхема является широко распространенной, поэтому не возникает сложностей с ее программированием, отладкой и технической поддержкой.

Основными критериями при выборе ЖКИ были его графические возможности, сложность подключения к внешним контроллерам, наличие собственной символьной таблицы, возможность послойного вывода информации, разрешение и цена. Наилучшим вариантом среди имеющихся стал ЖКИ WG16080A-YYH-V со встроенным микроконтроллером LC7981.

Для преобразования входных сигналов, снимаемых для исследования, необходим АЦП с достаточно высоким быстродействием, небольшими размерами, невысокой стоимостью и простотой в использовании. Для данных целей, с учетом приведенных условий, был выбран АЦП TLC5540, производимый фирмой Texas Instruments. АЦП TLC5540 - быстродействующий 8 битный АЦП, который осуществляет преобразование с производительностью до 40 MSPS. Прибор изготовлен по полупроводниковой КМОП технологии, что позволяет осуществлять высокоскоростное преобразование при малой потребляемой мощности. Типовая ширина полосы пропускания входного аналогового сигнала составляет 75 МГц, что делает этот прибор идеальным для применения в различных системах. Встроенные резисторы позволяют сформировать 2 В опорное напряжение из 5 В напряжения питания, что позволяет снизить количество внешних элементов. Цифровые выходы имеют высокоомное состояние. Прибор TLC5540 для работы требует только однополярного источника питания + 5 В.

Поскольку входы ПЛИС Spartan-3E XS100 и выходы АЦП TLC5540 имеют различные рабочие уровни напряжений (в частности выходное напряжение сигналов TLC5540 выше входных напряжений ПЛИС FPGA Spartan-3E), то было решено использовать 10 битный буфер-преобразователь уровней SN74CBTD3861DW. Буфер SN74CBTD3861 является 10битной высокоскоростной TTL-совместимой шиной коммутации.

Для предварительного усиления входного сигнала необходимы операционные усилители, включенные встречно. Усилитель должен обеспечивать качественное усиление без искажения сигнала, иметь небольшие размеры, обладать низкой стоимостью. Данным условиям удовлетворяет микросхема AD8032, вмещающая в себя два операционных усилителя, питание которых двуполярное, но соединенное на общие входы. Так же отличается низким потреблением питания.

2.7. Описание узлов двухканального цифрового осциллографа

Цифровой двухканальный осциллограф состоит из трех основных функциональных узлов:

1) Узел отображения графической информации
2) Узел обработки информации
3) Узел приема информации.

Рассмотрим подробнее узел отображения информации. Данный узел состоит из жидкокристаллического дисплея WG16080A-YYH-V. WG16080A-YYH-V это универсальный ЖК-контроллер ИС, который может отображать текст и графику на LCD-дисплей. Динамический вывод графики на ЖК-дисплей размером 160x80 пикселей. WG16080A-YYH-V может отображать текст, графику, прокрутку изображения в любом. WG16080A-YYH-V запоминает текст, коды символов и графику во внешний буфер памяти. Основные функции контроллера включают в себя передачу данных от управляющего микропроцессора в буфер памяти, чтения данных из памяти, преобразование данных для отображения на дисплее и генерация временных сигналов для буфера памяти. На рисунке 19 приведена блок-схема ЖК-дисплея с интегрированным контроллером

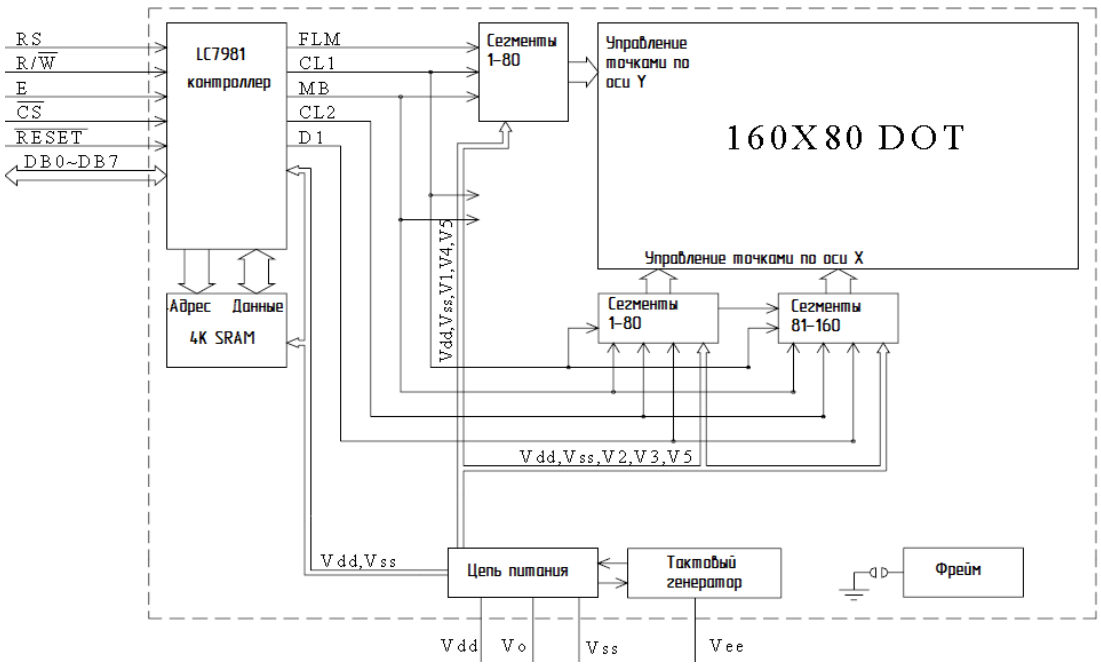

Рисунок 19 – Блок-схема ЖК-дисплея с интегрированным контроллером WG16080A-YYH-V

Данный дисплей с интегрированным контроллером был выбран исходя из того, что необходимо отображать динамическую графическую информацию одновременно со статической. WG16080A-YYH-V работает при напряжении 5 вольт.

На рисунках 20 и 21 приведены временные диаграммы сигналов чтения и записи данных микроконтроллером LC9781.

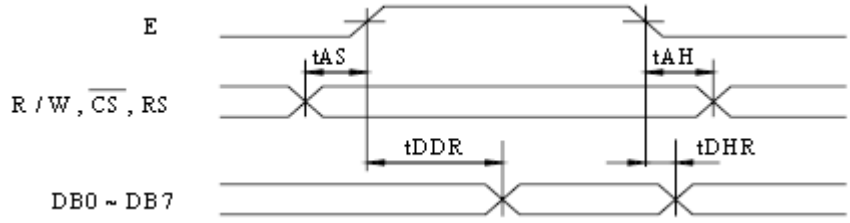

Рисунок 20 – Временная диаграмма чтения данных из памяти

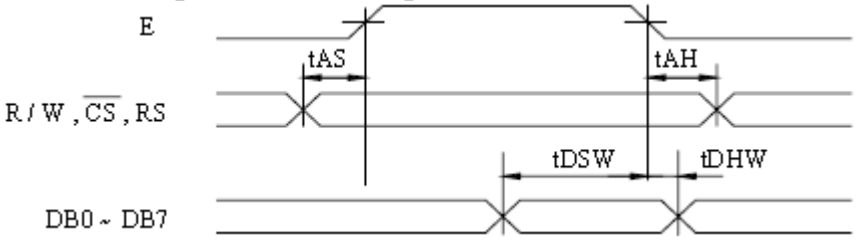

Рисунок 21 – Временная диаграмма записи данных в память

В таблице 1 приведены временные значения для диаграмм чтения и записи данных. Буквы R и W, указанные в скобках, обозначают цикл, чтение или запись.

Таблица 1. Временные значения сигналов чтения и записи.

Наименование	Обозначение	Минимальное значение (нс)	Максимальное значение (нс)
Время установки адреса	tAS	90	
Время удержания адреса	tAH	10	
Время задержки данных (R)	tDDR	-	140
Время удержания данных (R)	tDHR	10	
Время задержки данных (W)	tDSW	220	
Время удержания данных (W)	tDHW	20	

Узел обработки информации состоит из ПЛИС Spartan3E XC3S100E. Данная ПЛИС имеет в своем составе 100 тыс. системных вентилей, 2160

логических ячеек, 4 блока умножения 18x18, 72 Кбит блоки ОЗУ, 15 Кбит распределенная память, 2 блока синхронизации.

Семейство Spartan-3E является дальнейшим развитием семейства Spartan-3. Увеличение отношения логической ёмкости к количеству блоков ввода-вывода позволило существенно снизить себестоимость кристаллов в перерасчете на одну логическую ячейку.

Благодаря своей низкой стоимости, ПЛИС FPGA семейства Spartan-3E идеально подходят для применения в небольших инициативных проектах.

Семейство Spartan-3E может с успехом заменить и превзойти ASIC (Application-Specific Integrated Circuit - специализированная интегральная микросхема). ПЛИС семейства Spartan-3E позволяют сократить сроки разработки, а также обладают большей гибкостью по сравнению с обычными микросхемами ASIC. Кроме того, вследствие программируемости ПЛИС FPGA у разработчика существует возможность вносить изменения в проект в готовом устройстве, не прибегая к замене комплектующих, что также невозможно осуществить, используя ASIC. Далее приведены основные характеристики выбранной ПЛИС:

- Интерфейсные контакты с технологией SelectIO, работающие при различных значениях напряжения и стандартах:
- 376 контактов ввода-вывода или 156 дифференциальных сигнальных пар ввода-вывода;
- Поддержка однополюсных сигнальных стандартов ввода-вывода LVCMOS, LVTTL, HSTL и SSTL;
- Поддержка 3,3-, 2,5-, 1,8-, 1,2-В стандартов ввода-вывода;
- Полнофункциональные дифференциальные вводы-выводы LVDS, RSDS, мини-LVDS и HSTL/SSTL;
- Передача данных со скоростью до 622 Мбит/с по одной дифференциальной паре ввода-вывода;
- Улучшенная поддержка DDR;
- Поддержка DDR SDRAM до 333 Мб/с;

- Гибкие логические ресурсы:
- Мультиплексоры, позволяющие реализовать логические функции более четырех переменных, не используя дополнительных 4-LUT;
- Логика ускоренного переноса;
- Расширенные блоки умножителя, каждый блок 18x18 бит, с возможностью конвейеризации;
- Порт JTAG IEEE 1149,1/1532 для программирования и отладки;
- Иерархическая архитектура памяти SelectRAM:
- Устранение расфазировки синхроимпульсов
- Синтез частот, умножение, деление;
- Фазовый сдвиг с высоким разрешением;
- Широкий спектр частот (от 5 МГц до 333 МГц);
- Восемь глобальных тактовых входов и по восемь дополнительных на каждой половине кристалла ПЛИС;
- Конфигурационный интерфейс для подключения стандартных ППЗУ:
- x8 или x8/x16 NOR флэш-ППЗУ с параллельным интерфейсом;
- Полная поддержка САПР ISE и WebPACK;
- Поддержка 32/64бит, 33МГц PCI;

В узел приема информации входят микроконтроллер ATmega16 и TLC5540.

Рассмотрим микросхему ATmega16. Данная микросхема является 8-разрядным микроконтроллером с 16 Кб внутрисистемно программируемой Flash-памяти. Отличительные особенности данной микросхемы приведены ниже:

- 130 высокопроизводительных команд, большинство команд выполняется за один тактовый цикл
- 32 8-разрядных рабочих регистра общего назначения
- Встроенный 2-цикловый умножитель
- Энергонезависимая память программ и данных

- 16 Кбайт внутрисистемно программируемой Flash памяти (In-System Self-Programmable Flash)
- Внутрисистемное программирование встроенной программой загрузки
- Обеспечен режим одновременного чтения/записи (Read-While-Write)
- 1 Кбайт встроенной SRAM
- Программируемая блокировка, обеспечивающая защиту программных средств пользователя
- Интерфейс JTAG (совместимый с IEEE 1149.1)
- Возможность сканирования периферии, соответствующая стандарту JTAG
- Программирование через JTAG интерфейс: Flash, EEPROM памяти, перемычек и битов блокировки
- Два 8-разрядных таймера/счетчика с отдельным предварительным делителем, один с режимом сравнения
- Один 16-разрядный таймер/счетчик с отдельным предварительным делителем и режимами захвата и сравнения
- Счетчик реального времени с отдельным генератором
- 8-канальный 10-разрядный аналого-цифровой преобразователь
- 8 несимметричных каналов
- 7 дифференциальных каналов (только в корпусе TQFP)
- дифференциальных канала с программируемым усилением в 1, 10 или 200 крат (только в корпусе TQFP)
- Байт-ориентированный 2-проводный последовательный интерфейс
- Программируемый последовательный USART
- Последовательный интерфейс SPI (ведущий/ведомый)
- Встроенный аналоговый компаратор
- Сброс по подаче питания и программируемый детектор кратковременного снижения напряжения питания
- Шесть режимов пониженного потребления: Idle, Power-save, Power-down, Standby, Extended Standby и снижения шумов ADC
- Выводы I/O и корпуса

- 32 программируемые линии ввода/вывода
- Рабочие напряжения 4,5 - 5,5 В
- Рабочая частота 0 - 16 МГц.

На рисунке 22 приведена блок-схема микроконтроллера ATmega16.

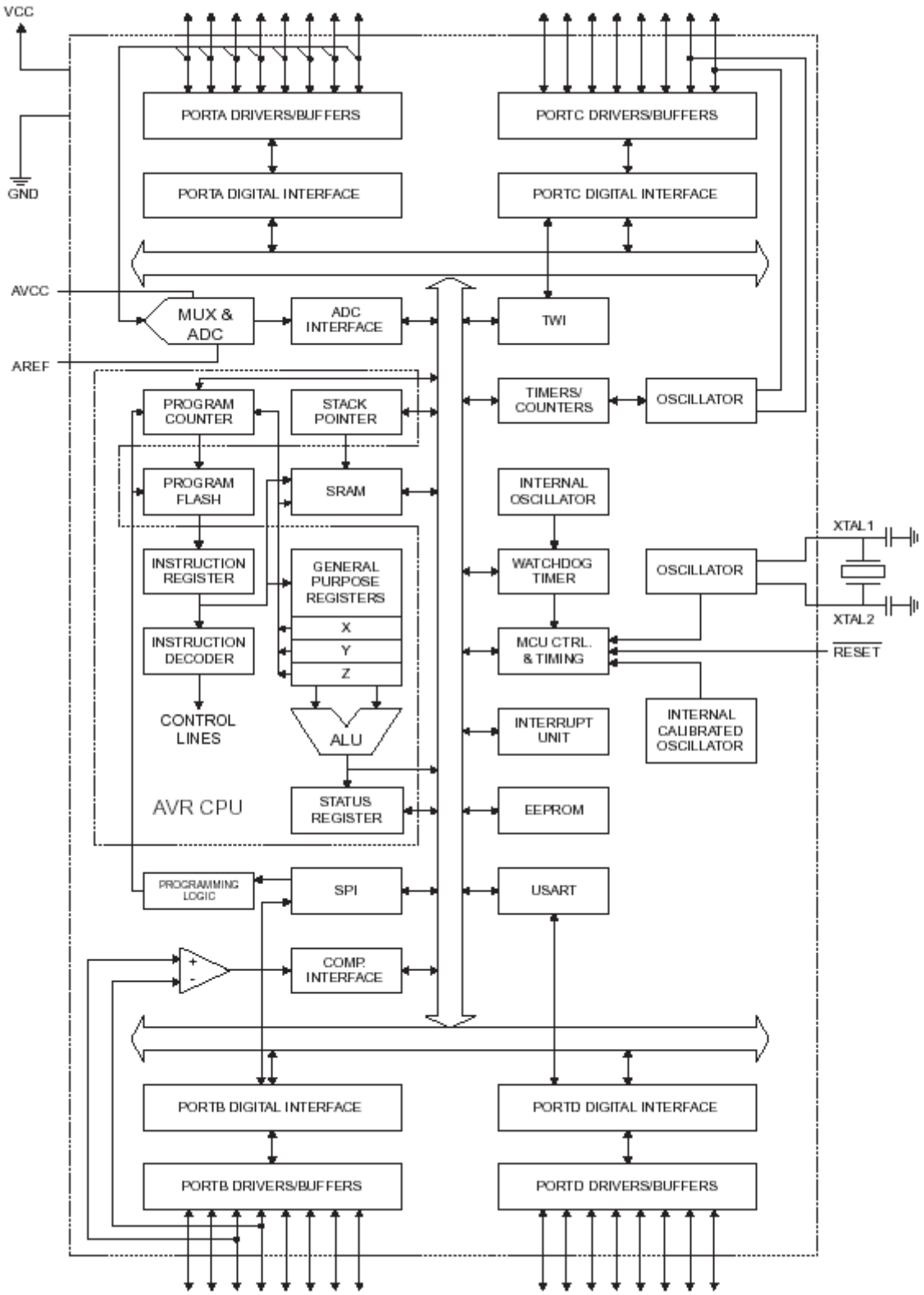

Рисунок 22 – Блок-схема микроконтроллера ATmega16

Далее рассмотрим микроконтроллер TLC5540. Данный микроконтроллер является быстродействующим 8-битным аналого-цифровым преобразователем. Ниже приведены основные характеристики:

- 8 битное разрешение
- Дифференциальная ошибка линейности
- ±0.3 LSB, максимум ±1 LSB при 25°C
- не более ±1 LSB во всем температурном диапазоне
- Максимальная интегральная ошибка линейности
- ±0.6 LSB, максимум ±0.75 при 25°C
- не более ±1 LSB во всем температурном диапазоне
- Максимальная производительность 40 MSPS
- Встроенное УВХ
- Однополярное питание + 5 В
- Типовая потребляемая мощность 85 мВт
- Ширина полосы входного аналогового сигнала 75 МГц
- Встроенные УВХ

TLC5540 - быстродействующий 8 битный АЦП, который осуществляет преобразование с производительностью до 40 MSPS. Прибор изготовлен по полупроводниковой КМОП технологии, что позволяет осуществлять высокоскоростное преобразование при малой потребляемой мощности. Типовая ширина полосы пропускания входного аналогового сигнала составляет 75 МГц, что делает этот прибор идеальным для применения в различных системах. Встроенные резисторы позволяют сформировать 2 В опорное напряжение из 5 В напряжения питания, что позволяет снизить количество внешних элементов. Цифровые выходы имеют высокоомное состояние. TLC5540 для работы требует только однополярного источника питания + 5 В.

Выводы

В данной главе проведена разработка двухканального цифрового осциллографа. Создан алгоритм работы устройства и алгоритм функционирования программного обеспечения. Алгоритмы позволяют выявить ключевые места в работе программного обеспечения и устройства. Написано на языке C++ программное обеспечение для микроконтроллера ATmega16.

3. Расчет параметров двухканального цифрового осциллографа

Надежность РЭА – один из важнейших параметров эксплуатации подобных устройств в различных условиях. Надёжность — свойство объекта сохранять во времени в установленных пределах значения всех параметров, характеризующих способность выполнять требуемые функции в заданных режимах и условиях применения, технического обслуживания, хранения и транспортирования (ГОСТ 27.002—89). В данной главе проводятся расчеты надежности разработанного устройства и потребляемая им мощность.

3.1. Расчет надежности двухканального цифрового осциллографа

Составим перечень элементов электрической схемы. Статистические данные о надежности отдельных элементов сведем в таблицу X.

Таблица 2. Перечень элементов

Позиция	Наименование элемента	Количество	Интенсивность отказов λ_i, 10^{-6}/час
C3 .. C4	КМ5Б, 22 пФ	2	0,01
C9 .. C18	КМ5Б, 100 нФ	10	0,01
C68	КМ5Б, 47 нФ	1	0,01
C69	КМ5Б, 100 нФ	1	0,01
R9	С1-4, 1 Вт	1	0,002
R17 .. R19	С1-4, 1 Вт	3	0,002
R22	С2-23, 1 Вт	1	0,05
R24 .. R26	С1-4, 1 Вт	3	0,002
R29	С2-23, 1 Вт	1	0,05
R37 .. R41	С1-4, 1 Вт	5	0,002
BQ1	Кварц	1	0,05
DD7	ATmega16	1	0,2
DD1	TLC5540	1	0,05
DD3	SN74CBTD	1	0,05
DD2	TLC5540	1	0,05
DD4	SN74CBTD	1	0,05
	ЖК-индикатор	1	0,05
			$\Sigma\lambda_i = 7{,}22 \times 10^{-6}$

Рассчитаем вероятность безотказной работы за время t = 500 часов

$P = e^{-\sum_{i=1}^{n}\lambda_i t} = e^{-0{,}000515} = 0{,}995871.$

Средняя наработка на отказ равна

$$\int_0^\infty e^{-\sum_{i=1}^{n}\lambda_i t}\,dt = -\frac{1}{\sum\lambda_i} e^{-\sum\lambda_i t}\Big|_0^\infty = \frac{1}{\sum\lambda_i} \approx 10^6\ ч$$

3.2. Расчет потребляемой мощности двухканальным цифровым осциллографом

По принципиальной электрической схеме основными устройствами, потребляющими мощность, являются:

1. Контроллер ATmega16 (потребляемая мощность определяется по документации)
2. Контроллер SN74CBTD (потребляемая мощность определяется по документации)
3. Контроллер TLC5540 (потребляемая мощность определяется по документации)
4. ЖК-индикатор (потребляемая мощность определяется по документации)
5. Резисторы

Сведем в таблицу 3 исходные данные для расчета мощности:

Таблица 3. Исходные данные для расчета мощности

Элемент	Исходные данные
R9	10 кОм
R16…R19	10 кОм
R20	30 кОм
R21	10 кОм
R22	30 кОм
R2…R26	10 кОм
R27	30 кОм
R28	10 кОм
R29	30 кОм
R36	4.7 кОм
R37…R41	5 кОм

Дискретными компонентами являются резисторы.

Будем считать, что низкий уровень сигнала на входе схемы по D+ будет определяться малым напряжением 0,3 В. Тогда максимально возможная потребляемая резисторами R9, R16 .. R37 мощность при U_{ref}=3 В будет равна:

$$P_{R2} = \frac{(U_{ref} - U_{D+})}{R9} = 0{,}00274 \text{ Вт}$$

Аналогично рассчитываем другие резисторы.

Расчет мощности, потребляемой микросхемами

Мощность, потребляемая контроллером ATmega16, равна:

$P_{D1} = U_{пит.} I_{потр.} = 5$ В*20 мА = 100 мВт

Мощность, потребляемая контроллером SN74CBTD, равна:

$P_{D2} = U_{пит.} I_{потр.} = 5$ В*128 мкА = 0.75 мВт.

Так как их две, то общая мощность равна 1.5 мВт.

Мощность, потребляемая контроллером SN74CBTD, равна:

$P_{D3} = U_{пит.} I_{потр.} = 5$В * 75 мкА = 0.375 мВт. Так кА их тоже две, то суммарная мощность будет равна 0.75 мВт.

Мощность, потребляемая ЖК-индикатором равна:

$P_{ЖК} = U_{пит.} I_{потр.} = 5$В * 0.35 мА = 1.75 мВт

Расчет суммарной потребляемой мощности

Суммарная потребляемая мощность равна: $P = \Sigma P_i = 279{,}164$ мВт

Выводы

Как видим, такая надежность системы полностью удовлетворяет требованиям технического задания, в котором требуемая надежность в течение 500 часов работа равна 0,90.

Данный контроллер потребляет относительно небольшую мощность и может поддерживать питание от шины USB, не используя вторичные источники электропитания.

4. Экспериментальное исследование разработанного двухканального цифрового осциллографа

Экспериментальное исследование разработанных устройств является одним из важных этапов создания устройства. Данный этап позволяет протестировать все основные функции разработанного устройства, проверить выполняемость алгоритма, выявить ошибки, допущенные в процессе написания программного обеспечения, проверить эргономику устройства.

4.1. Разработка методики тестирования макета двухканального цифрового осциллографа

Для корректной работы ДЦО необходимо провести тестирование осциллографа. Тестирование должно включать в себя проверку всех функциональных возможностей ДЦО – проверка исследованием одного и двух сигналов в режимах одного и двух каналов, исследование сигнала с изменением временной развертки, исследование сигналов различной формы (пилообразный сигнал, синусоидальный и прямоугольный), исследование сигналов разной амплитуды. Блок схема тестирования приведена на рисунке 23.

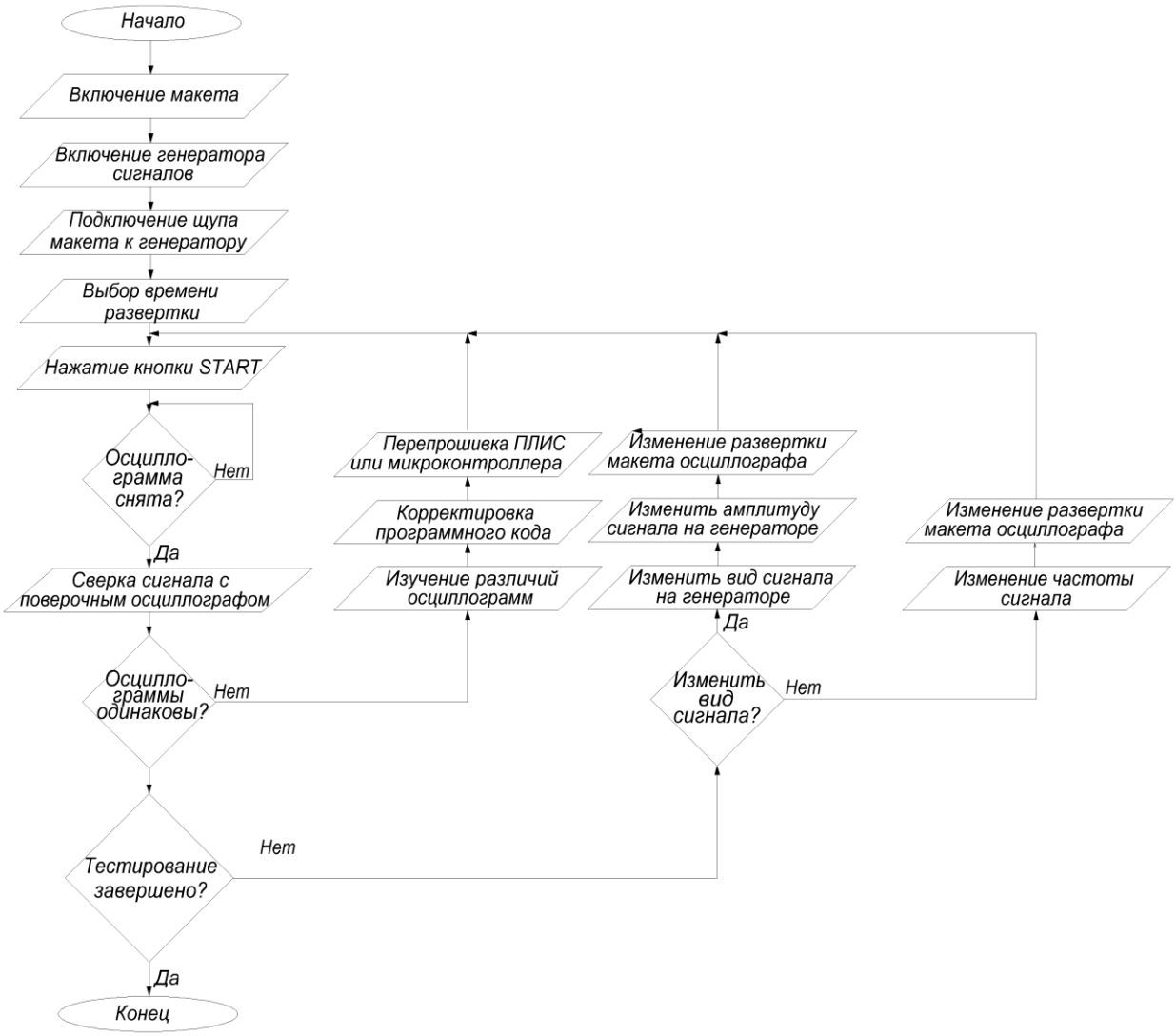

Рисунок 23 – Блок схема тестирования макета устройства

Для проверки работоспособности макета ДЦО необходим генератор сигналов и поверочный осциллограф. Далее выполняются следующие действия:

1) При помощи кнопок управления на макете ДЦО выставляется минимальная развертка;
2) На генераторе сигналов выставляется амплитуда сигнала в 2.5В, прямоугольная форма сигнала и частота, соответствующая половине частоты развертки выбранного режима для получения качественной картинки и оценки работы макета ДЦО;
3) Щуп макета ДЦО подключается к генератору сигналов необходимо нажать кнопку RESET для начала сбора сигнала и вывода его на ЖКИ. Исследуемый сигнал будет высвечиваться на дисплее до повторного нажатия кнопки RESET;
4) Для поверки снятого сигнала, щуп поверочного осциллографа подключается к генератору сигналов и снимается эталонный сигнал;
5) Сверка сигнала снятого при помощи поверочного осциллографа с сигналом снятым при помощи макета ДЦО.

Действия с 1-го по 5-е повторяются. При каждом повторении изменяется развертка макета ДЦО в сторону уменьшения и изменяется генерируемый сигнал по частоте. Затем действия с 1-го по 5-е повторяются для сигнала пилообразной формы и синусоидальной формы. После чего тестирование повторяется для сигналов с амплитудой 5В.

4.2. Тестирование работоспособности макета двухканального цифрового осциллографа

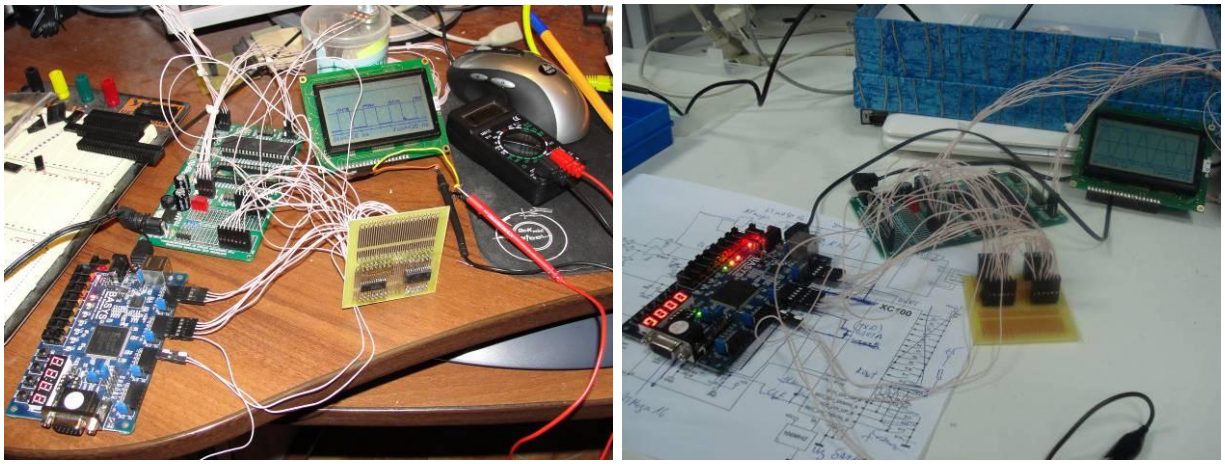

Рисунок 24 – Общий вид макета разрабатываемого устройства

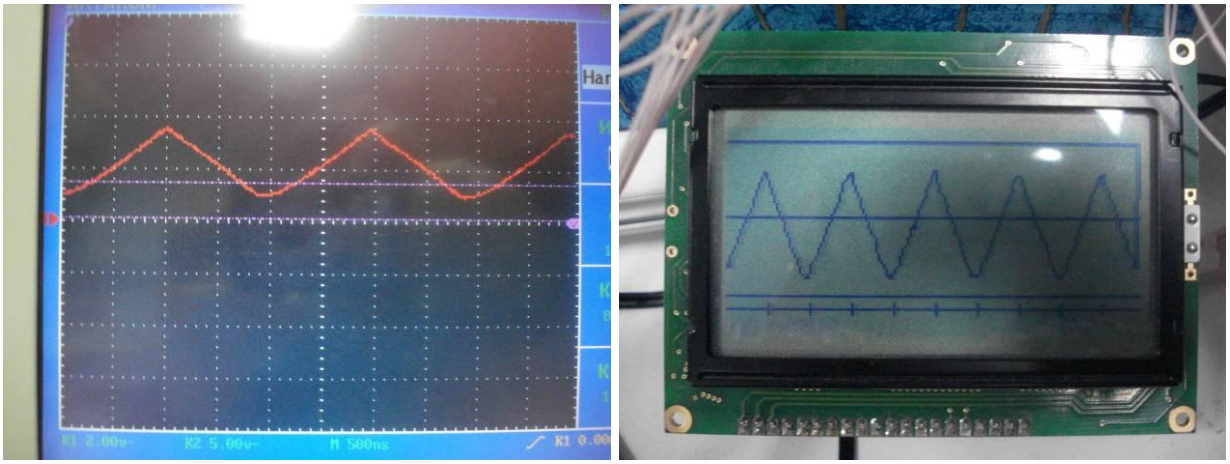

Рисунок 25 – Вид пилообразного сигнала оцифрованного с помощью лабораторного и разрабатываемого осциллографа

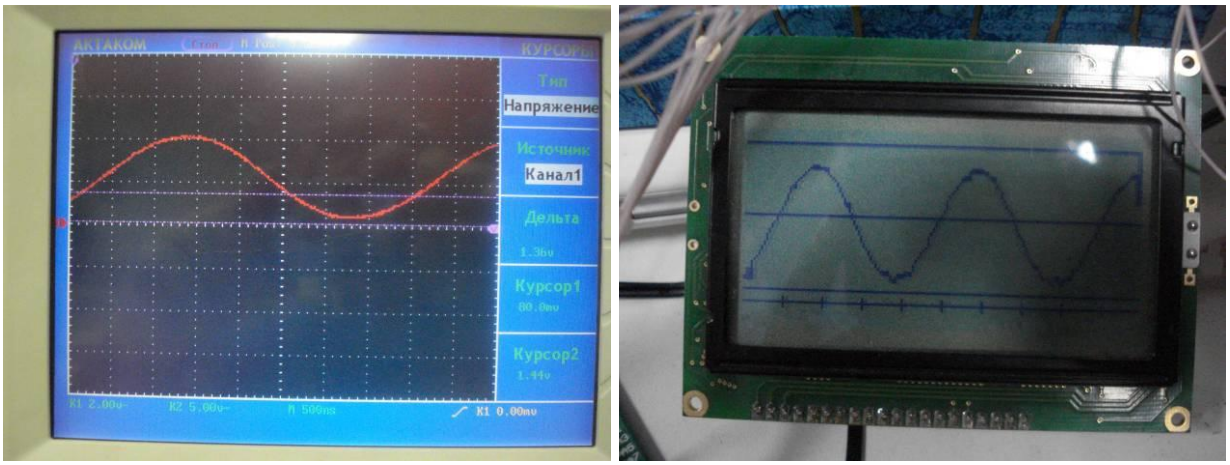

Рисунок 26 – Вид синусоидального сигнала оцифрованного с помощью лабораторного и разрабатываемого осциллографа

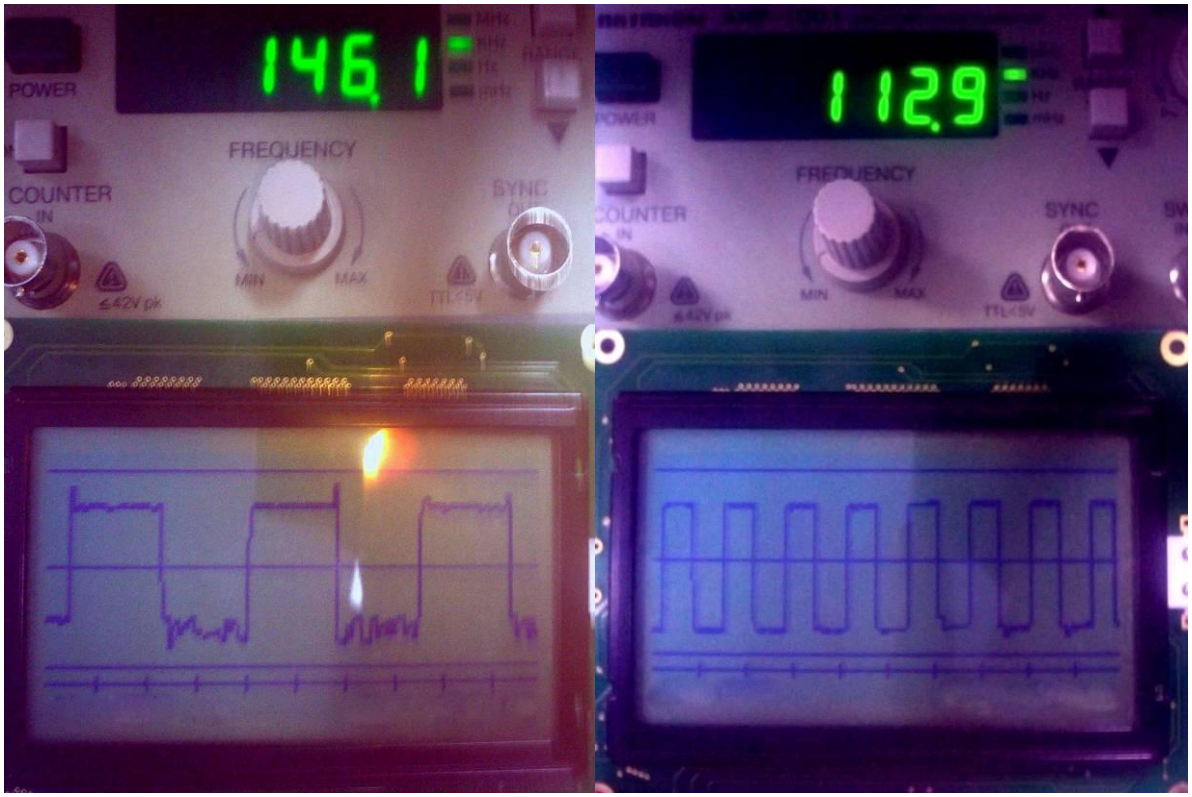

Рисунок 27 – Меандр частотой порядка 100 – 150 КГц, снятый с частотами дискретизации 5.0 МГц и 2.5 МГц

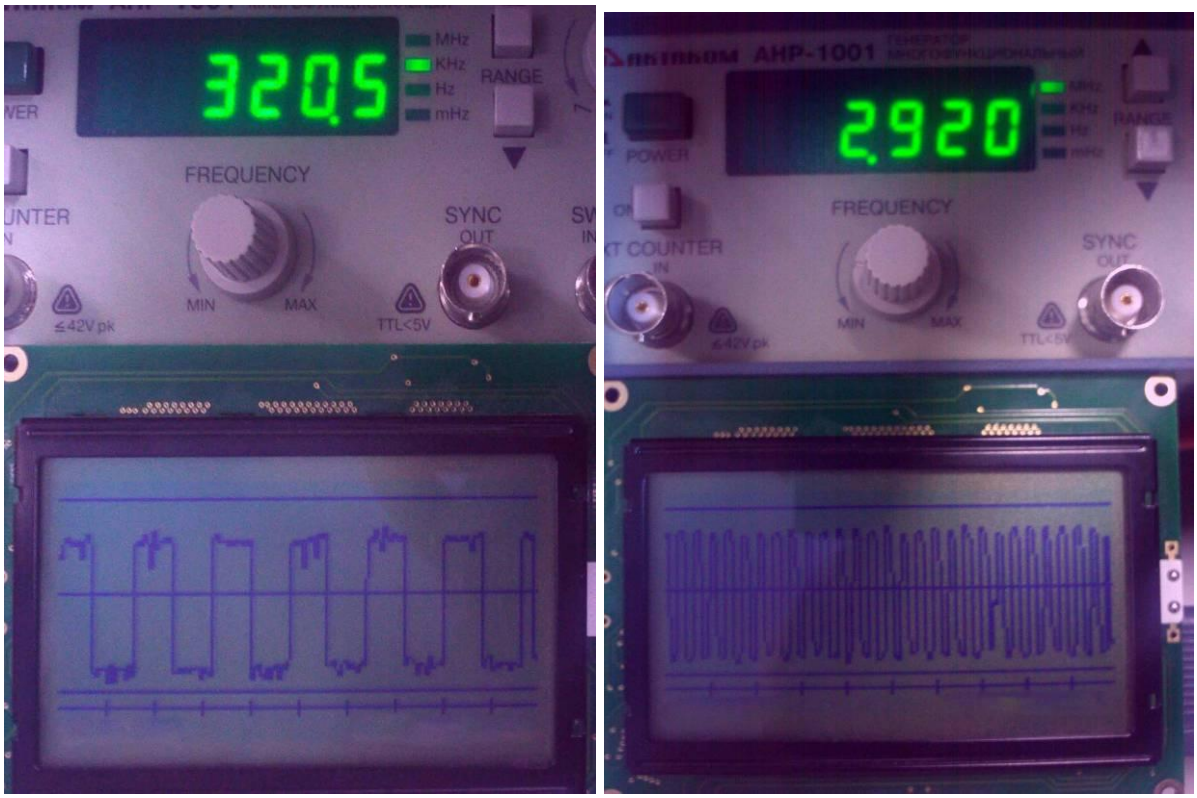

Рисунок 28 – Меандр частотами порядка 320 КГц и 3 МГц, снятый с частотами дискретизации 12.5 МГц и 25 МГц

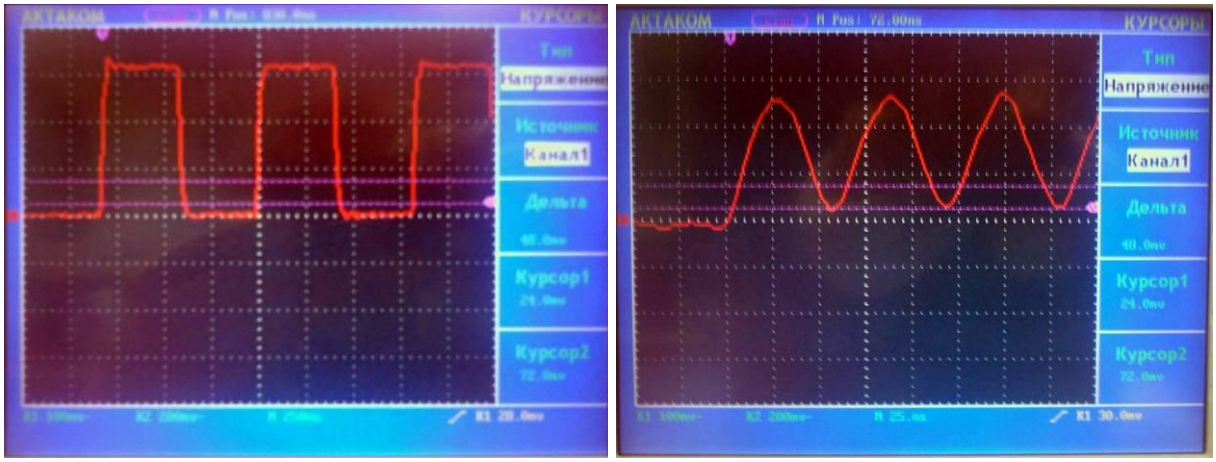

Рисунок 29 – Тактовый сигнал генерируемый ПЛИС для получения данных с микросхемы АЦП при 1.0 МГц и 25.0 МГц

Выводы

В данной главе была разработана методика тестирования ДЦО. Методика позволяет полностью протестировать все рабочие режимы разработанного ДЦО и проверить работоспособность ДЦО, а так же его соответствие техническому заданию. Проведенная проверка макета ДЦО в соответствии с разработанной методикой тестирования показала, что разработанный ДЦО полностью соответствует техническому заданию, позволяет исследовать сигналы с частотой до 3 МГц, амплитудой до 10В любой формы (пилообразной, прямоугольной и синусоидальной). Обязательное тестирование ДЦО в соответствии с разработанной методикой позволяет выявить все неточности.

Заключение

Разработанный двухканальный цифровой осциллограф является незаменимым устройством для инициативных групп и индивидуальных разработчиков, в случаях, когда нет необходимости в дорогих решениях с большим набором функций, либо нет финансовых средств для их приобретения. Разработанный осциллограф обладает всеми необходимыми функциями – измерение сигналов различной формы, достаточно широкого диапазона частот, достаточная амплитуда входного сигнала, временная развертка. Также к достоинствам относятся небольшие габариты, возможность одновременного исследования двух каналов, низкая стоимость компонентов и простота реализации двухканального цифрового осциллографа.

Для экспериментального исследования был собран и протестирован макет двухканального цифрового осциллографа. Макет обладал характеристиками аналогичными характеристикам разработанного двухканального цифрового осциллографа. Экспериментальные исследования показали, что собранный макет отвечает необходимым требованиям и может использоваться для исследования широкого спектра сигналов, а значит и разработанный двухканальный цифровой осциллограф так же отвечает требованиям.

В дальнейшем необходимо расширить функциональные возможности разработанного двухканального цифрового осциллографа как в программном плане, так и в плане эргономики и исследуемых сигналов. Рекомендуется увеличить напряжение исследуемого сигнала, получать исследуемый сигнал на дисплее более высокого разрешения, создать схему автономного питания от элементов питания АА либо ААА, либо иных, разработать эргономичный и простой интерфейс управления двухканальным цифровым осциллографом.

Список использованных источников

1. А.С.Кузнецов, Трехканальный осциллограф – М.:Энергия, 1999
2. Коллектив авторов, Отечественные осциллографы – М.:Энергия, 2007
3. Коллектив авторов, Метрологическое оборудование – Челябинск, 2007
4. П. Хоровиц, У. Хилл. Искусство схемотехники – М.: Мир, 1998
5. Суворова Е., Шейнин Ю. Проектирование цифровых систем на VHDL. - СПб.:BHV, 2003
6. Бибило П.Н. Основы языка VHDL – М.: Солон-Р, 2002
7. http://www.atmel.com/
8. http://www.ti.com/
9. Белов А.В. Самоучитель разработчика устройств на микроконтроллерах AVR – СПб.: Наука и Техника, 2008
10. Конспект лекций по курсу «Конструкторское проектирование электронной аппаратуры», Шерстнев В.В, 2008 г.
11. Конспект лекций по курсу «Системотехника ЭВС, комплексы и сети», Шпиев В.А., 2009 г.
12. http://www.chip-dip.ru
13. http://www.adclab.ru
14. http://www.atmel.com
15. http://www.aldatasheet.com
16. http://www.eosystems.ro/eoscope/eoscope_en.htm
17. http://vellman.com

Приложения

Приложение 1. Листинг прошивки для ATmega16 с кодами цифр, букв и знаков для вывода на ЖКИ.

```
/*! \file font5x7.h \brief Graphic LCD Font (Ascii Characters). */

#ifndef FONT5X7_H
#define FONT5X7_H

#include <avr/pgmspace.h>

// standard ascii 5x7 font
// defines ascii characters 0x20-0x7F (32-127)
//static unsigned char __attribute__ ((progmem)) Font5x7[] = {
static unsigned char Font5x7[] PROGMEM = {
      0x00, 0x00, 0x00, 0x00, 0x00,// (space)
      0x00, 0x00, 0x5F, 0x00, 0x00,// !
      0x00, 0x07, 0x00, 0x07, 0x00,// "
      0x14, 0x7F, 0x14, 0x7F, 0x14,// #
      0x24, 0x2A, 0x7F, 0x2A, 0x12,// $
      0x23, 0x13, 0x08, 0x64, 0x62,// %
      0x36, 0x49, 0x55, 0x22, 0x50,// &
      0x00, 0x05, 0x03, 0x00, 0x00,// '
      0x00, 0x1C, 0x22, 0x41, 0x00,// (
      0x00, 0x41, 0x22, 0x1C, 0x00,// )
      0x08, 0x2A, 0x1C, 0x2A, 0x08,// *
      0x08, 0x08, 0x3E, 0x08, 0x08,// +
      0x00, 0x50, 0x30, 0x00, 0x00,// ,
      0x08, 0x08, 0x08, 0x08, 0x08,// -
      0x00, 0x60, 0x60, 0x00, 0x00,// .
      0x20, 0x10, 0x08, 0x04, 0x02,// /
      0x3E, 0x51, 0x49, 0x45, 0x3E,// 0
      0x00, 0x42, 0x7F, 0x40, 0x00,// 1
      0x42, 0x61, 0x51, 0x49, 0x46,// 2
      0x21, 0x41, 0x45, 0x4B, 0x31,// 3
      0x18, 0x14, 0x12, 0x7F, 0x10,// 4
      0x27, 0x45, 0x45, 0x45, 0x39,// 5
      0x3C, 0x4A, 0x49, 0x49, 0x30,// 6
      0x01, 0x71, 0x09, 0x05, 0x03,// 7
      0x36, 0x49, 0x49, 0x49, 0x36,// 8
      0x06, 0x49, 0x49, 0x29, 0x1E,// 9
      0x00, 0x36, 0x36, 0x00, 0x00,// :
      0x00, 0x56, 0x36, 0x00, 0x00,// ;
      0x00, 0x08, 0x14, 0x22, 0x41,// <
      0x14, 0x14, 0x14, 0x14, 0x14,// =
      0x41, 0x22, 0x14, 0x08, 0x00,// >
      0x02, 0x01, 0x51, 0x09, 0x06,// ?
      0x32, 0x49, 0x79, 0x41, 0x3E,// @
      0x7E, 0x11, 0x11, 0x11, 0x7E,// A
      0x7F, 0x49, 0x49, 0x49, 0x36,// B
      0x3E, 0x41, 0x41, 0x41, 0x22,// C
      0x7F, 0x41, 0x41, 0x22, 0x1C,// D
      0x7F, 0x49, 0x49, 0x49, 0x41,// E
      0x7F, 0x09, 0x09, 0x01, 0x01,// F
      0x3E, 0x41, 0x41, 0x51, 0x32,// G
      0x7F, 0x08, 0x08, 0x08, 0x7F,// H
      0x00, 0x41, 0x7F, 0x41, 0x00,// I
      0x20, 0x40, 0x41, 0x3F, 0x01,// J
      0x7F, 0x08, 0x14, 0x22, 0x41,// K
      0x7F, 0x40, 0x40, 0x40, 0x40,// L
      0x7F, 0x02, 0x04, 0x02, 0x7F,// M
      0x7F, 0x04, 0x08, 0x10, 0x7F,// N
      0x3E, 0x41, 0x41, 0x41, 0x3E,// O
```

```
        0x7F, 0x09, 0x09, 0x09, 0x06,// P
        0x3E, 0x41, 0x51, 0x21, 0x5E,// Q
        0x7F, 0x09, 0x19, 0x29, 0x46,// R
        0x46, 0x49, 0x49, 0x49, 0x31,// S
        0x01, 0x01, 0x7F, 0x01, 0x01,// T
        0x3F, 0x40, 0x40, 0x40, 0x3F,// U
        0x1F, 0x20, 0x40, 0x20, 0x1F,// V
        0x7F, 0x20, 0x18, 0x20, 0x7F,// W
        0x63, 0x14, 0x08, 0x14, 0x63,// X
        0x03, 0x04, 0x78, 0x04, 0x03,// Y
        0x61, 0x51, 0x49, 0x45, 0x43,// Z
        0x00, 0x00, 0x7F, 0x41, 0x41,// [
        0x02, 0x04, 0x08, 0x10, 0x20,// "\"
        0x41, 0x41, 0x7F, 0x00, 0x00,// ]
        0x04, 0x02, 0x01, 0x02, 0x04,// ^
        0x40, 0x40, 0x40, 0x40, 0x40,// _
        0x00, 0x01, 0x02, 0x04, 0x00,// `
        0x20, 0x54, 0x54, 0x54, 0x78,// a
        0x7F, 0x48, 0x44, 0x44, 0x38,// b
        0x38, 0x44, 0x44, 0x44, 0x20,// c
        0x38, 0x44, 0x44, 0x48, 0x7F,// d
        0x38, 0x54, 0x54, 0x54, 0x18,// e
        0x08, 0x7E, 0x09, 0x01, 0x02,// f
        0x08, 0x14, 0x54, 0x54, 0x3C,// g
        0x7F, 0x08, 0x04, 0x04, 0x78,// h
        0x00, 0x44, 0x7D, 0x40, 0x00,// i
        0x20, 0x40, 0x44, 0x3D, 0x00,// j
        0x00, 0x7F, 0x10, 0x28, 0x44,// k
        0x00, 0x41, 0x7F, 0x40, 0x00,// l
        0x7C, 0x04, 0x18, 0x04, 0x78,// m
        0x7C, 0x08, 0x04, 0x04, 0x78,// n
        0x38, 0x44, 0x44, 0x44, 0x38,// o
        0x7C, 0x14, 0x14, 0x14, 0x08,// p
        0x08, 0x14, 0x14, 0x18, 0x7C,// q
        0x7C, 0x08, 0x04, 0x04, 0x08,// r
        0x48, 0x54, 0x54, 0x54, 0x20,// s
        0x04, 0x3F, 0x44, 0x40, 0x20,// t
        0x3C, 0x40, 0x40, 0x20, 0x7C,// u
        0x1C, 0x20, 0x40, 0x20, 0x1C,// v
        0x3C, 0x40, 0x30, 0x40, 0x3C,// w
        0x44, 0x28, 0x10, 0x28, 0x44,// x
        0x0C, 0x50, 0x50, 0x50, 0x3C,// y
        0x44, 0x64, 0x54, 0x4C, 0x44,// z
        0x00, 0x08, 0x36, 0x41, 0x00,// {
        0x00, 0x00, 0x7F, 0x00, 0x00,// |
        0x00, 0x41, 0x36, 0x08, 0x00,// }
        0x08, 0x08, 0x2A, 0x1C, 0x08,// ->
        0x08, 0x1C, 0x2A, 0x08, 0x08 // <-
};

#endif
```

Приложение 2. Листинг программы для микроконтроллера ЖКИ.

```
/* lc7981.c: Source code for the LC7981/HD61830 graphics lcd driver.
 * The hardware port defines can be found in lc7981.h. */

#include "lc7981.h"
#include "graphics.h"

/**
 * Strobes the Enable control line to trigger the lcd to process the
 * transmitted instruction.
 */
void lcd_strobe_enable(void) {
      lcd_enable_high();
      __asm("nop;"); __asm("nop;"); __asm("nop;");
      lcd_enable_low();
      __asm("nop;"); __asm("nop;"); __asm("nop;");
}

/**
 * Waits for the busy flag to clear, which should take
 * around the maximum time for an instruction to complete.
 * Note, LCD operation is kind of sensitive to this configuration. If the
delay
 * is too fast, the LCD will miss some pixels when it is really put through
 * a stress test. This dela time seems to work great.
void lcd_wait_busy(void) {
      _delay_us(3);
}

 * Older implementation of lcd_wait_busy() that checked the busy flag in
 * hardware. I found that it always hanged after plotting a byte to the
screen,
 * so I took up the delay version above.
 *
 */
void lcd_wait_busy(void) {
      char counter = 0;
      unsigned char data;
      // Set RW and RS high
      lcd_rw_high();
      lcd_rs_high();
      __asm("nop;"); __asm("nop;"); __asm("nop;");
      // Wait until busy flag on last bit of the data port clears.
      do {
            counter++;
            lcd_enable_high();
            __asm("nop;"); __asm("nop;"); __asm("nop;");
            data = LCD_DATA_PORT;
            __asm("nop;"); __asm("nop;"); __asm("nop;");
            lcd_enable_low();
            __asm("nop;"); __asm("nop;"); __asm("nop;");
      //    LED_Blic(100);
      } while (data & 0x80 && counter <= 100);
      if (counter == 100) _delay_us(3);
}

/**
 * Writes a raw instruction to the LCD.
 * @param command The 4-bit instruction code.
 * @param data The 8-bit paramater/data to the specified instruction.
 */
void lcd_write_command(unsigned char command, unsigned char data) {
      /* Wait for the busy flag to clear */
      lcd_wait_busy();
```

```
	/* Set RW low, RS high to write the instruction command */
	lcd_rw_low();
	lcd_rs_high();
	/* Instruction commands are a maximum of 4 bits long, so
	 * just mask off the rest. */
	LCD_DATA_PORT = (command&0x0F);
	__asm("nop;"); __asm("nop;"); __asm("nop;");
	__asm("nop;"); __asm("nop;"); __asm("nop;");
	lcd_strobe_enable();
	__asm("nop;"); __asm("nop;"); __asm("nop;");
	__asm("nop;"); __asm("nop;"); __asm("nop;");

	/* Set RW low, RW low to write the instruction data */
	lcd_rw_low();
	lcd_rs_low();
	LCD_DATA_PORT = data;
	__asm("nop;"); __asm("nop;"); __asm("nop;");
	__asm("nop;"); __asm("nop;"); __asm("nop;");
	lcd_strobe_enable();
	__asm("nop;"); __asm("nop;"); __asm("nop;");
	__asm("nop;"); __asm("nop;"); __asm("nop;");
}

/**
 * Initializes the LCD chip select, display off and reset signals.
 */
void lcd_init(void)
{
	LCD_CTRL_DDR |= (1<<LCD_CTRL_CS)|(1<<LCD_CTRL_DISP)|(1<<LCD_CTRL_RST);

	lcd_cs_low();
	lcd_disp_high();

	lcd_rst_low(); _delay_ms(50);
	lcd_rst_high(); _delay_ms(50);
}

/**
 * Initializes the LCD in graphics mode.
 * Uses a character pitch of 8 (8 bits are plotted whenever a byte is drawn)
 */
void lcd_graphics_init(void) {
	unsigned char commandData;

	/* Set the data direction registers apprioriately */
	LCD_DATA_DDR = 0xFF;
	LCD_CTRL_DDR |= (1<<LCD_CTRL_RS)|(1<<LCD_CTRL_RW)|(1<<LCD_CTRL_E);

	/* Assert all control lines to low */
	lcd_rw_low();
	lcd_rs_low();
	lcd_enable_low();

	/* Send mode configuration command with
	 * Toggle Display On, Master, Mode Graphics bits set */
	commandData = LCD_MODE_ON_OFF | LCD_MODE_MASTER_SLAVE | LCD_MODE_MODE;
	lcd_write_command(LCD_CMD_MODE, commandData);

	// Send the set character pitch command with horizontal
	// character pitch of 8 (so 8 pixels are painted when we draw)
	commandData = LCD_CHAR_PITCH_HP_8;
	lcd_write_command(LCD_CMD_CHAR_PITCH, commandData);

	// Send the number of characters command with the total
	// number of graphics bytes that can be painted horizontally
	// (width/8)
```

```
	commandData = (LCD_WIDTH/8)-1;
	lcd_write_command(LCD_CMD_NUM_CHARS, commandData);

	// Set the time division
	commandData = 128-1;
	lcd_write_command(LCD_CMD_TIME_DIVISION, commandData);

	// Set the display low/high start address to 0x00 (left corner)
	commandData = 0x00;
	lcd_write_command(LCD_CMD_DISPLAY_START_LA, commandData);
	lcd_write_command(LCD_CMD_DISPLAY_START_HA, commandData);

	// Reset the cursor to home 0x00 (left corner)
	commandData = 0x00;
	lcd_write_command(LCD_CMD_CURSOR_LA, commandData);
	lcd_write_command(LCD_CMD_CURSOR_HA, commandData);
}

/**
 * Moves the LCD cursor to the specified coordinates.
 * @param x The new x coordinante of the cursor.
 * @param y The new y coordinante of the cursor.
 */
void lcd_graphics_move(unsigned short x, unsigned short y) {
	unsigned short pos;

	/* Calculate the raw address in terms of bytes on the screen */
	pos = ((y*LCD_WIDTH)+x)/8;

	/* Move the cursor to the new address */
	lcd_write_command(LCD_CMD_CURSOR_LA, pos&0xFF);
	lcd_write_command(LCD_CMD_CURSOR_HA, pos>>8);
}

/**
 * Draws a byte to the LCD at the current LCD's cursor location.
 * @param data The byte to draw. The pixels are drawn MSB to LSB.
 */
void lcd_graphics_draw_byte(unsigned char data) {
	lcd_write_command(LCD_CMD_WRITE_DATA, data);
}

/**
 * Plots a byte at the specified coordinates.
 * @param x The x coordinante of the byte to be drawn.
 * @param y The y coordinante of the byte to be drawn.
 * @param data The byte to draw. The pixels are drawn MSB to LSB.
 */
void lcd_graphics_plot_byte(unsigned short x, unsigned short y, unsigned char
data) {
	lcd_graphics_move(x, y);
	lcd_graphics_draw_byte(data);
}

/**
 * Plots a pixel at the specified coordinates.
 * @param x The x coordinante of the pixel.
 * @param y The y coordinante of the pixel.
 * @param state PIXEL_ON to set the pixel, otherwise pixel will be cleared.
 */
void lcd_graphics_plot_pixel(unsigned short x, unsigned short y, unsigned
char state) {
	unsigned char pos;

	lcd_graphics_move(x, y);
```

```
	/* Since lcd_graphics_move() moves the cursor to a particular
	 * byte, not bit, we need the relative distance to the specified
	 * bit we are going to set/clear. */
	pos = x%8;

	if (state == PIXEL_ON)
	{
		lcd_write_command(LCD_CMD_SET_BIT, pos);
	}
	else
	{
		lcd_write_command(LCD_CMD_CLEAR_BIT, pos);
	}
}

/**
 * Clears the LCD screen
 */
void lcd_graphics_clear(void) {
	unsigned short i;
	/* Move cursor to home (top left corner) */
	lcd_graphics_move(0, 0);
	/* Draw empty bytes to ocucpy the entire screen */
	for (i = 0; i < ((LCD_WIDTH*LCD_HEIGHT)/8); i++)
		lcd_graphics_draw_byte(0x00);
}
```

Приложение 3. Листинг программы передачи данных по UART.

```
/*! \file uart.c \brief UART driver with buffer support. */
//

#include <avr/io.h>
#include <avr/interrupt.h>

#include "buffer.h"
#include "uart.h"

// UART global variables
// flag variables
volatile u08   uartReadyTx;              ///< uartReadyTx flag
volatile u08   uartBufferedTx;           ///< uartBufferedTx flag
// receive and transmit buffers
cBuffer uartRxBuffer;                    ///< uart receive buffer
cBuffer uartTxBuffer;                    ///< uart transmit buffer
unsigned short uartRxOverflow;           ///< receive overflow counter

#ifndef UART_BUFFERS_EXTERNAL_RAM
      // using internal ram,
      // automatically allocate space in ram for each buffer
      static char uartRxData[UART_RX_BUFFER_SIZE];
      static char uartTxData[UART_TX_BUFFER_SIZE];
#endif

typedef void (*voidFuncPtru08)(unsigned char);
volatile static voidFuncPtru08 UartRxFunc;

// enable and initialize the uart
void uartInit(void)
{
      // initialize the buffers
      uartInitBuffers();
      // initialize user receive handler
      UartRxFunc = 0;

      // enable RxD/TxD and interrupts
      outb(UCR, BV(RXCIE)|BV(TXCIE)|BV(RXEN)|BV(TXEN));

      // set default baud rate
      uartSetBaudRate(UART_DEFAULT_BAUD_RATE);
      // initialize states
      uartReadyTx = TRUE;
      uartBufferedTx = FALSE;
      // clear overflow count
      uartRxOverflow = 0;
      // enable interrupts
      sei();
}

// create and initialize the uart transmit and receive buffers
void uartInitBuffers(void)
{
      #ifndef UART_BUFFERS_EXTERNAL_RAM
            // initialize the UART receive buffer
            bufferInit(&uartRxBuffer, uartRxData, UART_RX_BUFFER_SIZE);
            // initialize the UART transmit buffer
            bufferInit(&uartTxBuffer, uartTxData, UART_TX_BUFFER_SIZE);
      #else
            // initialize the UART receive buffer
            bufferInit(&uartRxBuffer, (u08*) UART_RX_BUFFER_ADDR,
UART_RX_BUFFER_SIZE);
            // initialize the UART transmit buffer
```

```
            bufferInit(&uartTxBuffer, (u08*) UART_TX_BUFFER_ADDR,
UART_TX_BUFFER_SIZE);
      #endif
}

// redirects received data to a user function
void uartSetRxHandler(void (*rx_func)(unsigned char c))
{
      // set the receive interrupt to run the supplied user function
      UartRxFunc = rx_func;
}

// set the uart baud rate
u16 uartSetBaudRate(u32 baudrate)
{
      // calculate division factor for requested baud rate, and set it
      u16 bauddiv = ((F_CPU+(baudrate*8L))/(baudrate*16L)-1);
      //u16 bauddiv = F_CPU/(baudrate*16L)-1;
      outb(UBRRL, bauddiv);
      #ifdef UBRRH
      outb(UBRRH, bauddiv>>8);
      #endif
      return bauddiv;
}

// returns the receive buffer structure
cBuffer* uartGetRxBuffer(void)
{
      // return rx buffer pointer
      return &uartRxBuffer;
}

// returns the transmit buffer structure
cBuffer* uartGetTxBuffer(void)
{
      // return tx buffer pointer
      return &uartTxBuffer;
}

// transmits a byte over the uart
void uartSendByte(u08 txData)
{
      // wait for the transmitter to be ready
      while(!uartReadyTx);
      // send byte
      outb(UDR, txData);
      // set ready state to FALSE
      uartReadyTx = FALSE;
}

// gets a single byte from the uart receive buffer (getchar-style)
int uartGetByte(void)
{
      u08 c;
      if(uartReceiveByte(&c))
            return c;
      else
            return -1;
}

// gets a byte (if available) from the uart receive buffer
u08 uartReceiveByte(u08* rxData)
{
      // make sure we have a receive buffer
      if(uartRxBuffer.size)
      {
```

```
            // make sure we have data
            if(uartRxBuffer.datalength)
            {
                  // get byte from beginning of buffer
                  *rxData = bufferGetFromFront(&uartRxBuffer);
                  return TRUE;
            }
            else
            {
                  // no data
                  return FALSE;
            }
      }
      else
      {
            // no buffer
            return FALSE;
      }
}

// flush all data out of the receive buffer
void uartFlushReceiveBuffer(void)
{
      // flush all data from receive buffer
      //bufferFlush(&uartRxBuffer);
      // same effect as above
      uartRxBuffer.datalength = 0;
}

// return true if uart receive buffer is empty
u08 uartReceiveBufferIsEmpty(void)
{
      if(uartRxBuffer.datalength == 0)
      {
            return TRUE;
      }
      else
      {
            return FALSE;
      }
}

// add byte to end of uart Tx buffer
u08 uartAddToTxBuffer(u08 data)
{
      // add data byte to the end of the tx buffer
      return bufferAddToEnd(&uartTxBuffer, data);
}

// start transmission of the current uart Tx buffer contents
void uartSendTxBuffer(void)
{
      // turn on buffered transmit
      uartBufferedTx = TRUE;
      // send the first byte to get things going by interrupts
      uartSendByte(bufferGetFromFront(&uartTxBuffer));
}
// UART Transmit Complete Interrupt Handler
UART_INTERRUPT_HANDLER(SIG_UART_TRANS)
{
      // check if buffered tx is enabled
      if(uartBufferedTx)
      {
            // check if there's data left in the buffer
            if(uartTxBuffer.datalength)
            {
```

```
                // send byte from top of buffer
                outb(UDR, bufferGetFromFront(&uartTxBuffer));
            }
            else
            {
                // no data left
                uartBufferedTx = FALSE;
                // return to ready state
                uartReadyTx = TRUE;
            }
        }
        else
        {
            // we're using single-byte tx mode
            // indicate transmit complete, back to ready
            uartReadyTx = TRUE;
        }
}

// UART Receive Complete Interrupt Handler
UART_INTERRUPT_HANDLER(SIG_UART_RECV)
{
        u08 c;

        // get received char
        c = inb(UDR);

        // if there's a user function to handle this receive event
        if(UartRxFunc)
        {
            // call it and pass the received data
            UartRxFunc(c);
        }
        else
        {
            // otherwise do default processing
            // put received char in buffer
            // check if there's space
            if( !bufferAddToEnd(&uartRxBuffer, c) )
            {
                // no space in buffer
                // count overflow
                uartRxOverflow++;
            }
        }
}
```