**Введение**

В данной курсовой работе стоит задача разработать специализированное цифровое устройство - процессор CORDIC, являющееся отдельным законченным модулем для расчёта значений обратной тригонометрической функции arcSin(Z) и функции возведения в степень числа arCth(Z).

Устройство такого рода может использоваться в сложной комплексной системе, реализующей расчёт координат, расстояния или размера объекта, либо может найти своё применение в системах навигации и позиционирования. Такие приборы широко используются в различных сферах жизнедеятельности, начиная от строительных площадок и заканчивая сверхточными системами позиционирования объектов в космосе.

Данное устройство должно сочетать в себе как высокое быстродействие, так и необходимую точность вычисления. Исходя из задания, в основе работы устройства лежит метод Волдера «Цифра за цифрой». Этот метод представляет собой ряд итерационных формул, в результате расчёта которых происходит процесс поворота вектора до нужного угла.

Общая идея метода сводится к следующему. Последовательным умножением аргумента на заранее выбранные константы, приблизить аргумент с заданной точностью для одних функций к единице, для других функций - к нулю. Однако, для того, чтобы само значение функции при этом оставалось неизменным, необходимо одновременно совершать эквивалентные действия над выбранными константами. При выборе значений констант особым образом удается существенно упростить вычисления значений функции.

Промоделировав математическую модель метода Волдера в пакете Scilab, ее необходимо реализовать практически на интегральных микросхемах. Построить такое устройство можно как на отдельных микросхемах низкой степени интеграции выполняющих определённые функции, либо на программируемой логической матрице, либо на готовом микроконтроллере.

Каждый из этих способов имеет свои преимущества и недостатки, поэтому необходимо выбрать наиболее эффективный и экономичный среди них.

Для построения устройства на ПЛИС необходимо его смоделировать в САПР на языке VHDL. При использовании микроконтроллера необходимо написать программу либо на языке ASSEMBLER, либо на любом другом языке программирования высокого уровня, например C++. Таким образом, разработка и построение цифрового устройства является комплексной и весьма сложной задачей, требующей глубоких знаний во многих компьютерных дисциплинах.

Большинство цифровых систем строится на микропроцессорах либо на микроконтроллерах. И при помощи микропроцессорных систем происходит управление различными технологическими процессами и отдельными операциями. Данные системы практически универсальны, так как они имеют очень высокое быстродействие, и достаточную разрядность для выполнения точных расчетов на производстве. А так же используя в данных системах ППЗУ возможно при помощи одной компьютерной системы управление различным оборудованием. То есть необходимо изменение только программы управления.

**1. АНАЛИЗ ПОСТАВЛЕННОЙ ЗАДАЧИ**

В курсовой проекте необходимо реализовать специализированный математический процессор для вычисления элементарных функций arcSin(Z) и arCth(Z).

Эти функции можно реализовать методом Волдера «Цифра за цифрой», метод Меджита и др. Оба эти метода основаны на ряде итерационных формул, которые приведены в таблице 1.1. Основное их различие заключается в том, что в методе Меджита на первом этапе сразу рассчитываются все направляющие коэффициенты. В данном устройстве будем использоваться метод Волдера «Цифра за цифрой», так как он требует меньших аппаратных затрат и обладает достаточной точностью вычислений.

В таблице 1 приведены итерационные формулы для вычисления заданных элементарных функций по методу Волдера «Цифра за цифрой».

**Табл.1 -** Метод Волдера для расчёта ArcSin(z).

|  |  |
| --- | --- |
| ***Вычисление*** | |
|  |  |
| ***Вычисление*** | |
|  |  |

Данное устройство должно являться частью комплексного прибора, поэтому оно должно иметь разъём для подключения и все соответствующие интерфейсные сигналы.

Устройство получает данные от главного модуля и передаёт ему обратно результат. Данные передаются в формате с фиксированной точкой. На целую часть отводится 7 разрядов и 1 знаковый, на дробную часть также отводится 16 разрядов. Передача осуществляется по интерфейсу I2C(TWI). Выбран именно этот интерфейс так как в нём реализовано распознавание пакетов (начало, конец, распознавание собственных адресов для каждого устройства, что является хорошим способом и альтернативным вариантом для предотвращения коллизий).

**1.1 Расчет разрядности представления данных и числа итераций**

Произведём расчёт числа итераций и разрядности в математическом пакете SciLab.

Определим диапазон допустимых значений заданных элементарных функций. Исследуем итерационные формулы для вычисления Θ = arcth Z:



Тогда границы значений функции, при которых метод Волдера является верным, лежит в следующем пределе:



При этом Z для функции Θ = arch Z должно находиться в следующем диапазоне:



Определим диапазон допустимых значений аргументов функции 2Z, а затем диапазон значений само функции.



Тогда границы значений функции, при которых метод Волдера является верным, лежит в следующем пределе:



Следовательно функция будет изменяться в диапазоне:

.



Рассчитаем минимальное число итераций при погрешности преобразования равной 0,01% и максимальной выходной величине 4,685:



Тогда количестово разрядов данных составит:

.



**1.2 Разработка итерационных алгоритмов вычисления функции в математическом пакете SciLab**

Промоделируем в пакете математического моделирования MatLab метод Волдера «Цифра за цифрой» для анализа его работоспособности и определения погрешности вычислений.

В цикле подсчитаем значение абсолютной погрешности при изменении количества итераций от 1 до n и при изменении разрядности данных (дробная часть задается от 1 до 16 бит).

Листинг программы для исследования погрешности при изменении числа итераций:

//== function arch ====

//function [archZ] = arch(Z,N)

Q0 = 0;

x0 = 0.6072529;

y0 = 0;

N=20;

Z= 0.512

acos\_ist = asin(Z)

for n=1:N

Q(1)=Q0+atan(2^(-0));

r(1)=sign(Z-y0);

x(1)=x0-y0\*2^(-0);

y(1)=y0+x0\*2^(-0);

for i=1:n

r(i)=sign(Z-y(i));

// ARTH =atan(2^(-i));

Q(i+1)= Q(i)+r(i)\*atan(2^(-i));

x(i+1)= x(i)-r(i)\*y(i)\*2^(-i);

y(i+1)= y(i)+r(i)\*x(i)\*2^(-i);

end

arcsinZ(n)= Q(n);

asin\_ist = asin(Z);

pogr(n)=(asin\_ist-arcsinZ(n))/asin\_ist\*100;

end

scf(1);

clf();

plot(pogr);

xgrid();

//endfunction

//////////////////////////////////////////////////////////////

Q0 = 0;

x0 = 0.6072529;

y0 = 0;

for n=1:20

m=3;

Q(1)=Q0+atan(2^(-0));

Q(1)=conv(n,m,Q(1));

r(1)=sign(Z-y0);

x(1)=x0-y0\*2^(-0);

x(1)=conv(n,m,x(1));

y(1)=y0+x0\*2^(-0);

y(1)=conv(n,m,y(1));

for i=1:20

r(i)=sign(Z-y(i));

// ARTH =atan(2^(-i));

Q(i+1)= Q(i)+r(i)\*atan(2^(-i));

Q(i+1)=conv(n,m,Q(i+1));

x(i+1)= x(i)-r(i)\*y(i)\*2^(-i);

x(i+1)=conv(n,m,x(i+1));

y(i+1)= y(i)+r(i)\*x(i)\*2^(-i);

y(i+1)=conv(n,m,y(i+1));

end

arcsinZ(n)= Q(20);

asin\_ist = asin(Z);

pogr(n)=(asin\_ist-arcsinZ(n))/asin\_ist\*100;

end

scf(2);

clf();

plot(pogr);

xgrid();

Запустив приведенную программу, мы получим график зависимости погрешности от числа итераций:

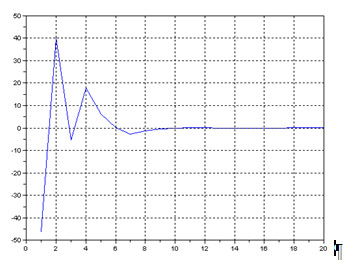


Рис.1.1 - График абсолютной погрешности для функции Θ = arcsin Z при изменяющемся числе итераций

По полученным графикам видно как ведёт себя погрёшность преобразования. При количестве итераций меньше 10 погрешность достигает 5%. По заданию требуется, чтобы погрешность составляла меньше 1%, это достигается при n = 13 и более.

Листинг программы для изменения разрядности данных:

clear

clc

//== function convert real to bin ====

function [x\_bin] = conv(n,m,x)

//преобразование целой части

x\_int = abs(int(x));

x\_tmp=x\_int;

for j=1:m

x\_tmp = x\_tmp/2

if (x\_tmp-int(x\_tmp))==0

arr\_tmp(j)=0

else

arr\_tmp(j)=1

x\_tmp = int(x\_tmp)

end

end

x\_cel = 0;

for j=1:m

x\_cel = x\_cel+arr\_tmp(j)\*2^(j-1)

end

//преобразование дробной части

x\_real = x- x\_int;

for i=1:n,

x\_real = x\_real \* 2;

if x\_real<1

arr(i)=0;

else

arr(i)=1;

x\_real=x\_real-1;

end,

x\_real1=1-int(x\_real)

end

x\_new=0;

for i=1:n

x\_new=x\_new+arr(i)\*2^(-i)

end

x\_bin=x\_cel+x\_new

endfunction

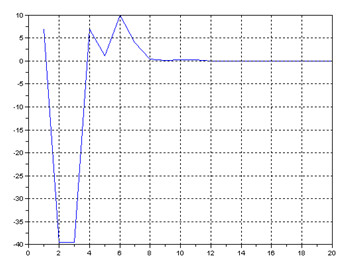


Рис.1.2 - График абсолютной погрешности для функции Θ = arch Z при изменении разрядности дробной части

Изменяя количество бит, приходящихся на дробную часть, исследуется влияние разрядности данных на точность вычислений. Из графика видно, что необходимая точность достигается при разрядности дробной части 13 и более.

Исследуем точность итерационных формул для функции arcth Z в математическом пакете SciLab. Ниже приведен код программы, которая вычисляет погрешность вычисления методом Волдера заданной функции при изменении числа итерации и разрядности дробной части.

Листинг программы:

clear

clc

//== function convert real to bin ====

function [x\_bin] = conv(n,m,x)

//преобразование целой части

x\_int = abs(int(x));

x\_tmp=x\_int;

for j=1:m

x\_tmp = x\_tmp/2

if (x\_tmp-int(x\_tmp))==0

arr\_tmp(j)=0

else

arr\_tmp(j)=1

x\_tmp = int(x\_tmp)

end

end

x\_cel = 0;

for j=1:m

x\_cel = x\_cel+arr\_tmp(j)\*2^(j-1)

end

//преобразование дробной части

x\_real = x- x\_int;

for i=1:n,

x\_real = x\_real \* 2;

if x\_real<1

arr(i)=0;

else

arr(i)=1;

x\_real=x\_real-1;

end,

x\_real1=1-int(x\_real)

end

x\_new=0;

for i=1:n

x\_new=x\_new+arr(i)\*2^(-i)

end

x\_bin=x\_cel+x\_new

endfunction

//========================================

function [arth\_i]=arth(x)

arth\_i = 1/2\*log((1+x)/(1-x));

endfunction

//== function arcth ====

//function [arcthZ] = arcth(Z,N)

Q0 = 0;

x0 = 1;

Z=2.25;

y0 = Z;

N=40;

Q(1)=Q0+arth(2^(-1));

r(1)=sign(1-y0);

x(1)=x0-r(1)\*y0\*2^(-1);

y(1)=y0-r(1)\*x0\*2^(-1);

for n=1:N

for k=2:n

i=1+int((k-1)/2);

r(k)=sign(1-y(k-1));

ARTH =arth(2^(-i));

Q(k)= Q(k-1)+r(k)\*ARTH;

x(k)= x(k-1)-r(k)\*y(k-1)\*2^(-i);

y(k)= y(k-1)-r(k)\*x(k-1)\*2^(-i);

end

arthZ(n)= Q(n)

y\_ist = 1/2\*log((1+Z)/(1-Z))

// z\_i = (exp(y\_ist)+exp(-y\_ist))/2

pogr(n)=(y\_ist-arthZ(n))/y\_ist\*100;

end

scf(1);

clf();

plot(pogr);

xgrid();

//endfunction

for n=1:20

m=3;

Q(1)=Q0+arth(2^(-1));

Q(1)=conv(n,m,Q(1));

r(1)=sign(Z-x0);

x(1)=x0+r(1)\*y0\*2^(-1);

x(1)=conv(n,m,x(1));

y(1)=y0+r(1)\*x0\*2^(-1);

y(1)=conv(n,m,y(1));

last=26;

for k=2:last

i=1+int((k-1)/2);

r(k)=sign(Z-x(k-1));

ARTH =arth(2^(-i));

ARTH=conv(n,m,ARTH);

Q(k)= Q(k-1)+r(k)\*ARTH;

Q(k)=conv(n,m,Q(k));

x(k)= x(k-1)+r(k)\*y(k-1)\*2^(-i);

x(k)=conv(n,m,x(k));

y(k)= y(k-1)+r(k)\*x(k-1)\*2^(-i);

y(k)=conv(n,m,y(k));

end

archZ(n)= Q(last);

y\_ist = log(Z+sqrt(Z^2-1));

pogr(n)=(y\_ist-archZ(n))/y\_ist\*100;

end

scf(2);

clf();

plot(pogr);

xgrid();

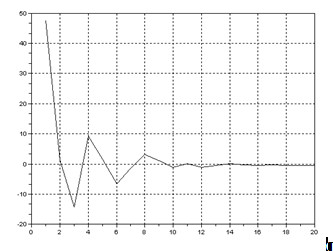


Рис.1.3 - График абсолютной погрешности для функции arcthZ при изменяющемся числе итераций

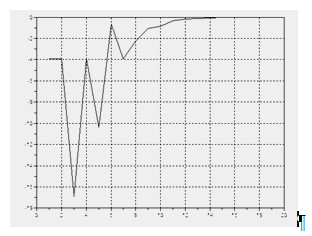


Рис.1.4 - График абсолютной погрешности для функции arcthZ при изменении разрядности дробной части

Из приведенных графиков видно, что минимальное число итерации должно составить 12, а минимальное число разрядов – 12. То есть рассчитанное число итераций и разрядность данных соответствуют результатам моделирования.

**2. РАЗРАБОТКА СТРУКТУРНОЙ СХЕМЫ УСТРОЙСТВА**

Устройства подключается через разъём к Управляющей компьютерной системе(УКС), от которой оно будет получать управляющие сигналы, данные и которому оно будет в ответ выдавать результат вычисления.

Анализируя поставленную задачу можно определить основные блоки включающие в себя устройство.

Так сигналы от УКС будут поступать на устройство ввода-вывода информации (УВВ), которое может присутствовать только абстрактно в нашем устройстве.

Из устройства ввода вывода данные будут поступать в ОЗУ, а управляющие сигналы непосредственно в блок управления. Блок управления (УУ) должен синхронизировать и контролировать работу всего модуля в целом. При этом он должен вырабатывать сигналы чтения и записи из ОЗУ и ПЗУ, а также сигналы управления блоком вычислений (ОБ).

В ОЗУ хранятся входные величины, промежуточные данные и результат. В ПЗУ может храниться микропрограмма вычисления, а также таблицы значений констант таких как Arth(2-i), log(1+ri2-i) и 2-i, .

Это один из возможных вариантов структурной схемы. Ее можно реализовать на ПЛИС или на интегральных схемах средней степени интеграции. Реализации на ПЛИС более дорогая и трудоемкая в сравнении с другими способами. На интегральных микросхемах средней степени интеграции устройство получится более дорогое, так как требуется большое количество дополнительных микросхем, и будет обладать меньшим быстродействием, чем при реализации на микроконтроллере. Поэтому мы остановимся на реализации устройства на микроконтроллере

**2.1 Обоснование достаточности аппаратных средств и программных ресурсов**

Данное устройство будет реализовано на микроконтроллере AVR фирмы ATMEL , так как они наиболее распространённые и доступные по цене. Для реализации данного устройства достаточно использовать 8-ми разрядный микроконтроллер ATmega16, который по своим параметрам вполне удовлетворяет всем требованиям проекта и имеет запас программных и аппаратных средств для введения дополнительных функций и усовершенствования данного специализированного процессора.

AT mega16 обладает следующими характеристиками:

* 8-разрядный высокопроизводительный AVR микроконтроллер с малым энергопотреблением
* Прогрессивная RISC архитектура

**1.** 130 высокопроизводительных команд, большинство команд выполняется за один тактовый цикл,

**2.** 32 8-разрядных рабочих регистра общего назначения

Полностью статическая работа

**3.** Производительность приближается к 16 MIPS (при тактовой частоте 16 МГц)

**4.** Встроенный 2-цикловый перемножитель

* Энергонезависимая память программ и данных

1. 16 Кбайт внутрисистемно программируемой Flash памяти (In-System Self-Programmable Flash) :

-обеспечивает 1000 циклов стирания/записи

- дополнительный сектор загрузочных кодов с независимыми битами блокировки

- Внутрисистемное программирование встроенной программой загрузки

- Обеспечен режим одновременного чтения/записи (Read-While-Write)

2. 512 байт EEPROM:

- Обеспечивает 100000 циклов стирания/записи

3. 1 Кбайт встроенной SRAM

- Программируемая блокировка, обеспечивающая защиту программных средств пользователя

* Интерфейс JTAG (совместимый с IEEE 1149.1)
* Программируемый последовательный USART
* Последовательный интерфейс SPI (ведущий/ведомый)
* Специальные микроконтроллерные функции

1.Сброс по подаче питания и программируемый детектор кратковременного снижения напряжения питания

2.Встроенный калиброванный RC-генератор

3. Внутренние и внешние источники прерываний

4. 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

40-выводной корпус PDIP и 44-выводной корпус TQFP

* Рабочие напряжения 4,5 - 5,5 В
* Рабочая частота 0 - 16 МГц
* Корпус: DIP-40

Контролер имеет три внешних прерывания, подключаемых к одному из портов. Обмен информацией с УКС можно производить по интерфейсу SPI.

**2.2 Выбор и обоснования режимов работы узлов микроконтроллера**

Для ввода и вывода данных будем использовать интерфейс SPI (Serial Peripheral Interface) - полнодуплексный скоростной синхронный трёхпроводной интерфейс. В используемом МК интерфейс SPI реализован на старших выводах порта В.

SPI в реализации Atmel имеет четыре задаваемые программно скорости передачи, может передавать байты от старшего к младшему биту и наоборот, обнаруживает ошибки пакета.

За работу интерфейса отвечают 3 регистра:

* Регистр SPCR - регистр управления

SPIE - разрешение прерывания по окончанию передачи.

SPE - SPI включен. Если установлен этот бит, то выводы #SS, MISO,

MOSI и SCK работают как выводы SPI, иначе - как простые выводы порта.

DORD - направление передачи данных. Если установлен, то передача идёт с младшего бита, если сброшен - со старшего.

MSTR - если установлен, контроллер работает как Master, если сброшен - как Slave. Управляется также выводом #SS, если он настроен на ввод - при подаче "0" на #SS бит MSTR сбрасывается.

CPOL - определяет уровень на выходе SCK в режиме ожидания - SCK = CPOL

CPHA - если установлен, передача и приём бита производится по обратному фронту, если сброшен - по прямому.

SPR1, SPR0 - делитель тактовой частоты (F - частота генератора контроллера).

SPR1SPR0Частота

00F/4

01F/16

10F/64

11F/128

* Регистр SPSR - регистр состояния
* Регистр SPDR - регистр данных

Также мы испольуем линии порта D, на выводы которого подаются внешние прерывания – PD2(INT0) и PD3(INT1) .

Для задания работы портов используются регистры DDR и PORT, соответствующих портов. Режимы работы портов приведены в таблице1.

В те моменты, когда устройство не востребовано, будем переводить его в спящий режим Power Down, выход из которого осуществляется по прерыванию. Данный режим работы описан в таблице 2.

Прерывания будем обрабатывать по переднему фронту, для этого занесём соответствующее значение в регистр MCUCR.

**2.3 Формирование тактовых импульсов**

Тактировать микроконтроллер можно несколькими способами: использовать внешний тактирующий сигнал, либо подключать внешний кварцевый резонатор или RC-цепочку, либо использовать встроенный генератор с внутренней RC- цепочкой. Всё зависит от той задачи, которую мы решаем.

Использование встроенного RC-генератора с внутренней время задающей RC-цепочкой является наиболее экономичным решением, так как при этом не требуется никаких внешних компонентов.

Внутренний RC-генератора микроконтроллеров семейства мега может работать на нескольких фиксированных частотах.

Частота работы внутреннего RC-генератора определяется значениями битов CKSEL3-0 согласно табл.3.

В разрабатываемом устройстве высокая частота, а соответственно и высокая скорость работы не нужна, поэтому вполне достаточно будет выбрать частоту тактирования равную 4.0 МГц.

**2.4 Организация сброса**

В данном случае нет смысла использовать отдельную кнопку сброса, сброс будет осуществляться по включению питания либо по внешнему сигналу сброса от устройства управления. При включении устройства подаётся питание на контроллер и автоматически осуществляется сброс, при выключении устройства, просто снимается питание с контроллера.

Для предотвращения сбоев работы контролера и выполнения программы при скачке напряжения, используем встроенную схему сброса при снижении питания BOD, которая отслеживает напряжение питания. Если работа этой схемы разрешена, то при снижении питания ниже некоторого уровня она переводит контроллер в состояние сброса. Когда напряжение вновь увеличится до порогового значения, запускается таймер задержки сброса. После формирования задержки tTOUT внутренний сигнал сброса снимается и происходит запуск микроконтроллера.

Включением/выключением схемы BOD управляет конфигурационная ячейка BODEN. Для разрешения работы схемы эта ячейка должна быть запрограммирована в «0». Порог срабатывания VBOT определяется состоянием конфигурационной ячейки BODLEVEL, при «1» порог срабатывания 2.7 В, при «0» порог 4 В.

Для уменьшения вероятности ложных срабатываний порог напряжения переключения схемы имеет гистерезис, равный 50 мВ. Кроме того, срабатывание схемы BOD происходит только в том случае, если период провала больше 2 мкс.

Задание длительности задержки сброса tTOUT определяется значением конфигурационных ячеек, и включает в себя две составляющих: ts – выход на рабочий режим и стабилизация частоты тактового генератора, tr – для установки напряжения питания. При использовании встроенного RC-генератора с внутренней RC-цепочкой при включённой схеме BOD, биты конфигурации SUT1-0 установлены в значение «00», при этом ts= 6 тактов, tr –соответственно не используется.

**2.5 Схема включения микроконтроллера**

В микроконтроллере AT mega16 используется напряжение 5 В, подаваемое на вход VCC. Вход AREF используется для подключения внешнего фильтрующего конденсатора для повышения помехозащищенности.

**3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ МИКРОКОНТРОЛЛЕРА**

**3.1 Алгоритм работы устройства**

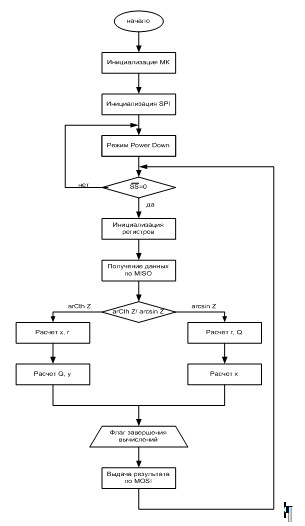


Рис.3.1 - Алгоритм работы устройства

Описание алгоритма программы микроконтроллера:

1. Подача напряжения питания на устройство, автоматически происходит сброс и запуск внутреннего генератора тактовых импульсов;
2. Инициализация режимов работы контролера и его внутренних устройств (портов, интерфейсов и т.д.);
3. Переход в энергосберегающий режим ожидания Power Down;
4. Ожидание прерывания INT1, выводящего контролер из спящего режима;
5. При приходе сигнала SS происходит загрузка регистров SPI и прием данных по MISO от УКС.
6. Анализ полученных данных. Если получена команда 1 – запуск вычисления функции arcth Z, если получена команда 2 – функции arcsin Z.
7. Начало вычисления заданной функции:
   1. 1 этап вычислений - расчет ri , Xi (для функции arcth Z) или ri , Qi (для функции arcsin Z);
   2. 2 этап вычислений – расчет Yi , Qi или Xi
8. Установка флага завершения вычислений;
9. Выдача результата по MOSI;
10. Ожидание прерывания либо INT0 переводящего в спящий режим контроллер, либо INT1 запускающего следующий расчёт.

**3.2 Разработка программного обеспечения**

Листинг программы на языке C++ для микроконтроллера ATmega16:

//== Библиотеки ========================================

#include <mega16.h>

#include <delay.h>

#include "ctype.h"

#include "stdlib.h"

//== Global Variables =====================================

#define TRUE 1

#define FALSE 0

#define COM1 0x01//команда начала вычислении arCthZ

#define COM2 0x02 //команда начала вычислении arcsinZ

#define GICRMask 0xC0 //разрешение прерываний Int0 Int1

#define MCUCRMask 0xCF //спящий режим POWER Down прерывания по низкому уровню

#define nMCUCRMask 0x0F //запрет режима POWER Down

// биты для настройки SPI

#define MOSI 5 //выходный данные передаются по 5 биту порта В

#define SCK 7 // импульсы синхронизации передаются по 7 биту порта В

#define SS 4 //бит упарвляющий передачей данных

#define SPIE 7 //разрешение прерываний по SPI

#define SPE 6 // включение SPI

#define MSTR 4 //МК в режиме мастер

#define SPR0 0 // делитель на 16

#define CLRBIT(ADDR,BIT) (ADDR&=~(1<<BIT))

#define SETBIT(ADDR,BIT) (ADDR|=(1<<BIT))

unsigned char com;//переменная для хранения полученной команды

unsigned char DATA[4];//массив данных, в котором хранится полученное значение Z

unsigned char DATA\_SEND[4];//массив данных, в котором хранится результат вычислений

unsigned char DDR\_SPI; //переменная для настройки работы порта В

unsigned char SPIF = 0;//флаг завершения передачи/приема данных по SPI

unsigned char FlagInt1 = 0//флаг получения прерывания INT1 – вывод МК из спящего режима

unsigned char FlagInt0 = 0;//флаг получения прерывания INT0 – перевод МК в спящий режим

unsigned char FlagCalcReady = 0; //флаг завершения вычислений

//== Const =============================================

// Table of 2^(-i) ---------------------------------------------

float dva[15]={0.5,0.25,0.125,0.0625,0.03125,0.015625,

0.0078125,0.00390625,0.001953125,0.0009765625,

0.00048828125,0.000244140625, 0.0001220703125, 0.00006103515625,

0.00003051758125};

// Table of Arth -------------------------------------------------

loat ath[13]={ 0.5493061,0.2554128,0.1256572,0.0625816,0.0312602,0.0156263,

0.0078127,0.0039063, 0.0019531,0.0009766,0.0004883,0.0002441,

0.0001221 };

float log1[13]={0.5849625, 0.3219281,0.169925, 0.0874628, 0.0443941, 0.0223678,

0.0112273, 0.0056245, 0.0028150, 0.0014082, 0.0007043, 0.0003522,

0.0001761};

float log2[13]={1, 0.4150375, 0.1926451, 0.0931094, 0.0458037, 0.0227201, 0.0113153,

0.0056466, 0.0028205, 0.0014096, 0.0007046, 0.0003523, 0.0001761};

//=====================================================

void GlobalInitialize(void)

{

#asm ("cli");

DDRB = DDR\_SPI;

PORTB = 0xD0;

DDRD = 0x00; //PortD as input

PORTD = 0x0C; //подключение резисторов подтяжки к выводам PD2, PD3

GICR=GICRMask;

MCUCR=nMCUCRMask;

#asm ("sei");

}

//=====================================================

void Init\_SPI\_SLAVE(void)

{

//настройка интерфейса в режим подчиненный

DDR\_SPI=(1<<MOSI); //формируем маску для порта В: передача битов по MOSI

//прием по MISO, тактовый сигнал и сигнал выбора МС на ввод

SPCR |= (1 << SPIE); //разрешение прерывания по SPI

SPCR |= (1 << SPE); //включение интерфейса

}

//== функция приема сообщений ============================

void SPI\_SlaveReceive(void)

{

unsigned char i;

for(i=0; i<1; i++)

{

while(!(SPSR & (1<<SPIF))); //ждем завершения передачи 1-го байта

com = SPDR;

SPIF = FALSE;

}

for(i=1; i<5; i++)

{

while(!SPIF); //ждем завершения передачи байта

DATA[i-1] = SPDR;

SPIF = FALSE;

}

}

//== функция передачи данных =============================

void SPI\_SlaveSend(void)

{

unsigned char i;

for (i = 0; i<4; i++)

{

SPDR = DATA\_SEND[3-i]; //сохр данный в регистре данных SPI

while(!SPIF); //ждем завершения передачи

SPIF = FALSE; //установка флага завершения передачи в 0

}

FlagCalcReady = 0;

}

//====преобразования данных в формат с плавающей запятой =======

float char\_to\_Float(void)

{

float tmp=0;

float a=255;

tmp = (DATA[3]\*a);//преобразование целой части

tmp=tmp+DATA[2];

tmp=tmp+(DATA[1]/a);//преобразование дробной части

tmp=tmp +(DATA[0]/a/a);

return tmp;

}

//=====================================================

void Float\_to\_char(float tmp)

{

int data\_tmp=0;

data\_tmp=(int)tmp;

DATA\_SEND[3]=data\_tmp>>8;

DATA\_SEND[2]=data\_tmp;

data\_tmp=(int)((tmp-data\_tmp)\*65025);

DATA\_SEND[1]=data\_tmp>>8;

DATA\_SEND[0]=data\_tmp;

}

//=====================================================

float arCth(float Z)

{

float aCh;

float X0=1.45235,X1=0,Y0=0,Y1=0,Q0=0,Q1=0;

unsigned char i,n;

for(n=1;n<=26;n++)//число итераций 26

{

i = 1 +((n-1)>>1);

if ((Z-Q0)>=0) //определение знака итерации

{

Q1=Q0 + ath[i-1]; //вычисление Z

X1=X0 + Y0\*dva[i-1]; //вычисление Xi=arcthZ

Y1=Y0 + X0\*dva[i-1]; //вычисление Yi=sh(arChZ)

}

else

{

Q1=Q0 - ath[i-1]; //вычисление Z

X1=X0 - Y0\*dva[i-1];//вычисление Xi=arcthZ

Y1=Y0 - X0\*dva[i-1]; //вычисление Yi=sh(arChZ)

}

//сохранение предыдущих значений

Q0=Q1;

X0=X1;

Y0=Y1;

}

aCh = Q1;

FlagCalcReady = 1;

return aCh;

}

//=====================================================

float arcsinZ (float Z)

{

float as;

float X0=1.0, X1=0.0, Q0=0.0, Q1=0.0;

unsigned char i,n;

for(n=1;n<=26;n++)

{

i = 1 + ((n-1)>>1);

if ((Z-Q0)>=0) //определение знака итерации

{

Q1=Q0 + log1[i]; //вычисление угла

X1=X0 + X0\*dva[i]; //вычисление Xi

}

else

{

Q1=Q0 - log2[i]; //вычисление Z

X1=X0 - X0\*dva[i]; //вычисление Xi

}

Q0=Q1;

X0=X1;

}

as = X1;

FlagCalcReady =1;

return as;

}

//=====================================================

void main(void)

{

unsigned char nSS; //сигнал выбора микросхемы

float Z, ans;

Init\_SPI\_SLAVE(); //инициализация SPI

GlobalInitialize(); //настройка портов ввода-вывода

while(1)

{

if (FlagInt0) //обработка прерывания Инт0

{

FlagInt0=0;

MCUCR=MCUCRMask; //разрешение включения спящего режима

#asm ("SLEEP");} //переход в спящий режим

}

if (FlagInt1)

{

FlagInt1=0;

nSS = PORTB & 0x10; //маска для выделения бита PORTB4

if (nSS == 0)

{

SPI\_SlaveReceive();

if (com == COM1)

{

Z = char\_to\_Float();

ans = arCth(Z);

}

if (com == COM2)

{

Z = char\_to\_Float();

ans = arcsin (Z );

}

}

if (FlagCalcReady)

{

Float\_to\_char(ans);

SPI\_SlaveSend();

}

}

}

//=interrupt==============================================

interrupt [EXT\_INT0] void INT0\_interrupt(void) //обработка прерывания Int0

{

FlagInt0 = 1; //установка флага

}

interrupt [EXT\_INT1] void INT1\_interrupt(void) //обработка прерывания Int1

{

FlagInt1 = 1; //установка флага

MCUCR=nMCUCRMask; //запрет спящего режима

}

**4. АНАЛИЗ РАБОТОСПОСОБНОСТИ ПРОГРАММЫ**

Откомпилировав написанную программу в CodeVision, проанализируем её работу в симуляторе AVRStudio4.

В пошаговом режиме будем тестировать программу, подавая в нужные моменты на входы контроллера соответствующие сигналы. По имеющимся в отладчике часам определим время получения одного значения.

На рис.4.3 показаны результаты вычислений функции arch при получении следующей посылки: com = 0x01, Z = 1.7997576 (после преобразования посылки в формат с плавающей запятой). В результате вычислений получено: arch Z = 1,1927513. Истинное значение составляет arch Z = 1,1927565. Погрешность составляет:, что находится в пределах заданной.



Ниже отображены результаты вычислений пр получении другой посылки от УКС. В ней com = 0x02, Z = 1.0468 (после преобразования посылки в формат с плавающей запятой). В результате вычислений получено: arcsinZ = 2,0657325. Истинное значение составляет arcsinZ = 2,06594. Погрешность составляет:, что находится в пределах заданной.



По значению переменной StopWatch можно подсчитать время всего преобразования:

T=7141,5 мкс, что составляет 7,1 мс и вполне удовлетворяет техническому заданию.

**Заключение**

В данной работе был промоделирован и реализован процессор CORDIC. Данный процессор способен выполнять вычисление различных элементарных функций, таких как тригонометрические, логарифмические и другие. Однако в данном случае стояла задача реализовать только функции arCth Z и arcsinZ.

Вычисление заданной функции устройство выполняет в соответствии с техническим заданием с заданной точностью и в течении заданного периода времени.

Структурная схема, разработанная в этой работе, конечно, не единственно возможная и, тем более, не максимально производительная. Но на ее примере можно усвоить основные принципы построения специализированных цифровых вычислительных систем, таких как процессор CORDIC.

Данное устройство является универсальным и может быть использовано в любом комплексном устройстве. Для согласования работы данного устройства с ведущим устройством не возникает никаких сложностей, так как оно может работать асинхронно, принимать и генерировать прерывания.