ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Государственное образовательное учреждение высшего профессионального образования
«**ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**»

О.С. Качин, А.С.Каракулов

РАЗРАБОТКА АЛГОРИТМОВ УПРАВЛЕНИЯ ДЛЯ МИКРОПРОЦЕССОРНЫХ ЭЛЕКТРОПРИВОДОВ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Рекомендовано к печати
Редакционно-издательским советом
Томского политехнического университета

Издательство
Томского политехнического университета
Томск 2009

УДК 68-83-52
ББК 3859
К31

Качин О.С., Каракулов А.С.

К31 Разработка алгоритмов управления для микропроцессорных электроприводов. Лабораторный практикум: учебное пособие / О.С. Качин, А.С.Каракулов. – Томск: Издательство Томского политехнического университета, 2009. – 104 с.

В учебном пособии рассмотрены основные методы создания управляющего программного обеспечения для встроенных систем электроприводов. Рассматриваются алгоритмы реализации цифровых регуляторов, логических систем управления, а также специализированных процедур, применяемых в электроприводе.

Предназначено для студентов, обучающихся по специальности 140604 – «Электропривод и автоматика промышленных установок и технологических комплексов», и магистров, обучающихся по магистерской программе 140611 – «Электроприводы и системы управления электроприводов» направления подготовки 140600 – «Электротехника, электромеханика и электротехнологии».

УДК 68-83-52
ББК 3859
Рекомендовано к печати Редакционно-издательским советом
Томского политехнического университета

Рецензенты

Кандидат технических наук,
руководитель сектора разработки электромеханических устройств
ООО «Элетим»
С.В.Рикконен

Кандидат технических наук, доцент
Северской государственной технологической академии
С.Н. Кладиев

Введение

Целью практикума является обучение следующему:

1. Использовать среды моделирования Симулинк для моделирования процессов.
2. Использовать готовые компоненты среды Симулинк для создания моделей.
3. Создавать собственные компоненты среды Симулинка для реализации моделей, отсутствующих в Симулинке.
4. Разрабатывать приложение управления процессом моделирования.
5. Разработка структур логического управления электроприводами.
6. Разработка элементарных цифровых регуляторов.

Выполнение заданий по практикуму требует элементарных знаний по программированию, теории электропривода и теории автоматического регулирования.

Практикум содержит 7 занятий. Результатом работы по каждому занятию должен быть отчет. В отчете должны быть приведены результаты моделирования, доказывающие наличие выполненных заданий для самостоятельной работы, а также используемые коды, необходимые для самостоятельно проведенного моделирования.

Занятие 1. Создание моделей в среде Симулинк

Среда визуального моделирования Симулинк содержит большое количество блоков, которые представляют собой математические модели различных устройств, а также блоки, реализующие типовые звенья, применяемые в технических системах. Кроме того, в среде Симулинк содержатся блоки, которые позволяют визуализировать процессы моделирования и формировать различные сигналы задания.

Для запуска среды Симулинк необходимо запустить приложение Матлаб и нажать на панели инструментов кнопку «Simulink». В результате появляется окно **Simulink Library Browser,** в котором раскроется список групп компонентов библиотеки Симулинка**.** В данном окне выбрать иконку «Create a new model» (создать новую модель). В результате откроется окно редактирования модели (рис.1).

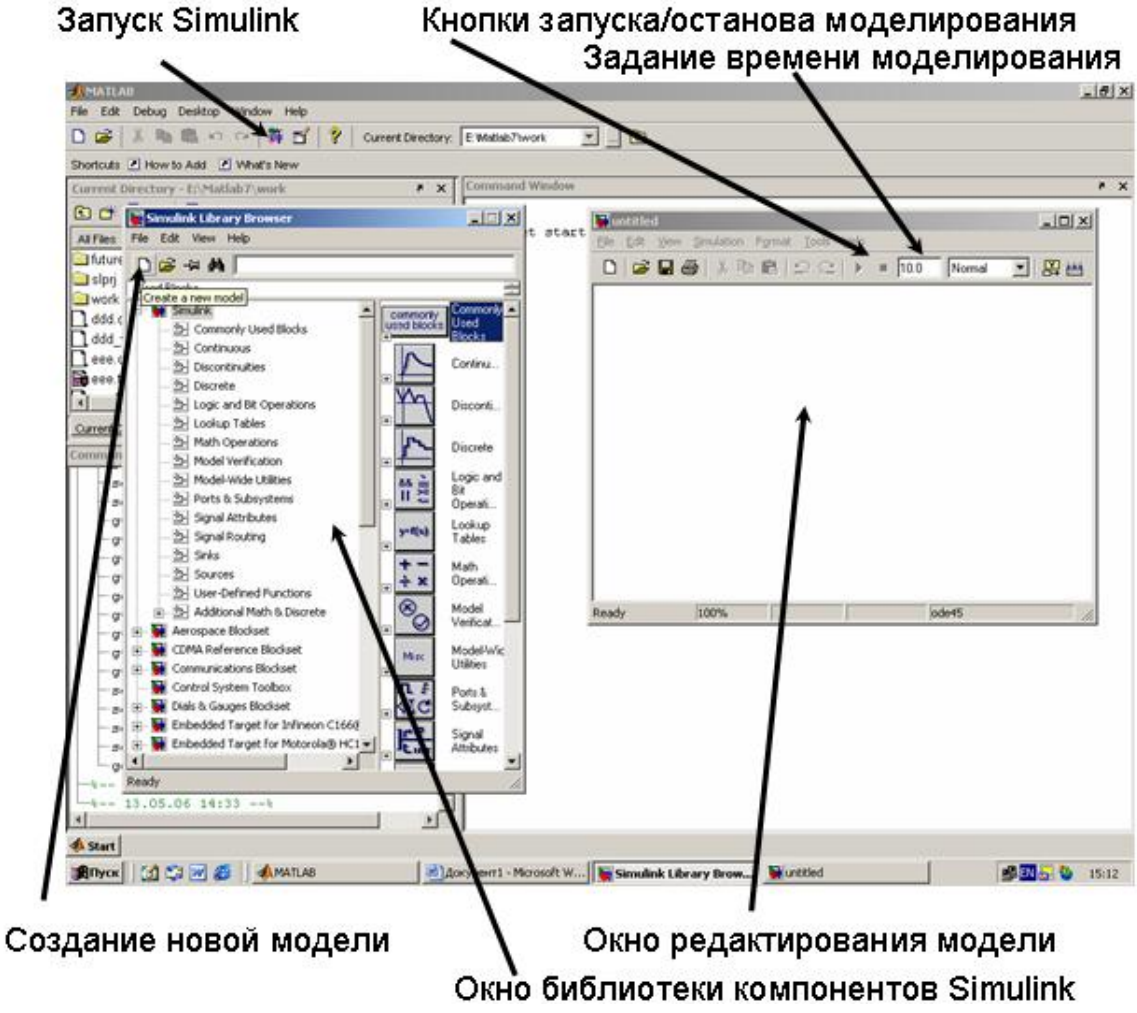

Рис.1

В окно редактирования моделей возможно помещать различные компоненты, предлагаемые библиотекой компонентов Симулинк. Для примера, перетащите из библиотеки компонентов (группа Simulink) осциллограф (Scope, подгруппа Sinks) и генератор синусоидальных сигналов (Sine Wave, подгруппа Sources) и соедините их между собой (см.рис.2.).

После двойного нажатия на компоненте Scope появится окно визуализации координат.

После двойного нажатия на компоненте Sine Wave появится окно параметров настройки выходного сигнала (амплитуда, частота и т.д.).

Нажмите кнопку запуска моделирования (см.рис.1). Симулинк смоделирует процессы в модели за время, задаваемое в окне «Задание времени моделирования» (рис.1).

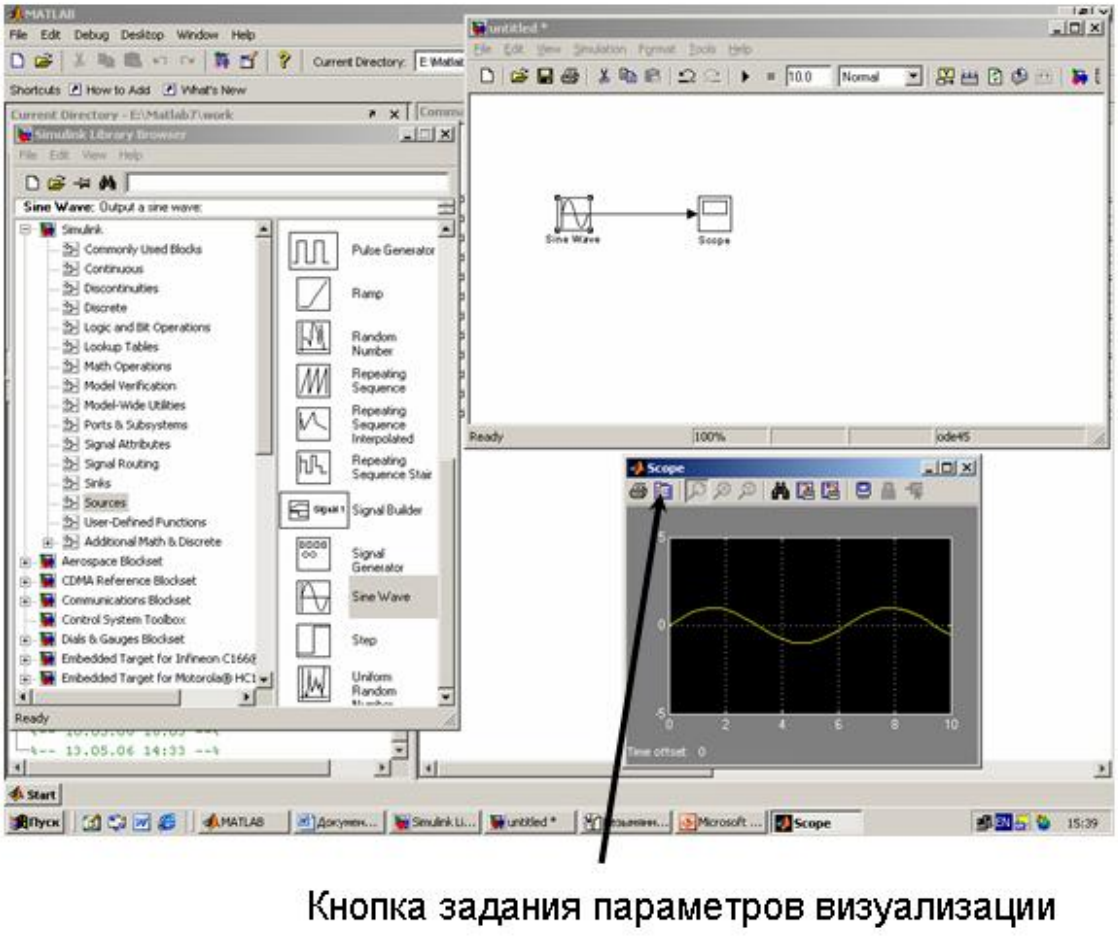

Рис.2

Примечание. Если в параметр «Задание времени моделирования» задать значение «inf», то становится возможным наблюдать моделируе-

мые процессы бесконечно долго (до нажатия кнопки останова моделирования, показанной на рис.1.)

Задания для самостоятельной работы

1. Выясните, какие еще компоненты задания сигналов предлагает Симулинк, приведите 5 примеров с осциллограммами. Каким образом можно визуализировать несколько процессов с помощью одного компонента Scope?
2. Каким образом можно отмасштабировать выводимый на экран компонента Scope сигнал по времени и по амплитуде (исследуйте данный компонент с помощью кнопки задания параметров визуализации, рис.2)?

Как правило, при моделировании берется исходный сигнал, пропускается через моделируемую систему, которая преобразует исходный сигнал, и после этого выходы моделируемой системы визуализируются для последующего анализа. В выполненной модели было показано задание исходного сигнала с последующей визуализацией. Выполним преобразование сигнала с одновременной визуализацией исходного и преобразованного сигнала. В качестве преобразователя сигнала используем интегратор, который находится в подгруппе Continuous. Моделируемая система и результаты показаны на рис.3.

Как можно видеть на полученных осциллограммах, сигналы удовлетворяют равенству $\int \sin x\, dx = -\cos x$

Задание для самостоятельной работы

1. Смоделируйте реакцию апериодического звена 1-ого порядка на синусоидальный входной сигнал частотой 3 Гц и установите влияние постоянной времени данного звена на выходной сигнал (используйте компонент задания передаточной функции).

В рассмотренных примерах действия производились с абстрактными математическими компонентами и компонентом визуализации Scope. Для моделирования работы электромеханических систем воспользуемся группой SimPowerSystems. В качестве примера рассмотрим моделирование процесса пуска асинхронного двигателя при подаче на его обмотки напряжения 380 В, 50 Гц.

Как правило, при моделировании электромеханических систем необходимо создать:

- модель источника электроэнергии

- модель электромеханического преобразователя (двигателя)
- модель механизма, который приводится в действие двигателем
- модель системы управления координатами двигателя.

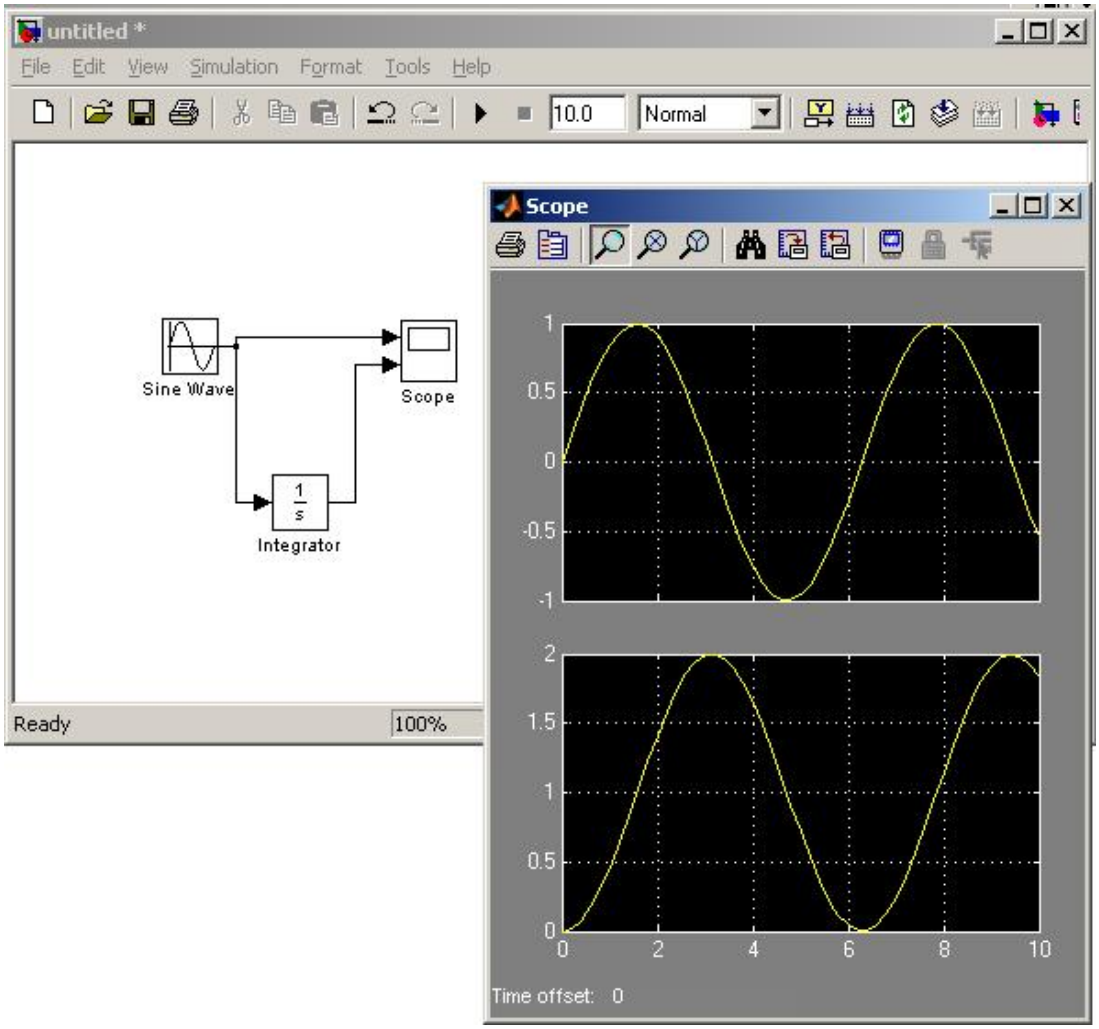

Рис.3

Модель трехфазной электрической сети (источника электроэнергии) можно собрать следующим образом (используя группу SimPowerSystems/Electrical Sources, компонент AC Voltage Source):

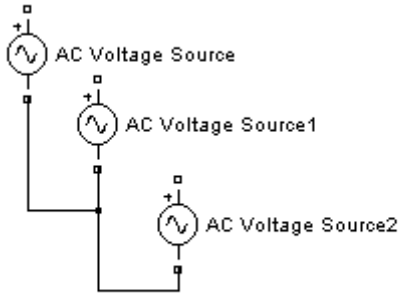

Рис.4

В созданной модели необходимо назначить для каждого компонента напряжение 310 Вольт (амплитудное значение фазного напряжения) и частоту 50 Гц. Сдвиг по фазам – 0, 240, 120 градусов для каждого компонента соответственно.

Для того, чтобы модель с примененными компонентами заработала, необходимо (в обязательном порядке!) установить компонент измерения напряжения Voltage Measurement из группы SimPowerSystems/ Measurements (рис.5).

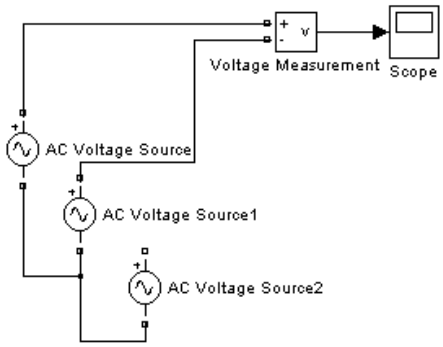

Рис.5

В качестве модели двигателя воспользуемся компонентом Asynchronous Machine SI Units, находящийся в подгруппе SimPowerSystems/Machines. В свойствах компонента необходимо установить тип ротора как короткозамкнутый (Rotor type - Squirrel Cage). Параметры схемы замещения, предлагаемые для редактирования, в рамках данной задачи моделирования изменению не подлежат.

Модель двигателя выдает вектор значений рассчитанных координат. Для визуализации конкретных координат модели двигателя необходимо применить блок Machines Measurement Demux из подгруппы

SimPowerSystems/Machines. В свойствах компонента необходимо выбрать подключение к асинхронному двигателю, а также установить «галочки» напротив тех координат, которые подлежат визуализации: ток статора, электромагнитный момент, скорость ротора. К компоненту Machines Measurement Demux становится возможным подключать стандартные компоненты визуализации, например Scope. Полная модель двигателя и электрической сети, а также результаты моделирования процесса включения двигателя, представлены на рис.6.

Перед началом моделирования необходимо задать время моделирования 0,35 секунд, а также установить метод расчета ode23tb через меню окна редактирования Simulation\Configuration Parameters, строка Solver.

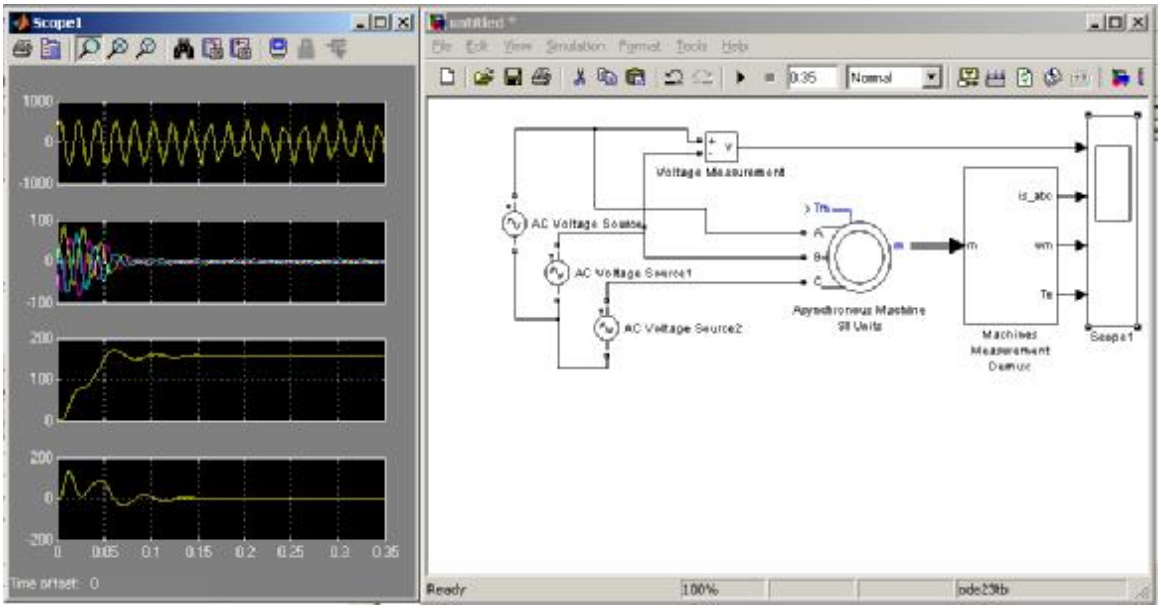

Рис.6

Для моделирования наличия механизма в системе нагрузим модель двигателя по входу Tm, для чего к данному входу подсоединим компонент задания константы Constant из подгруппы Simulink/Sources (рис.7). Значение константы будет устанавливать момент нагрузки на валу двигателя, в данном случае рекомендуется установить ее равным значению 40 Нм.

Смоделируйте процесс пуска двигателя при задании значительного момента нагрузки, например 100 Нм. Как будет видно из осциллограммы, двигателю не хватает электромагнитного момента для преодоления момента нагрузки и он не может запуститься. Более того, при таком моменте нагрузки скорость двигателя становится отрицательной, что сви-

детельствует об активном характере момента нагрузки, что характерно для таких механизмов как кран, лифт и пр.

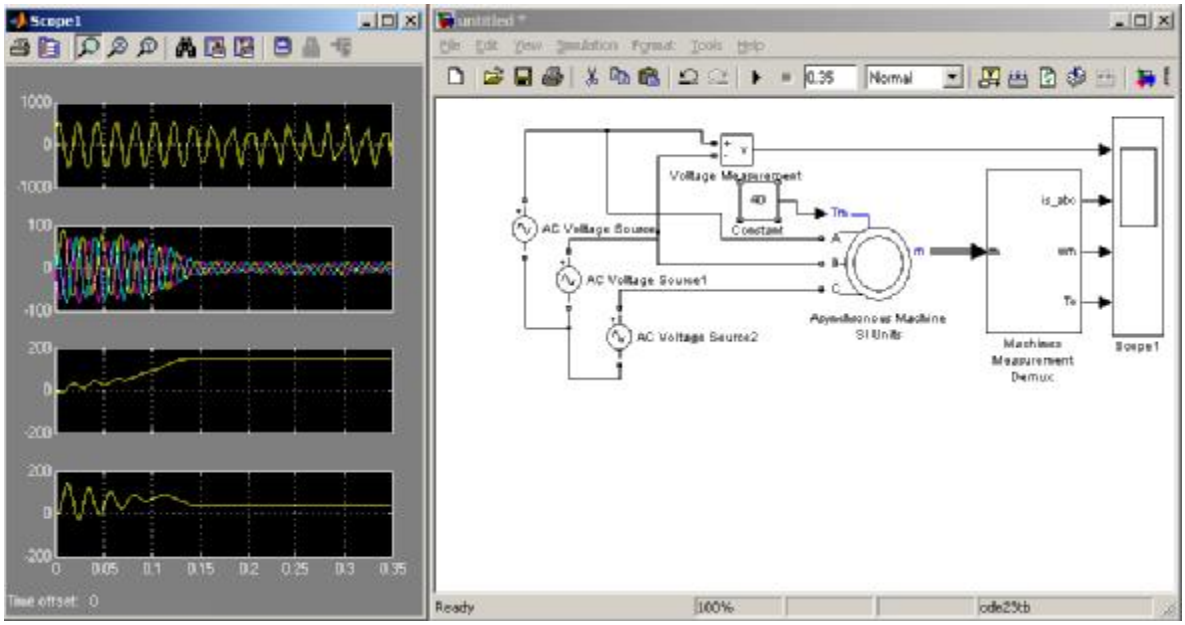

Рис.7

Для конвейеров, задвижек и прочих механизмов характерен реактивный момент нагрузки, который всегда направлен против направления вращения двигателя, а при отсутствии момента двигателя момент нагрузки равен нулевому значению.

Для моделирования механизма с реактивным моментом сопротивления необходимо разработать специальный компонент, так как такой компонент в стандартной библиотеке Симулинка не предусмотрен.

Создание собственных компонентов в Симулинке возможно благодаря наличию стандартных компонентов S-функций. Данные компоненты позволяют пользователю определять самому количество входов/выходов компонентов, а также устанавливать соотношения между входными и выходными величинами посредством заданных пользователем формул и алгоритмов.

Перед тем, как работать с механизмом S-функций Симулинка, необходимо произвести настройку компилятора Матлаба, для чего в командной строке Матлаба (найти значок “>>” в главном окне Матлаба) ввести команду *mex –setup*(**обязательно поставить пробел перед знаком «-»!!!**) и ответить на заданные вопросы. Пример ответов показан ниже.

*>> **mex -setup***
Please choose your compiler for building external interface (MEX) files:

*Would you like mex to locate installed compilers [y]/n? **y***

Select a compiler:
[1] Lcc C version 2.4 in E:\MATLAB7\sys\lcc
[0] None

*Compiler: **1***
Please verify your choices:
Compiler: Lcc C 2.4
Location: E:\MATLAB7\sys\lcc
*Are these correct?([y]/n): **y***

Try to update options file: D:\Documents and Settings\karakulov\Application Data\MathWorks\MATLAB\R14\mexopts.bat
From template: E:\MATLAB7\BIN\WIN32\mexopts\lccopts.bat

Done . . .

Примечание. Настройку компилятора в Матлабе достаточно проделать только 1 раз.

Прежде чем создать собственный компонент, разберем простейший пример с применением S-функций, в котором происходит домножение входной величины на коэффициент, равный «2».

Откройте новое окно редактирования модели и создайте связку компонентов, показанную на рис.8. Компонент S-Function Builder находится в подгруппе Simulink/User Defined Functions.

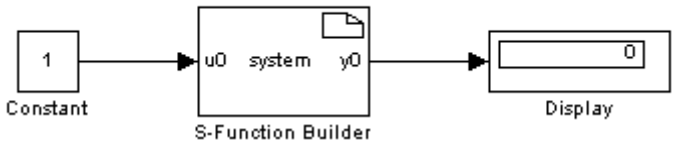

Рис.8

Двойным щелчком по компоненту S-Function Builder раскройте окно его свойств (рис.9). Задайте имя файла, в котором будет находится содержимое компонента, в строке S-function name. Откройте закладку Outputs, удалите предлагаемое текстовое содержание и введите строку

*y0[0]=2*u0[0];*

Здесь:

y0 - обозначает первый выход компонента (отсчет идет с нуля), *y0[0]* обозначает что выходной вектор имеет размерность 1 (то есть одно число, не массив!).

u0 обозначает первый вход компонента (отсчет идет с нуля), *u0[0]* обозначает что входной вектор имеет размерность 1 (то есть одно число, не массив).

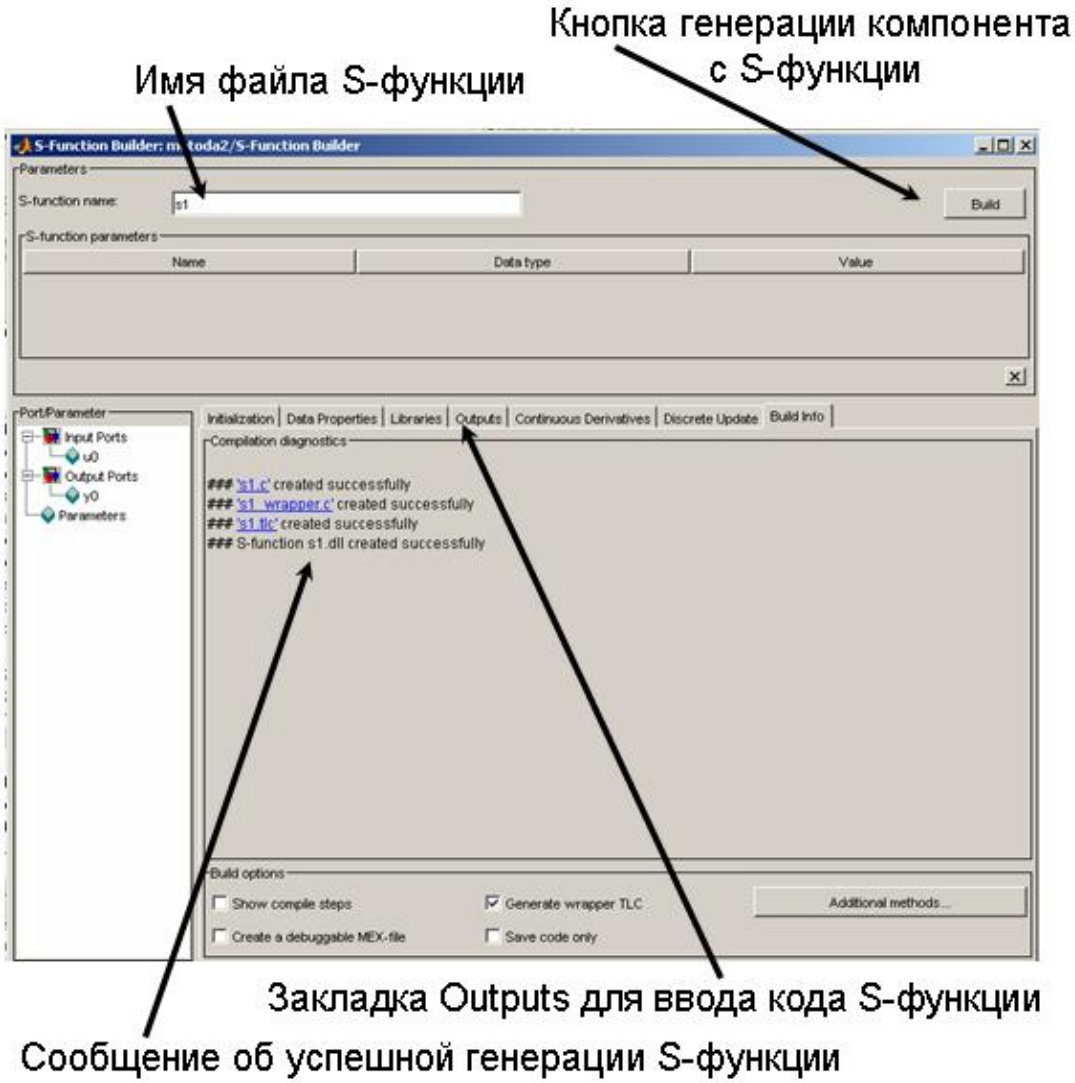

Рис.9

Таким образом, в рассматриваемом примере входная переменная в компоненте помножается на 2 и произведение выдается на выход.

Для генерации компонента необходимо нажать кнопку Build (построить).

Запустите моделирование. В результате в показаниях компонента Display можно видеть удвоенное значение компонента Constant. Таким образом, S-функция позволяет генерировать компоненты, алгоритм функционирования которых определяет пользователь. При генерации алгоритмов необходимо использовать синтаксис языка С.

Усложним задачу. Допустим, что необходимо создать компонент, который перемножает 2 входных сигнала и на выход выдает их произведение. Для этого в компонент необходимо добавить 2-ой вход. Откройте окно свойств S-функции, закладку Data Properties и во внутренней закладке Inputs нажмите кнопку добавления 2-ого входа (рис.10) с именем *u1*.

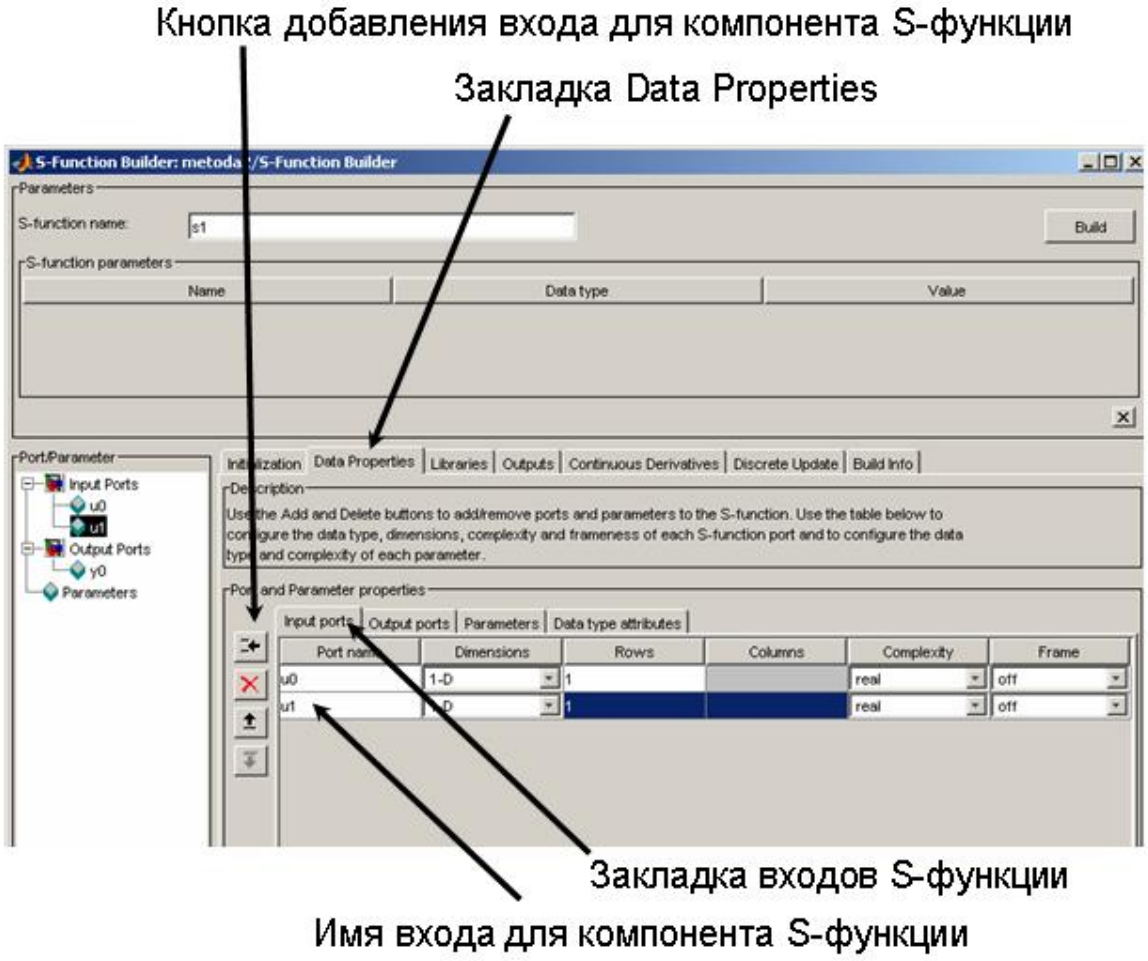

Рис.10.

В закладке Outputs необходимо формулу заменить на

*y0[0]=u1[0]*u0[0];*

то есть использовать для перемножения оба входа.

После генерации новой S-функции у компонента автоматически появится второй вход, в который необходимо завести константу (рис.11).

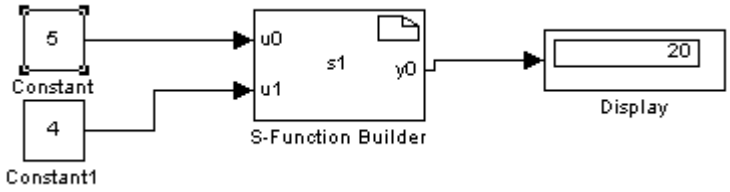

Рис.11.

Задание для самостоятельной работы

1. На базе рассмотренного примера, сгенерируйте компонент S-функции с двумя выходами, второй выход должен показывать значение, обратное по знаку сигналу первого выхода.

Вернемся к созданию компонента, реализующего реактивный момент сопротивления. Алгоритм функционирования такого алгоритма будет следующий:

1. Если скорость положительная, то момент сопротивления равен заданному.
2. Если скорость отрицательная, то момент сопротивления равен заданному с обратным знаком.
3. Если скорость равна «0», то момент сопротивления равен 0.

Компонент должен иметь 2 входа:

1. Вход задания момента сопротивления u0[0].
2. Вход сигнала скорости u1[0].

Компонент должен иметь 1 выход: на выходе формируется момент нагрузки двигателя y0[0].

Реализация на языке С данного алгоритма будет следующая:

```
if (u1[0]>0) y0[0]=u0[0];
if (u1[0]<0) y0[0]= - u0[0];
if (u1[0]==0) y0[0]=0;
```

Создадим такой компонент, установим его в модель (рис.12) и промоделируем его работу, в том числе и при больших значениях момента (>100Нм).

Примечание. В связи с некоторыми особенностями работы Симулинка, при расчете S-функций для сложных нелинейных систем (в частности, моделей электромеханических преобразователей), необходимо установить строго периодичный вызов S-функций (в закладке Initialization выбрать Sample mode: Discrete, время дискретизации (параметр SampleTimeValue) 0.0001 секунда)

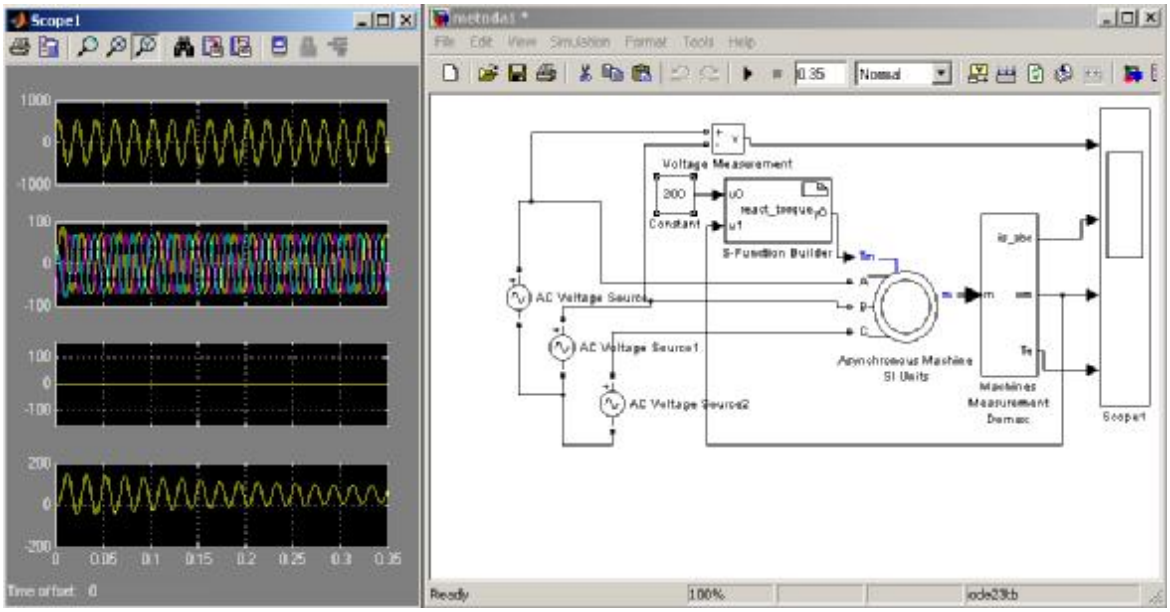

Рис.12

Задание для самостоятельной работы.

1. Проверьте корректность работы компонента, изменяя чередование фаз двигателя, нагрузку, в том числе и знак нагрузки.
2. Сохраните созданную модель под именем «metoda1»

В качестве модели системы управления применим модель простейшего способа управления асинхронным двигателем – посредством 3-фазного пускателя, выполненного на базе механической контактной группы. Библиотека Симулинк предлагает компонент Breaker (подгруппа SimPowerSystems/Elements), который моделирует работу механического контакта. Данный компонент имеет 2 входа и 1 выход.

Вход, который отвечает за управление контактором, обозначен как «>». При подаче на данный вход значения «0» контактор считается ра-

зомкнутым, в остальных случаях он замкнут. На рис.13 показано управление тремя контакторами посредством компонента константы. Вход (расположен слева от блока), обозначенный «□», предназначен для подачи сигнала напряжения на контактор. Выход (расположен справа от блока), обозначенный «□», предназначен для выдачи сигнала напряжения с контактора.

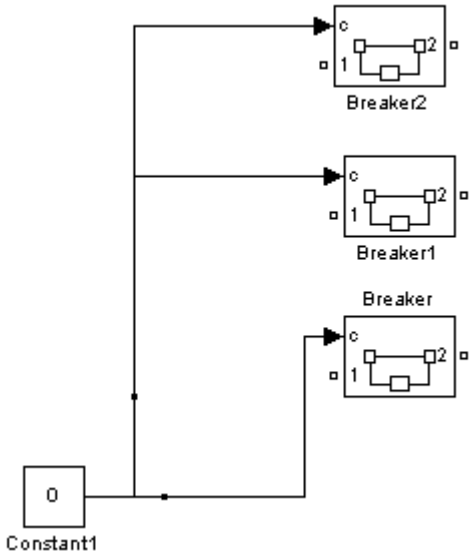

Рис.13

Применим схему, показанную на рис.13, для подачи сигнала напряжения на асинхронный двигатель. Для этого в разрыв между моделью источника электрической энергии и двигателя установим 3 компонента Breaker с управлением от компонента задания константы (см.рис.14).

Для моделирования работы полученной схемы необходимо задать время моделирования как inf (бесконечность). Свойства компонента Breaker задать следующие:

Breaker resistance – 0.000001 Ом

Initial state – 0

Snubber resistance - 1e2 Ом

Snubber capcacitance - inf

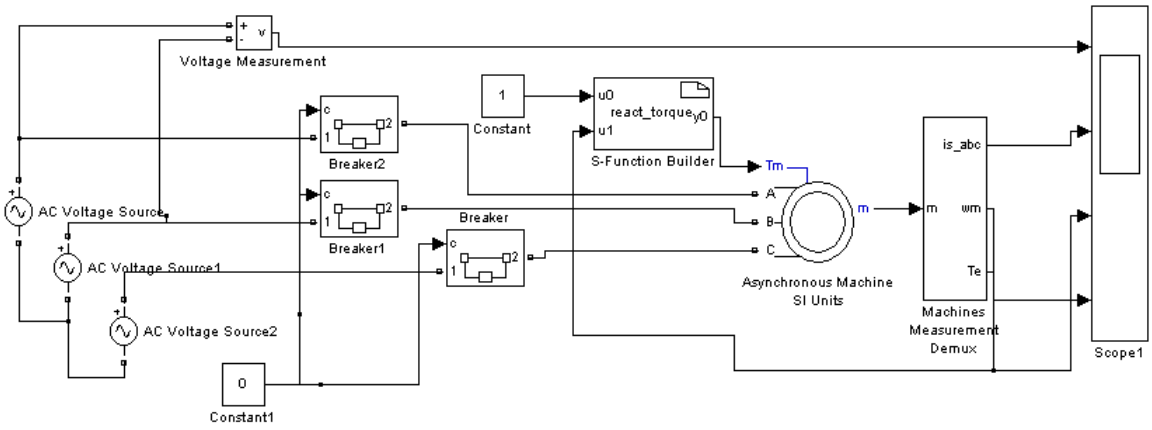

Рис.14

Задание для самостоятельной работы

Запустите модель с обесточенным двигателем, подайте управляющий сигнал на контакторы при помощи компонента Constant и зафиксируйте переходный процесс возникающий при пуске двигателя. Не останавливая процесса моделирования, обесточьте двигатель и зафиксируйте переходный процесс, возникающий при этом.

Матлаб предлагает возможности по созданию приложения пользовательского интерфейса к разработанным моделям. Данные возможности осуществляются с помощью приложения Матлаба GUIDE. Суть такого интерфейса заключается в том, что изменение параметров и значений компонентов можно производить не непосредственно в редакторе модели, а с помощью внешнего окна, стандартного для приложений Windows. В рассматриваемом примере мы будем задавать момент нагрузки и включение/выключение двигателя из внешнего окна, посредством стандартной кнопки и слайдера (линейка «прокрутки») Windows.

Для использования возможностей создания пользовательского интерфейса запустите мастер создания приложений GUIDE (кнопка «GUIDE» на панели инструментов основного окна Матлаба). В результате запустится окно мастера приложений (рис.15).

Нажмите кнопку «ОК», после чего раскроется редактор пользовательского интерфейса (рис.16).

В редакторе создайте 1 слайдер и 2 кнопки, выбрав их на левой панели и растянув в поле редактора. Слайдер будет отвечать за задание момента нагрузки. Левая кнопка будет отвечать за старт двигателя, правая – за обесточивание (рис. 16).

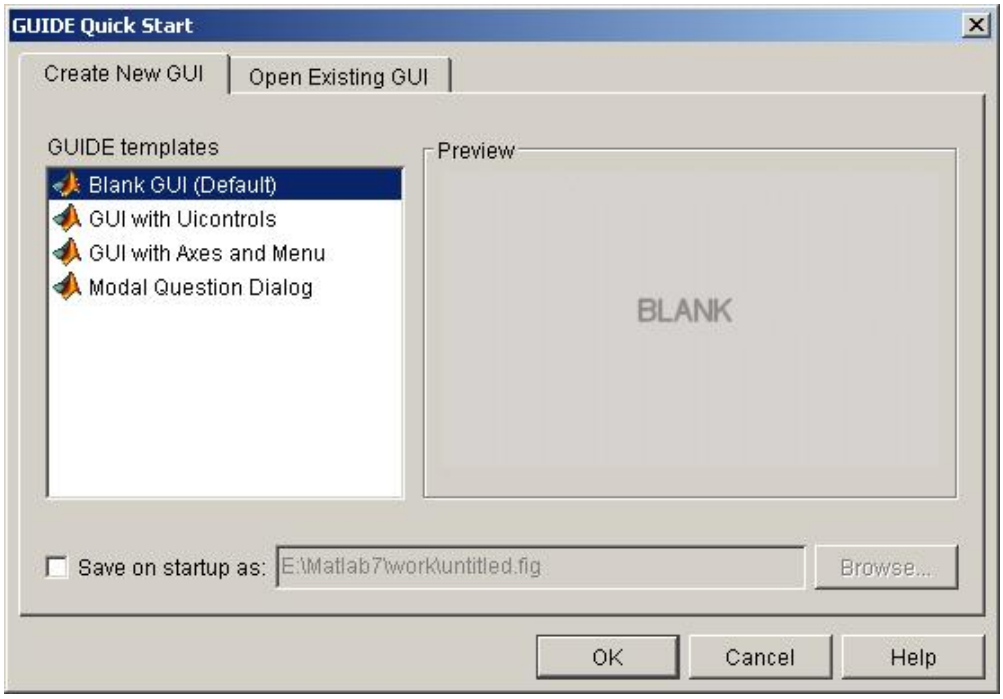
GUIDE Quick Start
Create New GUI
Open Existing GUI
GUIDE templates
Blank GUI (Default)
GUI with Uicontrols
GUI with Axes and Menu
Modal Question Dialog
Preview
BLANK
Save on startup as:
E:\Matlab7\work\untitled.fig
Browse...
OK
Cancel
Help

Рис.15

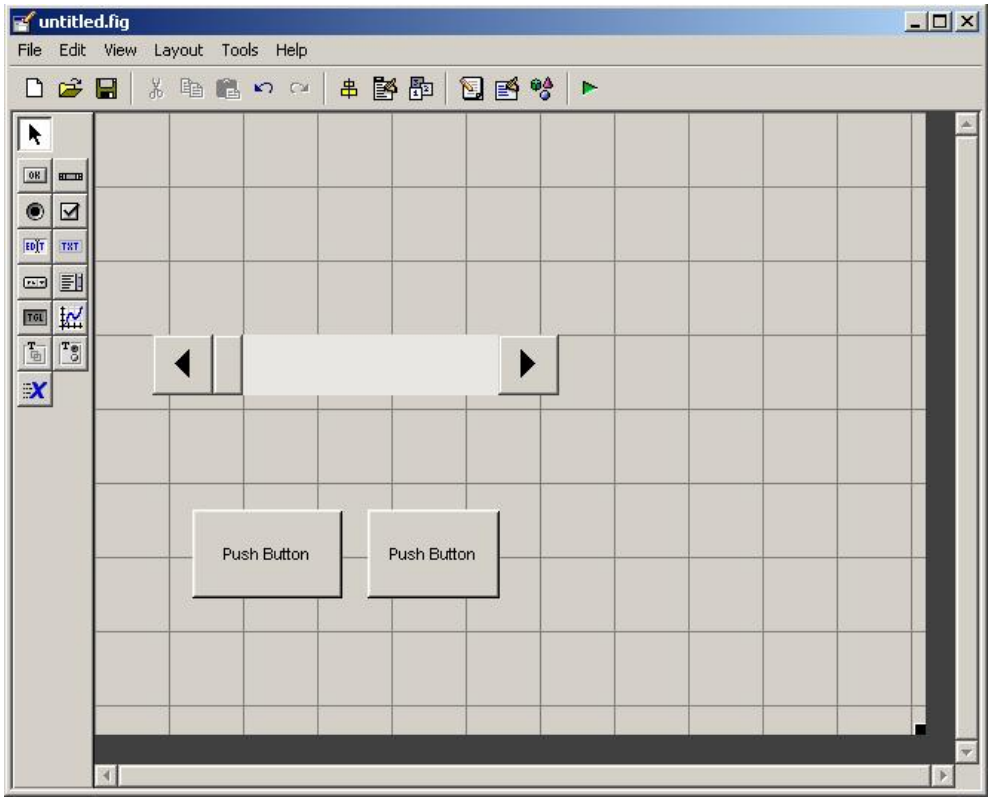
untitled.fig
File Edit View Layout Tools Help
Push Button
Push Button

Рис.16.

Вызовите окно свойств левой кнопки с помощью двойного клика по ней. В свойстве «String» впишите название кнопки исходя из ее назначения. Аналогично поступите с правой кнопкой.

Сохраните файл редактора интерфейса, при этом автоматически появится окно редактирования скриптов (кодов) Матлаба. Найдите в раскрывшемся скрипте Матлаба строку:

function pushbutton1_Callback(hObject, eventdata, handles)

Данная строка описывает наличие реакции на нажатие кнопки *pushbutton1* (в данном случае это одна из кнопок интерфейса).

После данной строки можно написать желаемый код, который должен выполняться при нажатии левой кнопки. В данном случае необходимо с помощью данной кнопки подать напряжение на модель двигателя, то есть присвоить значению константы единицу.

Присвоение значения компоненту осуществляется с помощью команды Матлаб *set_param* и заданием соответствующих параметров команды. Например, если необходимо задать значение «1» для компонента константы Constant1, то необходимо подать команду

set_param('metoda1/Constant1','Value','0')

где:

metoda1 – имя файла, под которым Симулинк сохранил созданную модель в окне редактирования модели.

Constant1 – имя компонента задания константы (см.в редакторе модели).

Value – свойство, отвечающее за выходной сигнал компонента

0 – присваемое значение для свойства *Value*

Примечание. Получить имя файла и компонента можно следующим образом: выделите требуемый компонент в текущей модели и введите в командную строку Матлаба команду *gcb*.

Считывание значения с компонента осуществляется с помощью команды Матлаб get_param. Например, если необходимо вывести значение компонента *Constant1* в элемент *Static Text* графического интерфейса, то необходимо сгенерировать в скрипте следующие команды*:*

text_val=get_param('metoda1/Constant1','value')

set(handles.text1, 'String', text_val);

В данном примере первая строка считывает значение константы и сохраняет его в переменной *text_val*, а вторая строка осуществляет вывод этого значения на экран в элемент *Text1*.

Компонент Constant1 отвечает за включение/выключение контакторов в рассматриваемой модели.

Таким образом, для подачи команд на движение необходимо наличие в текстовом редакторе следующих строк:

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
set_param('metoda1/Constant1','Value','1')


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
set_param('metoda1/Constant1','Value','0')
```

Данные строки позволяют присвоить значения 0 или 1 константе управления при нажатии соответствующей кнопки.

Примечание. Набор символов, начинающихся с % означает комментарий и не выполняется в скрипте.

Для задания момента сопротивления необходимо сначала считать значения со слайдера, а затем по известной технологии выставить значение константы, отвечающей за задание момента сопротивления. Считывание параметра слайдера осуществляется с помощью команды *get*. Пример задания момента сопротивления для рассматриваемой модели будет следующим:

```
function slider1_Callback(hObject, eventdata, handles)
slider_value = get(hObject,'Value')
set_param('metoda1/Constant','Value',num2str(slider_value*100))
```

Примечание. Первую строку текста необходимо отыскать в скрипте, остальные две ввести вручную. Функция *num2str* обеспечивает перевод числового типа данных в текстовый.

После окончания редактирования скрипта закройте окно редактора скриптов.

В окне редактирования интерфейсов на панели инструментов нажмите кнопку RUN (запустить). Расположите окна появившегося приложения пользовательского интерфейса и редактора модели таким же образом, как показано на рис.17. Нажимая кнопки и передвигая бегунок

слайдера, убедитесь в изменении соответствующих значений констант. Запускать моделирование не обязательно.

Примечание. Если в компоненте задания константы показывается значок ***-C-*** **,** то необходимо увеличить размеры компонента на экране для обеспечения полного вывода значения константы.

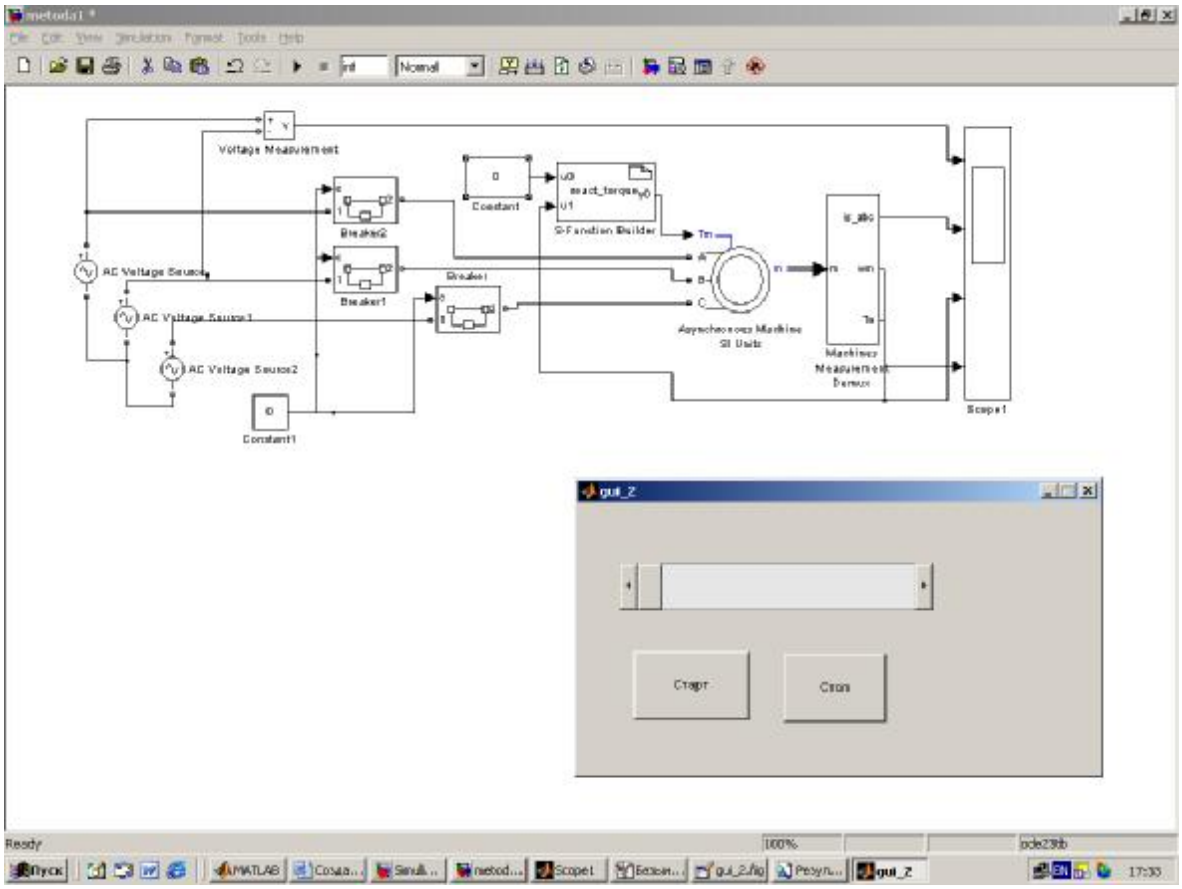

Рис.17

Задание для самостоятельной работы

1. Запустите модель с бесконечным временем моделирования и убедитесь, что созданное приложение позволяет управлять процессом моделирования.

2. Разработайте модель, в которой будет осуществляться изменение чередование фаз двигателя (а соответственно направление скорости вращения) посредством дополнительных коммутаторов. Добавьте в приложение управления соответствующие компоненты, необходимые для включения реверса двигателя.

3. В редакторе интерфейса добавьте компонент *Static Text* и обеспечьте отображение значения задаваемого момента нагрузки.

Если вы выполнили задание, то результат моделирования и модель показаны на рис.18, а соответствующая часть скрипта будет выглядеть как (файл модели Симулинка в данном случае назван как *«metoda4»*):

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
set_param('metoda4/Constant3','Value','1')
set_param('metoda4/Constant2','Value','0')
set_param('metoda4/Constant1','Value','1')

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
set_param('metoda4/Constant1','Value','0')
set_param('metoda4/Constant2','Value','0')
set_param('metoda4/Constant3','Value','0')

% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
slider_value = get(hObject,'Value')
set_param('metoda4/Constant','Value',num2str(slider_value*100))

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
set_param('metoda4/Constant1','Value','0')
set_param('metoda4/Constant2','Value','1')
set_param('metoda4/Constant3','Value','1')
```

Примечание. В данном примере необходимо применить раздельное управление для двух компонентов Breaker прямого включения, двух компонентов Breaker реверсивного включения, и одного компонента общей фазы (компоненты Constant1, Constant2, Constant3 на рис.18 соответственно).

Для того, чтобы выводить значение слайдера в элемент Static Text, необходимо в фунцию

function slider1_Callback(hObject, eventdata, handles)

вписать строку

*set(handles.text1, 'String', get(hObject,'Value')*100);*

В данном случае в элементе GUI *Static Text* будет показываться значение слайдера, домноженное на 100.

Более правильным решением будет прямой вывод в элемент *Static Text* значения из компонента модели. Пример задания значения компонента константы через слайдер и одновременное считывание значения константы с визуализацией посредством компонента GUI *Static Text* приведен ниже:

```
function slider1_Callback(hObject, eventdata, handles)
slider_val=get(hObject,'Value')
set_param('metoda4/Constant','value', num2str(slider_val*100));
text_val=get_param('metoda4/Constant','value')
set(handles.text1, 'String', text_val);
```

Полученная модель является визуально громоздкой ввиду наличия большого количества компонентов на экране. Для объединения компонентов в единый блок Симулинк предлагает возможность использовать компонент подсистемы с именем Subsystem (группа Simulink, подгруппа Ports & Subsystems).

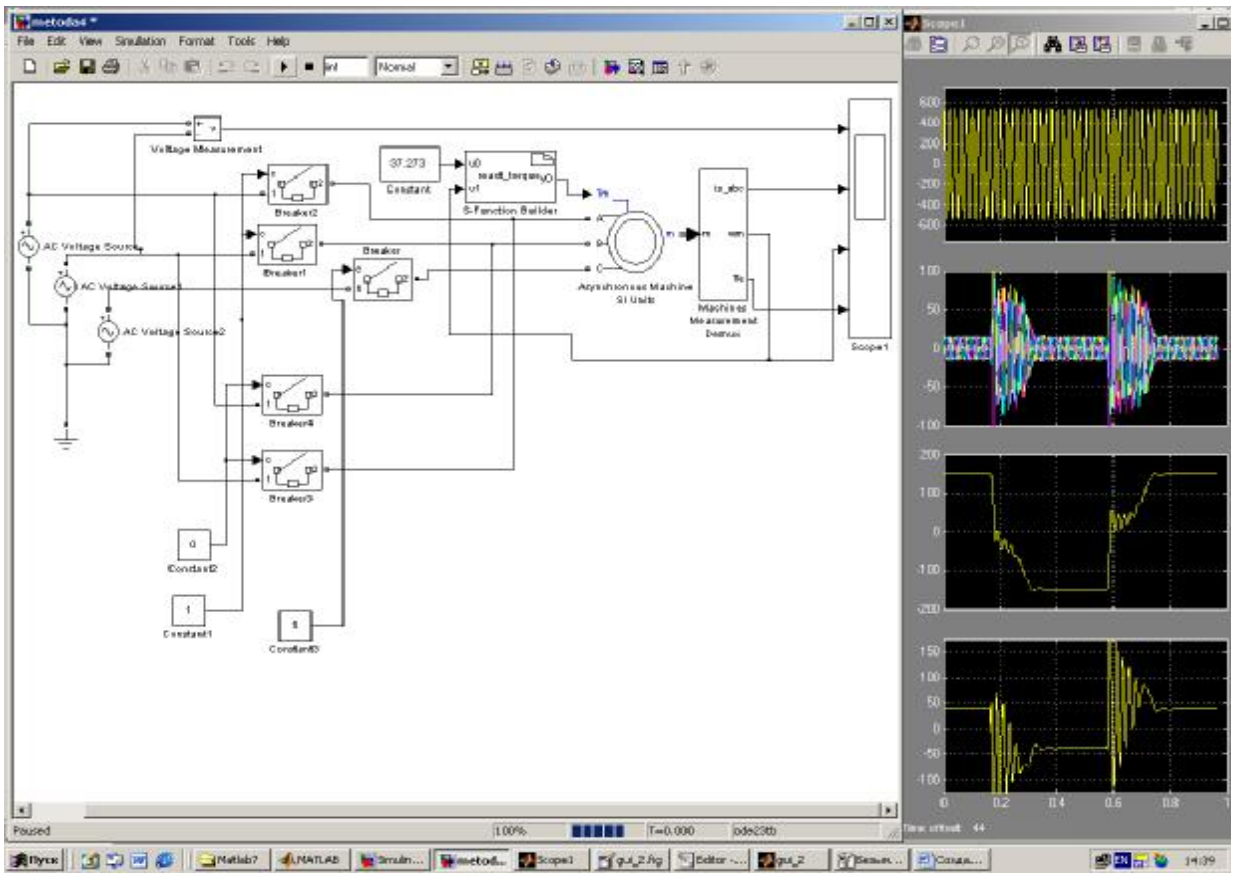

Рис.18

Для изучения работы данного компонента откройте новое окно редактора модели и соберите систему, показанную на рис.19.

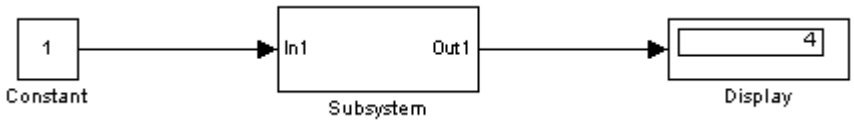

Рис.19

Сделайте двойной щелчок мыши по компоненту Subsystem, в результате откроется окно редактирования подсистемы (содержимое окна приведено на рис.20.).

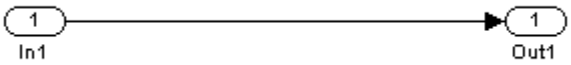

Рис.20

Как можно видеть, данная подсистема передает входной сигнал с порта входа in1 без изменений на порт выхода out1. Разорвем линию, соединяющую порты in1 и out1 (выделите мышью линию и нажмите клавишу Delete). Установим в разрыв компонент Gain (домножение на коэффициент), который находится в группе Simulink, подгруппа Commonly Used Blocks, и соединим его вход и выход с портами (рис.21.). Задайте значение коэффициента равным «5» (окно свойств компонента откроется после двойного щелчка мышью по нему).

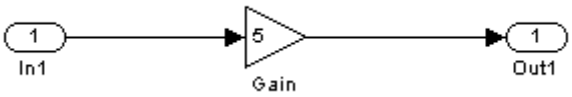

Рис.21

Вернемся к предыдущему окну редактирования модели с подсистемой. Для удобства понимания действия подсистемы присвоим ее имя, показывающее её суть. Для этого под компонентом на слове Subsystem щелкните мышью, удалите имеющееся название подсистемы и введите «multiplication 5» (домножение 5), после чего щелкните указателем мыши над компонентом. В результате компонент приобрел комментарий, способствующий лучшему пониманию работы всей модели.

Примечание. Аналогичные действия можно проводить с любым компонентом. Не рекомендуется печатать названия русскими буквами.

Запустите моделирование, в результате в компоненте Display появится значение, соответствующее перемножению исходной константы на коэффициент подсистемы.

Реализованная подсистема имеет 1 вход и один выход и относится к классу SISO (single input – single output). Обычно подсистемы имеют несколько входов и выходов (системы MIMO – multyinput - multyoutput). Технология установки дополнительного выхода выглядит следующим образом:

1. Откройте окно подсистемы.
2. Выделите мышью выход out1 .
3. скопируйте компонент out1 в буфер обмена (например, с помощью комбинации клавиш Ctrl-C).
4. Вставьте скопированный компонент из буфера обмена в окно редактирования подсистемы (например, с помощью клавиши Ctrl-V).
5. Расположите появившийся компонент Out2 рядом с компонентом Out1.
6. Убедитесь, что на компоненте исходной подсистемы появился дополнительный выход Out2.

Установка дополнительного порта входа происходит аналогично.

Для демонстрации действия созданной многовыходной подсистемы, выполните указанные действия, после чего соберите схему модели, показанную на рис. 22

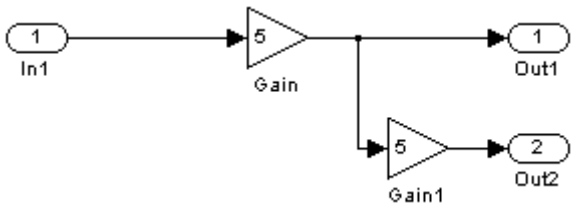

Рис.22

Подсоедините к новому выходу компонент Display (рис.23) и запустите моделирование.

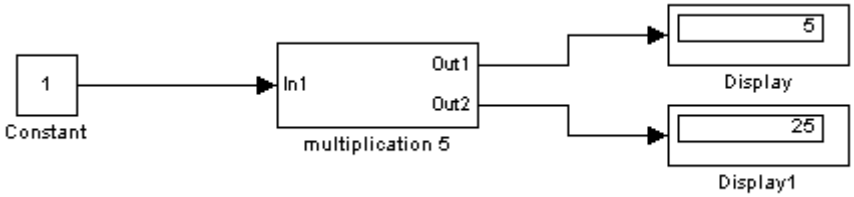

Рис.23

Таким образом, с помощью компонента подсистем Subsystem можно разгрузить окно редактирования от большого количества компонентов, объединив некоторые из них в функционально законченные блоки.

Задание для самостоятельной работы.

В разработанной модели создайте подсистемы, в которых объединены константы задания работы реверсивной и нереверсивной контакторных групп.

Варианты индивидуальных заданий (выполняются на основе модели рисунка 18)

1. Доработать модель таким образом, чтобы напряжение трехфазной сети менялось посредством слайдера интерфейса GUI.
2. Доработать модель таким образом, чтобы момент нагрузки изменялся в зависимости от скорости вращения двигателя (с увеличением скорости момент нагрузки возрастает).
3. Разработать S-функцию, позволяющую в данной модели определять наличие обрыва фазы двигателя.
4. Разработать S-функцию, позволяющую в данной модели реализовать времятоковую защиту.
5. Используя выход модели двигателя по углу поворота, разработать систему, обесточивающую двигатель при достижении заданного значения угла поворота.
6. Вывести текущий режим работы («останов», «движение вперед», «движение назад») на окно интерфейса GUI в текстовом формате.
7. Вывести значение действующего тока двигателя на окно интерфейса GUI.
8. Вывести значение действующего напряжения электрической сети на окно интерфейса GUI.

Занятие 2. Разработка систем логического управления электроприводами

Рассмотрим структуру логического управления на примере электропривода позицирования. В данном примере требуется создать систему логического управления, позволяющую автоматически производить останов при достижении одного из двух заданных положений механизма. В качестве реального механизма будет служить задвижка трубопровода (рис.1), где требуется производить автоматическое отключение двигателя при достижении положения «Открыто» при открытии и положения «Закрыто» при закрытии.

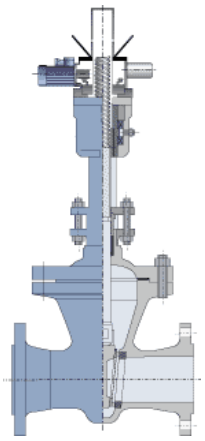

Рис. 1.

Техническое задание для рассматриваемого примера будет следующее:

1. Электропривод должен отрабатывать команды «Открыть», «Закрыть», «Стоп» с остановом при достижении заданного положения.
2. Электропривод должен выдавать сигнализацию о достижении конечных положений «Открыто» и «Закрыто»
3. Электропривод должен выдавать сигнализацию о своем состоянии – «Движение на открытие», «Движение на закрытие», «Остановлен».

Для разработки систем логического управления Симулинк предлагает модуль Stateflow (найдите его группу в библиотеке компонентов).

В группе Stateflow находится единственный компонент под именем Chart. Данный компонент позволяет реализовывать конечный авто-

мат - одно из самых широкоприменяемых устройств для логического управления.

Конечный автомат является событийно-управляемым устройством. В таких системах происходит переход между заданными заранее состояниями, смена состояний выполняется вследствие возникновения различных событий.

Например, при управлении двигателем в реверсивном режиме можно выделить 3 логических состояния:

1. Остановлен
2. Движение вперед
3. Движение назад

Событиями управления, по которым происходит смена указанных состояний, служат команды «Остановить», «Двигаться вперед» и «Двигаться назад».

Каждое состояние предполагает выполнение каких-то специфичных действий, характерных для данного состояния: в состоянии «Остановлен» необходимо снять напряжение с двигателя, в состоянии «Движение вперед» необходимо подать напряжение прямой последовательности, в состоянии «Движение назад» необходимо подать напряжения обратной последовательности. Пример графического представления данного конечного автомата показан на рис.2.

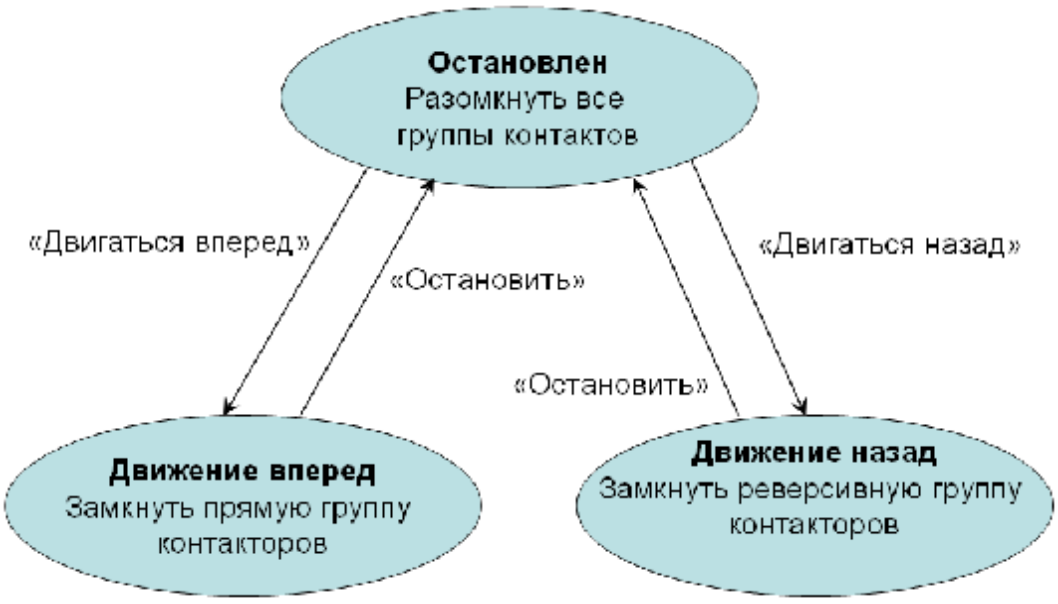

Рис.2

Откройте окно редактирования модели и поместите компонент Chart из группы Stateflow. Сделайте двойной клик по компоненту, и перед вами откроется редактор диаграмм, в котором определяются все состояния и устанавливаются переходы между ними.

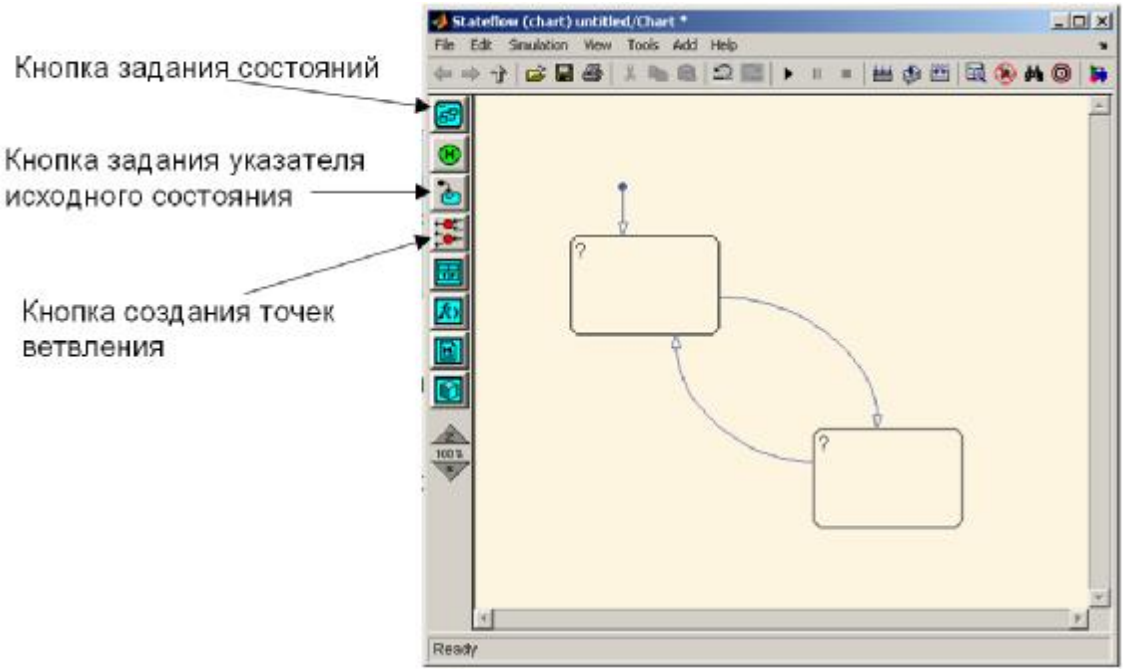

Рис.3

При помощи кнопки задания состояний создайте 2 прямоугольника. Подведите указатель мыши к краю прямоугольника и нажав на левую кнопку мыши, вытяните стрелку от одного прямоугольника до другого. Таким образом устанавливаются переходы между состояниями. С помощью кнопки задания исходного состояния, нарисуйте стрелку, показывающую на верхний прямоугольник.

В результате вы должны получить диаграмму, аналогичную с рис.3.

Кликните в прямоугольнике по знаку «?» и получите возможность задания имени состояния (рис.4.). Назовем одно состояние stop (остановлен), другое move (движение)

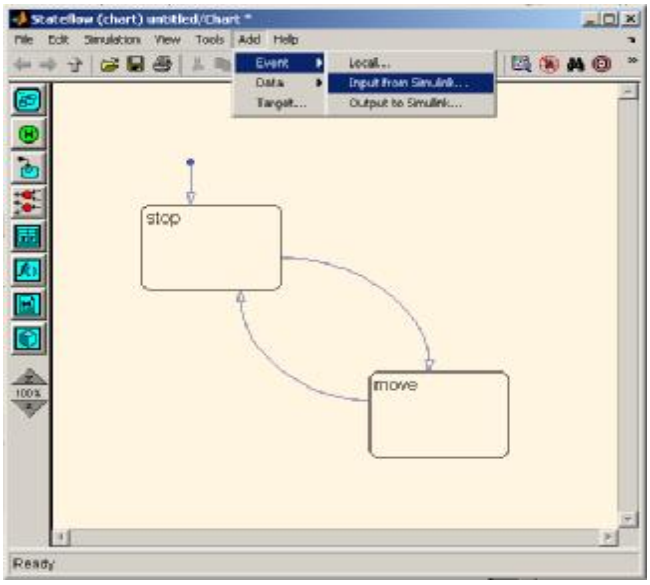

Рис.4

Для того, чтобы осуществлялись переходы между состояниями, необходимо обеспечить ввод событий в систему. Выполните через меню окна редактора диаграмм команду **Add-Event-Input from Simulink…** (Рис.5), т.е. события мы будем имитировать средствами Симулинка.

В появившемся окне (рис.) задайте имя события (my_event_1) и тип события в поле Trigger (по нарастающему фронту, т.е. Rising), нажмите кнопку OK.

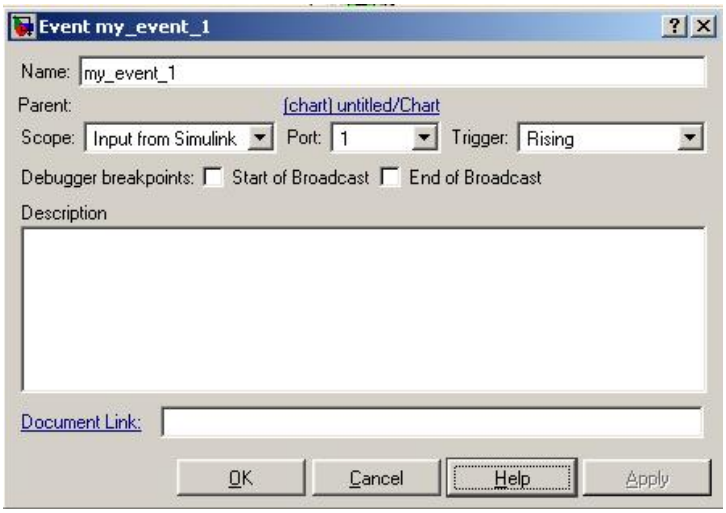

Рис.5

Аналогичным образом добавьте событие my_event_2 и тип события Falling (спадающий фронт).

Таким образом, мы задали два типа событий, вызывающие реакцию системы – на нарастающий фронт и спадающий фронт. Зададим события перехода между состояниями на диаграмме. Для этого необходимо выделить стрелку и вместо появившегося знака вопроса ввести имя события (рис.6)

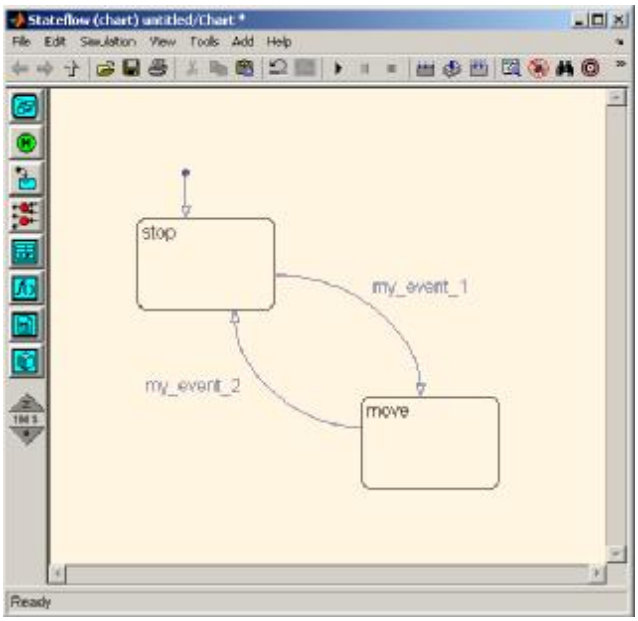

Рис.6

Так как события будут имитироваться средствами Симулинка, необходимо доработать модель системы в редакторе моделей (рис.7). Установите два компонента констант со значениями «1» и «-1». Установите ручной переключатель сигналов (группа Симулинк, подгруппа Signal Routing, компонент Manual Switch). В связи с тем, что в компоненте Chart мы установили 2 события (my_event_1 и my_event_2), необходимо превратить сигнал в вектор посредством мультиплексора ((группа Симулинк, подгруппа Signal Routing, компонент Mux).

Прежде чем запустить моделирование, зададим задержку при осуществлении переходов для того, чтобы существовала возможность наблюдать работу разработанного конечного автомата. Для этого необходимо подать команду Tools-Debug… (рис.8)

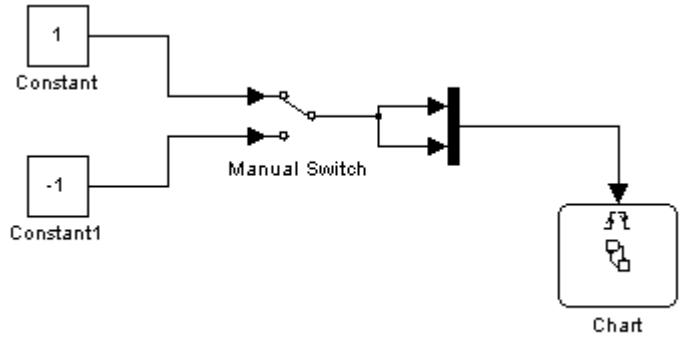

Рис.7

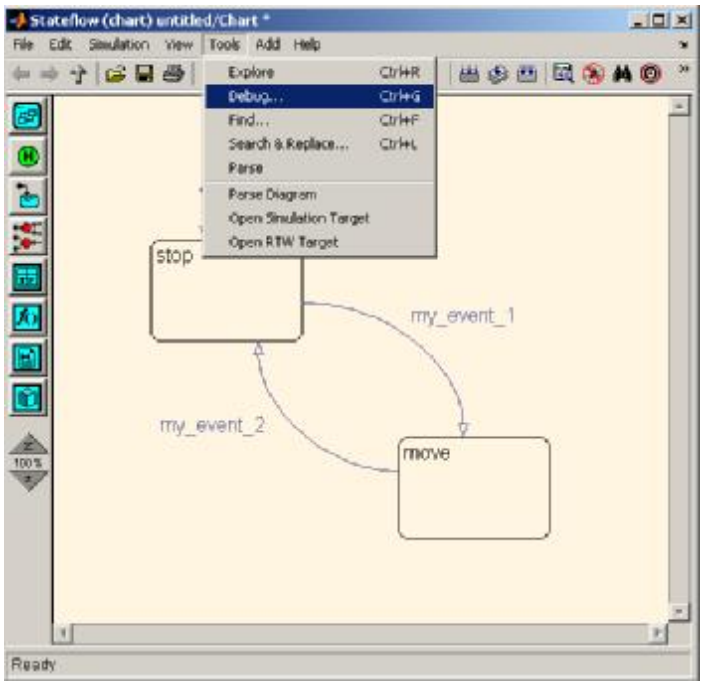

Рис.8

В появившемся окне необходимо включить анимацию (Animation – enabled) и установить время задержки (Delay) как 0,6 секунды (рис.9)

Задайте бесконечное время моделирования (inf), расположите окно редактора модели и редактора диаграмм рядом и запустите моделирование. Периодически делая двойной клик по переключателю сигналов, убедитесь в осуществлении заданных переходов между состояниями (активный переход или состояние выделяется синим цветом). Остановите моделирование.

Устройство управления, не выдающее во внешнюю среду никаких сигналов, является бесполезным. Добавим выход к разработанному ко-

нечному автомату. Данный выход будет показывать необходимость включения контакторов (значение «1») или отключения (значение «0»).

Для добавления выхода необходимо подать команду **Add-Data-Output to Simulink** (рис.10).

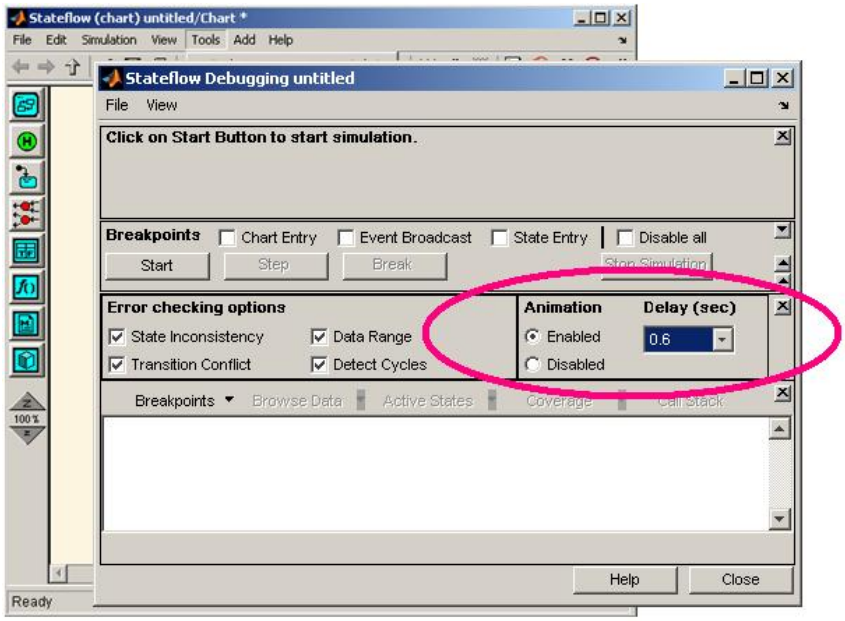

Рис.9

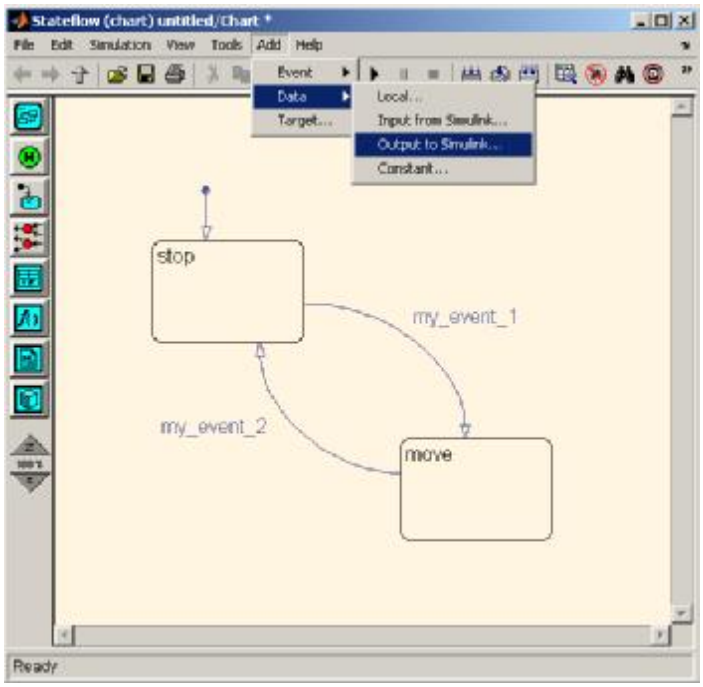

Рис.10

В появившемся окне задайте имя выхода (поле Name) как contactor (название с маленькой буквы!). В окне редактора в компоненте Chart появится выход с именем contactor. Данный выход необходимо соединить с компонентом Display для визуализации результата работы разрабатываемого конечного автомата (рис.11). В редакторе диаграмм необходимо определить действия по отношению данного выхода. Для этого в прямоугольниках, обозначающих состояния, необходимо задать (см.рис.11):

1. en:contactor=0 для состояния stop
2. en:contactor=1 для состояния move

Примечание. В каждом состоянии можно задавать действия над введенными переменными, для этого необходимо поставить метку «en: » и прописать желаемое действие.

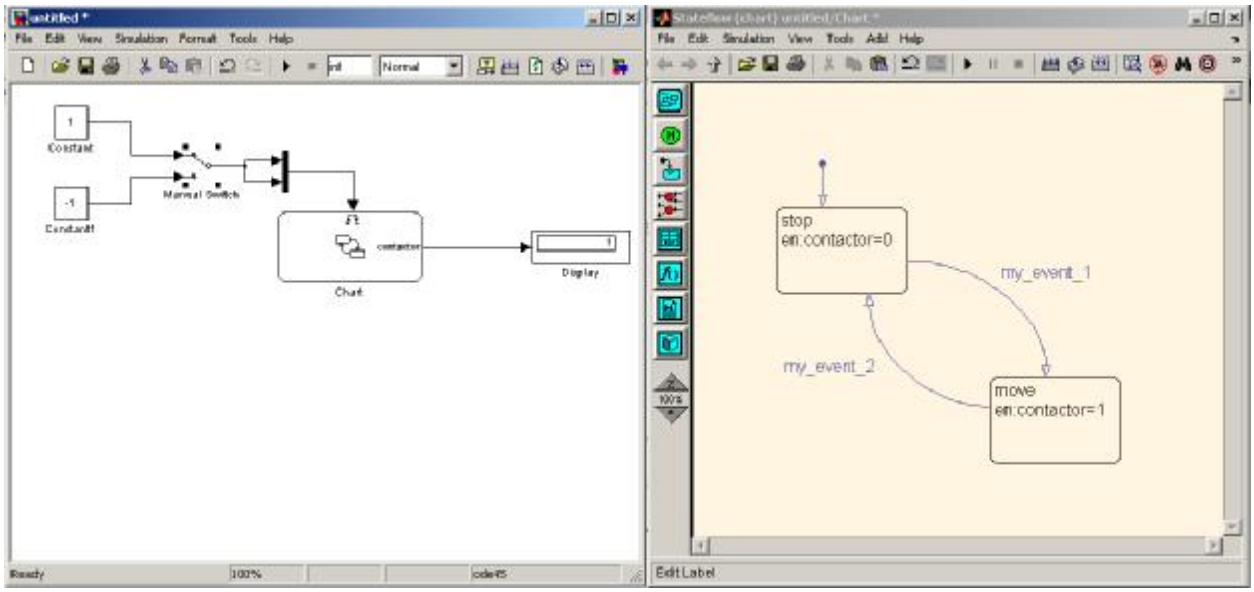

Рис.11

Запустите моделирование и убедитесь в изменении показания компонента Display при формировании событий с помощью компонента Manual Switch.

Усложним рассматриваемый пример за счет введения ветвления при осуществлении переходов и добавлении входа для данных. Допустим, что в системе существует датчик температуры двигателя. При превышении заданной температуры необходимо запрещать пуск двигателя. Для введения входа для датчика температуры необходимо подать команду **Add-Data-Input from Simulink** (рис.12)

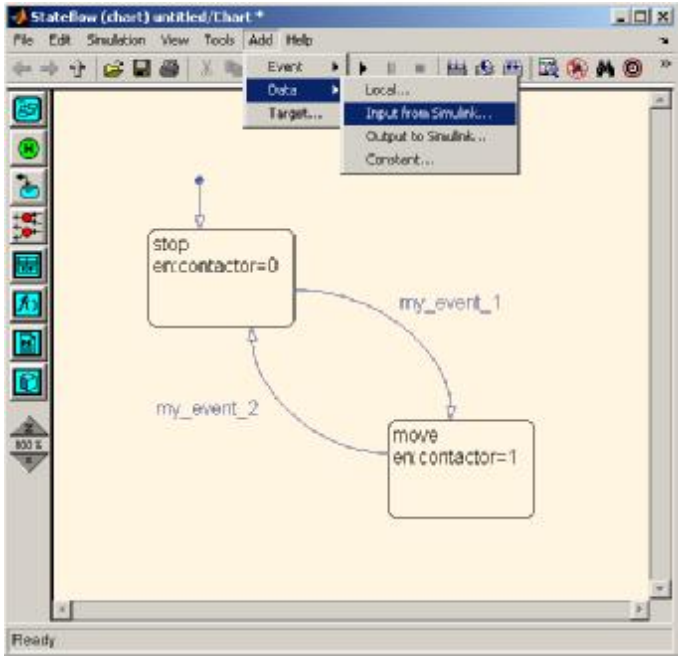

Рис.12

В появившемся окне задайте имя входа (поле Name) как temperature и нажмите ОК. В окне редактора модели к появившемуся входу temperature компонента Chart прикрепите константу для задания температуры (рис.13).

В редакторе диаграмм, используя кнопку создания точек ветвления (рис.3), создайте диаграмму как на рис.13 В отличии от событий, при формировании условных переходов по входу данных (в данном случае temperature) необходимо применять знаки «[ ]» (см. рис.13).

Промоделируйте работу конечного автомата (имитируя события через Симулинк) при задании температуры более 100 градусов и менее 100 градусов.

Примечание. При разработке конечных автоматов лучше использовать для входных воздействий не события, а данные. Но так как конечный автомат является событийно управляемой системой, то для активации работы автомата необходимо постоянно создавать периодические события, например, путем подключения генератора синусоидальных сигналов к событийному входу.

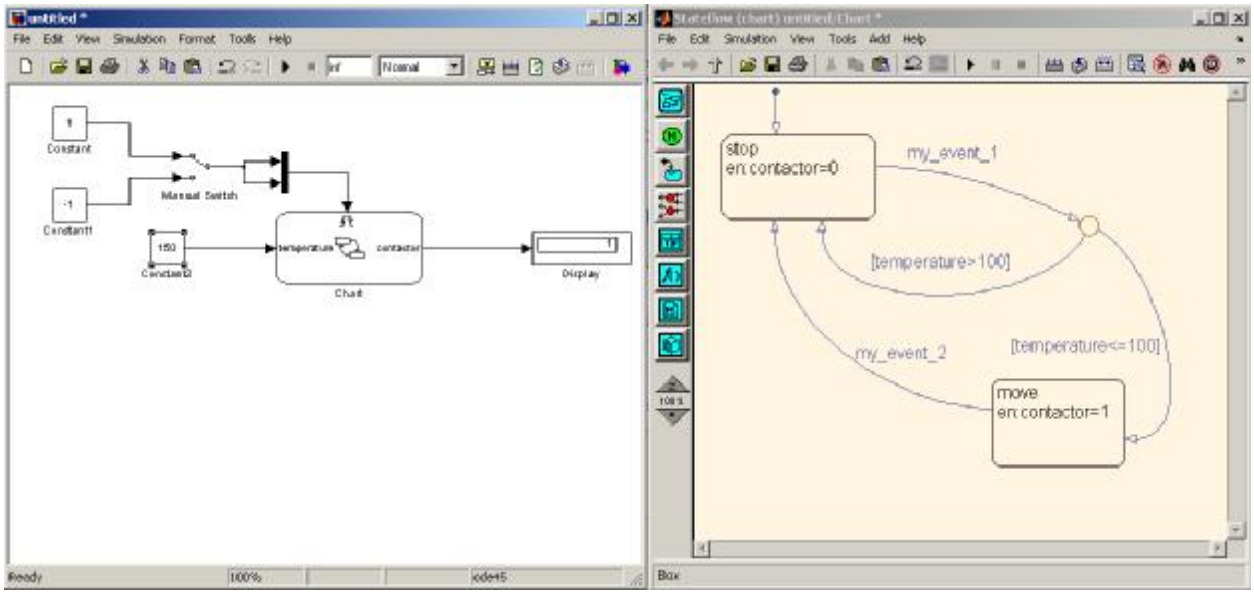

Рис.13

Задание для самостоятельной работы.

1. Разработайте конечный автомат согласно рис.2. Для активации автомата примените синусоидальный генератор. В результате конечный автомат должен иметь 2 выхода (управление прямым и реверсивным контакторами), 1 событийный вход (для генератора синусоидальных сигналов), 3 входа данных для управления (команды «Вперед», «Назад», «Стоп») и 3 состояния. . В качестве кнопок управления для подачи команд и индикаторов состояний используйте элементы группы Dials&Gauges Blockset, библиотека Global Majic ActiveX library (Buttons – кнопку, LEDs - светоиндикаторы).
2. На базе предыдущей реализации конечного автомата, разработайте конечный автомат для управления электроприводом задвижки согласно представленному ранее техническому заданию. В качестве сигнала датчика положения используйте выход модели двигателя по повороту вала двигателя
3. Изменение момента нагрузки в задвижке происходит согласно нагрузочной диаграмме в функции от положения клина задвижки. Модифицируйте S-функцию, отвечающую за механизм таким образом, чтобы она отражала реальную работу задвижки.

В реальных системах управления применяется программный код. Данный код может быть реализован различными методами:

1. Метод логической функции

2. Табличный метод

3. Алгоритмический метод.

Рассмотрим создание программы логического управления алгоритмическим методом.

В качестве языка программирования будем использовать язык С, краткие сведения о котором приведены ниже (этих сведений достаточно для написания S-функций в среде Симулинк)

Основными средствами языка С, которые будут использоваться в данном практикуму, являются следующие:

– Операторы +, -, *, /, <, >, <=, >=, = =,*!*=,
– Комментарии вида / * ... * /; //
– Набор встроенных функций;
– Оператор *if , if … else ….*

В рассматриваемом языке существует ключевые слова, которые нельзя использовать для названий переменных и функций. Эти слова следующие: *var, void, main, while, return, if, else.*

В языке определены комментарии одного вида: такой комментарий начинается с символа /* и заканчивается символами */. Между звездочкой и слешем не должно быть никаких пробелов. Любой текст, расположенный между начальными и конечными символами комментария, программой игнорируется.

Комментарии могут находиться в любом месте программы и могут быть многострочными (начинаться в одном месте, а заканчиваться в другом). Комментарии не могут быть вложенными. То есть в одном комментарии не может находиться другой.

Если необходимо закомментировать одну строчку программы, можно использовать сочетание //.

При объявлении перменных необходимо руководствоваться следующим: первый символ должен быть обязательно буквенным, или символом подчеркивания, последующие символы должны быть буквами, цифрами или символами подчеркивания. Например: *temp, count_ , set523* - это правильная запись, а *5count, t!set, r...temp* – неправильная

Общая форма объявления переменной имеет такой вид:

float static x = 0; - для задания переменной «x» в формате плавающей запятой и начальной инициализацией «0»

int static y; для задания переменной «y» в формате целого числа.

В языке предусмотрены основные математические операции:
- - Вычитание, также унарный минус (умножает операнд на -1)
+ - Сложение
* - Умножение
/ - Деление

Приоритет выполнения арифметических операций следующий:

Высокий: - (унарный минус)
Средний: * /
Низкий: + -
Операции с одинаковым приоритетом выполняются слева направо. Используя круглые скобки можно изменить порядок вычислений, они придают операции (или последовательности операции) наивысший приоритет.
Пример:
```
int  x = -30, y = 0;
x = x * 5 - 7 / (35 + 6); /* Здесь переменная примет значение -150.017 */
y = x - 10 * 20; /* Здесь переменная примет значение -350.017 */
```

В языке предусмотрены следующие операции сравнения:
> - Больше чем
>= - Больше или равно
< - Меньше чем
<= - Меньше или равно
!= - Не равно
== - Равно

Как и в арифметических выражениях, для изменения порядка выполнения операций сравнения можно использовать круглые скобки.

Общая форма оператора *if* следующая:
```
if (выражение) { оператор; }
else { оператор; }
```

Здесь оператор может быть одним оператором или блоком операторов. Фраза else может отсутствовать.

Если выражение истинно, то выполняется оператор, следующий за if. В противном случае выполняется оператор, следующий за else.

Пример реализации оператора *if*:
```
int x = 30, y1 = 0, y2 = 0;
if (x > 0)  y1 = 10; /* Переменная примет значение 10 */
```

if (x == 20) y2 = 5;} / Переменная примет значение 20 */*
else y2 = 20;

Разработка программы управления может быть сведена к следующему:

1. Выбираются логические режимы работы электропривода, характерные и необходимые для достижения цели функционирования электропривода.
2. На основе выделенных логических режимов работы формируются компоненты, позволяющие достичь цели каждого из режимов.
3. В каждом компоненте количественно определяется условие достижения цели работы компонента.
4. Для каждого компонента разрабатывается процедура управления, которая позволяет достигать требуемую область цели.
5. Для ключевых событий компонента (достижения цели, выход за рабочую область наблюдения координат и т.д.) формируется номер того компонента, который позволяет наиболее эффективно решать задачи управления при текущем состоянии координат и событиях системы.

Разработанное по данной методике управление может быть представлено графически. На основании выделенных компонентов и условий переходов рисуется граф системы. В узлах графа показываются функции управления и восстановления координат, на ребрах – условия формирования событий переходов между узлами. Каждое ребро является однонаправленной стрелой, указывающее направление перехода для данного события. Предлагаемый алгоритм образует каркас управления системы, позволяющий выделить все логические состояния системы управления и соблюсти ограничения на работоспособность процедур. Каркас задает иерархию выполнения алгоритмов для последовательного достижения целей управления, и таким образом формирует последовательно-иерархическое управление электроприводом.

Реализация логического управления по приведенному техническому заданию, будет выглядеть следующим образом:

float static out_o=0;
float static out_z=0;
float static out_oing=0;
float static out_zing=0;
float static command_=0;
float static speed=0;
float static position=50;

```
float static mode=1;
float static a=0;

void main()
{while(1)
 { if (mode==0)
        { speed=0; out_oing=0;out_zing=0;
        if (position==0) {out_z=1;out_o=0;}
        if (position==100) {out_z=0;out_o=1;}
        if (command_==1) {mode=1;}
        if (command_==2) {mode=2;}
        }

 if (mode==1)
        { speed=50; out_z=0;
        out_o=0; out_oing=1;
        if (command_==0) {mode=0;}
         position=position+1;
         if (position>100) position=100;
        if (position==100) { mode=0; }
        }

 if (mode==2)
         { speed=-50;
        out_zing=1;
        out_z=0;out_o=0;
         if (command_==0) {mode=0;}
         position=position-1;
         if (position<0) position=0;
         if (position==0) { mode=0;}
         } command_=55; a=0; while(a<55555) { a=a+1; }
}
}
```

Задание для самостоятельной работы

Реализуйте данное управление в виде S-функции для управления моделью электропривода задвижки.

Занятие 3. Создание цифровых регуляторов

В данной работе продемонстрирован процесс создания различных цифровых регуляторов. Созданные цифровые регуляторы предназначены для управления скоростью двигателя постоянного тока. В целях упрощения, контур тока исключен.

Перед тем, как перейти к разработке регулятора для двигателя, попробуем разработать систему регулирования тока в RL-цепочке посредством цифрового регулятора. Для начала разработаем модель RL цепочки за счет численного решения дифференциального уравнения: *U=R*i+L*di/dt*. Для решения данного уравнения на компьютере его необходимо дискретизировать, то есть перейти к уравнению в конечно-разностной форме и решить его. Технология такого перехода будет показана на примере апериодического звена, которое, как известно, описывается практически той же передаточной функцией, что и RL-цепочка:

$$W(p) = \frac{1}{1 + T_\phi p},$$

где T_ϕ – постоянная времени инерционного звена (фильтра). Если обозначить входной сигнал как $x(t)$, а выходное управляющее воздействие, подаваемое на вход объекта управления, как $y(t)$, то передаточной функции инерционного звена будет соответствовать дифференциальное уравнение:

$$y(t) + T_\phi \frac{dy(t)}{dt} = x(t).$$

Особенность расчетов любой цифровой системы, в том числе и при моделирование на ЭВМ в среде Симулинк, состоит в том, что от момента получения информации о состоянии входных величин, в данном случае $x(t)$, до момента выдачи, рассчитанного в соответствии с заданным алгоритмом значения управляющего воздействия $y(t)$, проходит определенное время. Следующую выборку данных можно получить только спустя некоторое время, называемое **интервалом квантования** по времени или **периодом дискретизации** по времени *T*. Таким образом, входная информация доступна только в фиксированные моменты времени $t = kT$, где k – целое число 0, 1, 2, ..., представляющее собой номер очередной выборки данных. Значения входной $x(t)$ и выходной $y(t)$ функций определены только в дискретные моменты времени $t = kT$. Такие дискретные функции называются **решетчатыми**, а уравнения, связывающие между собой дискретные переменные - **дискретными**

разностными уравнениями. Согласно теореме Котельникова (или Шеннона в западном варианте), если цифровой регулятор или цифровой фильтр имеет частоту дискретизации по крайней мере вдвое большую, чем максимальная частота входного сигнала, то он может быть реализован без существенных погрешностей на базе своего аналогового прототипа. При выполнении этого условия существует переход от дифференциального уравнения к разностному — значения непрерывных величин заменяются дискретными, а значения дифференциалов непрерывных функций — так называемыми **разностями**, то есть величинами приращений дискретных переменных за период дискретизации *T*.

Аналогом дифференциала первого порядка для непрерывной функции при этом является **первая обратная разность** (первая левая разность):

$$\nabla f(k) = f(k) - f(k-1).$$

Для перехода от непрерывного уравнения к разностному заменим непрерывные переменные дискретными, дифференциалы этих переменных – разностями, а приращение времени dt – величиной интервала квантования по времени T. Получим:

$$y(k) + \frac{y(k) - y(k-1)}{T} \cdot T_\phi = x(k).$$

После преобразований дискретное уравнение примет вид:

$$y(k) = \frac{T}{T + T_\phi} \cdot x(k) + \frac{T_\phi}{T + T_\phi} \cdot y(k-1).$$

Обозначим коэффициенты:

$$k_0 = \frac{T}{T + T_\phi} \quad \text{и} \quad k_1 = \frac{T_\phi}{T + T_\phi}.$$

Тогда, с учетом полученных коэффициентов k_0 и k_1, дискретное уравнение инерционного звена выглядит следующим образом:

$$y(k) = k_0 \cdot x(k) + k_1 \cdot y(k-1)$$

Задание для самостоятельной работы

Перейдите от дифференциального уравнения RL-цепочки *U=R*L+L*di/dt* к конечно-разностной форме и получите формулу для расчета текущего значения тока.

Реализуем модель RL цепочки посредством механизма S-функции Симулинка. На вход такой S-функции будет подаваться сигнал напряжения, на выходе – сигнал тока. Текст такой S-функции приведен ниже.

```
float static R=3.6;
float static L=0.034;
```

float static i=0;
float static di=0;
float static u=0;
float static t=0;
float static dt=0.001;

u=u0[0];// забираем входной сигнал - напряжение
t=t+dt; // расчитываем текущее время
*i=i+(u-R*i)/L*dt; // расчитываем ток*
y0[0]=i;// выдача результата

Примечание

1. При повторном запуске модели Симулинка, необходимо заново перетранслировать S-функцию посредстом нажатия кнопки Build в окне S-function builder для задания нулевых начальных условий модели. Если применяются несколько компонентов S-функций, необходимо перетранслировать заново каждый перед началом каждого моделирования.
2. *float static* используется для задания имени новой переменной, применяемой в модели.

При генерации S-функции установите дискретный характер вычислений компонента, период дискретизации 0.001 секунда.

Результаты моделирования показаны на рис.1.

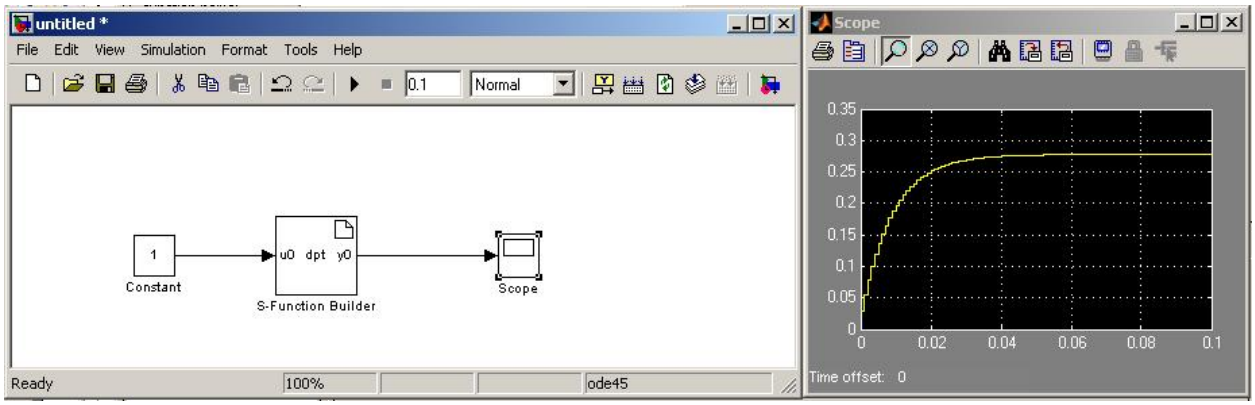

Рис.1.

Повторим моделирование с увеличенным напряжением (рис.2.):

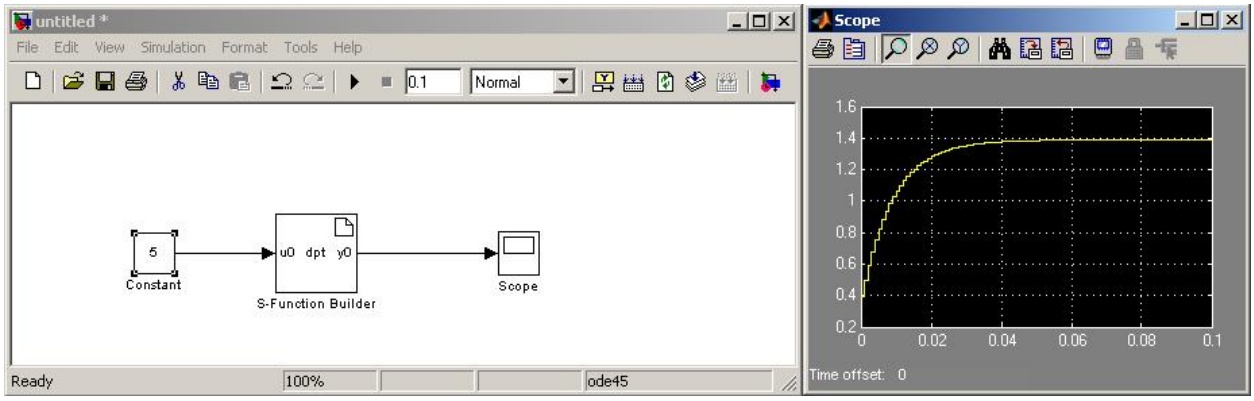

Рис.2

Как можно видеть, при увеличении входного сигнала происходит соответствующее увеличение выходного сигнала, форма выходного сигнала одинакова и соответствует реальному поведению тока в RL-цепочке.

Для регулирования тока можно применить простейший цифровой регулятор, который работает по принципу: если выходной сигнал меньше заданного, то увеличиваем напряжение, если больше – то уменьшаем.

В классической теории автоматического управления принято управлять по ошибке (рассогласованию между входной и выходной координатами). С учетом этого, указанный принцип несколько модифицируется:

если ошибка по току меньше нуля, то уменьшаем текущее напряжение на определенное значение, если больше – то увеличиваем.

Для реализации регулятора создадим S-функцию reg, содержимое которой представлено ниже.

```
float static input=0;
float static output=0;

input=u0[0];
if (input<0) output=output-0.01;
if (input>0) output=output+0.01;
y0[0]=output;
```

Создадим систему управления током RL-цепочки с использованием этих разработанных S-функций. Модель системы управления показана на рис.3. Как можно видеть, для регулирования используется сигнал об-

ратной связи по току, который вычитается из заданного сигнала. При этом в обратной связи применен компонент Z^{-1}, который обеспечивает задержку в передаче выходного сигнала на сумматор на одну дискрету, то есть на сумматор подается значение, которое было рассчитано при предыдущем шаге расчета.

Задание для самостоятельной работы

В показанной модели установите время моделирования inf (бесконечность), запустите моделирование, и изменяя задания на ток, наблюдайте его изменение в модели RL-цепочки.

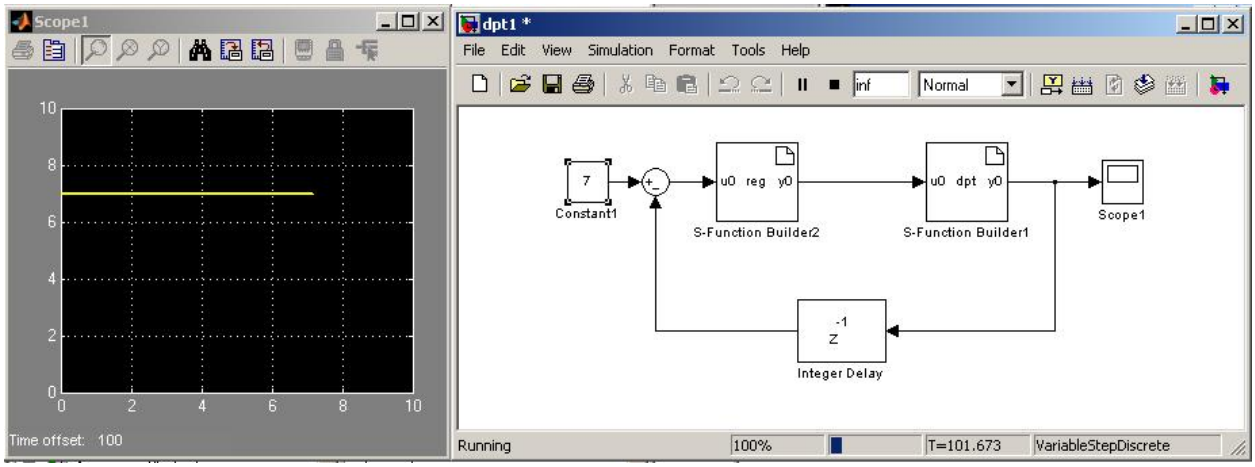

Рис.3

Перейдем теперь к разработке цифровой системы управления двигателем постоянного тока. За основу возьмем двигатель 2ПБ132МГ со следующими характеристиками:

Номинальная мощность $P_{\text{дв.н}} = 1{,}6$ кВт.

Номинальное напряжение $U_{\text{дв.н}} = 220$ В.

Сопротивление якорной цепи $R_{\text{яц}} = 3{,}553$ Ом.

Индуктивность якорной цепи $L_{\text{яц}} = 0{,}034$ Гн.

Момент инерции двигателя $J_{\text{дв}} = 0.038$ кг $\cdot$ м^2.

Коэффициент ЭДС при номинальном потоке возбуждения $c = 1.82 \dfrac{\text{В} \cdot \text{с}}{\text{рад}}$.

Изначально необходимо создать модель двигателя.

ДПТ может быть описан дифференциальными уравнениями

$$u - e = i \cdot r + L\frac{di}{dt},$$

$$M - M_c = J\frac{d\omega}{dt},$$

где u – напряжение на якоре, В;

r, L – активное сопротивление, Ом; индуктивность якорной цепи, Гн;

ω – угловая скорость вращения двигателя, $\frac{\text{рад}}{\text{с}}$;

J – момент инерции якоря двигателя;

e – ЭДС двигателя, В;

M, M_C – момент двигателя и статической нагрузки, Н·м;

Момент и ЭДС двигателя определяются по выражениям:

$M = c \cdot i$;

$e = c \cdot \omega$

Задание для самостоятельной работы

Преобразуйте дифференциальные уравнения, описывающие поведение двигателя постоянного тока в конечно-разностную форму и предложите формулу для расчета тока и скорости двигателя.

Реализация модели двигателя постоянного тока с помощью S-функции может быть выполнена следующим образом:

```
 float static R=3.6;
float static L=0.034;
float static i=0;
float static u=0;
float static t=0;
float static dt=0.001;

float static w=0;
float static Mc=0;
float static Me=0;
float static c=1.82;
float static J=0.038+0.0;
```

```
u=u0[0];// забираем входной сигнал - напряжение
Mc=u1[0];

t=t+dt; // расчитываем текущее время
i=i+(u-R*i-c*w)/L*dt; // расчитываем ток
Me=i*c;
w=w+(Me-Mc)/J*dt;

y0[0]=i;// выдача результата
y1[0]=w;
```

Для регулирования скорости применим тот же самый регулятор, который был разработан для RL-цепочки. С учетом ограничения на максимальное напряжение, которое можно приложить к двигателю, введем программное ограничение на выходное значение регулятора. Текст программы такого регулятора с ограничением выходного значения показан ниже:

```
 float static input=0;
float static output=0;

input=u0[0];
if (input<0) output=output-0.01;
if (input>0) output=output+0.01;
if (output>220) output=220;
if (output<-220) output=-220;
y0[0]=output;
```

Задание для самостоятельной работы

Соберите модель, показанную на рис.4., установите время моделирования 10 секунд и смоделируйте пуск двигателя при различных заданиях на скорость (компонент Constatnt) и заданиях на момент (компонент Constant2).

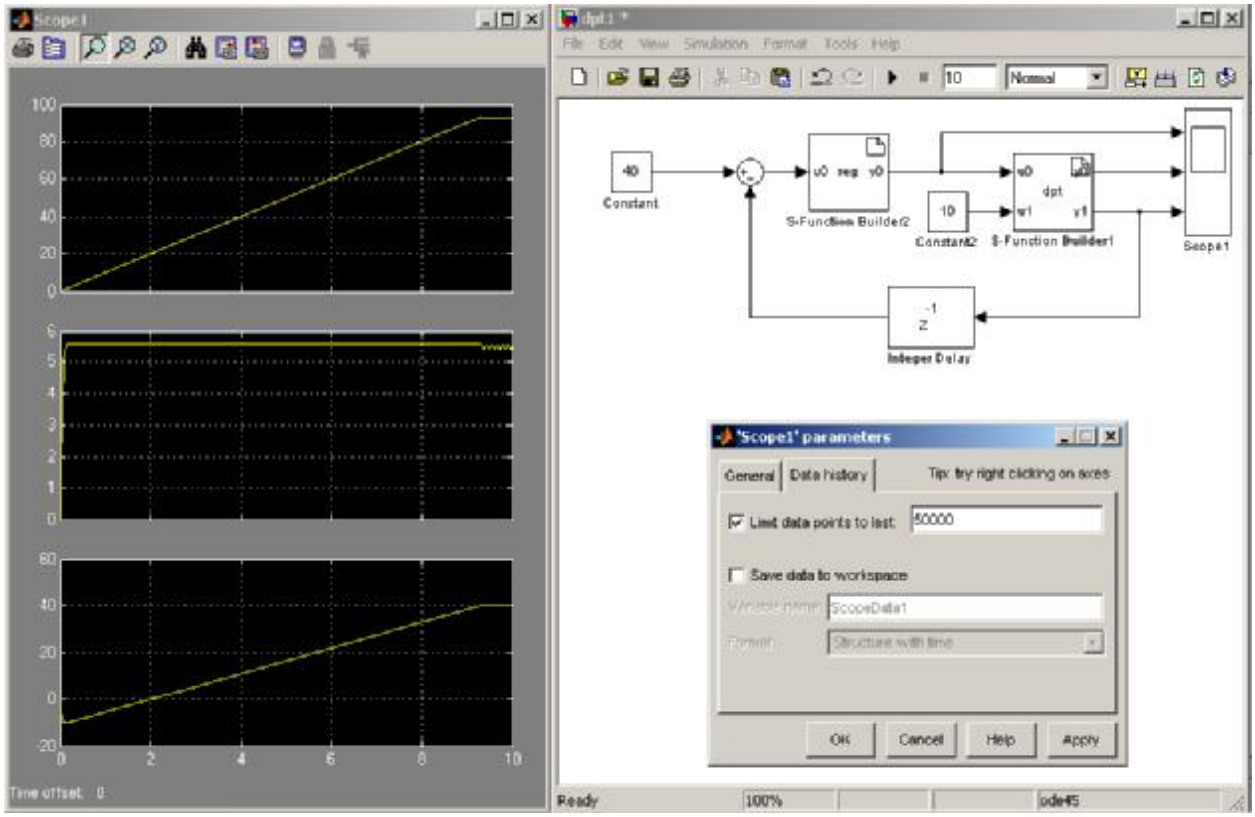

Рис.4.

Как можно увидеть при выполнении задания, двигатель выходит на заданную скорость при задании различных моментов, то есть система астатична, что объясняется интегрирующими свойствами примененного регулятора. Недостатком данного регулятора является низкое быстродействие – время выхода на заданную скорость составляет порядка 10 секунд (см.рис.4).

Задание для самостоятельной работы

Предложите свой способ увеличения быстродействия системы.

Применим пропорциональный регулятор с коэффициентом усиления 10. Для этого содержимое S-функции регулятора должно принять следующий вид:

```
float static input=0;
float static output=0;

input=u0[0];
output=10*input;
```

if (output>220) output=220;
if (output<-220) output=-220;
y0[0]=output;

Установите время моделирования 0,2 секунды. В результате должны получиться графики переходных процессов, идентичные представленным на рис.5. Как видно из графиков, установившаяся скорость отличается от заданной (42 против 50). При увеличении нагрузки (рис.6) различие увеличивается. Данный пример демонстрирует отсутствие астатизма П-регулятора.

Задание для самостоятельной работы

Предложите способ уменьшения ошибки регулирования при применении П-регулятора.

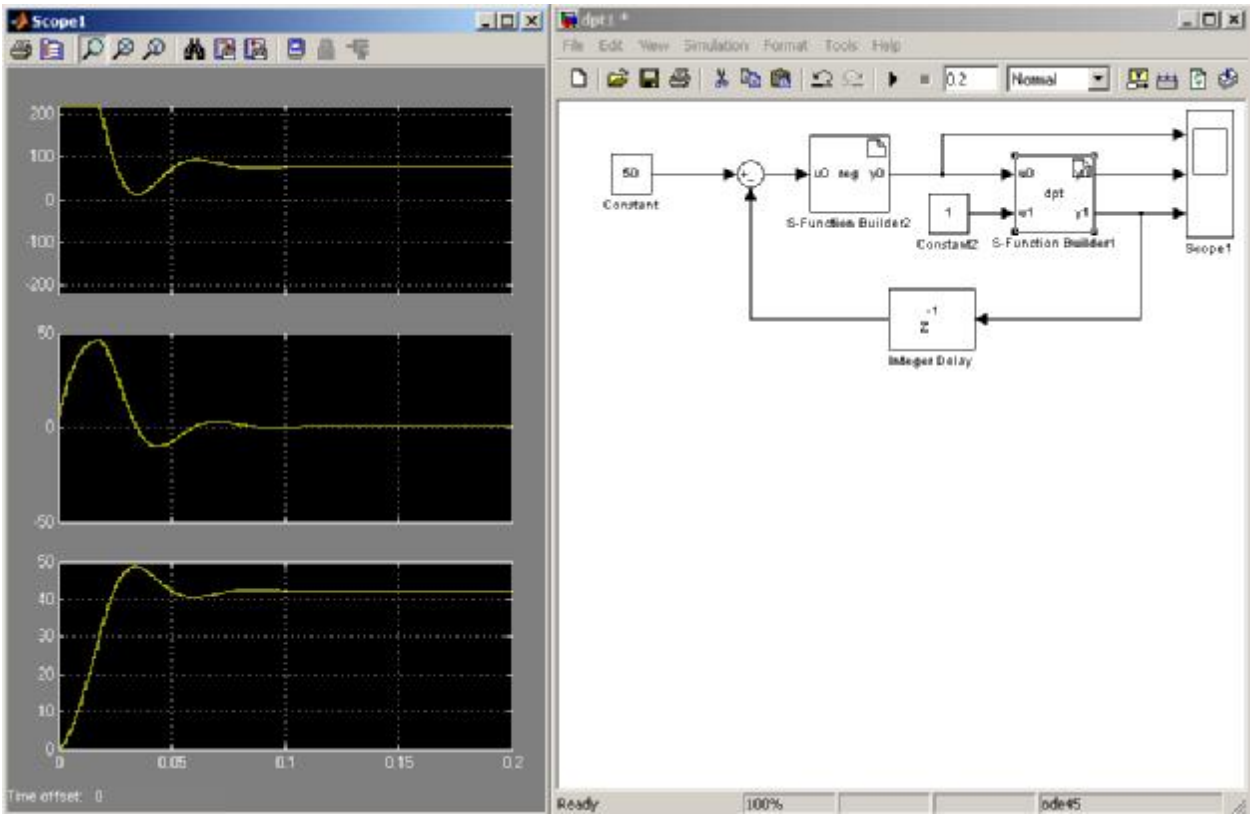

Рис.5

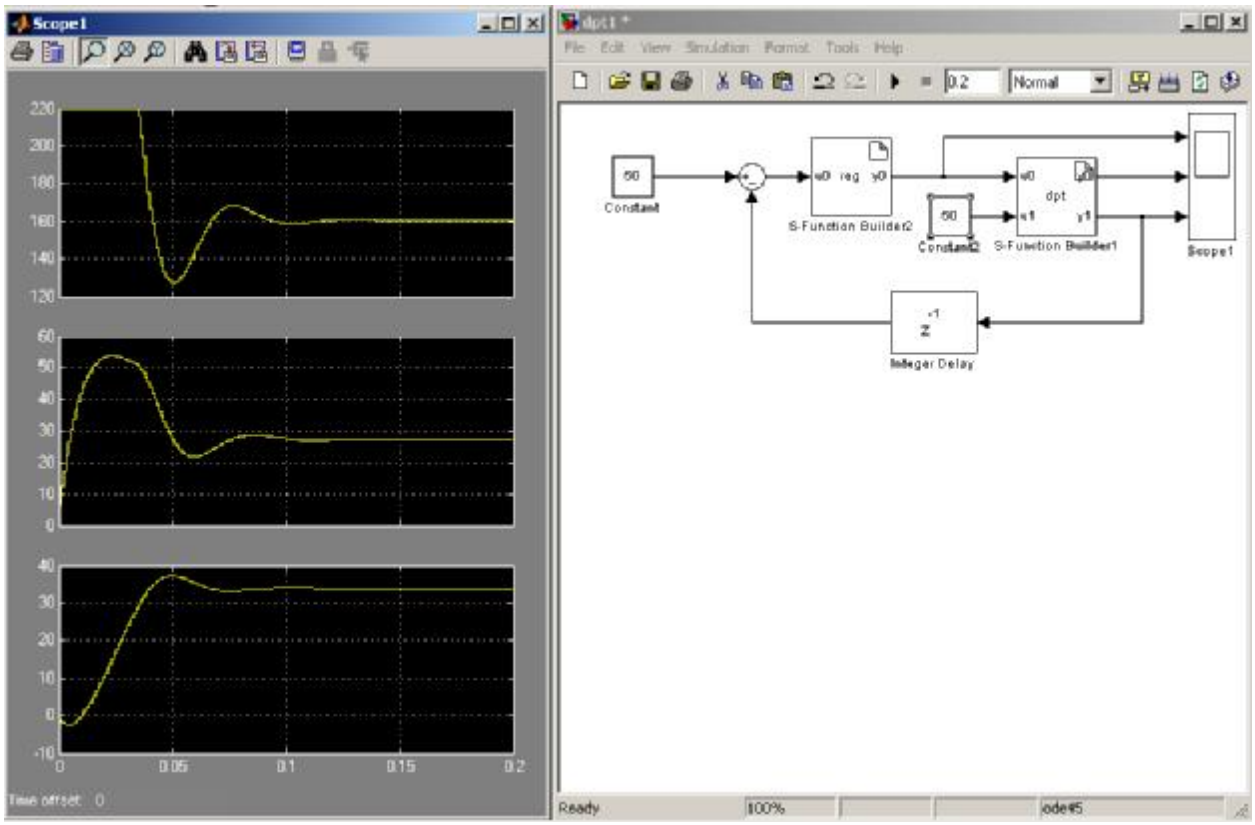

Рис.6

Еще одним простейшим регулятором является релейный. Его реализация будет выглядеть следующим образом:

```
float static input=0;
float static output=0;

input=u0[0];
if (input<0) output=-220;
if (input>=0) output=220;

if (output>220) output=220;
if (output<-220) output=-220;
y0[0]=output;
```

Релейный регулятор характеризуется резкими переключениями при регулировании, что ведет к колебательности процессов при срабатываний регулятора (см.рис.7).

Задание для самостоятельной работы

1. Упростите предложенный регулятор.
2. Изменяя задание на скорость и момент нагрузки, определите, является ли релейный регулятор астатичным.
3. Предложите способы устранения колебаний координат управляемой системы, возникаемых при работе релейного регулятора.

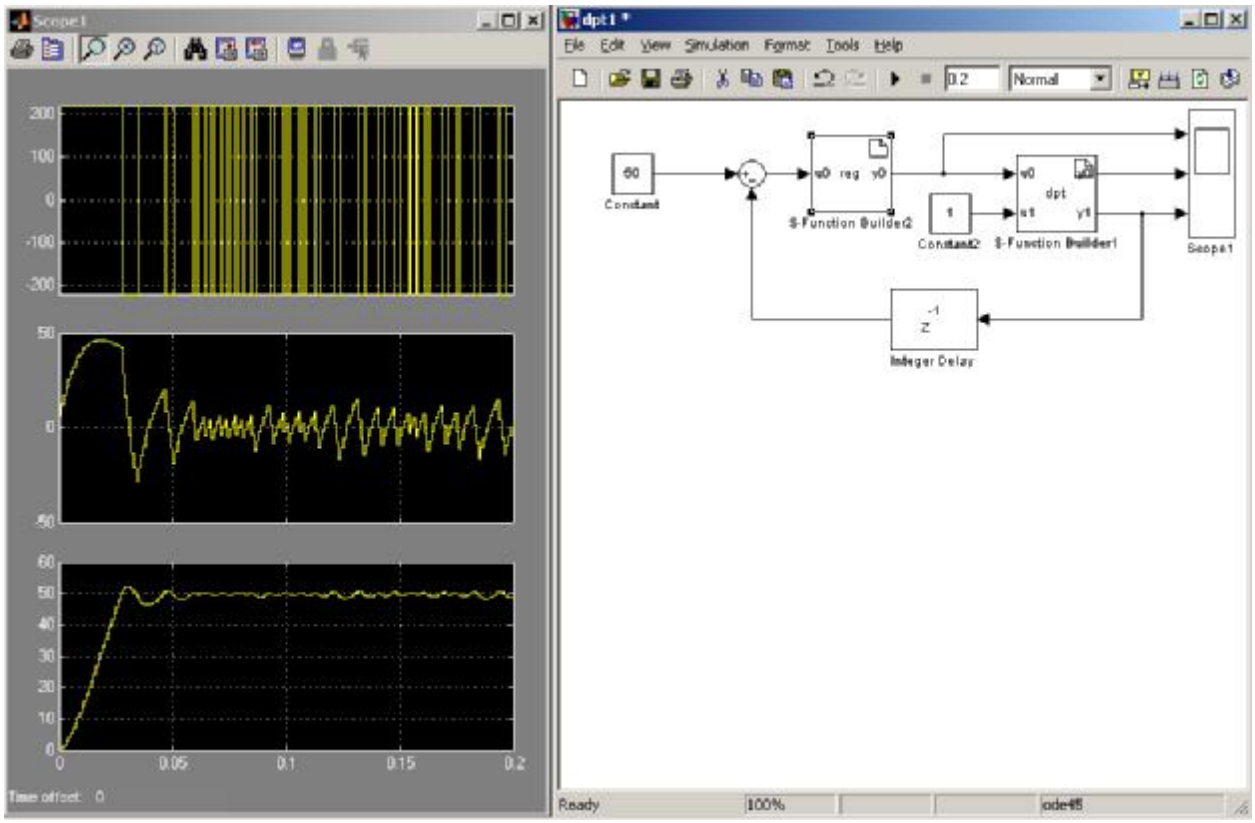

Рис.7

Одним из способов устранения недостатков работы релейного регулятора является введение зоны гистерезиса. Реализация такого регулятора будет следующей (ширина гистерезиса – от -3 до 3):

```
float static input=0;
 float static output=0;

 input=u0[0];
 if (input<-3) output=-220;
 if (input>3) output=220;
```

```
if (output>220) output=220;
if (output<-220) output=-220;
y0[0]=output;
```

Результат работы такого регулятора показан на рис.8.

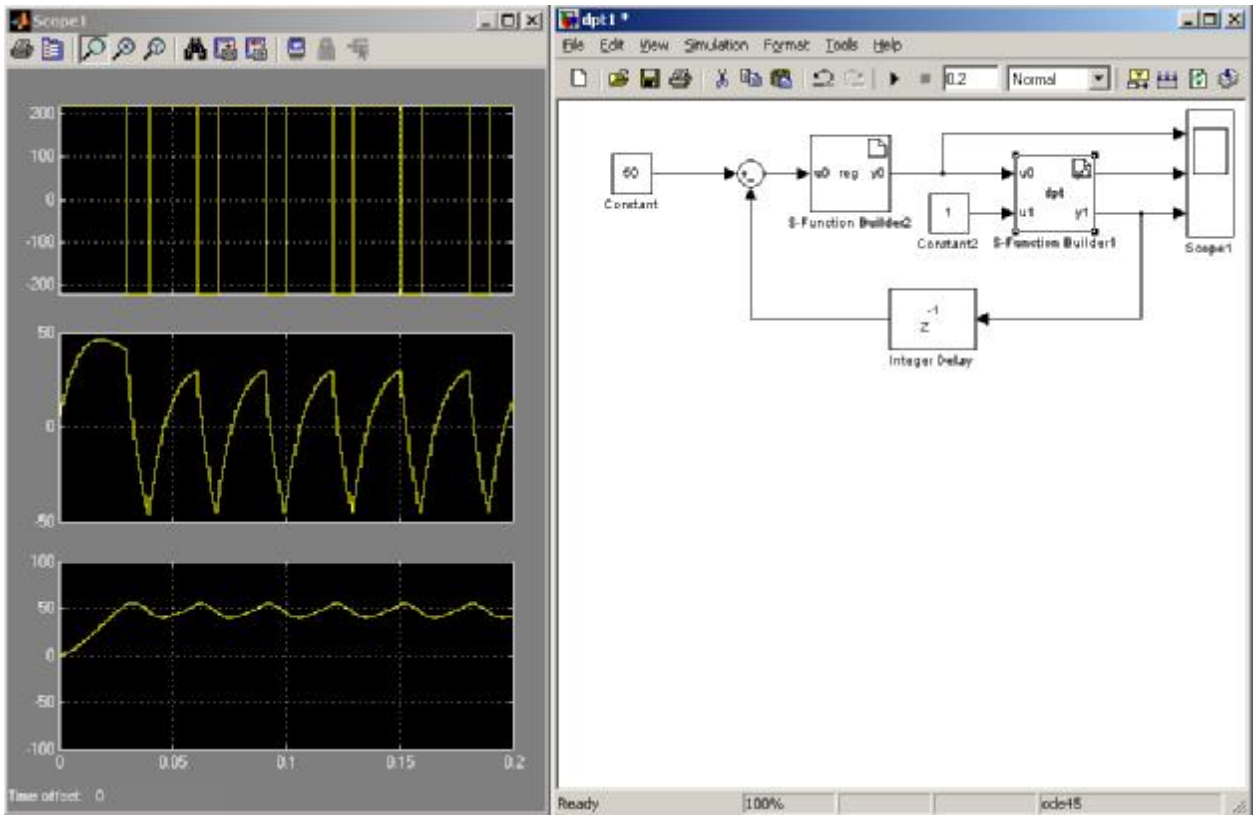

Рис.8

На практике очень распространен ПИ-регулятор.

Передаточная функция непрерывного ПИ-регулятора, реализуемого на аналоговой технике (например, на базе операционных усилителей), имеет вид:

$$W(p) = k_n + \frac{1}{T_u p},$$

где k_n – коэффициент передачи пропорциональной части, T_u – постоянная интегрирования. Если обозначить входной сигнал, который обычно является величиной рассогласования между заданным значением регулируемой величины и ее фактическим значением, снимаемым с датчика обратной связи, как $e(t) = x_з(t) - x_{oc}(t)$, а выходное управляющее воздействие, подаваемое на вход объекта управления, как $u(t)$, то пере-

даточной функции ПИ-регулятора будет соответствовать дифференциальное уравнение:

$$u(t) = k_n e(t) + \frac{1}{T_u}\int_0^t e(t)dt\,.$$

Продифференцируем исходное уравнение непрерывного ПИ-регулятора, чтобы избавиться от интеграла:

$$\frac{du(t)}{dt} = k_n \frac{de(t)}{dt} + \frac{1}{T_u} e(t)\,.$$

Для перехода от непрерывного уравнения к разностному заменим непрерывные переменные дискретными, дифференциалы этих переменных – разностями, а приращение времени dt – величиной интервала квантования по времени T. Получим:

$$\frac{u(k)-u(k-1)}{T} = k_n \frac{e(k)-e(k-1)}{T} + \frac{1}{T_u} e(k)\,.$$

После очевидных алгебраических преобразований разностное уравнение ПИ-регулятора примет вид:

$$u(k) = u(k-1) + \left(k_n + \frac{T}{T_u}\right)\cdot e(k) - k_n \cdot e(k-1)\,.$$

Обозначим коэффициенты:

$$k_0 = k_n + \frac{T}{T_u} \qquad \text{и} \qquad k_1 = k_n\,.$$

С учетом полученных коэффициентов уравнение будет выглядеть следующим образом:

$$u(k) = u(k-1) + k_0 \cdot e(k) - k_1 \cdot e(k-1)\,.$$

На рис.9 представлена структурная схема ПИ-регулятора с неявновыраженной интегральной составляющей и общим ограничением, где Звено $\boxed{Z^{-1}}$ — запаздывание на такт.

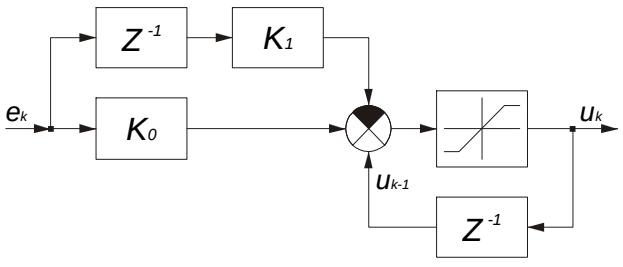

Рис.9

Реализация ПИ-регулятора в Симулинке выглядит следующим образом:

```
float static input=0;
float static output=0;
float static k0=2+0.001/0.01;
float static k1=2;
float static input_prev=0;

input=u0[0];
output=output+k0*input-k1*input_prev;
input_prev=input;

if (output>220) output=220;
if (output<-220) output=-220;
y0[0]=output;
```

Как видно из рис.10 и 11, при работе ПИ-регулятора ошибка регулирования для установившегося значения, в отличии от П-регулятора, равна практически нулевому значению при различных моментах сопротивления нагрузки.

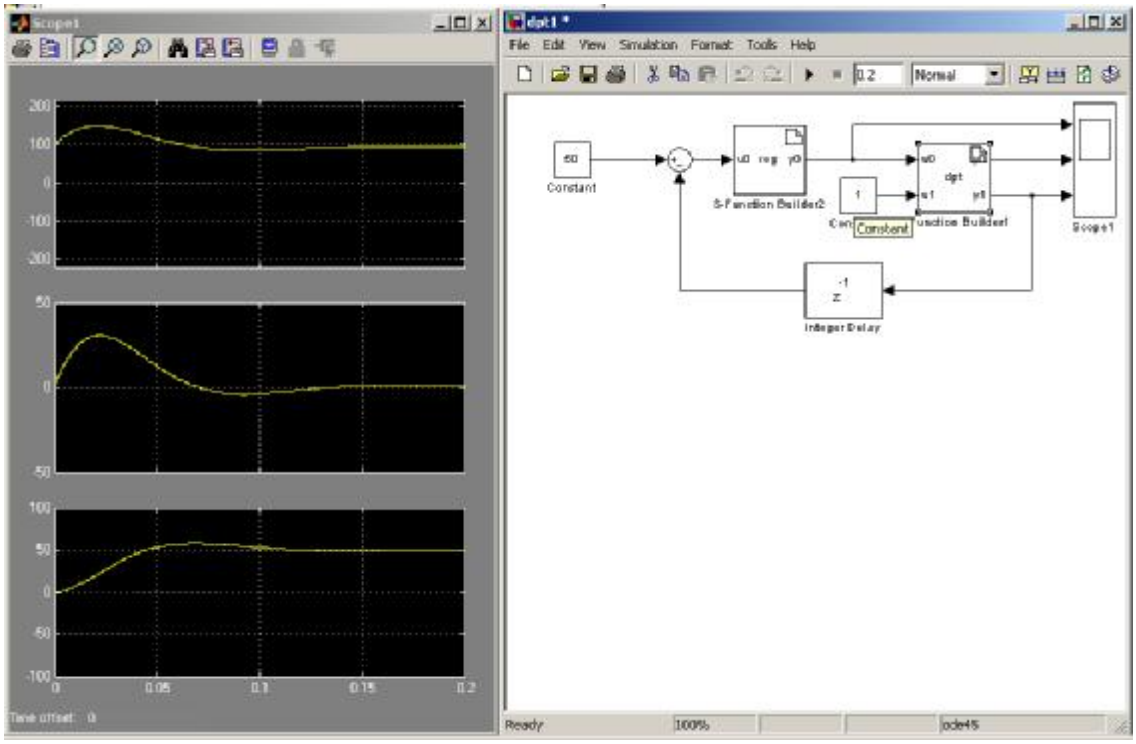

Рис.10

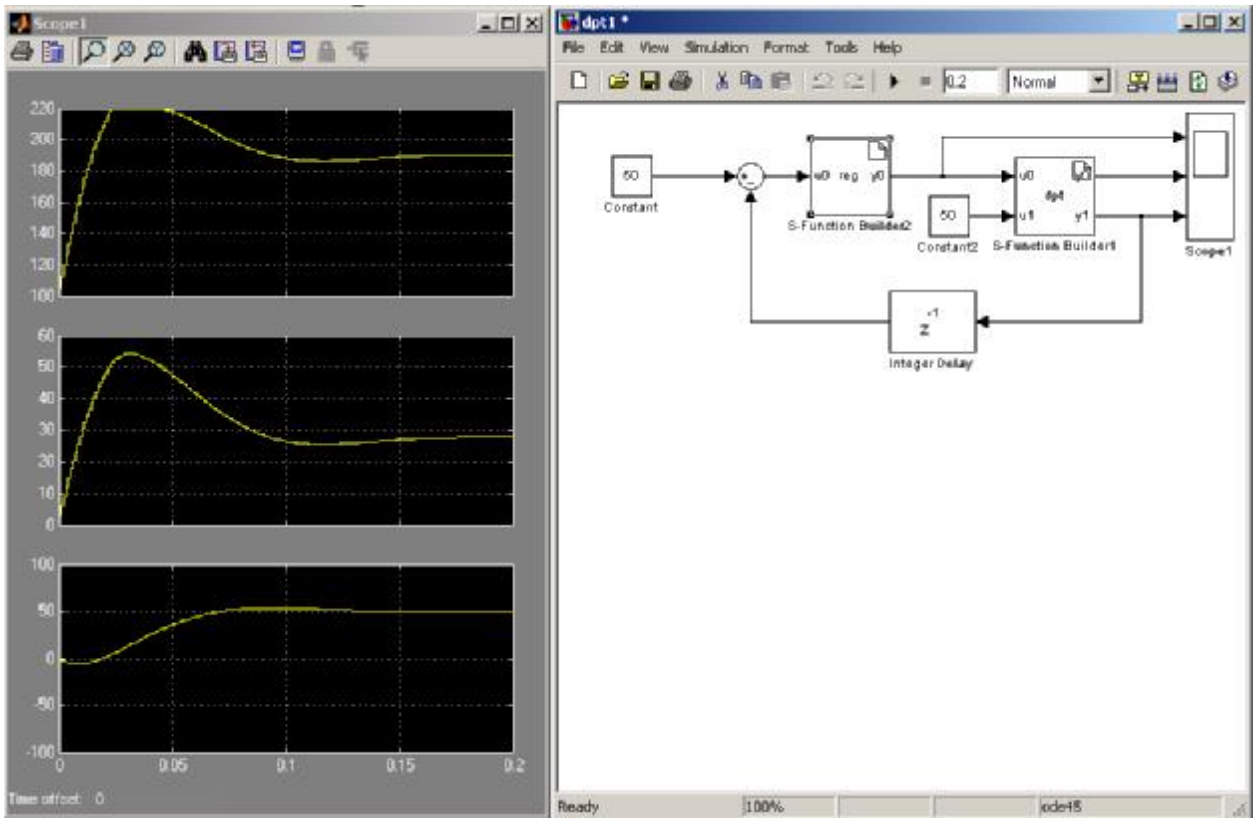

Рис.11

Задание для самостоятельной работы

Введите контур тока в рассматриваемую модель и реализуйте его на базе ПИД-регулятора.

Занятие 4. Среда программирования CodeComposerStudio. Использование симулятора

Среда программирования CodeComposerStudio (CCS) предназначена для создания программного обеспечения, запускаемого на DSP-процессорах фирмы Texas Instruments. CCS содержит все необходимые инструменты для набора и редактирования программ, конфигурирования ядра реального времени DSP|BIOS, получения машинного кода с использованием компилятора языка C, загрузки машинного кода в процессор, запуска и отладки программ, в том числе и в режиме реального времени. CCS позволяет работать как с реальным устройством, так и его

моделью в режиме симулятора. Симулятор позволяет заниматься отладкой программного обеспечения без наличия реального процессора, но при этом не поддерживает периферийные устройства на борту процессора и осуществляет вычисления со скоростью, отличной от скорости реального процессора. Таким образом, при работе с симулятором скорость выполнения программы будет в несколько раз меньше (поскольку компьютеру приходится моделировать работу всех системных устройств архитектуры ядра процессора), входные сигналы возможно только смоделировать, выходные сигналы не могут быть физически переданы во внешний мир и могут наблюдаться только по значениям переменных, невозможно смоделировать многозадачность. Однако для отработки логики работы расчетных процедур и некоторых других случаев симулятор оказывается доступным инструментом.

Изначально необходимо с помощью утилиты CodeComposerStudio Setup ввести в CCS применяемые устройства. На рисунке 1 (слева) показано, что CCS может работать с отладочной платой eZdsp и симулятором процессора F2812.

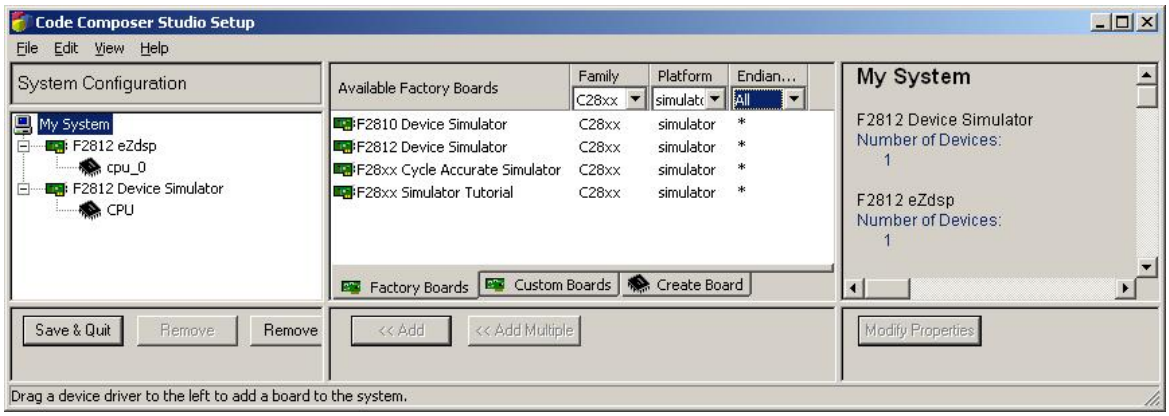

Рис.1

После запуска CCS в появившемся окне Parallel Debug Manager необходимо выбрать устройство, с которым будет происходить работа (в нашем случае это F2812 Device Simulator, см.рис.2).

Примечание. Для установки драйвера платы eZdsp необходимо воспользоваться диском, поставляемым с платой. Драйвер с диска устанавливается поверх установленной версии CCS.

После запуска основного окна CCS необходимо создать проект, создать текст программы, подключить его к проекту, оттранслировать,

загрузить его в память, запустить на исполнение, убедиться в правильности выполнения программы.

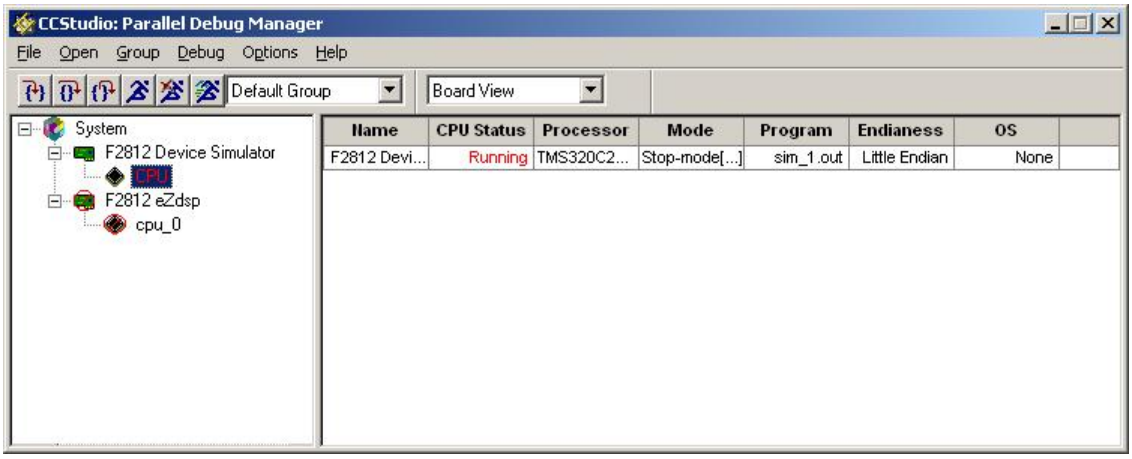

Рис.2

Ниже приведены инструкции по созданию простейшего приложения, позволяющего продемонстрировать работу простейшего цифрового фильтра.

1. Создать проект Project-New… В появившемся окне указать имя проекта, зафиксировать каталог проекта.
2. Создать текстовый файл File-New и ввести в него следующий код программы:

```
//для использования чисел с плавающей запятой
#include <float.h>
#include <math.h>
//объявление переменных программы
float t=0, //время
 u,          //входной сигнал
 filtr=0,    //отфильтрованный сигнал
 k0=0.1,   // коэффициент фильтра k0
 k1=0.9;   // коэффициент фильтра k1

void main()
{
 while(1)// организация безусловных циклов вычислений
   {        // начало цикла
    t=t+0.1; // имитация времени
```

```
    u=50*sin(t)+15*sin(100*t);// входной сигнал
    filtr=k0*u+k1*filtr;// отфильтрованный сигнал
   }   // конец цикла
}
```

3. Сохранить данный файл под именем sim_1, в качетсве типа файла выбрать из предложенного C/C++ Source file (*.c). Сохранение должно быть выполнено в каталог проекта.

4. Добавить только что сохраненный файл исходного текста программы в проект с помощью меню Project-Add files to project.

5. Добавить в проект аналогичным образом файл библиотеки rts2800.lib

6. Открыть окно настроек CCS через Option-Customize и на закладке **Program|Project Load** установить галочку для свойства Load Program After Build.

7. Выполнить трансляцию проекта через нажатие F7 или Project-Build. В случае успешного завершения трансляции и компоновки в появившемся окне сообщений будет показано:

```
---------------------------- sim_1.pjt - Debug ----------------------
Build Complete,
        0 Errors, 0 Warnings, 0 Remarks
```

После чего произойдет загрузка программы в память модели процессора.

8. Если программа содержит ошибки, их необходимо исправить и повторить трансляцию.

9. Для наблюдения за программой необходимо вывести окно WatchWindow, на нем открыть закладку Watch1, в клетки окна ввести переменные u и t.

10. Для наблюдения за формой изменения сигналов необходимо открыть графическое окно через View-Graph-TimeFrequency, в окне настроек сделать изменения согласно приведенному ниже окну (свойства Display Type – Dual Time, Start address – upper display - &u, Start address – lower display - &filtr, Acquisition Buffer Size – 1, DSP Data type – 32-bit floating points, см.рис.3).

11. Для отладки кода необходимо установить точку зондирования. Для этого необходимо щелкнуть левой клавишей мыши по строчке программы «t=t+0.1», затем правой клавишей и в появившемся контекстном меню выбрать Toggle Software Probe Point. Зелено-голубая точка появится слева от строчки, показывая

место установки точки зондирования (при выполнении данной строчки будет происходить обновление показаний привязанных к данной точке окон).

12. Для привязки данной точки зондирования к окнам отображения графиков и переменных (WatchWindow) необходимо вызвать окно привязки через Debug-ProbePoints, в появившемся окне щелкнуть мышью по слову NoConnection (для автоматической установки поля Location), после этого в поле Connect To выбрать Watch Window, нажать на окне кнопку Add. Аналогичным образом добавить вывод на графический дисплей. Нажать Ок.

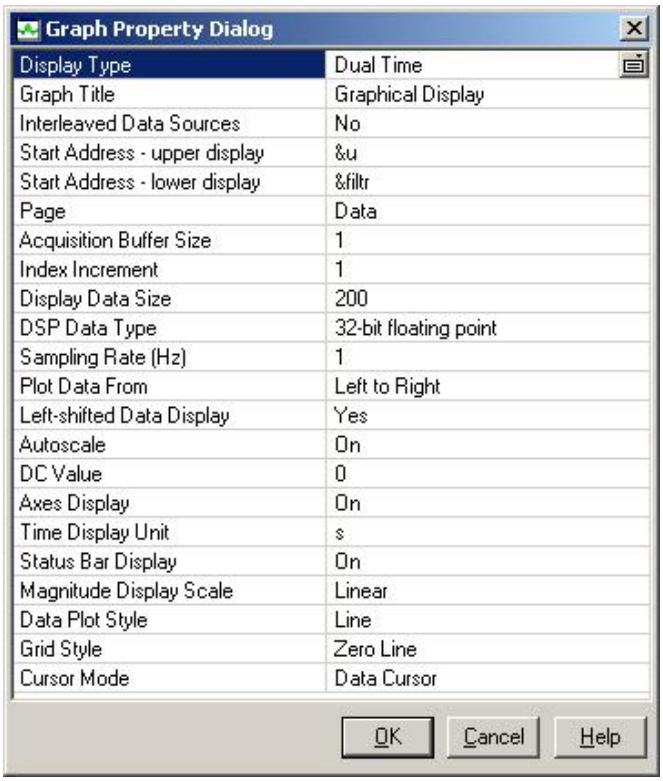

Рис.3

13. Для перезагрузки программы выполнить Project-Build.

14. Запустить программу клавишей F5 или через Debug-Run.

Результатом работы программы должно быть окно, показанное ниже(графический экран и WatchWindow показывают изменения переменных), см. рис.4:

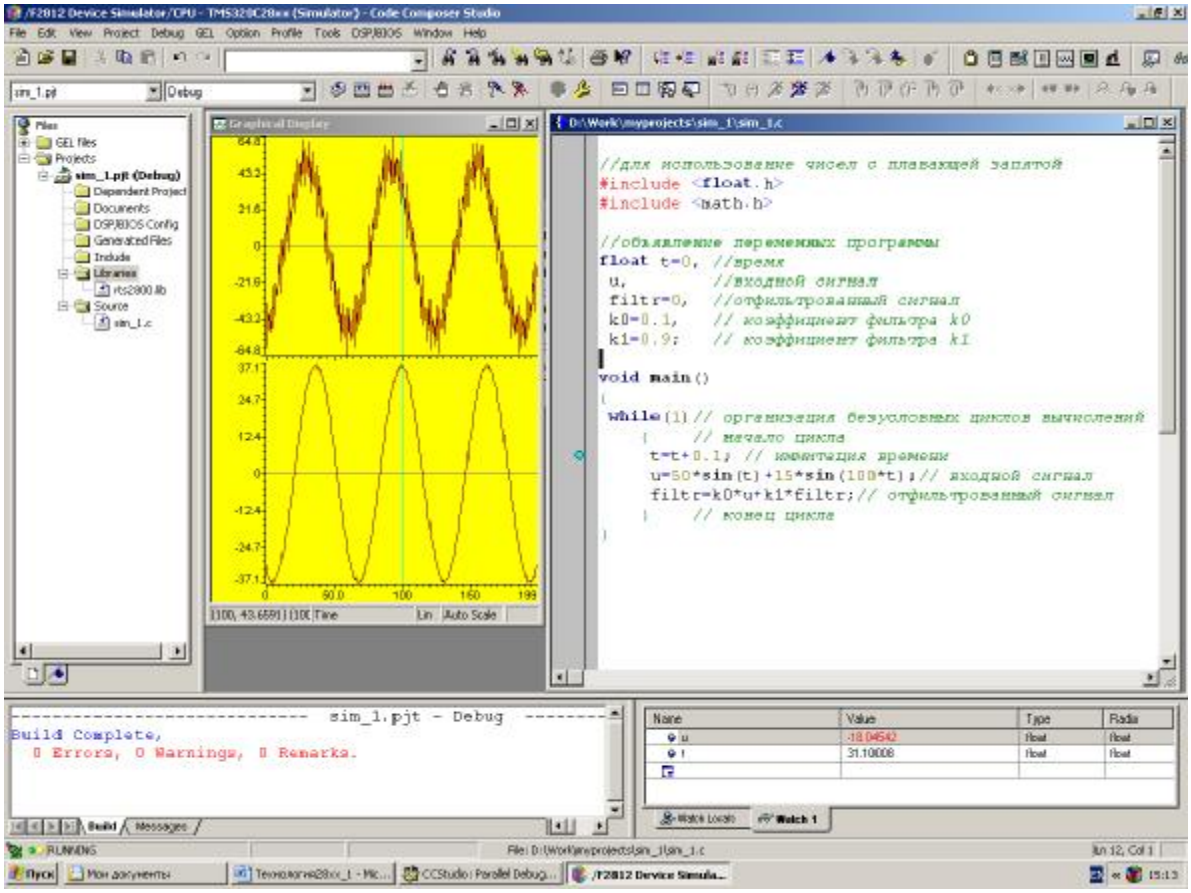

Рис.4

Далее показан более сложный пример использования симулятора. В примере показана система позицирования между двумя конечными положениями (лифт на 2 этажа), созданная на основе модели двигателя постоянного тока, ПИ-регулятора скорости, контура положения на базе П-регулятора и логической системы управления.

В примере создается зацикленный расчет работы системы. Происходит моделирование времени, работы двигателя постоянного тока, далее происходит расчет регуляторов скорости и положения, в завершение – система логического управления на основе смены состояний режимов работы в зависимости от координат системы и команд. Пример показывает возможность отладки программного обеспечения без наличия реальной системы управления, при этом затрагиваются основные моменты разработки процедур управления. Нижнее положение соответствует значению «0» в переменной pos, верхнее – «10». Срабатывание кнопок происходит при вводе в соответствующие переменные значения «1».

```
#include <float.h>// в файле используются расчеты
                          // с плавающей запятой

// декларирование используемых переменных
float t=0,dt=0.001;
float
w=0,i=0,u=0,R=3.6,L=0.034,Mc=1,Me=0,c=1.82,J=0.038,Mload;
float
k0=2+0.001/0.01,k1=2,input=0,input_prev=0,output=0,w_error=0;
float w_set=0;
float pos_set=0,pos=0,pos_error;
int down_button=0,stop_button=0,up_button=0,mode=0;
int up_lamp=0,down_lamp=0,down_move_lamp=0,up_move_lamp=0;


void main()// процедура расчета
{
m1:  // начало цикла
// модель времени
//-----------------------------------------------------------------
  t=t+dt;

// модель двигателя постоянного тока
//-----------------------------------------------------------------
   i=i+(u-R*i-c*w)/L*dt; // вычисление тока
   Me=i*c;// вычисление крутящего момента
   w=w+(Me-Mload)/J*dt;//вычисление скорости
   pos=pos+w*0.1;//вычисление положения

// модель реактивной нагрузки (лифт с противовесом)
//-----------------------------------------------------------------
   if (w<0) Mload=-Mc;else
   if (w>0) Mload=Mc;else Mload=0;

// регулятор скорости
//-----------------------------------------------------------------
   w_error=w_set-w;// расчет ошибки
   input=w_error;// передача сигнала ошибки на вход регулятора

   output=output+k0*input-k1*input_prev;// расчет регулятора
   input_prev=input;//запоминание текущего значения как
```

```
                // предыдущего
                // для следующего цикла расчета

  if (output>220) output=220;    // ограничение
                        // максимального значения
  if (output<-220) output=-220;

 u=output;    //соединение сигнала регулятора
                // и напряжение якоря двигателя

// регулятор положения
//-----------------------------------------------------------------

 pos_error=pos_set-pos;// расчет ошибки
w_set=pos_error*0.25;// расчет положения
                              // и передача задания на
                             // регулятор скорости

// управление сигнализацией положения
//-----------------------------------------------------------------
if (pos<1) down_lamp=1;else down_lamp=0;// лампа
                        //нижнего положения
  if (pos>9) up_lamp=1;else up_lamp=0;       // лампа
                              //верхнего положения

// режим работы «остановлен»
//-----------------------------------------------------------------
  if (mode==0)
   {
        pos_set=pos_set;
        u=0;//обесточиваем двигатель
        w_set=0;//останавливаем регулятор скорости
        down_move_lamp=0;     // выключили лампы
                              // движения вниз
        up_move_lamp=0;         // выключили лампы
                              //движения вверх
        if (up_button==1) mode=1;      //реакция на
                                    //нажатие
                                    //кнопки up
        if (down_button==1) mode=2; //реакция
                                    //на нажатие
```

```
                                    //кнопки down
        if (stop_button==1) mode=0;   // реакция
                                    // на нажатие
                                    // кнопки stop
  }

// режим подъема
//-----------------------------------------------------------------
  if (mode==1)
     {
                up_move_lamp=1;         // включили лампы
                                    //движения вверх
                down_move_lamp=0;     // выключили лампы
                                    //движения вниз
                pos_set=10;// задание целевого положения
                 if (up_button==1) mode=1; //реакция
                                    // на нажатие
                                    // кнопки up
                if (down_button==1) mode=0; //реакция
                                    //на нажатие
                                    // кнопки down
                if (stop_button==1) mode=0; // реакция
                                    // на нажатие
                                    // кнопки stop
                if (pos>pos_set) mode=0;// требуемое
                                //положение достигнуто
        }

// режим опускания
//-----------------------------------------------------------------
    if (mode==2)
        {
                up_move_lamp=0; // выключили
                                //лампы движения вверх
                down_move_lamp=1; // включили
                                //лампы движения вниз
                pos_set=0; // задание целевого положения
                if (up_button==1) mode=0; //реакция
                                // на нажатие кнопки up
                if (down_button==1) mode=2; //реакция
                                // на нажатие кнопки down
```

```
            if (stop_button==1) mode=0; // реакция
                                // на нажатие кнопки stop
            if (pos<pos_set) mode=0;//требуемое
                                //положение достигнуто
        }

    goto m1;//конец цикла
    }
```

При запуске программы и имитации включения кнопок возможно отслеживание отработки положения и срабатывания ламп сигнализации через окно WatchWindow. На графике, приведенном на рисунке 5, выводятся осциллограммы тока якоря и скорости.

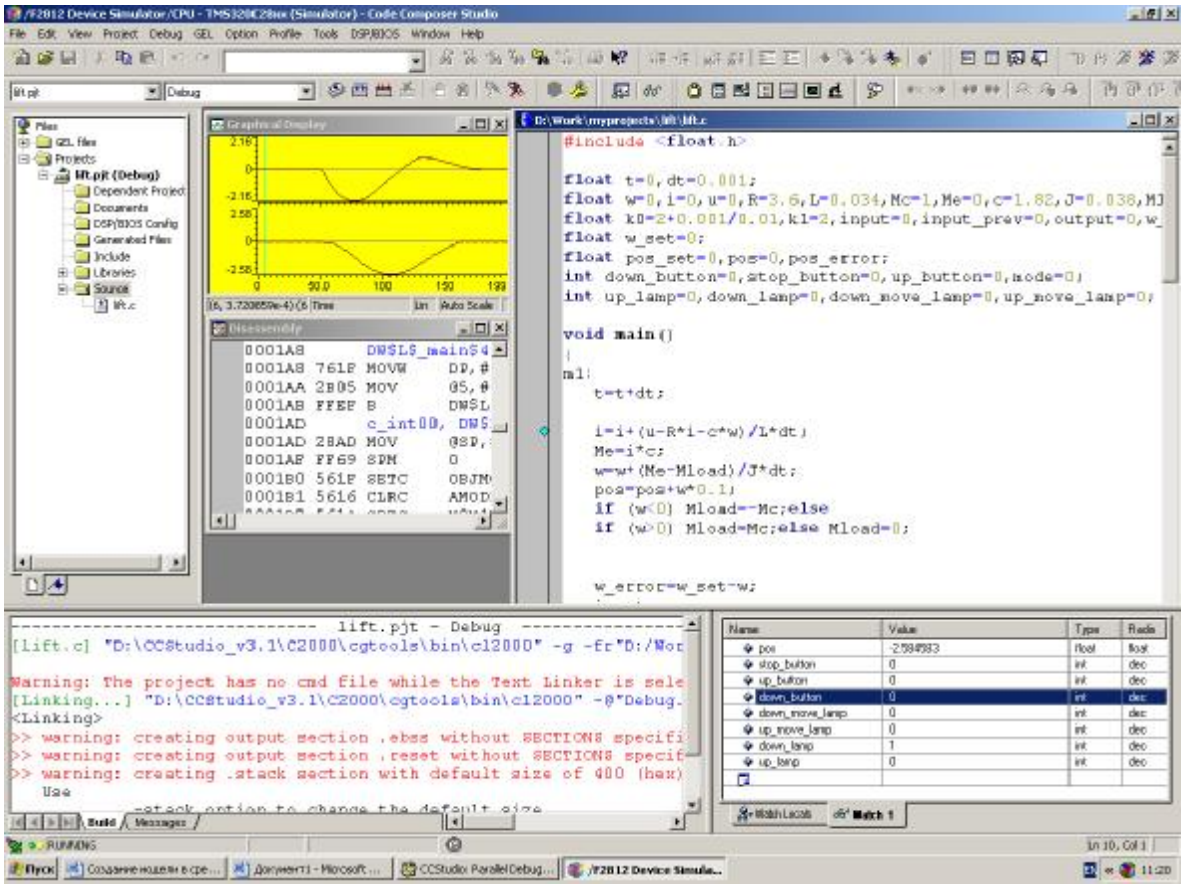

Рис.5

Внимание! В представленной системе содержатся ошибки различного характера (как минимум 3), которые не позволяют применить разработанную систему для управления реальным лифтом с гарантией

обеспечения безопасности доставки груза. В качестве задания для самостоятельной работы предлагается выявить эти ошибки и предложить способы по их устранению, а также установить в данную систему ПИ-регулятор тока

Занятие 5. Технология создания собственных функций для приложений

При программировании на языке С следует создавать новые функции по единой технологии. Ниже будет рассмотрен пример создания процедуры генерации пилообразного сигнала с заданным шагом до заданного значения. Данный пример строится на базе предыдущего проекта, в котором имеется сконфигурированная ранее операционная система.

Так как предполагается, что функции и данные будут объединены в структуру с одним именем, то такая структура будет носить название «объект».

Для начала определим данные, которые будут использоваться объектом. Определение необходимых ресурсов происходит посредством введения заголовочного файла (название имени кроме расширения совпадает с именем файла исходного текста используемых функций, предполагается что текст функции будет содержаться в отдельном файле).

Заголовочный файл состоит из 4-х частей:

1. объявление структуры как типа данных; в структуру входят все переменные функции, а также указатели на адреса процедур, применяющихся при исполнении функций, обычно это процедуры инициализации (init) и расчета (update или calc).
2. Определение типа данных для указателя на структуру п.1
3. Объявление прототипов процедур, применяемых при обслуживании и расчете функции.
4. Задание начальных констант в структуру.

Подобный подход позволяет запускать несколько копий объектов, каждый использует свой набор данных, равный структуре, при этом не происходит тиражирования кода в памяти программ, так как используется для всех копий объектов один код.

Сохраните следующий файл как заголовочный (с расширением *.h) в директорию «include»:

```
#ifndef __RAMP_H__
```

```
#define __RAMP_H__

typedef struct
{
  int step;
  int ramp_out;
  int max_out;
  int (*init)();
  int (*update)();
} RAMP;

typedef RAMP *RAMP_handle;

void RAMP_init(RAMP_handle);
void RAMP_update(RAMP_handle);

#define RAMP_DEFAULTS      {       1,\
                            0,\
                            100,\
                           (int (*)(int))RAMP_init,\
                           (int (*)(int))RAMP_update}
#endif
```

Примечание. При задании параметров по умолчанию следует четко соблюдать чередование символов «переброс каретки» и «\», в противном случае возможны ошибки компиляции. См.пример.

Для самих процедур функции необходимо создать отдельный файл, например следующего содержания:

```
#include "..\include\ramp.h"
/*  процедура инициализации */
void RAMP_init(RAMP *v)
{
 v->step=3;
}

/*  процедура расчета  */
void RAMP_update(RAMP *v)
{
  v->ramp_out=v->ramp_out+v->step;
```

```
  if (v->ramp_out>v->max_out) v->ramp_out=0;
}
```

В основном файле программы следует объявить экземпляры объекта.

```
RAMP ramp1=RAMP_DEFAULTS;
RAMP ramp2=RAMP_DEFAULTS;
```

В основной файл исходного текста проекта необходимо добавить вызовы функций объектов, например:

```
int a=0;

void main()
{
RAMP_init(&ramp1);
RAMP_init(&ramp2);

while (1)
 {
 a++;
 RAMP_update(&ramp1);
 RAMP_update(&ramp2);
 }
}
```

В результате будут генерироваться 2 пилообразных сигнала, шаг и максимальное значение которых можно задавать индивидуально.

В ходе разработки функций полезно пользоваться директивами компилятора, которые позволяют настроить ход компиляции. Рассмотрим на примере технологию применения директив.

Для того чтобы избежать предупреждения о переопределении константы, необходимо в начале каждого заголовочного файла вводить следующее:

```
#ifndef __LES1_H__
#define __LES1_H__
```

Для определения константы используем директиву *#define*. Допустим, необходимо, чтобы при компиляции происходило генерирование кода на сложение или вычитание. Для этого вводим

```
#define ADD 0
```

#define SUB 1

В дальнейшем ход компиляции будет определяться в зависимости от значений констант *ADD* или *SUB*.

```
#if ADD
        my1.c = my1.a + my1.b;
#endif

#if SUB
        my1.c = my1.a - my1.b;
#endif
```

Определять можно не только константы, но и макросы. Например, макрос возведения в квадрат будет выглядеть как

#define quad(x) ((x)(x))*

Вызов макроса возможен следующим образом:

```
float temp=0;
                temp = quad(3.5);
```

При компиляции все строки, определенные через #define, будут подменены на заданные в директиве значения.

Директива *#pragma* позволяет определить область памяти, куда будут размещены данные или код (например, в командном файле должна быть задана секция *sect*):

```
#pragma CODE_SECTION(www, "sect");
void www(void);
```

В данном случае при объявлении прототипа функции указывается ее размещение в памяти.

Полный пример заголовочного файла les1.h:

```
#ifndef __LES1_H__
#define __LES1_H__

#define ADD 0
#define SUB 1
#define Uint16 unsigned int
#define quad(x) ((x)*(x))


typedef struct
 {
        float a;
        float b;
```

```
        float c;
  } MY_STRUCT;


#define MY_STRUCT_DEFAULTS {2,\
                                                3,\
                                                0}

struct ABCD_bits{
 Uint16 a:4;
 Uint16 b:4;
 Uint16 c:4;
 Uint16 d:4;
};

typedef union ABCD_union {
  Uint16 all;
  struct ABCD_bits bit;
} ABCD_union;

#endif
```

Полный пример файла программы les1.c:

```
#include "les1.h"
#include <float.h>

MY_STRUCT my1=MY_STRUCT_DEFAULTS;

#pragma CODE_SECTION(www, "sect");
void www(void);

ABCD_union ax;

float temp=0;


void main()
{
while(1)
 {
```

```
 temp = quad(3.5);

 #if ADD
        my1.c = my1.a + my1.b;
 #endif

 #if SUB
        my1.c = my1.a - my1.b;
 #endif
 www();

 }
}

void www()
{
  ax.bit.c=1;
}
```

В данном примере также показан механизм использования объединений (union), который позволяет использовать одну и ту же область памяти для различных типов данных, в данном случае – это одновременно и беззнаковое целое, и группы битов по 4 бита в каждой. Такой механизм удобно использовать, например, для конфигурирования регистров либо побитно, либо целиком загрузкой целого известного значения.

Собственно разработанные процедуры полезно объединять в библиотеки, объединяющие процедуры пользователя по функциональному принципу (библиотека регуляторов, драйверов, сигналов и т.д.). Библиотека будет содержать объектный код, который будет подключен в основной проект при выполнении компоновки (выполняется при подаче команды project-build).

Для этого необходимо создать проект библиотеки Project-New, в появившемся окне указать имя библиотеки (автоматически будет создан каталог библиотеки, куда необходимо сохранять все ее исходные файлы), в поле типа проекта (Project type) выбрать Library (.lib). В окне настройки опции компиляции изменятся названия закладок (появится Archiver), см.рис.1.

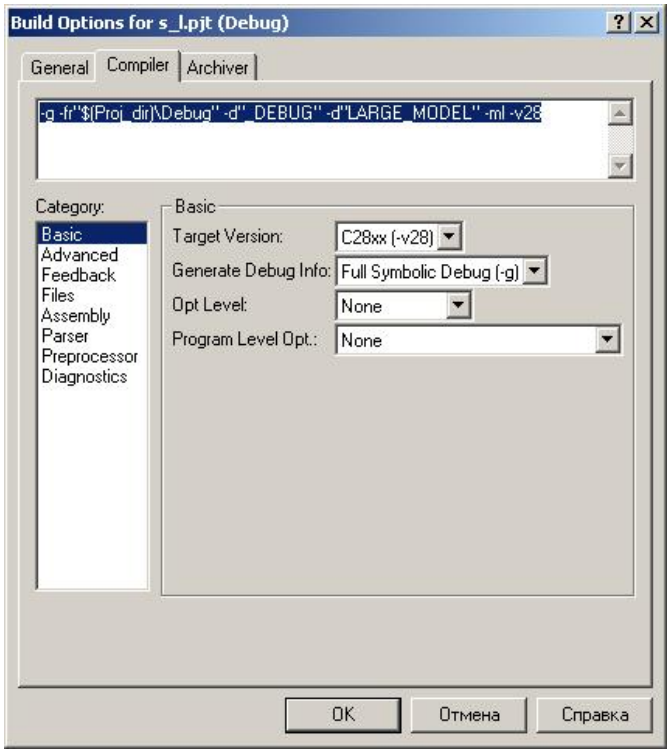

Рис.1

В проект необходимо добавить файл процедуры. При компиляции появится файл с расширением lib, имя которого будет одинаково с именем проекта.

Занятие 6. Использование пакета Матлаб для разработки программного обеспечения сигнального процессора

Ниже будет показано, каким образом возможно создавать программы для DSP TMS320F2812 используя средства Матлаб, в частности приложение Simulink и его блок-бокс Embedded target for TI C2000 DSP. В данном блок-боксе находятся средства конфигурирования системных регистров, регистров периферийных устройств, процедуры библиотеки псевдоплавающей запятой IQmath, процедуры библиотеки управления двигателем DMClib.

Необходимо иметь плату eZdsp2812 и установленные на компьютере MatlabR2006a, CCS3.1

Будет показано, каким образом можно создать программу в среде Simulink/Matlab, позволяющую плавно изменять яркость свечения светодиода платы eZdsp2812, с заданной периодичностью наращивая и снижая яркость свечения светодиода, то есть светодиод будет не загораться/гаснуть, а периодически плавно изменять яркость свечения во времени. Светодиод будет подключен к пину процессора IOPF15, работающего в режиме дискретного выхода. Соответственно будет иметься возможность только либо включить светодиод, либо выключить.

Яркость свечения будет изменяться визуально за счет изменения продолжительности включения светодиода за малый период времени. Последовательно изменяя скважность включения светодиода, за этот период времени удастся достигнуть визуального эффекта плавного изменения яркости.

Заданием на яркость свечения будет синусоидальный сигнал с частотой порядка 1 Гц. Параллельно будет запущен пилообразный сигнал высокой частоты (порядка 100 Гц). Сравнивая сигнал синусоиды и пилы, возможно определить время включения светодиода – если пилообразный сигнал больше значения синусоидального, то происходит включение светодиода, если меньше – то выключение. Соответственно светодиод будет «плавно» загораться с частотой 1 Гц.

Подразумевается, что вы уже знакомы с CCS и Симулником (в частности, механизмом создания)

Открываем Simulink, создаем в нем новое рабочее окно.

Находим компонент F2812 eZdsp (см.рисунок 1), перетаскиваем его в рабочее окно Simulink.

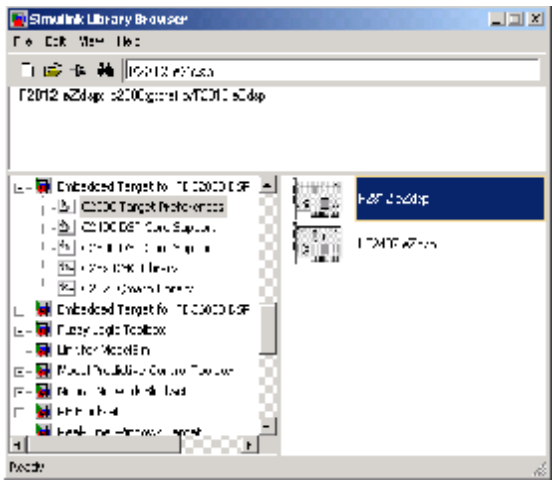

Рис.1

Раскрываем компонент, устанавливаем необходимые настройки согласно рисунку 2.

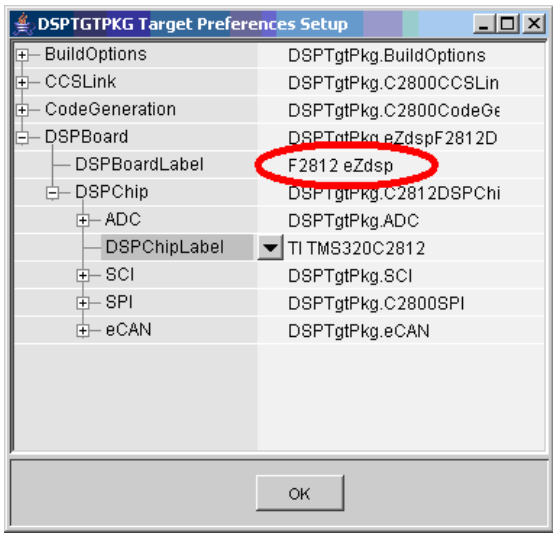

Рис.2

Собираем схему моделирования в Simulink согласно рисунку 3. В ней Subsystem – это подсистема для генерирования синусоидального сигнала на базе процедур с псевдофиксированной запятой из библиотек DMClib и IQmath_lib, Digital Output – готовый драйвер вывода сигнала на дискретную ножку, S-Function builder – блок генерирования пилообразного сигнала посредством S-функции, Relation Operator – стандартный компонент Simulink «если меньше, то…», Scope – стандартный компонент визуализации графиков Simulink.

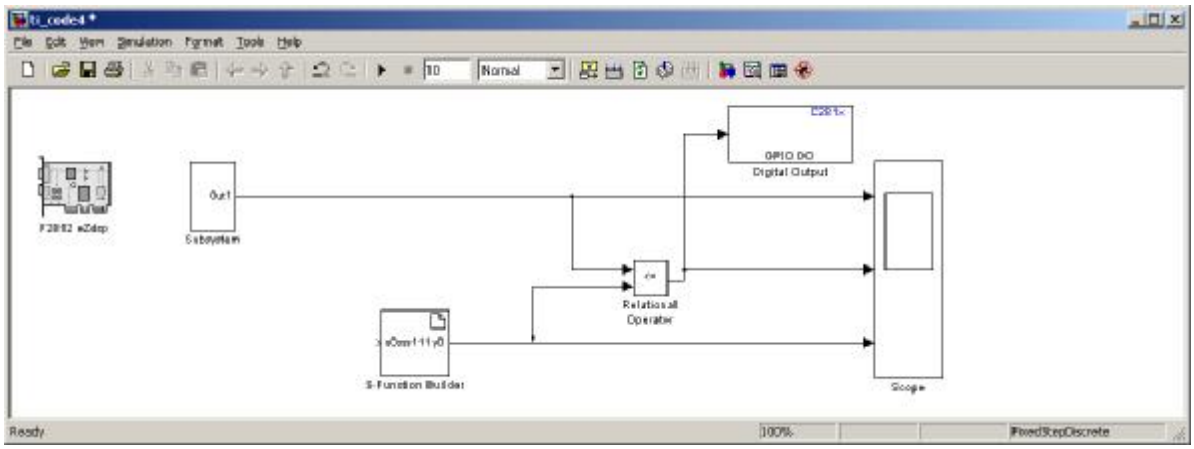

Рис.3

Драйвер дискретного выхода находится согласно рисунку 4:

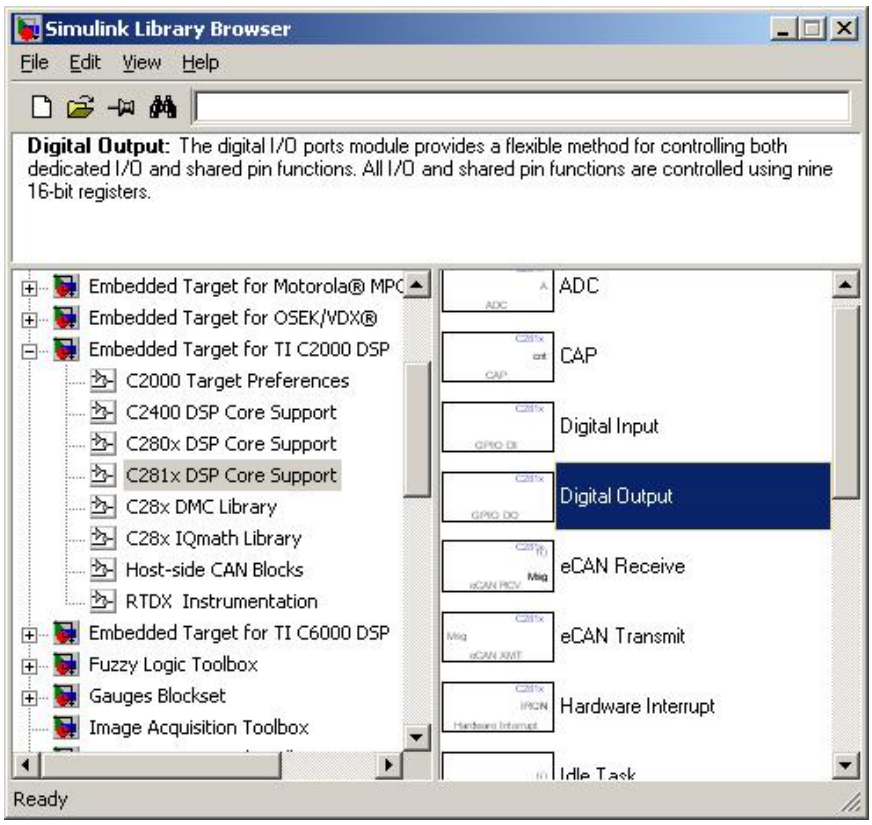

Рис.4

Выносим его в рабочее окно и двойным щелчком открываем его окно конфигурации, где необходимо выбрать IO port как GPIOF и установить галочку напротив bit14 и снять галочку напротив bit0 (см.рисунок 5). Последнее означает, что сигнал, пришедший на вход драйвера, будет выдан на ножку GPIOF14, сконфигурированную как дискретный выход.

Генератор пилы собираем через S-function builder, при этом необходимо выставить настройки вычисления кода S-функции как вычисление строго дискретное, с периодичностью 0,001с (см.рисунок 6)

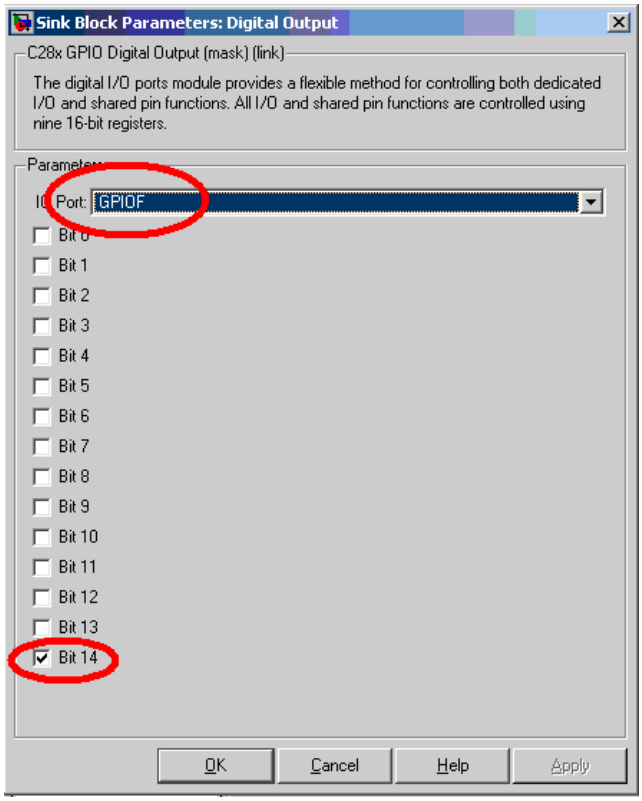
Sink Block Parameters: Digital Output
C28x GPIO Digital Output (mask) (link)
The digital I/O ports module provides a flexible method for controlling both dedicated I/O and shared pin functions. All I/O and shared pin functions are controlled using nine 16-bit registers.
Parameters
IO Port: GPIOF
Bit 0
Bit 1
Bit 2
Bit 3
Bit 4
Bit 5
Bit 6
Bit 7
Bit 8
Bit 9
Bit 10
Bit 11
Bit 12
Bit 13
Bit 14
OK
Cancel
Help
Apply

Рис.5

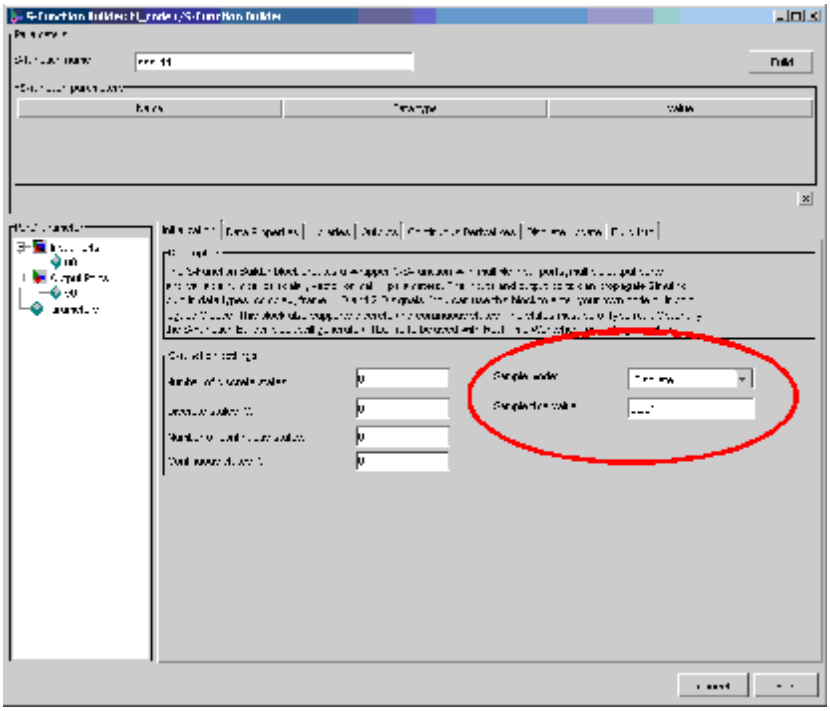

Рис.6

В закладке Output набираем текст программы генерации пилы:

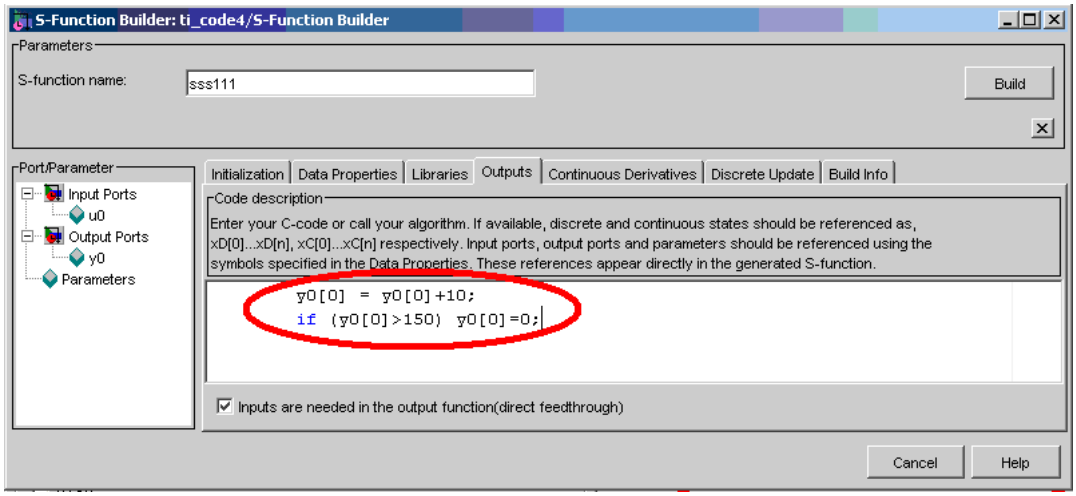

Рис.7

Для завершения построения S-функции необходимо установить ее имя (в данном случае показано имя sss111) и нажать клавишу Build (перед тем как нажать Build, необходимо в главном окне MatLab в Command Windows прописать mex -setup (обратить внимание на пробелы) и на предложенные вопросы ответить следующее: y, 1, y соответственно).

Переходим к построению блока генерации синусоидального сигнала. Так как вычисления Simulink выполняются в точном формате с плавающей запятой, а имеющийся DSP эффективно работает с целыми числами, собираем генератор синусоидального сигнала на базе процедур с псевдоплавающей запятой.

Примечание: при использовании блока constant необходимо в его свойствах (двойной клик мышью) на закладке Signal Data Types в поле output data type mode выбрать режим single.

Задания на такие процедуры происходят в формате с плавающей запятой, но перед тем как непосредственно передать значение в блок с фиксированной запятой, необходимо выполнить преобразование форматов, что осуществляется блоками Float to IQN.

Для генератора пилообразного сигнала, являющегося заданием изменения угла в диапазоне 0…2π для функции синуса от угла, используем процедуру RampGen из библиотеки DMC_lib (см.рисунок ниже)

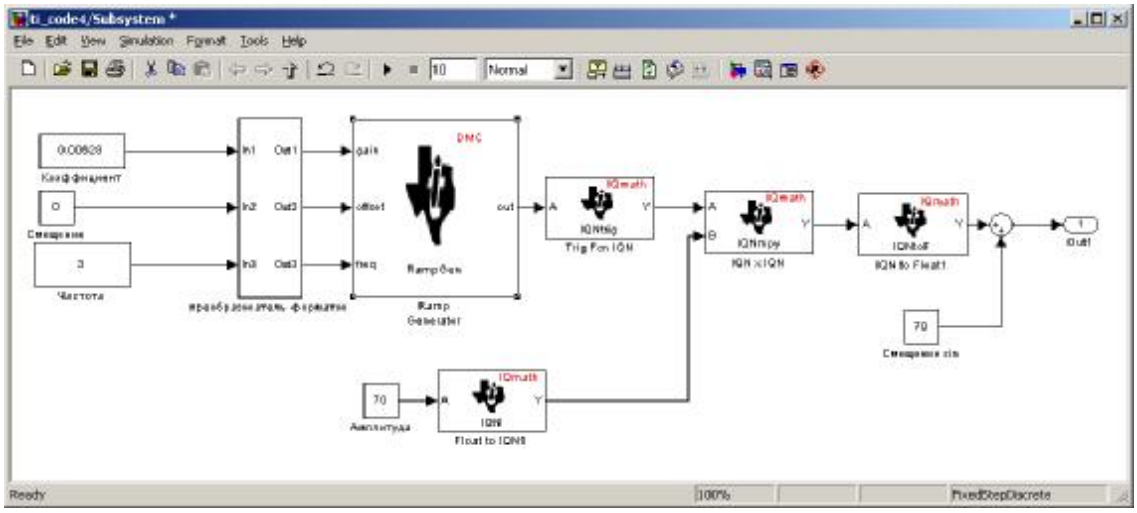

Рис.8

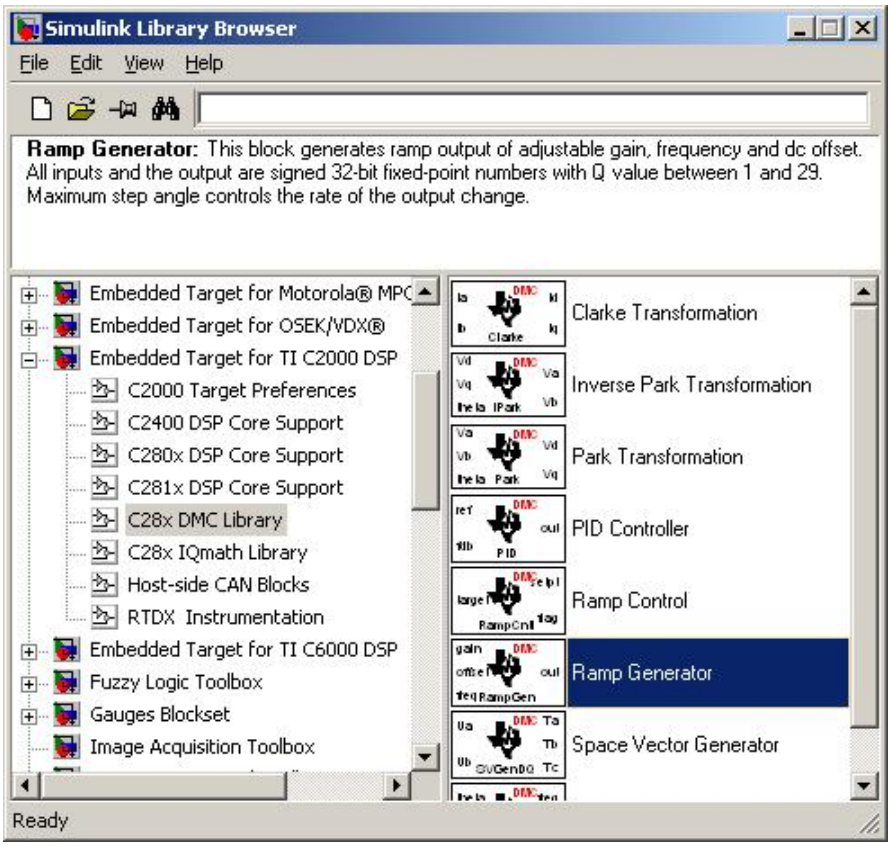

Рис.9

Для задания параметров (установить как на рисунке) модуля генератора пилообразного сигнала необходимо также выполнить преобразо-

вание форматов, здесь это выполнено в виде подсистемы «преобразователь форматов» (см.рисунок 10)

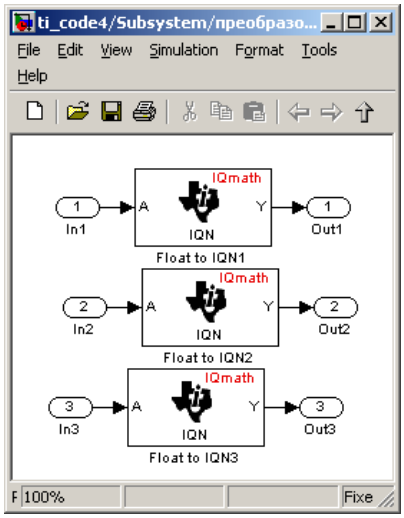

Рис.10

Для генерации синусоидального сигнала в функции от изменяющегося во времени угла используем стандартную процедуру генерации тригонометрического сигнала Trig Fcn IQN (см. рисунок 11).

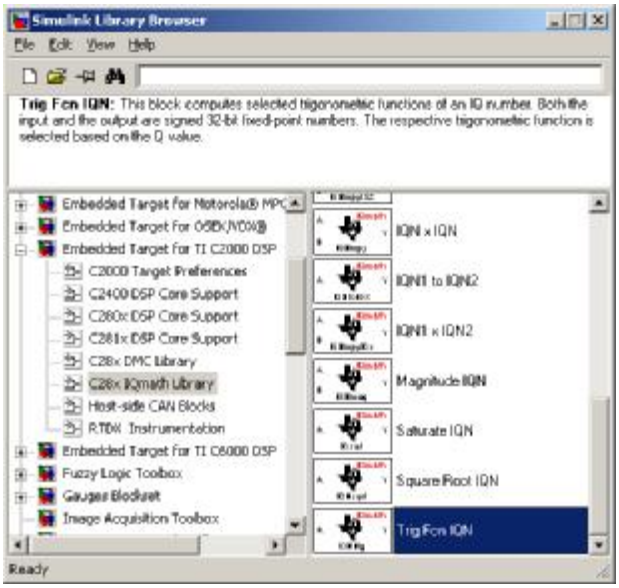

Рис.11

Для генерации проекта в CCS из Simulink необходимо воспользоваться приложением Matlab с названием Real-Time Workshop. Для этого заходим в настройки меню Simulation, Configuration Parameters рабочего окна в набранной схеме Simulink, выставляем следующие параметры: в поле System target file прописать ti_c2000_grt.tcl (см. рисунок 12):

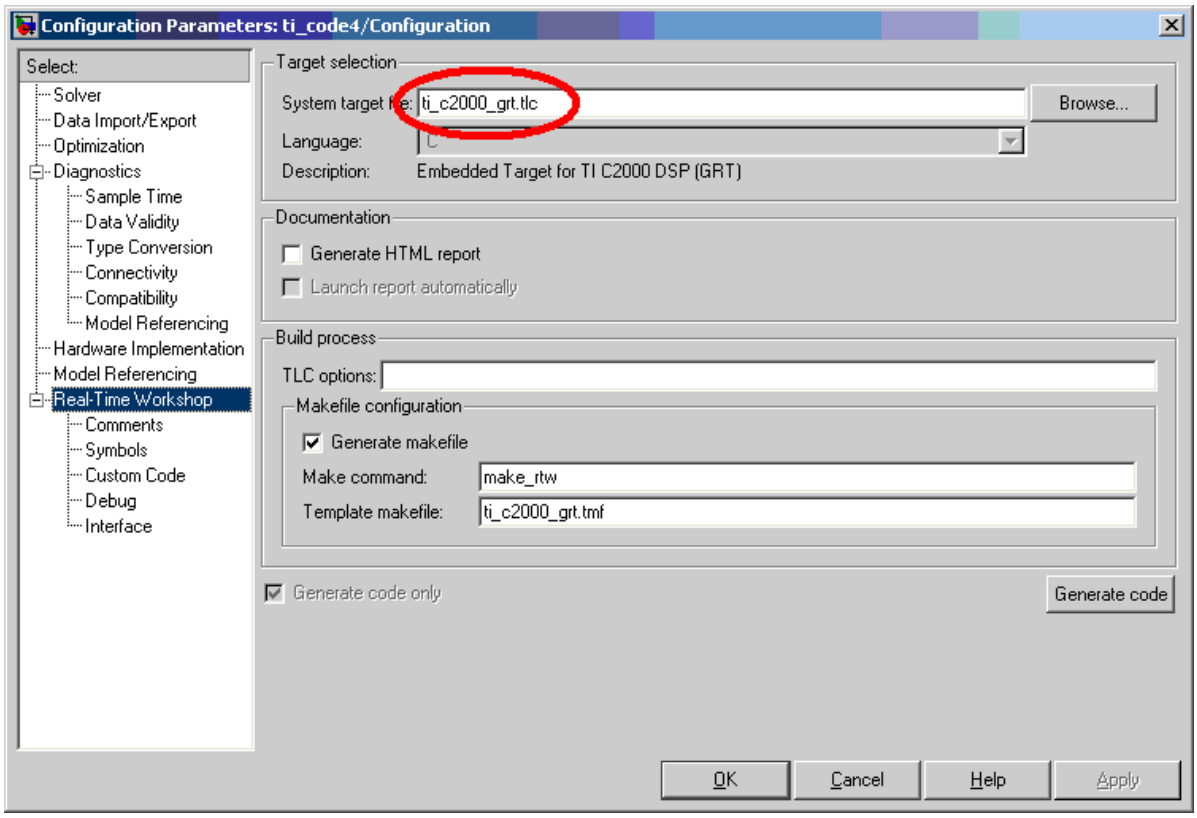

Рис.12

Перед тем как запустить генерацию проекта для CCS, существует возможность промоделировать работу будущего кода DSP. Для этого выставляем время моделирования в окне конфигурации как 10 секунд. Также необходимо выставить метод решения как дискретный (поле Solver), величину шага расчета задать как 0,001 секунда. Задание шага расчета также определяет шаг расчета в DSP (период дискретизации расчета). В данном примере с шагом расчета должен совпасть и шаг расчета S-функции.

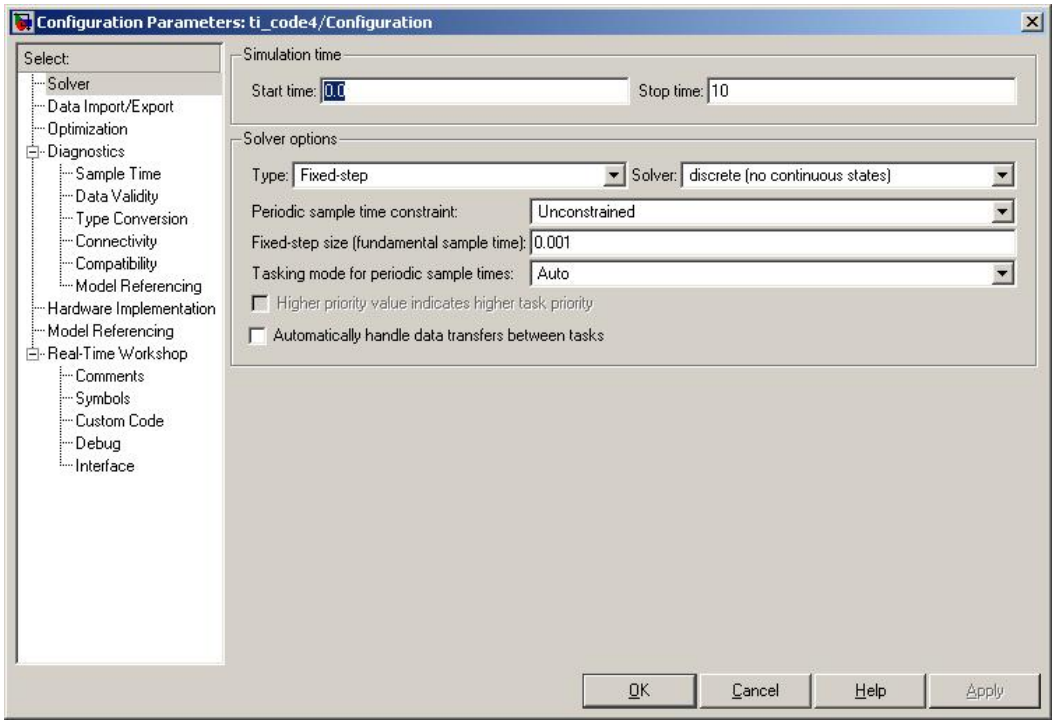

Рис.13

После запуска моделирования в окне SCOPE появится график, при увеличении которого можно понять принцип изменения скважности (см.рисунок 14, верхний график – синусоидальное задание на яркость, нижний – генератор пилообразного сигнала для сравнения, средний – результирующее смоделированное мигание светодиода).

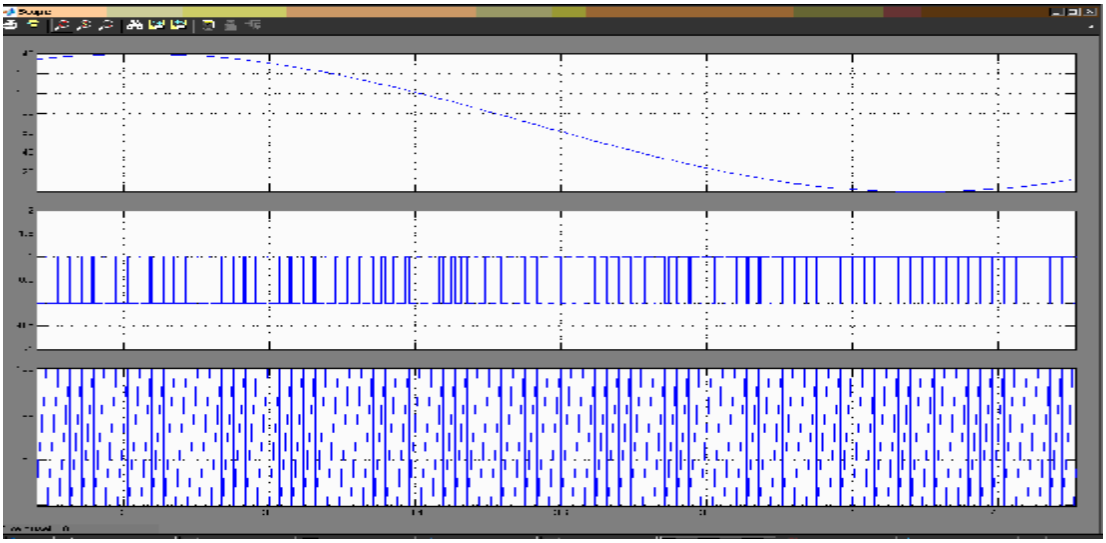

Рис.14

Для генерации кода проекта CCS необходимо вызвать окно конфигурации Simulation, Configuration Parameters и, убедившись в настройках согласно рисунку 15, нажать кнопку Generate code.

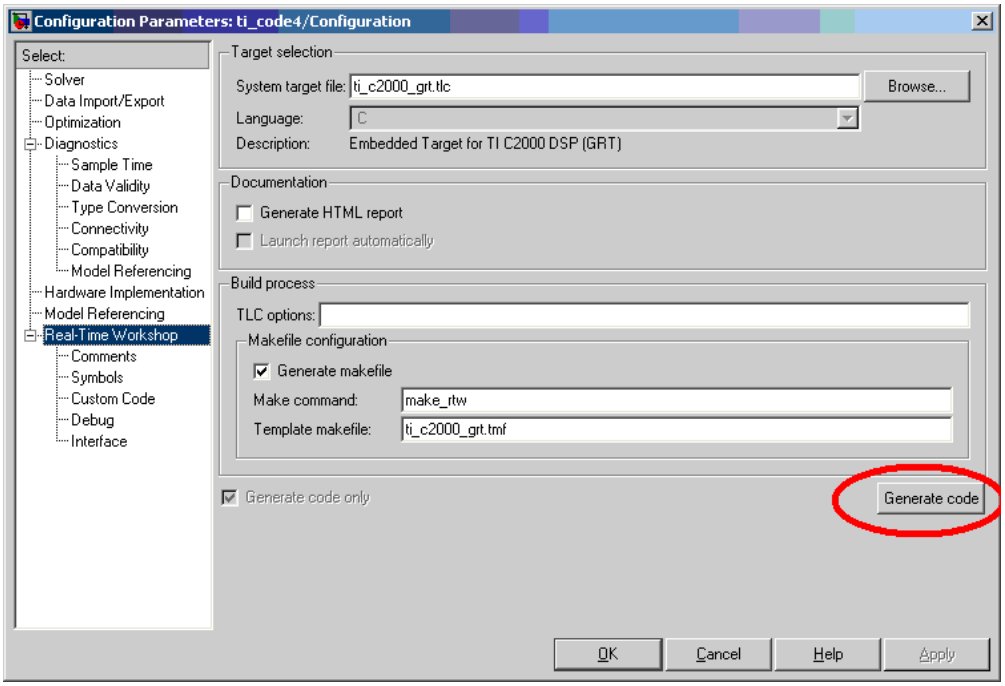

Рис.15

При нажатии этой кнопки запускается CCS, в нем создается проект с тем названием, которым была названа модель при создании в приложении Simulink из пакета MatLab, происходит компиляция и загрузка файла в процессор (плата должна быть предварительно подключена и проверена работоспособность). Запустив программу на выполнение в DSP, светодиод начинает визуально постепенно набирать/уменьшать яркость с периодичностью 1 секунда.

В результате было показано, что при программировании DSP можно использовать Simulink для следующих целей:

1. Автоматическое создание проекта с включением всех необходимых файлов
2. Конфигурация системных регистров
3. Конфигурации периферии (была показано конфигурация прерывания таймера и дискретный вывод)

4. Использование стандартных блоков (показан блок больше-равно) для программирования DSP

5. Создание собственных процедур для DSP на основе механизма S-функций

6. Использование псевдоплавающей запятой.

7. Использование процедур специальной библиотеки для управления электродвигателями.

Создание полноценных приложений посредством такого подхода невозможно, однако при быстром создании прототипов программ и проверки работы алгоритмов на основе использования готовых блоков Симулинка, данный подход является весьма полезным.

Примечание: проект для CCS автоматически создается в рабочей директории work пакета MatLab при генерировании кода. Перед повторным генерированием кода одного и того же проекта для исключения появления ошибок старый проект для CCS необходимо удалить вручную из рабочей директории MatLab work.

При генерации кода в CCS автоматически создается основной файл S-функции следующего содержания (название его зависит от названия S-функции в MatLab, в данном случае это sss111_wrapper.c), который находится в папке проекта Source рабочей директории work:

```
#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif
#include <math.h>
#define u_width 1
#define y_width 1
void sss111_Outputs_wrapper(const real_T *u0,
                    real_T *y0)
{
      y0[0] = y0[0]+10;
      if (y0[0]>150) y0[0]=0;
}
```

В теле программы (последние две строчки) прописан тот же текст программы генерации пилы, что был прописан нами в S-function builder при создании модели проекта в MatLab.

При генерации кода в CCS автоматически создается код блока «если не то» (Relational Operator) в основном файле eZdsp.c (названием файла является название модели проекта в MatLab), который находится в папке проекта Source рабочей директории work, состоящий из следующих строчек:

```
// вход на сравнение пилы с синусоидой
  /* RelationalOperator: '<Root>/Relational Operator' */
  eZdsp_B.RelationalOperator = ((real_T)rtb_Sum <= eZdsp_B.SFunctionBuilder);
// выход сигнала сравнения
/* S-Function Block: <Root>/Digital Output (c28xgpio_do) */
  {
    GpioDataRegs.GPFDAT.bit.GPIOF14 = eZdsp_B.RelationalOperator;
  }
```

Опять же при генерации кода в CCS автоматически создается код блока Subsystem (блока генерации синусоидального напряжения на базе процедур с псевдофиксированной запятой из библиотек DMClib и IQmath_lib) в основном файле eZdsp.c, который находится в папке проекта Source рабочей директории work, состоящий из следующих строчек:

```
/* Model output function */
static void eZdsp_output(int_T tid)
{
  /* local block i/o variables*/
  real32_T rtb_IQNtoFloat;
  real32_T rtb_Sum;
  int32_T rtb_FloattoIQN;
  int32_T rtb_IQNxIQN;
  int32_T rtb_TrigFcnIQN;
  int32_T rtb_FloattoIQN_d;

// блок преобразования форматов
  /* C28x IQmath Library (stiiqmath_iq) - '<S3>/Float to IQN' */
  {
    rtb_FloattoIQN = _IQ10 (eZdsp_P.k_Value);
  }
```

```
/* C28x IQmath Library (stiiqmath_iq) - '<S3>/Float to IQN1' */
{
  rtb_IQNxIQN = _IQ10 (eZdsp_P.smech_Value);
}

/* C28x IQmath Library (stiiqmath_iq) - '<S3>/Float to IQN2' */
{
  rtb_TrigFcnIQN = _IQ10 (eZdsp_P.frec_Value);
}

// генератор пилообразного сигнала
/* C28x DMC Library (tidmcrampgen) - '<S2>/Ramp Generator' */
{
  int32_T* angleregPtr = &eZdsp_DWork.RampGenerator_ANGLE_REG;

  *angleregPtr += _IQ10mpy (rtb_TrigFcnIQN, _IQ10(0.5));

  if (*angleregPtr > _IQ10(1))
  *angleregPtr -= _IQ10(1);
  else if (*angleregPtr < _IQ10(-1))
  *angleregPtr += _IQ10(1);

  rtb_FloattoIQN_d = _IQ10mpy (*angleregPtr++, rtb_FloattoIQN) + rtb_IQNxIQN;

  if (rtb_FloattoIQN_d > _IQ10(1))
  rtb_FloattoIQN_d -= _IQ10(1);
  else if (rtb_FloattoIQN_d < _IQ10(-1))
  rtb_FloattoIQN_d += _IQ10(1);
}

// процедура генерации тригонометрического сигнала
/* C28x IQmath Library (stiiqmath_iqtrig) - '<S2>/Trig Fcn IQN' */
{
  rtb_TrigFcnIQN = _IQ10sin(rtb_FloattoIQN_d);
}
// процедура преобразования форматов
/* C28x IQmath Library (stiiqmath_iq) - '<S2>/Float to IQN' */
{
```

```
  rtb_FloattoIQN_d = _IQ10 (eZdsp_P.amplit_Value);
 }

 //процедура перемножения сигнала амплитуды и тригонометрического сигнала
 /* C28x IQmath Library (stiiqmath_iqmpy) - '<S2>/IQN x IQN' */
 {
  rtb_IQNxIQN = _IQ10mpy (rtb_TrigFcnIQN, rtb_FloattoIQN_d);
 }

 // процедура обратного преобразования форматов
/*C28x IQmath Library (stiiqmath_iqtof) - '<S2>/IQN to Float' */
 {
  rtb_IQNtoFloat = _IQ10toF (rtb_IQNxIQN);
 }

 // блок суммирования сигналов
 /* Sum: '<S2>/Sum' incorporates:
  *  Constant: '<S2>/smech sin'
  */
 rtb_Sum = rtb_IQNtoFloat + eZdsp_P.smechsin_Value;
}
```

В данной работе приведено использование процедур с псевдофиксированной запятой из стандартных библиотек DMClib и IQmath_lib MatLab (подключение их показано на 5 странице данной методики).

Занятие 7. Разработка микропроцессорной системы управления двигателем постоянного тока

В данной работе будет продемонстрирован процесс разработки управляющего алгоритма микропроцессорной системы управления двигателем постоянного тока с независимым возбуждением. Система будет обеспечивать отработку заданной скорости вращения при различных моментах нагрузки. В качестве силовой схемы рассматривается однофазный управляемый выпрямитель.

Для разработки алгоритма управления сначала разработаем модель электропривода. Для этого в системе Симулинк открываем новое рабочее окно и вносим в него готовую модель двигателя постоянного тока DC Machine, которая находится в Simulink Library Browser, группа SimPowerSystems, подгруппа Machines. В целях проверки работоспособности модели двигателя подадим на якорь и обмотку возбуждения постоянное напряжение, для чего используем элементы DC Voltage Source, находящиеся в группе SimPowerSystems, подгруппа Electrical Sources. Для задания момента сопротивления на валу двигателя используем блок задания констранты Constant (находится в группе Simulink, подгруппа Commonly Used Blocks), для вывода координат двигателя используем осциллограф Scope, находящийся в группе Simulink, подгруппа Sinks. Полученная схема показана на рисунке 1. Если необходимо повернуть изображение блока, то нужно щелкнуть по нему правой кнопкой мыши и в выпавшем меню выбрать Format – Rotate block или Flip Block.

Устанавливаем время моделирования 0,5 секунд, запускаем моделирование и наблюдаем переходные характеристики по крутящему моменту, току и скорости двигателя. Если схема собрана правильно, то характеристики совпадут с показанными на рисунке 1

Задание для самостоятельной работы

1. Убедитесь, что при увеличении нагрузки установившаяся скорость становится меньше, а установившееся значение тока больше относительно первоначального варианта модели.
2. Убедитесь, что при увеличении напряжения на якоре двигателя установившаяся скорость возрастает относительно первоначального варианта модели.

Соберем теперь модель силовой схемы управления и подключим ее к двигателю и сети переменного тока. Силовая схема показан на рисунке 2 и состоит из двух диодов (неуправляемых полупроводников, находятся внизу) и двух тиристоров (полууправляемых полупроводников, находятся вверху). Данная схема представляет собой управляемый вы-

прямитель. Диоды и тиристоры находятся в подгруппе Power Electronics, где необходимы выбрать Thyristor и Diode.

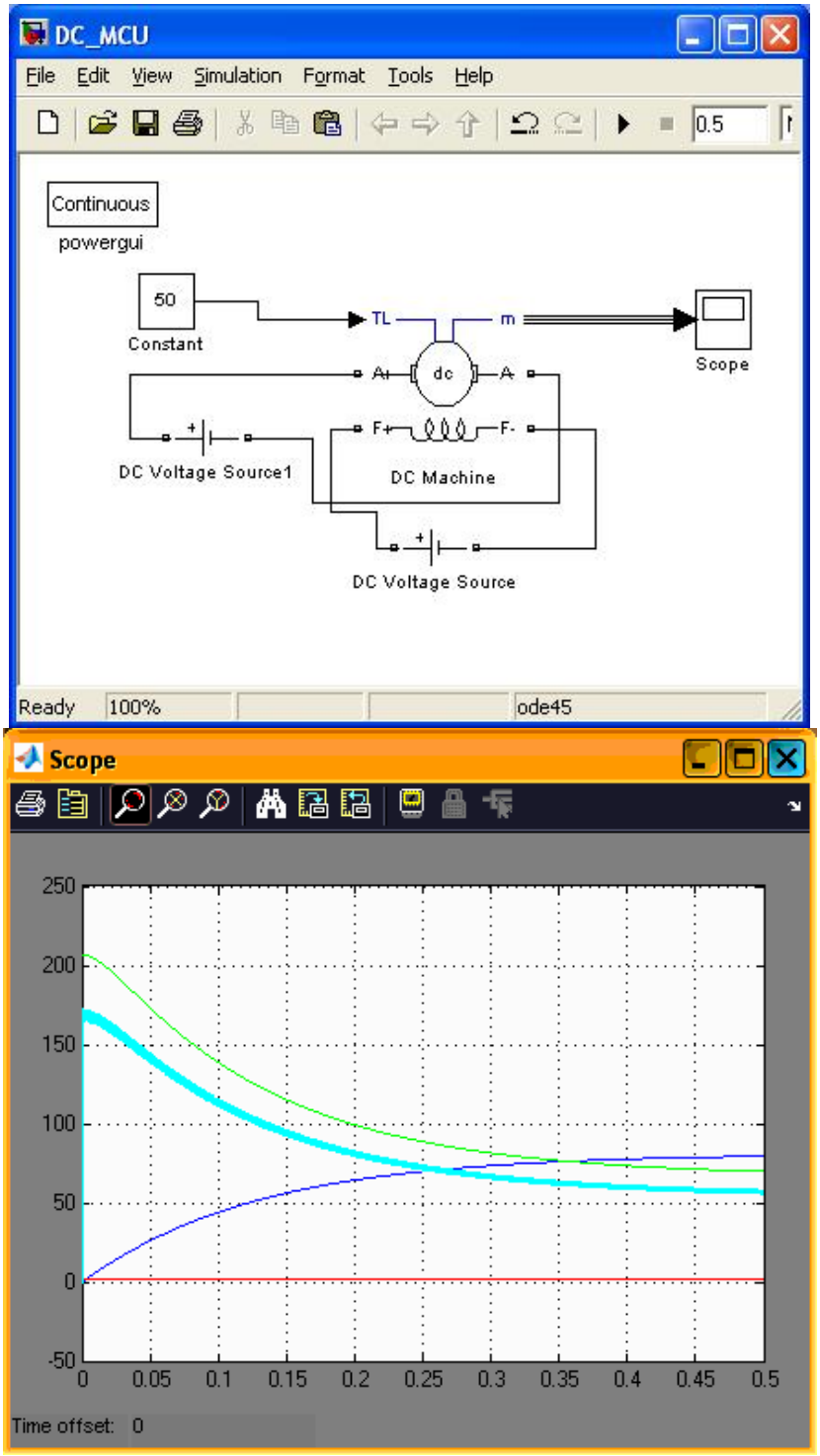

Рисунок 1

Подключаем средние точки моста к источнику переменного напряжения AC Voltage Source (находятся в группе SimPowerSystems, подгруппа Electrcai Sources), а выход поста подсоединяем к модели двигателя (см.рисунок 2). В компоненте AC Voltage Source необходимо установить частоту в значение «50» Гц. Для подачи сигнала открытия на тиристоры в целях проверки собранной модели используем блок константы со значением «1». Запускаем моделирование, результат должен совпасть с графиком на рисунке 2, на котором явно прослеживается влияние пульсаций выпрямленного напряжения в сигнале тока и момента.

Устанавливаем в систему модель датчика напряжения (см.рисунок 3), которая должна передавать сигнал напряжения в систему управления, причем сигнал напряжения содержит помехи. Для этого ставим блок Voltage Measurement (группа SimPowerSystems, подгруппа Measurements), соединяем его входы с клеммами модели источника напряжения. Выход измерителя вводим в сумматор (находится в группе Simulink, подгруппа CommonlyUsedBlocks), в котором происходит сложение с синусоидальным сигналом, имитирующем помехи в системе. Источник синусоидального сигнала находится в группе Simulink, подгруппа Sources, в блоке установить частоту синуса как 5000 рад/с, амплитуду – 15. Добавляем в имеющийся осциллограф еще один вход (через настройки осциллографа, в окне самого осциллографа найти пиктограмму Parameters, и установить в появившемся окне значение «2» в поле Number of Axes). Заводим на второй вход осциллографа сигнал с модели датчика напряжения, делаем это через мультиплексор, второй вход мультиплексора пока остается неподключенным. Если все сделано правильно, то результат будет аналогичным с рисунком 3: сигнал напряжения зашумлен высокочастотной гармоникой. Теперь модель готова для встраивания в нее системы управления.

Алгоритм системы управления будет реализован на базе S-функции, для чего необходимо в рабочее окно модели установить компонент S-function builder. Открываем окно S-Функции двойным нажатием на компонент, и устанавливаем имя S-функции (например, DC_motor_control). Данный компонент будет моделировать работу управляющего микропроцессора, который на основе поступающих данных из модели будет формировать импульсы на управляющие электроды тиристора. Как правило, микропроцессорные системы управления электроприводами работают с жестким периодом дискретизации, для чего устанавливаем дискретный режим работы компонента (Sample Mode - Discrete), а время дискретизации Sample time value – как 0.000055555.

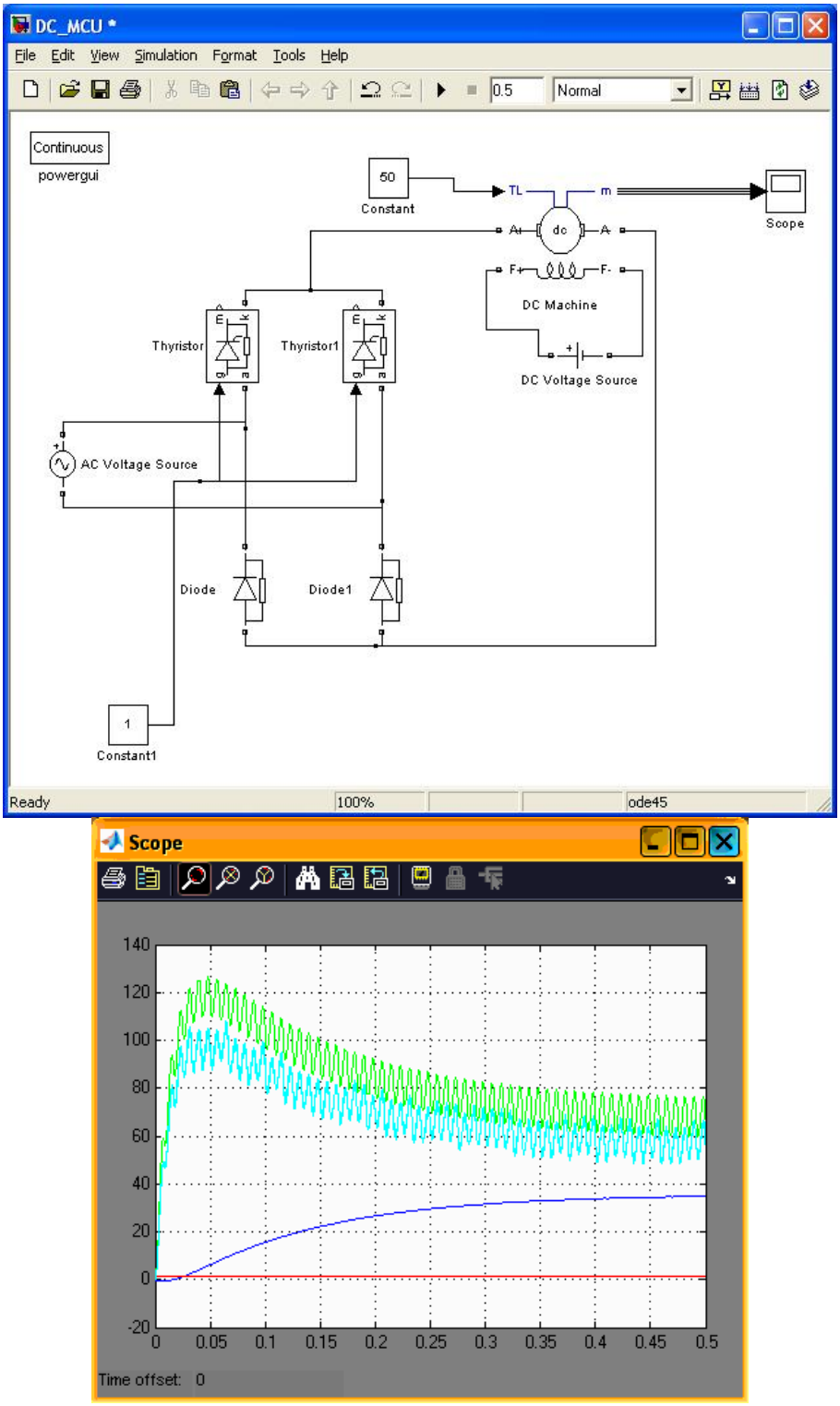
DC_MCU *
File Edit View Simulation Format Tools Help
0.5
Normal
Continuous
powergui
50
Constant
TL
m
Scope
A+
dc
A-
F+
F-
DC Machine
Thyristor
Thyristor1
DC Voltage Source
AC Voltage Source
Diode
Diode1
1
Constant1
Ready
100%
ode45
Scope
140
120
100
80
60
40
20
0
-20
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
Time offset: 0

Рисунок 2

Данное время для рассматриваемого примера было рассчитано из следующих соображений: контроль выдачи импульса тиристора будет осуществляться с погрешностью 1 градус, в каждом периоде напряжения 360 градусов, в каждой секунде – 50 периодов напряжения, а значит длительность одного градуса равна 1/360/50 = 0,000055555, или 1/18000 секунды, то есть микропроцессорная система будет 18000 раз в секунду принимать решение о том, выставить сигнал на открытие тиристора или убрать.

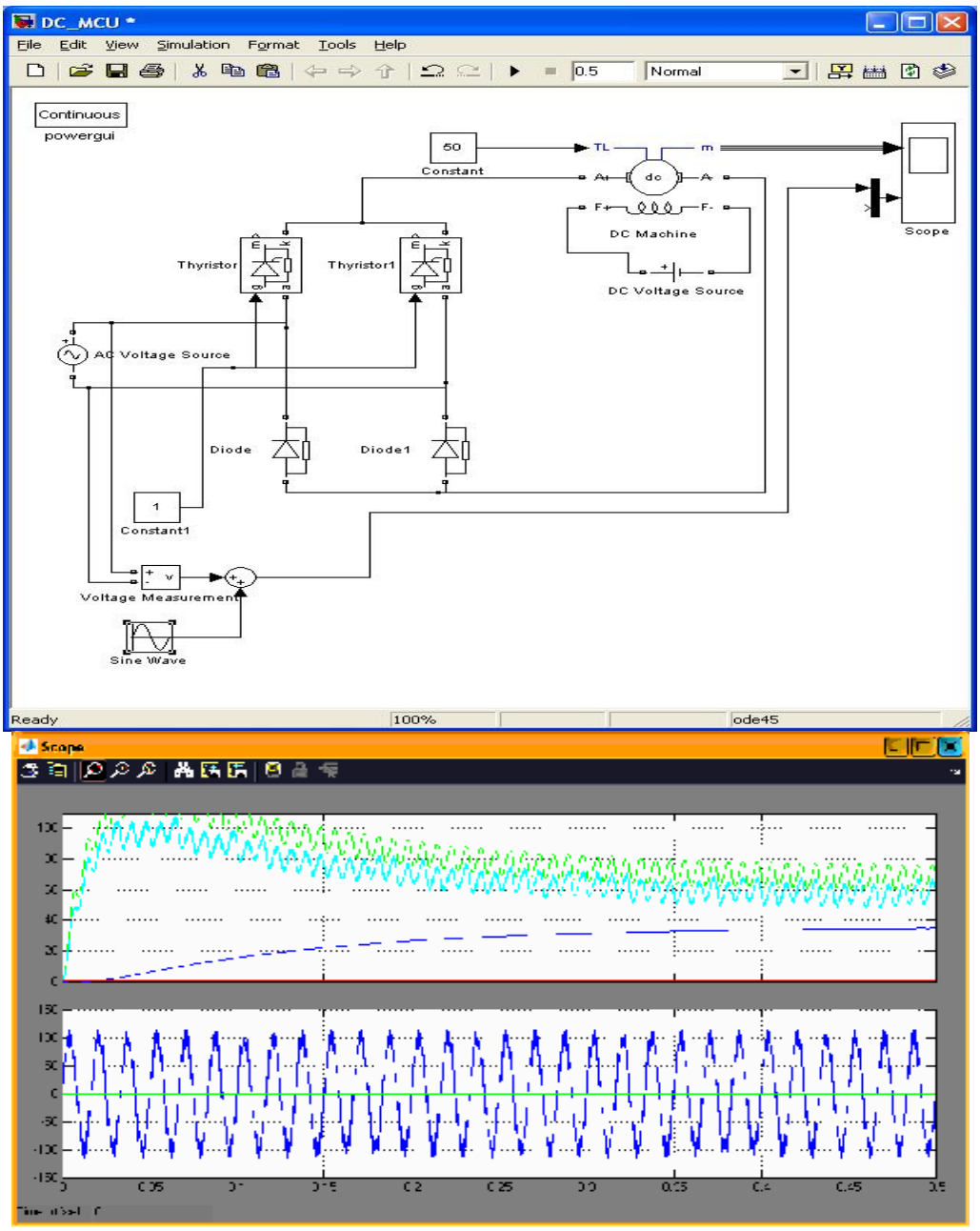

Рисунок 3

Также необходимо обеспечить ввод и вывод информации в модель микропроцессора. Для этого на закладке Data Properties устанавливаем в закладке Input Ports входные переменные set_angle (переменная для задания угла открытия тиристора) и u_net (переменная для ввода в систему информации о напряжении сети с целью синхронизации системы управления тиристором). На закладке Output Ports устанавливаем переменную out, в которую будет выдаваться сигнал для передачи на управляющий электрод тиристора.

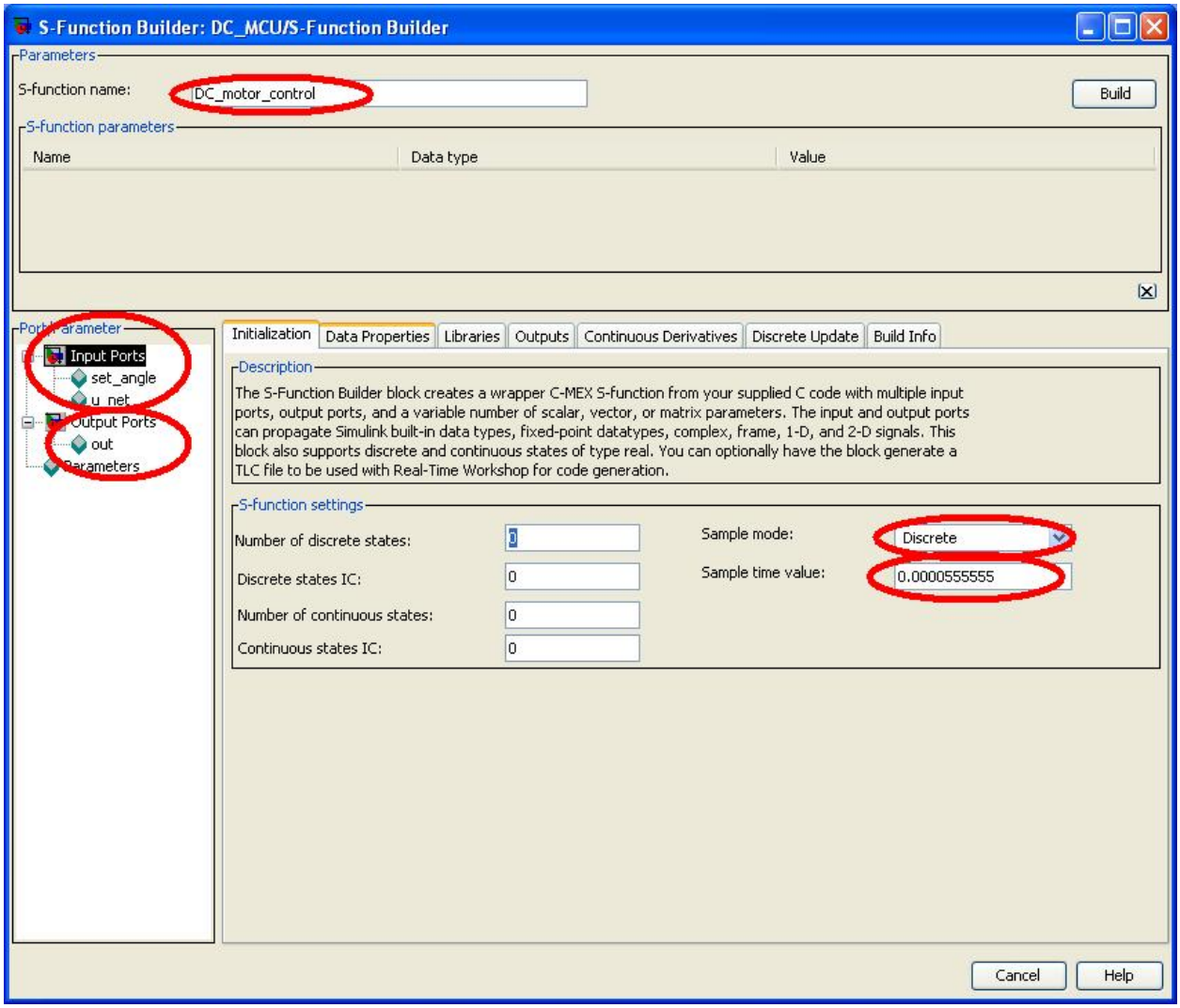

Рисунок 4

Подключаем модель микропроцессора к ранее созданной модели – для этого вводим в нее сигнал с модели датчика напряжения на вход u_net, для задания угла открытия тиристоров устанавливаем константу,

сигналы управления будем смотреть на осциллографе, заведя их через ранее неподключенный вход мультиплексора.

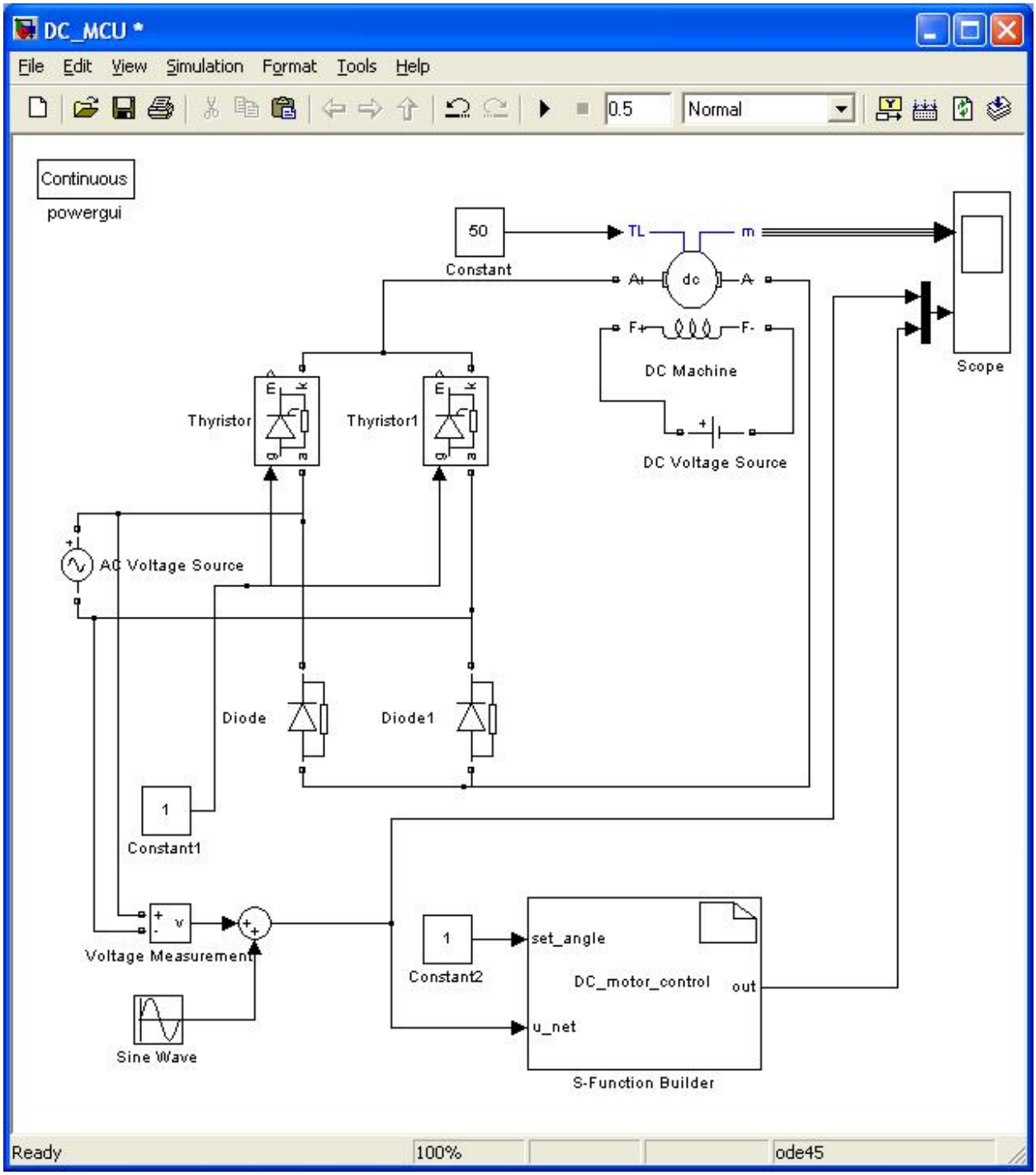

Рисунок 5

В закладку Output компонента S-функции вставляем алгоритм управления СИФУ (объявление переменных и сам алгоритм):

```
int static angle = 0; //переменная с инфомрацией о текущей угле фазы синусоиды с датчика напряжения
int static u = 0; // сигнал с датчика напряжения
int static u_prev = 0; // сигнал с дачтика напряжения, полученный в предыдущий раз

// СИФУ
u = u_net[0];// забираем сигнал с датчика напряжения
angle++; // увеличиваем текущее значение угла фазы сигнала с датчика напряжения
if (angle > set_angle[0]) out[0] = 100; // если угол больше чем заданный, то в включаем тиристор
   else out[0] = 0;// иначе - выключаем
if (u>=0) if (u_prev<0) angle = 0; //если положительный переход синусоиды через ноль, то обнуляем значение угла
if (u<=0) if (u_prev>0) angle = 0; //если отрицательный переход синусоиды через ноль, то обнуляем значение угла
u_prev = u;// запоминаем для следующего шага текущее значение с датчика напряжения
```

Основная цель функционирования такого алгоритма – выдача импульса открытия на управляющий электрод тиристора в соответствии с заданным углом открытия. Сам тиристор находится под переменным напряжением, поэтому для обеспечения открытия в заданный момент (задаваемый углом открытия) необходимо отслеживать фазу приложенного к нему напряжения. Когда напряжение переходит через нулевое значение, считается что угол тоже равняется нулю. Затем, угол начинает увеличиваться до 180 градусов, и в этот промежуток времени можно подавать управляющий импульс для его открытия.

Идея алгоритма заключается в следующем: при переходе сигнала напряжения через ноль необходимо обнулить значение угла. Переход может быть как положительный, так и отрицательный – в нашем случае это значение не имеет, так как импульс будет выдаваться сразу на оба тиристора – из них реально откроется только тот, который находится под своей полуволной напряжения, другой не сможет открыться из-за физического принципа действия. Каждый раз при запуске процедуры (а это будет происходить 18000 раз в секунду, об этом говорилось выше) происходит увеличение значение угла фазы синусоидального сигнала напряжения на единицу. Когда это значение становится больше заданного, происходит выдача значение 100 (возможно любое, большее ну-

ля), в противном случае выдается значение 0, которое не позволяет тиристору произвести открытие. Если все было сделано правильно, то для случая задания угла открытия 90 градусов показан рисунок 6: на нижнем графике показан сигнал с модели датчика напряжения и формируемые импульсы, строго следующие в середину полуволны сигнала.

Обязательным является объявление переменных алгоритма как static – это необходимо для того, чтобы при очередном вызове процедуры эти переменные не создавались заново, и соответственно, не теряли предыдущего значения.

Замыкаем теперь выход системы управления на управляющие электроды тиристоров (рисунок 7). Можно наблюдать поведение координат двигателя при «полуоткрытых» тиристорах.

Замкнем систему по скорости. Для этого убираем входной порт set_angle в S-функции, и вместо него устанавливаем два новых входа – задание скорости (set_speed) и сигнал обратной связи по скорости (fb_speed, fb – от английского feedback – «обратная связь»), см. рисунок 8.

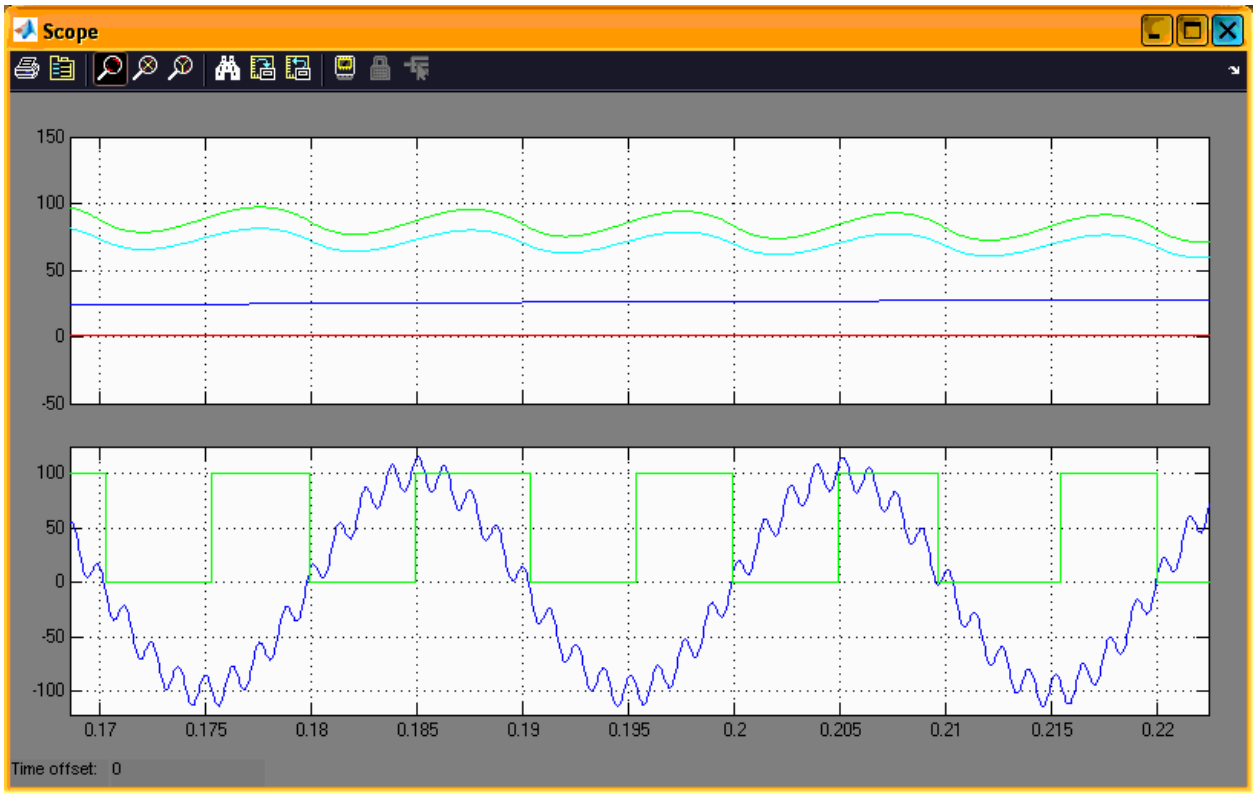

Рисунок 6

В закладку Outputs добавляем код регулятора скорости (ниже выделен шрифтом), в результате управляющий код выглядит следующим образом:

```
#define step 10 // шаг интегрирования
int static angle = 0;
int static u = 0;
int static u_prev = 0;
int static set_angle = 120;
int static counter = 0;
// регулятор скорости
counter++;
if (counter>36)
   {
      if (set_speed[0]>fb_speed[0]) set_angle = set_angle - step;
      if (set_speed[0]<fb_speed[0]) set_angle = set_angle + step;
      if (set_angle>170) set_angle = 170;
      if (set_angle<1) set_angle = 1;
      counter = 0;
   }


// СИФУ
u = u_net[0];
angle++;
if (angle > set_angle) out[0] = 100; else out[0] = 0;
if (u>=0) if (u_prev<0) angle = 0;
if (u<=0) if (u_prev>0) angle = 0;
u_prev = u;
```

Регулятор скорости построен по следующему принципу: если сигнал скорости двигателя (сигнал обратной связи) меньше заданного, то в таком случае начинаем уменьшать значение угла управления СИФУ, тем самым тиристоры открываются раньше, приложенное напряжение к двигателю больше, соответственно ток двигателя больше, он развивает больший момент и в конечном счете увеличивает скорость. Если сигнал скорости двигателя больше заданного, то в таком случае начинаем увеличивать угол открытия тиристора, уменьшая тем самым приложенное к двигателю напряжение, что приводит к снижению тока двигателя, а также его момента, в результате чего скорость падает. Изменение угла за одно вычисление происходит на величину *step*, которая задана в дан-

ном примере как константа (объявлена в начале директивой *#define*). Для того, чтобы увеличение или уменьшение шага не происходило бесконечно, значение выхода регулятора скорости (в данном случае это вход СИФУ, то есть переменная *set_angle*) ограничено минимальным и максимальным значением.

Так как двигатель обладает механической инерцией якоря, то нет необходимости рассчитывать управление с той же частотой, на которой рассчитывается управление тиристорами. Для снижение частоты расчета регулятора скорости применена переменная *counter*, которая каждый раз, достигая значения 36, обнуляется, и при этом происходит вычисление значения регулятора скорости. В результате расчет происходит в 36 раз медленнее чем расчет алгоритма СИФУ, то есть частота дискретизации регулятора скорости равна 18000/36 = 500 Гц.

Выполняем операцию компиляции S-функции (нажатием кнопки Build), и если компиляция прошла без ошибок, то переходим к следующему шагу, в противном случае исправляем выявленные компилятором ошибки.

Как показывает осциллограф, компонент модели двигателя одновременно посылает на визуализацию вектор данных, состоящий из сигнала скорости тока и крутящего момента двигателя. Для выделения сигнала скорости используем компонент Bus selector (находится в группе Simulink, подгруппа Commonly Used Blocks), который позволяет выделить из шины, по которой идет вектор данных, только нужные сигналы. Подсоединяем вход компонента к шине выдачи сигналов координат двигателя (см. рисунок 9).

Двойным нажатием на компонент Bus selector вызываем окно настройки, в котором в области выбранных сигналов Selected signals удаляем имеющиеся сигналы с помощью кнопки Remove. Выделяем сигнал скорости Speed wm в области имеющихся сигналов Signals in the bus, и добавляем его в область выбранных сигналов с помощью кнопки Select (см. рисунок 10) Нажимаем кнопку OK для закрытия окна.

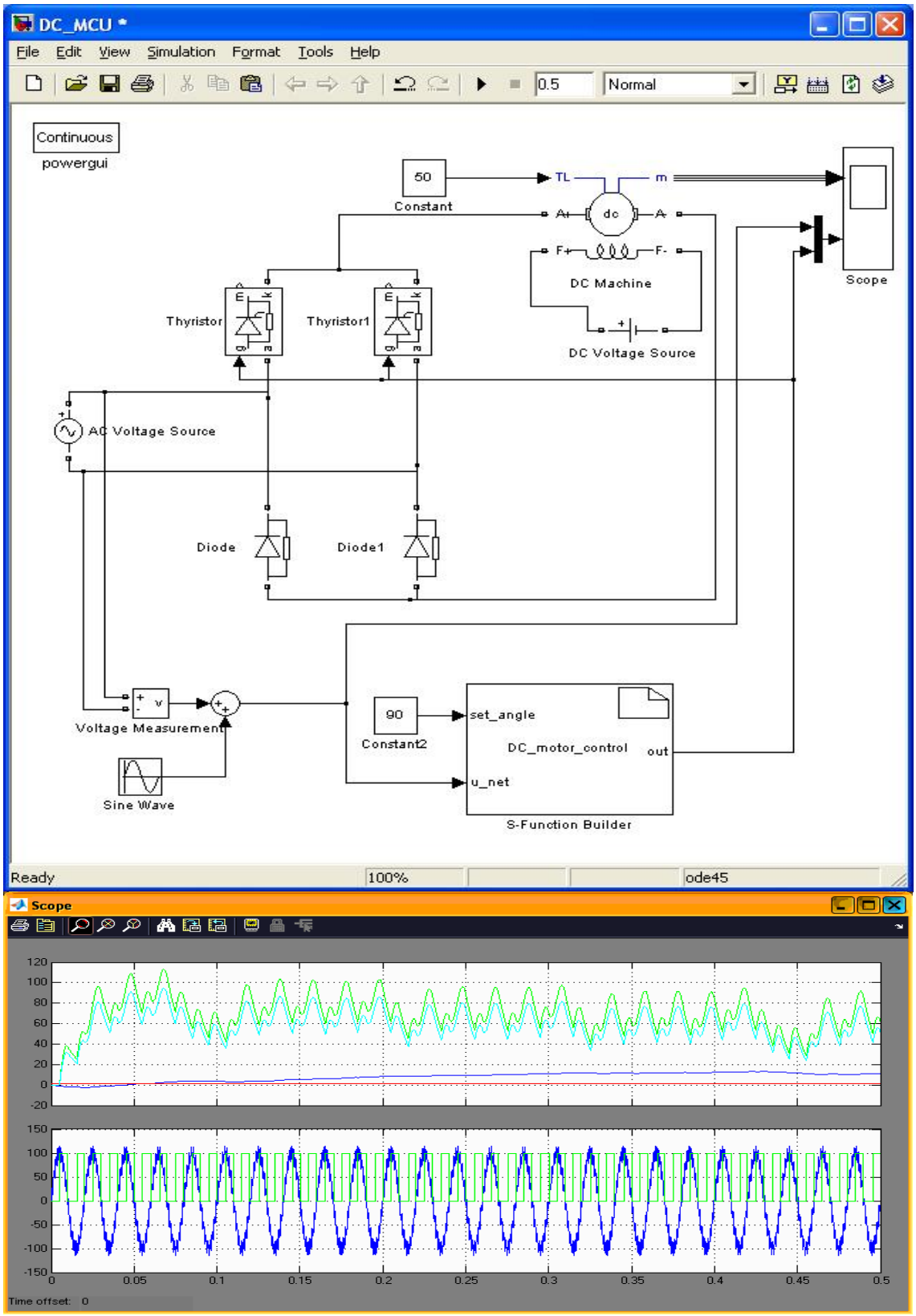
DC_MCU *
File Edit View Simulation Format Tools Help
0.5
Normal
Continuous
powergui
50
Constant
TL
m
A+
dc
A-
F+
F-
DC Machine
Scope
Thyristor
Thyristor1
DC Voltage Source
AC Voltage Source
Diode
Diode1
Voltage Measurement
90
Constant2
set_angle
DC_motor_control
out
u_net
Sine Wave
S-Function Builder
Ready
100%
ode45
Scope
120
100
80
60
40
20
0
-20
150
100
50
0
-50
-100
-150
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
Time offset: 0

Рисунок 7

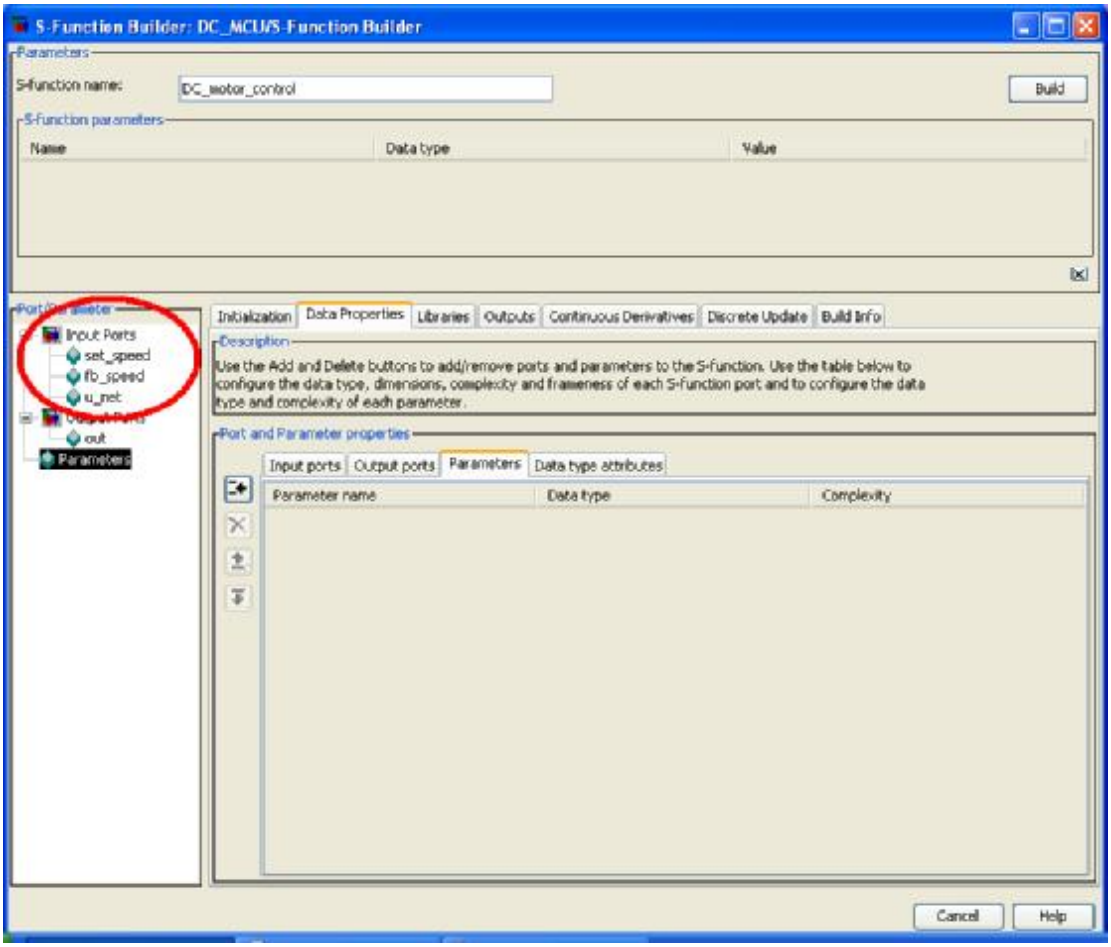

Рисунок 8

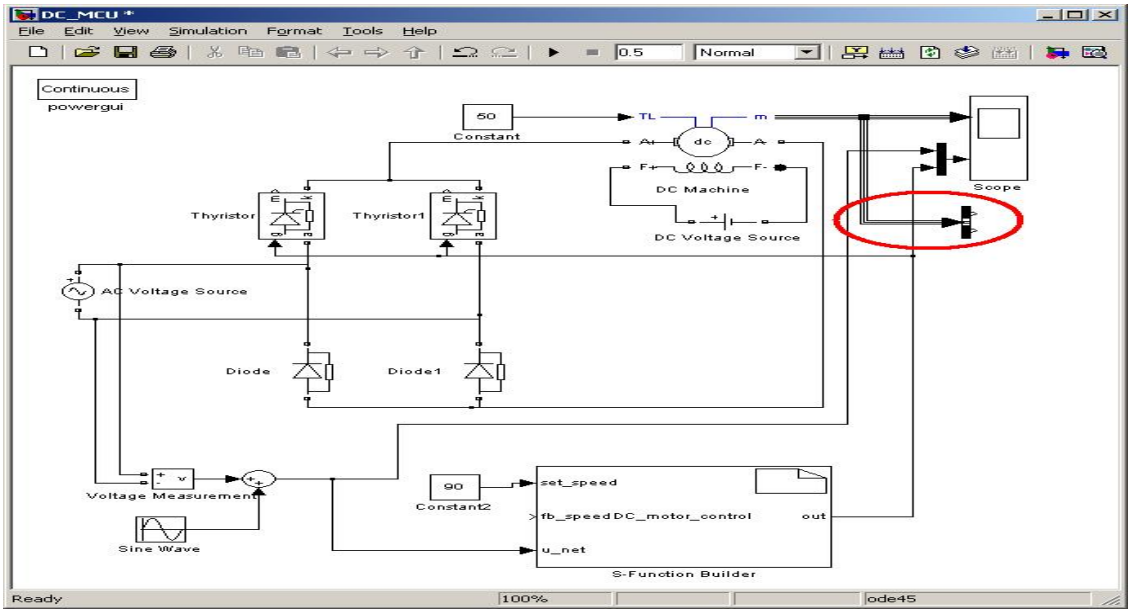

Рисунок 9

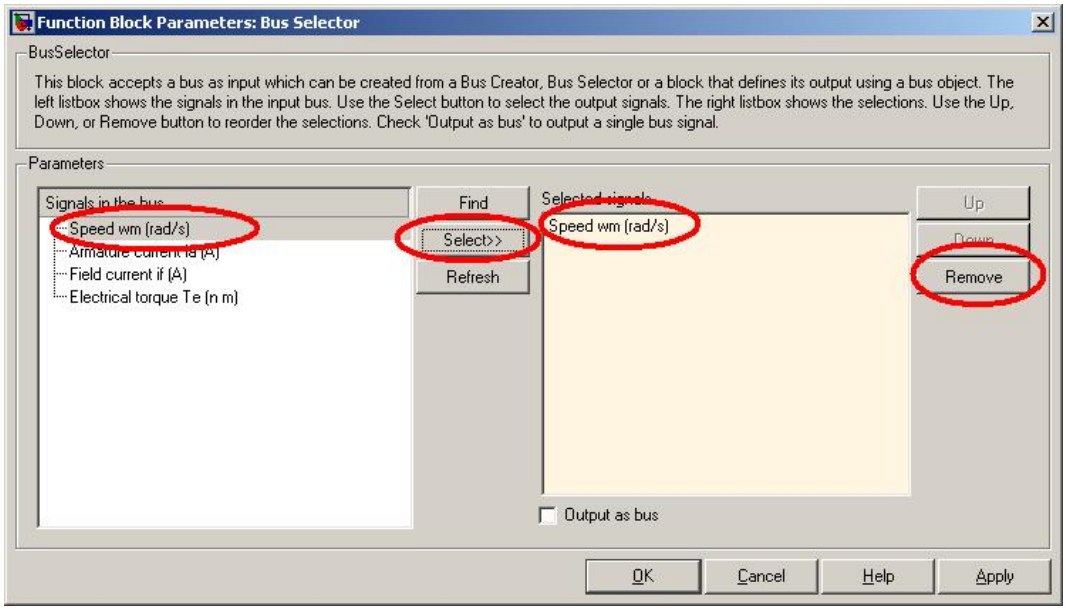

Рисунок 10

Соединяем выделенный сигнал скорости двигателя со входом S-функции, отвечающей за сигнал обратной связи fb_speed (см.рисунок 11).

Устанавливаем в компоненте AC Voltage Source значение амплитуды напряжения как 310 вольт (для 220 вольт фазного напряжения), см. рисунок 12.

Теперь система готова к запуска процесса проверки написанного управляющего кода. Устанавливаем время моделирования 1 секунда. Устанавливаем значение заданной скорости 150 (константа на входе set_speed S-функции), а задание на момент двигателя – 50 (константа на входе TL компонента модели двигателя). Запускаем моделирование, и убеждается в том что, система удерживает скорость в районе 150 рад/сек (см.рисунок 13). Пульсирующий характер скорости около заданного значения является следствием 3 причин – несовершенством регулятора скорости, несовершенством алгоритма СИФУ и примененной силовой схемой управления. Уменьшаем момент сопротивления до 20, и убеждаемся в том, что система теперь опять пытается удержать скорость в районе 150 рад/сек, однако период пульсаций стал больше. Проведите эксперименты для заданий скорости 200, 150, 100, 75 рад/сек при моментах нагрузки 20 и 50 Нм. Как покажут результаты моделирования, система «далека от совершенства».

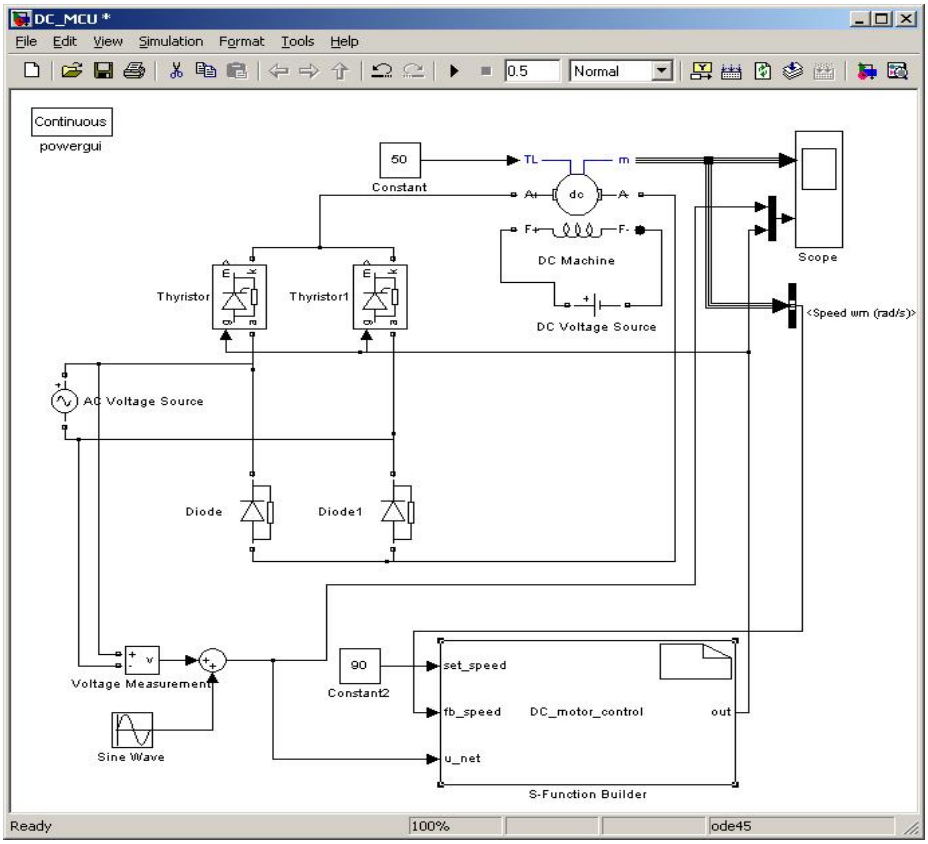

Рисунок 11

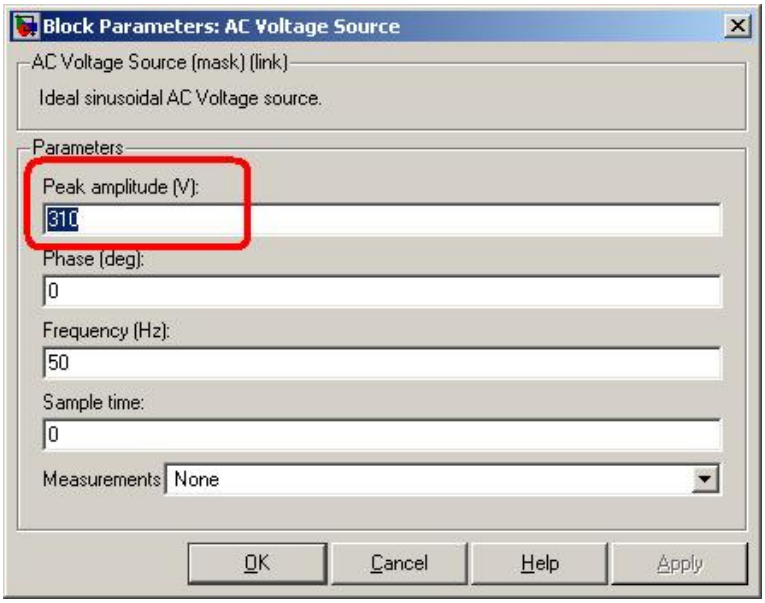

Рисунок 12

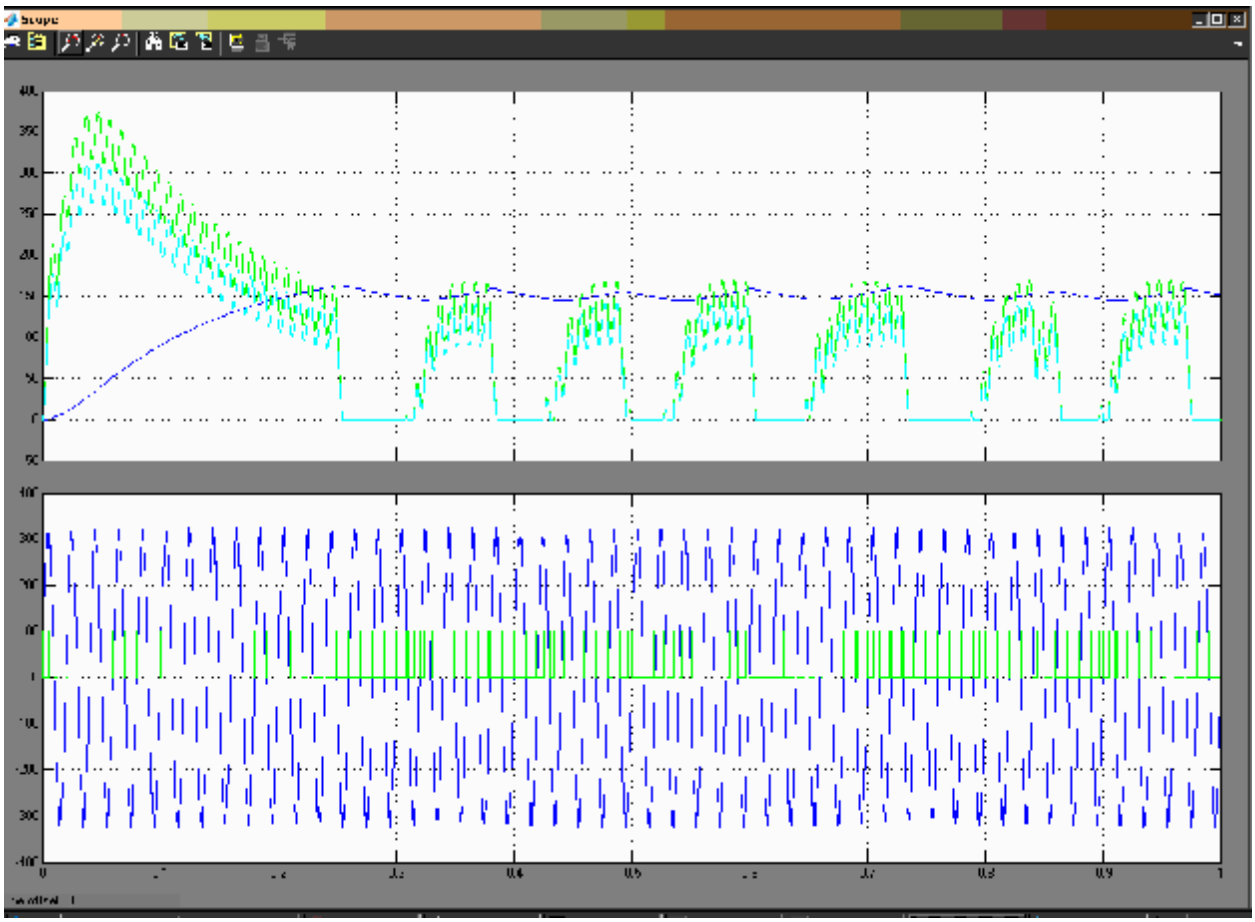

Рисунок 13

Задание для самостоятельной работы

Доработайте алгоритм управления с целью снижения пульсаций скорости при выходе на установившееся значение. Убедитесь в работоспособности предложенных вами решений для различных комбинаций задания скорости и момента сопротивления.

Одним из решений по улучшению работы алгоритма будет совершенствование алгоритма СИФУ. Применим не постоянное открытие тиристора вплоть до очередного перехода сигнала напряжения через ноль, а будем выдавать импульс только на время 8 градусов.

Для этого необходимо программу доработать до следующего вида (изменения касаются только той части, где происходит вычисление СИФУ, выделены шрифтом):

```
// СИФУ
u = u_net[0];
angle++;
if ((angle > set_angle)&&(angle < (set_angle+8)))
                        out[0] = 100; else out[0] = 0;
if (u>=0) if (u_prev<0) angle = 0;
if (u<=0) if (u_prev>0) angle = 0;
u_prev = u;
```

Повторим моделирование для заданий скорости 200, 150, 100, 75 рад/сек при моментах нагрузки 20 и 50 Нм и убедимся в том, что пульсации снизились, а диапазон регулирования увеличился.

В завершении, доработаем код до уровня, который позволит применить его в реальном микропроцессоре. Для этого надо убрать из алгоритма порты ввода/вывода, и оперировать только с объявленными в программе переменными. В результате в программе появляются переменны FbSpeed, SetSpeed, Out, а сама программа будет иметь следующий вид:

```
#define step 10 // шаг интегрирования
int static angle = 0;
int static u = 0;
int static u_prev = 0;
int static set_angle = 120;
int static counter = 0;
int static Out=170;
int static FbSpeed = 0;
int static SetSpeed = 0;

// секция присоединения входных портов S-функции к симулинку
u = u_net[0];
SetSpeed = set_speed[0];
FbSpeed = fb_speed[0];

// регулятор скорости
counter++;
if (counter>36)
   {
      if (SetSpeed>FbSpeed) set_angle = set_angle - step;
      if (SetSpeed<FbSpeed) set_angle = set_angle + step;
      if (set_angle>170) set_angle = 170;
```

```
      if (set_angle<1) set_angle = 1;
      counter = 0;
   }


// СИФУ
angle++;
if ((angle > set_angle)&&(angle < (set_angle+8))) Out = 100;
  else Out = 0;
if (u>=0) if (u_prev<0) angle = 0;
if (u<=0) if (u_prev>0) angle = 0;
u_prev = u;

// секция присоединения выходных портов S-функции к симулинку
out[0] = Out;
```

Содержание

Введение	3
Занятие 1. Создание моделей в среде Симулинк	4
Занятие 2. Разработка систем логического управления электроприводами	27
Занятие 3. Создание цифровых регуляторов	41
Занятие 4. Среда программирования CodeComposer-Studio. Использование симулятора	55
Занятие 5. Технология создания собственных функций для приложений	65
Занятие 6. Использование пакета Матлаб для разработки программного обеспечения сигнального процессора	71
Занятие 7. Разработка микропроцессорной системы управления двигателем постоянного тока	86

Учебное издание

КАЧИН Олег Сергеевич
КАРАКУЛОВ Александр Сергеевич

РАЗРАБОТКА АЛГОРИТМОВ УПРАВЛЕНИЯ ДЛЯ МИКРОПРОЦЕССОРНЫХ ЭЛЕКТРОПРИВОДОВ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Учебное пособие

Научный редактор
доктор технических наук,
профессор Р.Ф.Бекишев

Редактор *И.О. Фамилия*
Верстка *И.О. Фамилия*
Дизайн обложки *И.О. Фамилия*

Подписано к печати . .2008. Формат 60x84/16. Бумага «Снегурочка».
Печать Xerox. Усл. печ. л. 000. Уч.-изд. л. 000.
Заказ XXX. Тираж XXX экз.

Томский политехнический университет
Система менеджмента качества
Томского политехнического университета сертифицирована
NATIONAL QUALITY ASSURANCE по стандарту ISO 9001:2000

ИЗДАТЕЛЬСТВО ТПУ. 634050, г. Томск, пр. Ленина, 30.