ПРОЕКТИРОВАНИЕ КОМПЛЕКСОВ АППАРАТУРЫ СИСТЕМ УПРАВЛЕНИЯ ИСПЫТАНИЯМИ СЛОЖНЫХ ОБЪЕКТОВ. Часть 3

А.А. ЕГОРОВ, Д.А. СУРКОВ (ООО "АВИАТЭКС"), К.Ю. КИРПИЧЕВ (МАИ)

В статье проведен анализ взаимодействия графической среды программирования SCADE и микроконтроллера Atmega 168. Разработан алгоритм системы управления, программирование реализовано в среде SCADE, включая автоматическую кодогенерацию. Реализовано взаимодействие SCADE и Arduino. Представлены результаты проведенного моделирования и натурных испытаний системы.

Ключевые слова: *автоматизированное управление стендом; винтомоторная группа; БПЛА; мультикоптер; SCADE; Suite; Arduino IDE; среда графической разработки Processing; ATMEGA168; стандарт DO-178B; ARINC 661; жизненный цикл ПО; язык AVR; MaxMSP; MIT Media Lab; FSM – Finite-state machine; конечный автомат; циклограммы; массив данных; графики.*

В предыдущей статье [1] рассматривались программные средства системы автоматизированного управления стендом испытаний винтомоторной группы БПЛА. Представлена разработанная структурная схема стенда. Разработка системы управления велась на основе технологий модельно-ориентированного проектирования с использованием сред SCADE Suite; Arduino IDE и среды графической разработки Processing.

В настоящей статье будет проведен анализ взаимодействия графической среды программирования SCADE и микроконтроллера Atmega 168, представлен алгоритм системы управления и приведены результаты моделирования и натурных испытаний системы.

АНАЛИЗ ВЗАИМОДЕЙСТВИЯ ГРАФИЧЕСКОЙ СРЕДЫ ПРОГРАММИРОВАНИЯ SCADE И МИКРОКОНТРОЛЛЕРА ATmega 168

Основные свойства микроконтроллера ATmega 168

В основе разработанной системы управления используется платформа Arduino Nano 2.x, базирующаяся на микроконтроллере ATmega 168. Этот микроконтроллер широко применяется в различных задачах, он недорогой и легко приобретается.

Микроконтроллер – микросхема, предназначенная для управления электронными устройствами. Типичный микроконтроллер сочетает в себе функции процессора и периферийных устройств, может содержать ОЗУ и ПЗУ. По сути, это однокристальный компьютер, способный выполнять простые задачи. Большая часть выпускаемых в современном мире процессоров – микроконтроллеры.

Микроконтроллеры AVR семейства Mega, к которым относится ATmega 168, являются 8-разрядными микроконтроллерами с RISC-архитектурой. Они имеют электрически стираемую память программ (FLASH) и данных (EEPROM), а также разнообразные периферийные устройства. Следует отметить, что микроконтроллеры семейства Mega имеют самый богатый набор периферийных устройств по сравнению с микроконтроллерами других семейств.

ATmega 168 – низкопотребляющий 8 битный КМОП. Выполняя команды за один цикл, ATmega 168 достигает производительности 1 MIPS при частоте задающего генератора 1 МГц, что позволяет разработчику оптимизировать отношение потребления к производительности.

AVR ядро объединяет богатую систему команд и 32 рабочих регистра общего назначения. Все 32 регистра непосредственно связаны с арифметико-логическим устройством (АЛУ), что позволяет получить доступ к двум независимым регистрам при выполнении одной команды. В результате эта архитектура позволяет обеспечить в десятки раз большую производительность, чем стандартная CISC архитектура.

В микроконтроллерах AVR семейства Mega реализована Гарвардская архитектура, в соответствии с которой разделены не только адресные пространства памяти программ и памяти данных, но также и шины доступа к ним. Способы адресации и доступа к этим областям памяти также различны коллективами. Такая структура позволяет центральному процессору работать одновременно как с памятью программ, так и с памятью данных, что существенно увеличивает производительность. Каждая из областей памяти данных (ОЗУ и EEPROM) также расположена в своем адресном пространстве.

Микроконтроллер изготовлен по высокоплотной энергонезависимой технологии изготовления памяти компании Atmel. Встроенная ISP Flash позволяет перепрограммировать память программы в системе через последовательный SPI интерфейс программой-загрузчиком, выполняемой в AVR ядре, или обычным программатором энергонезависимой памяти.

Программа-загрузчик способна загрузить данные по любому интерфейсу, имеющемуся у микроконтроллера. При этом загруженная программа обрабатывается с обеспечением реального режима “Считывание при записи”. Объединив 8-ми битное RISK ядро с самопрограммирующейся внутри системы Flash памятью, корпорация Atmel сделала контроллер ATmega 168 мощным микроконтроллером, обеспечивающим большую гибкость и ценовую эффективность широкому кругу управляющих устройств.

ATmega 168 поддерживается различными программными средствами и интегрированными средствами разработки, такими как компиляторы C, макроассемблеры, программные отладчики/симуляторы, внутрисхемные эмуляторы и ознакомительные наборы.

Характеристики микроконтроллера ATmega 168:

- высококачественный низкопотребляющий 8-ми битный AVR микроконтроллер;
- передовая RISC архитектура;
- энергонезависимая память программ и данных;
- специальные характеристики микроконтроллера:
  - сброс при включении питания и детектор кратковременных пропаданий питания;
  - встроенный откалиброванный генератор;
  - внешние и внутренние источники прерывания;
  - пять режимов пониженного потребления: Idle, ADC Noise Reduction, Power-Save, Power-down и Standby.
- порты ввода – вывода и корпусное исполнение:
  - 23 программируемых линии портов ввода-вывода;
  - 32 выводные TQFP и MFL корпуса.
- диапазон напряжения питания: 4,5-5,5 В;
- коммерческий рабочий температурный диапазон;
- сверхнизкое потребление:
  - активный режим: 300 мкА при частоте 1 МГц и напряжении питания 1,8 В, 20 мкА при частоте 32 кГц и напряжении питания 1,8 В;
  - режим пониженного потребления: 0,5 мкА при напряжении питания 1,8 В.

Разработка алгоритма системы управления в среде SCADE

Логика системы автоматизированного управления стендом испытаний винтомоторной группы БПЛА реализована в виде конечного автомата в среде SCADE [2], структура которого представлена на рис. 1.

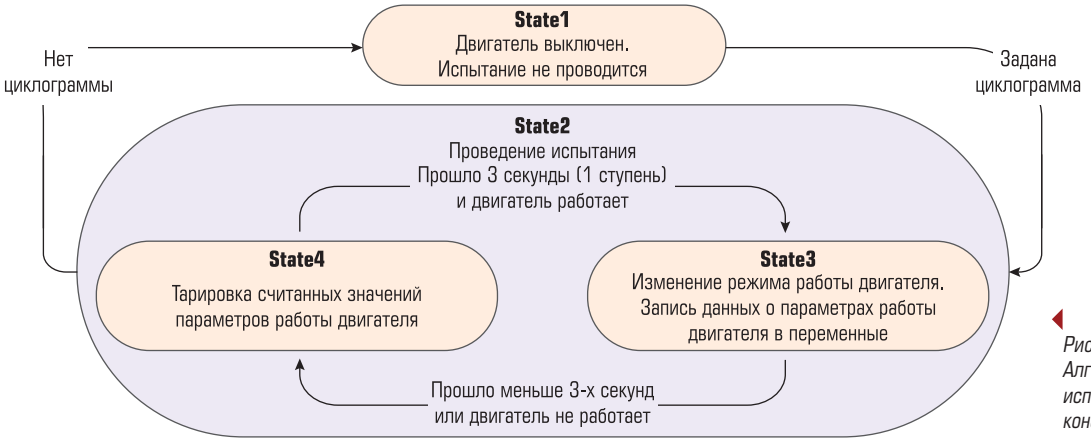


Рис. 1. Алгоритм проведения испытаний в форме конечного автомата

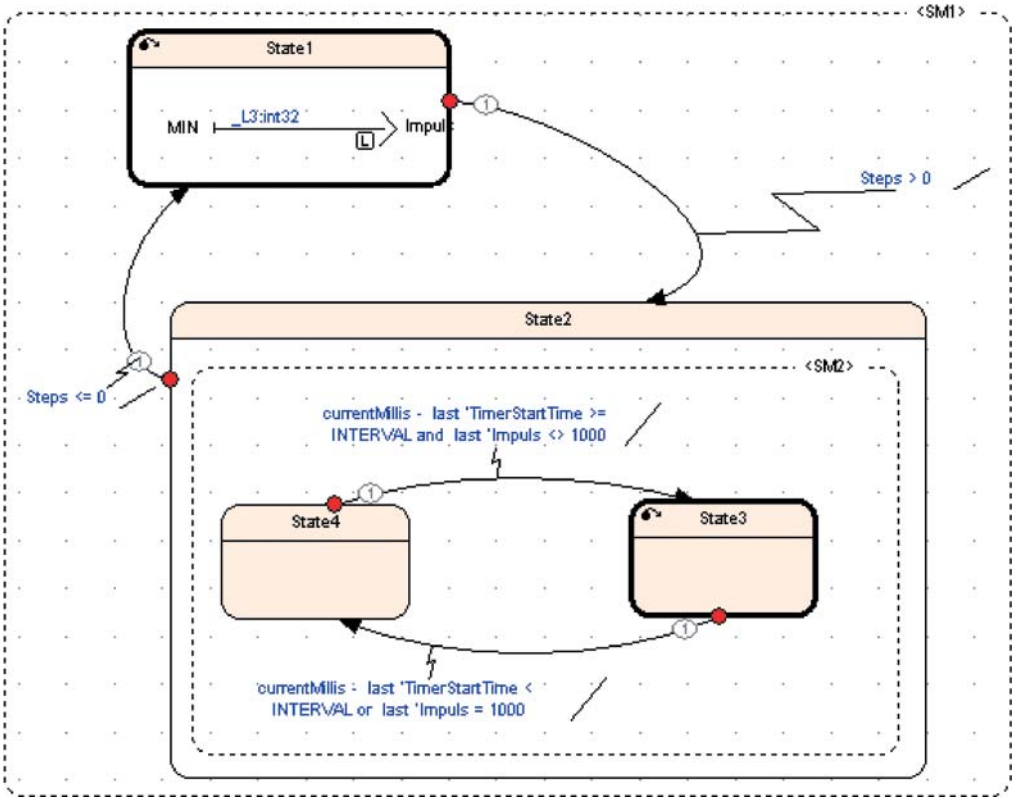


Рис. 2. Конечный автомат, реализованный с помощью ПО SCADE

Конечный автомат – это модель системы, содержащая конечное число ее состояний. Используется для представления и управления потоком выполнения каких-либо команд. Конечный автомат идеально подходит для реализации искусственного интеллекта, позволяя получить аккуратное решение без написания громоздкого и сложного кода.

Конечный автомат (FSM – Finite-state machine) это модель вычислений, основанная на гипотетической машине состояний. В один момент времени только одно состояние может быть активным. Следовательно, для выполнения каких-либо действий машина должна менять свое состояние.

Конечный автомат можно представить в виде графа, вершины которого являются состояниями, а ребра – переходы между ними. Каждое ребро имеет метку, информирующую о том, когда должен произойти переход. Например, на рис. 2 видно, что автомат сменит состояние State1 на состояние State2 при условии, что будет задана циклограмма на испытание.

Реализация конечного автомата начинается с выявления его состояний и переходов между ними. В конечном автомате, описывающем систему управления испытанием винтомоторной группы БПЛА есть 4 конечных состояния:

- State1 – испытание не проводится, двигатель выключен;
- State2 – испытание в процессе, двигатель работает.

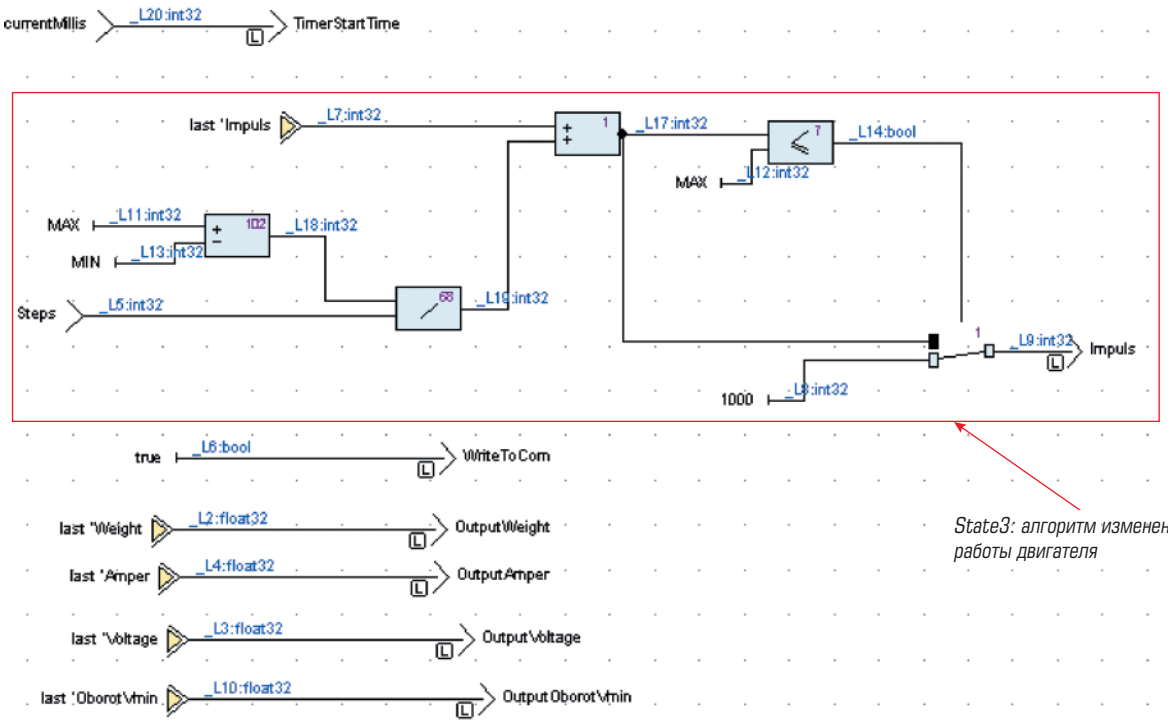
В состоянии State2, когда происходит испытание, есть два внутренних состояния:

- State3 – изменение режима работы двигателя;
- State4 – тарировка значений параметров испытаний.

Следующим шагом разработки программно-алгоритмического обеспечения была реализация этого конечного автомата в среде SCADE Suite.

Модель системы управления, реализованная с помощью конечного автомата, представлена на рис. 2.

Отправной точкой конечного автомата является состояние State1, которое остается активным до тех пор, пока не будет задано количество ступеней циклограммы – Steps. Когда это произойдет, состояние сменится на State2. Это состояние останется активным весь процесс проведения испытания.



State3: алгоритм изменения режима работы двигателя

Рис. 3. State3 конечного автомата

В состоянии State1 на выходную переменную Impuls подается значение, соответствующее имульсу, при воздействии которого сервопривод не приходит в движение.

В состоянии State2 описывается весь процесс испытания. В него входят состояния **State3** и **State4**. Реализация State3 показана на рис. 3. В данном состоянии промоделированы:

- алгоритм изменения режима работы двигателя;
- запись данных о параметрах его работы в выходные переменные;
- счетчик времени.

MAX и MIN являются константами, заданными в проекте, что показано на рис. 4.

Далее, с помощью блока “if ... else” происходит проверка, не превышает ли полученный импульс максимально допустимое значение, что проиллюстрировано на рис. 5. Если импульс равен или меньше максимального, его значение записывается в выходную переменную Impuls, если же оно превышает максимально возможное значение, на вход этой переменной подается значение 1000: при воздействии импульса данного значения двигатель прекращает свою работу.

В состоянии State3 также происходит запись значений параметров работы двигате-

Constant	Type	Value
A_KOEFTOKA	float32	0.074
ACP	float32	1023.0
B_KOEFTOKA	float32	37.88
COEF_HZ	float32	60.0
INTERVAL	int32	3000
MAX	int32	2000
MIN	int32	1140
NULL	int32	0
STEP_1	int32	1
TENZO_KOEF	float32	4.76
TENZO_NULL	float32	83750
VOLT_25	float32	25.0

Рис. 4. Константы проекта

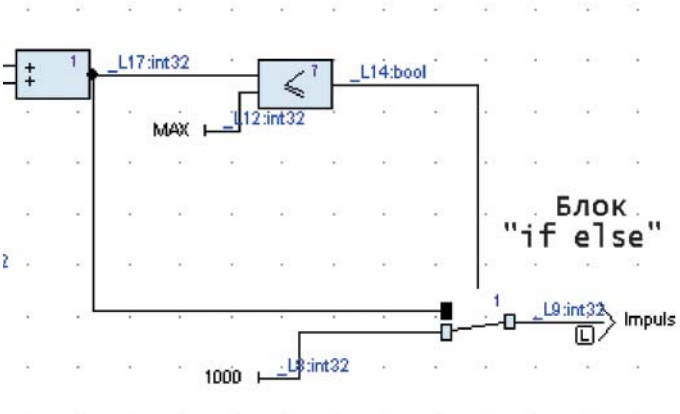


Рис. 5. State3: изменение режима работы двигателя

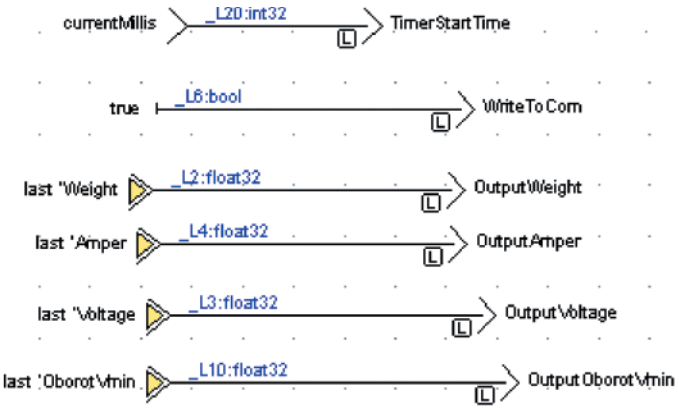


Рис. 6. State3: счетчик времени и запись значений в переменные

ля в выходные переменные, как показано на рис. 6. А именно, значения оборотов в минуту, тяги, тока и напряжения, считанные и от тарированные за предыдущую ступень циклограммы, присваиваются выходным переменным для каждого из параметров.

Кроме этого, в переменную TimerStartTime записывается значение текущего времени, прошедшего с начала испытаний, за которое отвечает переменная currentMillis.

Значение переменной WriteToCom в данном состоянии становится равным true. По факту изменения этой переменной система считывает выходные данные в алгоритме верхнего уровня.

После того, как состояние **State3** стало активно, а все прописанные в нем команды были выполнены, конечный автомат проверяет условие перехода к состоянию **State4**. Если время, прошедшее от момента активации **State3**, меньше необходимого интервала для одной ступени циклограммы, и если на двигатель поступает импульс, не равный 1000, конечный автомат переходит к состоянию **State4**, функции, исполняемые в данном состоянии, показаны на рис. 7.

В состоянии State4 происходит тарировка данных об испытании, считанных с датчиков. Входящие данные представляют собой отсчеты АЦП для параметров тока, напряжения и тяги, и значение частоты вращения для параметра оборотов двигателя. Значение переменной WriteToCom в данном состоянии равно false.

Тарировка параметров испытания производится с помощью дополнительных операторов.

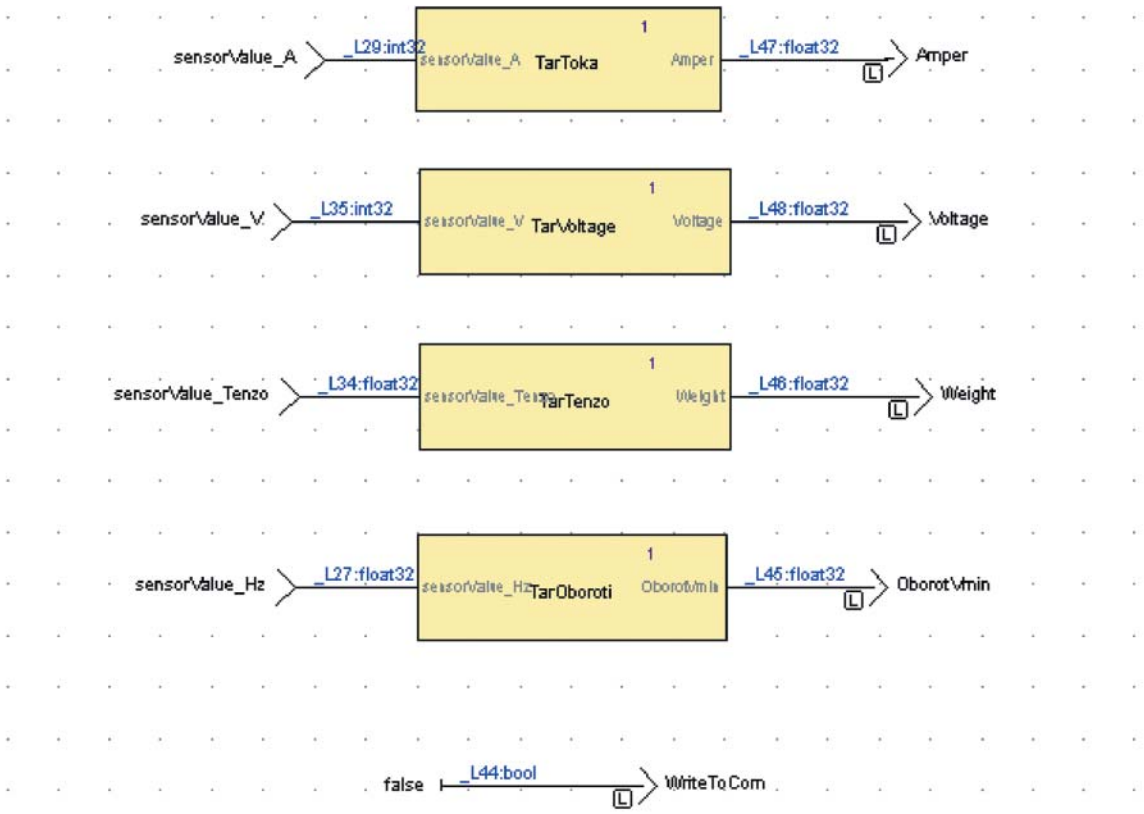


Рис. 7. State4: Тарировка данных с датчиков

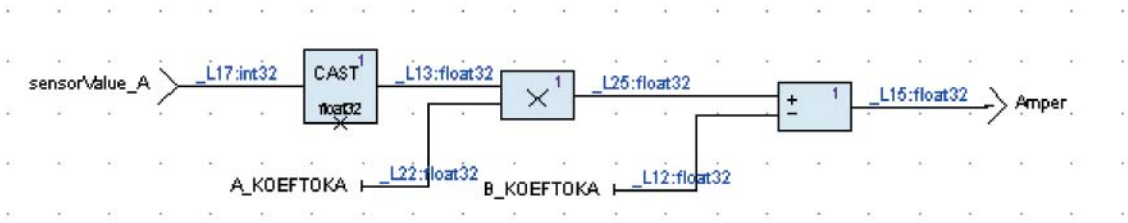


Рис. 8. State4: оператор для тарировки датчика тока

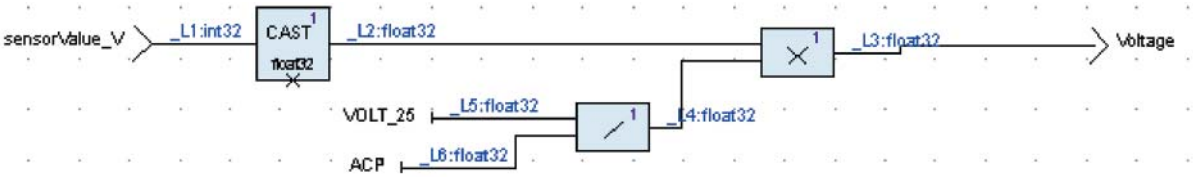


Рис. 9. State4: оператор для тарировки датчика напряжения

Рис. 10. State4: оператор для тарировки тензодатчика

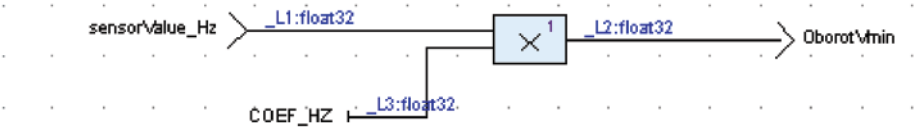


Рис. 11. State4: оператор для подсчета оборотов двигателя

Оператор для тарировки тока представляет собой модель уравнения для перевода отсчетов АЦП для датчика тока в амперы, как показано на рис. 8.

С помощью блока CAST входящее значение отсчетов АЦП sensorValue_A конвертируется из целочисленного типа данных в вещественный. Далее тарировка происходит по формуле:

$$I = 0{,}074 \cdot Count - 37{,}88.$$

Count = sensorValue_A. Коэффициенты формулы заданы в константах проекта, а именно A_KOEFTOKA = 0,074 и B_KOEFTOKA = 37,88. На выходе оператора получаем переменную Amper, значение которой соответствует значению текущего тока, потребляемого двигателем.

Оператор для тарировки датчика напряжения представлен на рис. 9.

Оператор представляет собой модель уравнения для переводов отсчетов АЦП в напряжение по формуле:

$$\text{Вольтаж} = Count \cdot \frac{5}{1023} \cdot 5.$$

VOLT_25 = 25, ACP = 1023, Count = sensorValue_V. На выходе данного оператора стоит переменная, значение которой равно напряжению блока питания в вольтах.

Оператор для тарировки тензодатчика и получения значения тяги показан на рис. 10.

Оператор представляет собой модель уравнения для перевода отсчетов АЦП для тензодатчика в граммы по формуле:

$$\text{Тяга} = (Count - 83\,750) \cdot 4{,}76.$$

Count = sensorValue_Tenzo. Коэфициенты TENZO_NULL = 83 750 и TENZO_KOEF = 4,76. На выходе данного оператора стоит переменная, значение которой равно тяге двигателя в граммах.

Оператор для подсчета оборотов двигателя показан на рис. 11. Чтобы узнать обороты в минуту данного двигателя, зная частоту его

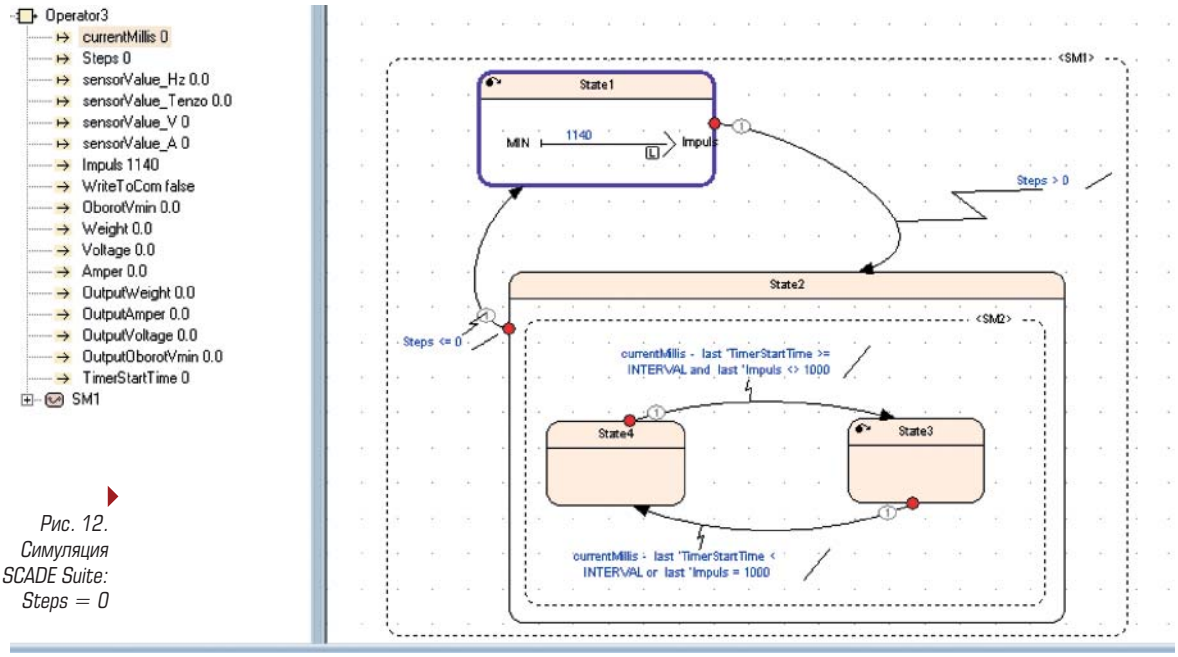


Рис. 12. Симуляция в SCADE Suite: Steps = 0

вращения, надо умножить эту частоту на 60 (количество секунд в минуте). В данной функции константа COEF_HZ = 60.

На выходе оператора переменная, значение которой равно оборотам в минуту.

После активации состояния State4 и выполнения всех его операторов, конечный автомат проверяет условие перехода к состоянию State3: если прошло 3 секунды (время ступени) с момента активации состояния или если последний импульс, поданный на двигатель, не был равен 1000, конечный автомат переходит в состояние State3.

Когда на двигатель будет подан импульс, равный 1000, испытание завершится.

Данный конечный автомат входит в состав оператора Operator3 проекта SCADESuite. После компиляции всех операторов, в случае отсутствия ошибок, программное обеспечение SCADE позволяет провести моделирование работы конечного автомата в режиме симуляции.

После запуска симуляции конечный автомат находится в состоянии State1, как показано на рис. 12. Переменная Steps = 0.

Изменим значение переменной Steps = 10 (рис. 13). Конечный автомат перешел в состо-

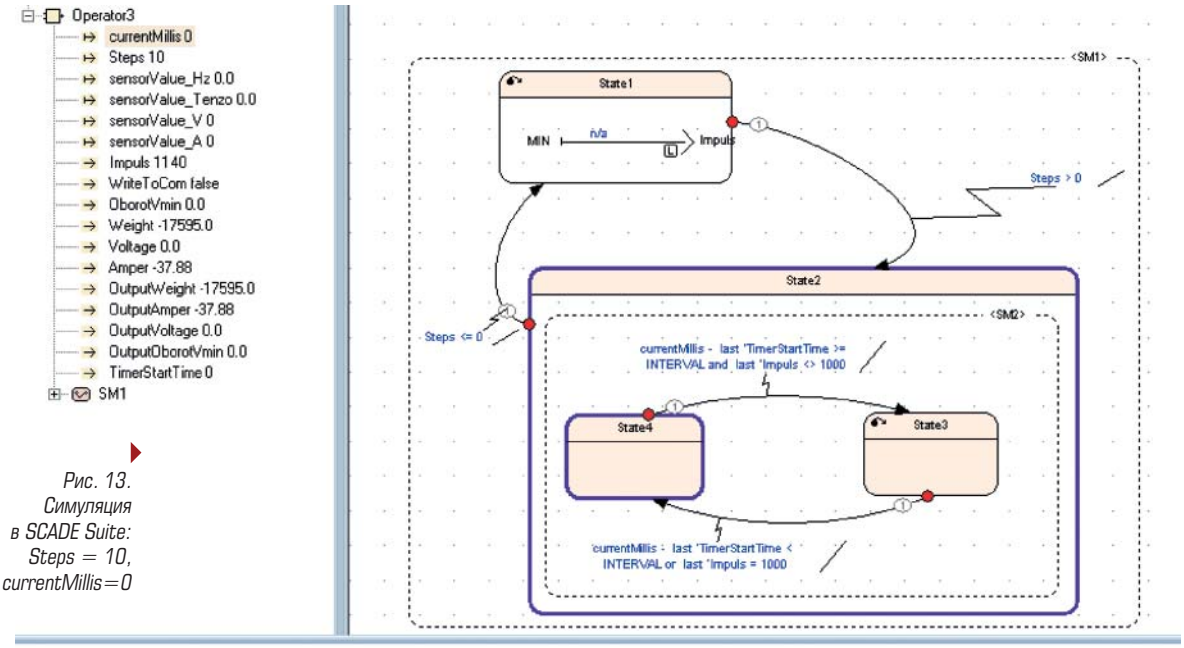


Рис. 13. Симуляция в SCADE Suite: Steps = 10, currentMillis=0

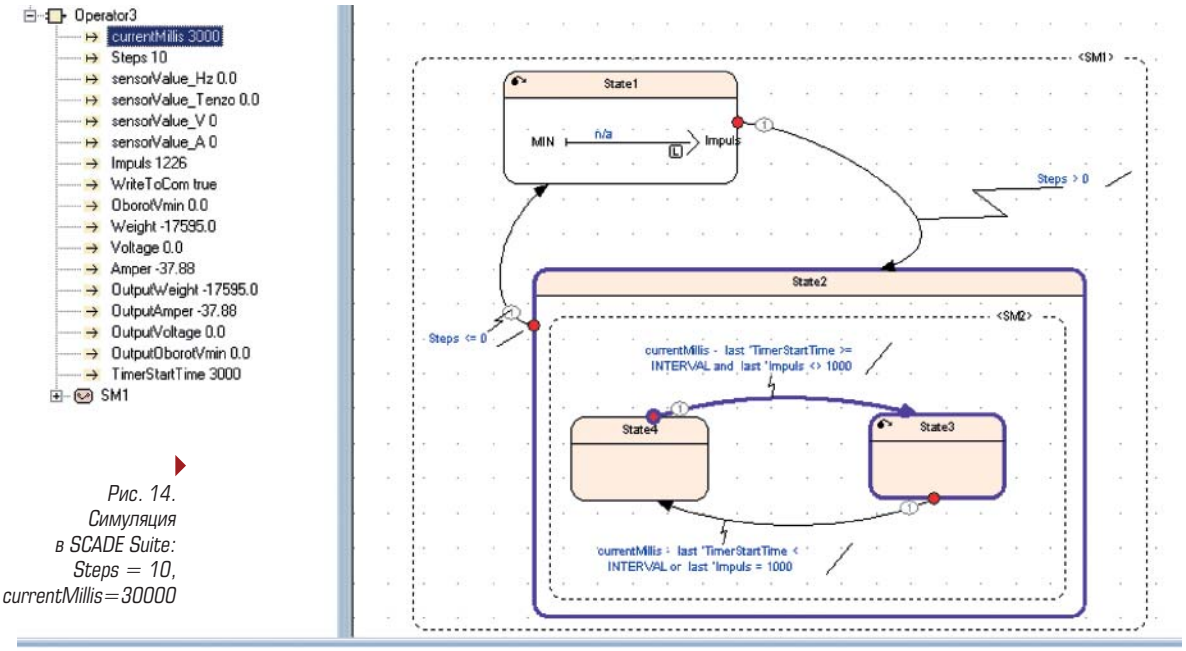


Рис. 14. Симуляция в SCADE Suite: Steps = 10, currentMillis=30000

яние State2 – State4. Конечный автомат будет находиться в этом состоянии до тех пор, пока не изменится значение переменной currentMillis, которая отвечает за отсчет времени в процессе проведения испытания. В данном состоянии выходные переменные TimerStartTime = 0 Impuls = MIN и WriteToCom = false.

Зададим вручную количество прошедших миллисекунд currentMillis = 3000 (рис. 14). Видим, что конечный автомат перешел в состояние State3. В этом состоянии переменные TimerStartTime = 3000, Impuls = 1226 и WriteToCom = true.

Однако, выполнив все функции состояния State3 и записав новое значение в переменную Impuls, отвечающую за смену режима работы двигателя, конечный автомат сразу переходит обратно в состояние State4, как показано на рис. 13.

После того, как конечный автомат сделает заданное количество переходов между состояниями, а переменная Impuls достигнет своего максимального значения, проведение испытания будет завершено, Impuls станет равна 1000 и конечный автомат вернется в состояние State4, пока не будет задана новая циклограмма на испытание (рис. 15).

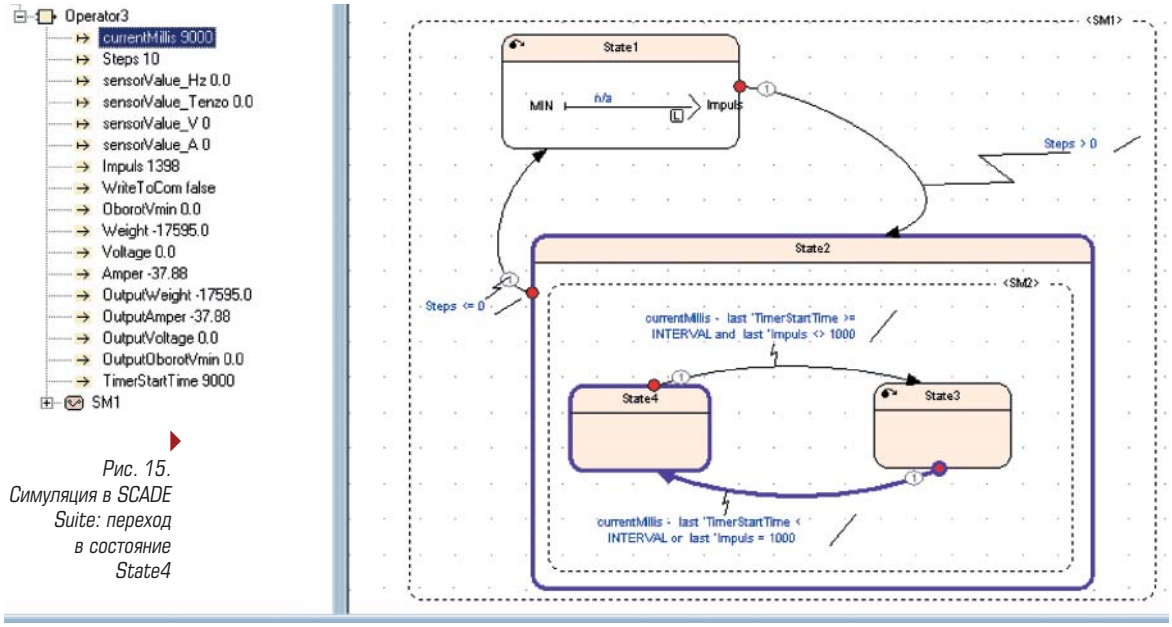


Рис. 15. Симуляция в SCADE Suite: переход в состояние State4

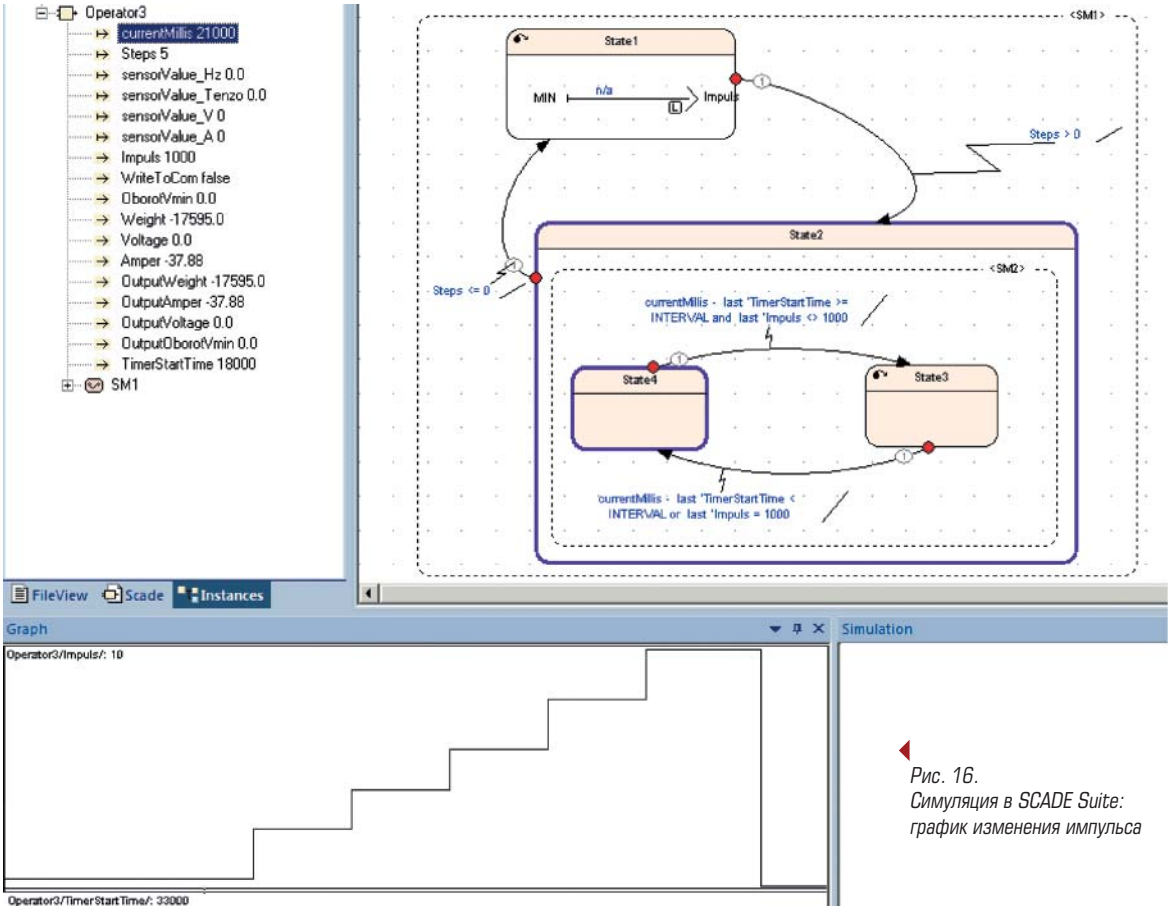


Рис. 16. Симуляция в SCADE Suite: график изменения импульса

Чтобы наглядно продемонстрировать изменение переменной Impuls в зависимости от изменяемого времени, зададим новую циклограмму для испытаний в режиме симуляции – Steps = 5 – и построим график переменной Impuls с помощью функции Graph. Теперь в процессе изменения значения переменной currentMillis, можно в графическом виде наблюдать как меняется переменная Impuls. На рис. 16 показано построение графика в окне симуляции. На оси *X* находятся значения выходной переменной TimerStartTime, на оси *Y* – значения переменной Impuls.

После анализа графика можно сделать вывод, что значение импульса, подающегося на двигатель, меняется ровно по ступеням циклограммы, а когда оно достигает максимального значения, импульс сбрасывается ниже минимального значения и больше не меняется, что соответствует окончанию испытания.

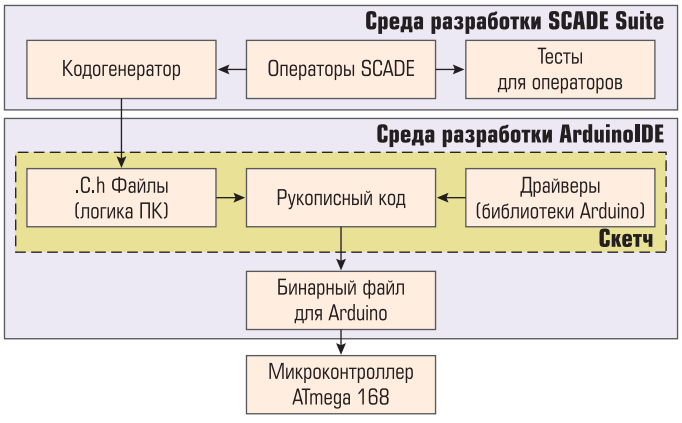


Рис. 17. Взаимодействие SCADE и Arduino

Взаимодействие микроконтроллера ATmega168 и SCADE

Микроконтроллер ATmega168 взаимодействует со средой разработки SCADE посредством платформы Arduino Nano.

Для взаимодействия с компьютером и программным обеспечением SCADE платформа Arduino Nano, построенная на микроконтроллере ATmega168, использует готовые драйверы для ввода и вывода данных, а также готовые библиотеки. Схема взаимодействия сред разработки SCADE и Arduino представлена на рис. 17.

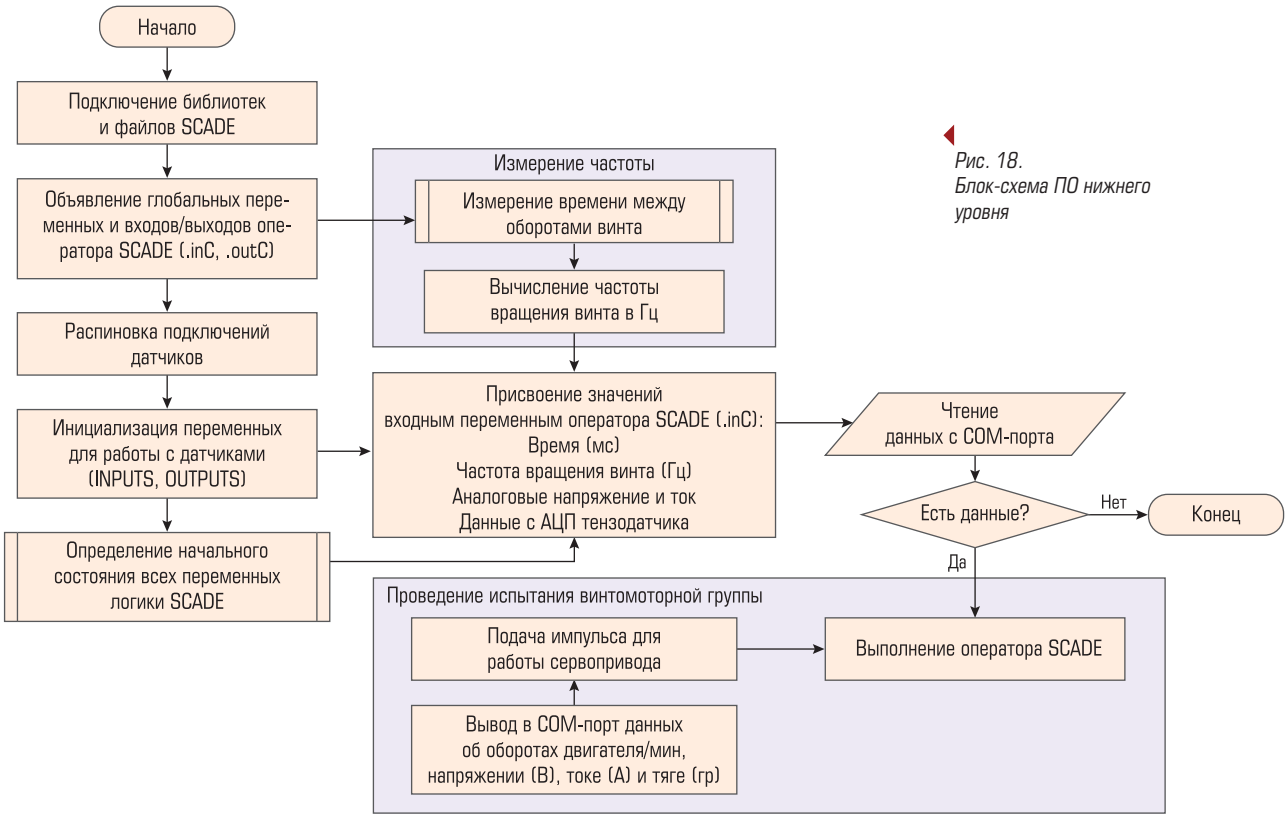


Рис. 18. Блок-схема ПО нижнего уровня

Операторы, созданные в SCADE Suite, с помощью кодогенератора KCG генерируются в файл с С-кодом формата ".c". Эти файлы подключаются в скетч Arduino вместе с библиотеками. Далее скетч дополняется необходимым рукописным кодом, с которым работают драйверы и библиотеки Arduino. Скетч загружается в плату Arduino Nano в виде бинарного файла, который использует микроконтроллер ATmega 168 для дальнейшей работы.

Разработка алгоритма и программы "нижнего уровня" системы. Блок-схема алгоритма "нижнего уровня" системы

Программное обеспечение (ПО) "нижнего" уровня системы работает согласно блок схеме, приведенной на рис. 18.

Опишем блок подключения библиотек и файлов SCADE: с помощью команды #include подключаем библиотеки для управления двигателем и для работы с тензодатчиком.

Также добавляем в проект файлы с С-кодом, сгенерированным в SCADESuite. Для этого после команды #include прописываем путь к нужным файлам. Они необходимы для работы операторов, смоделированных в SCADE.

Реализация блока подключения библиотек и файлов SCADE в тексте программы (Листинг 1):

```
//Библиотека для управления двигателем
#include <Servo.h>
//Библиотека для АЦП для тензодатчика
#include "Q2HX711.h"
//Подключение операторов. Файлы с кодом из SCADE Suite
#include "C:\Users\PC\SCADE\olja\stend_vint\KCG\Operator3.c"
#include "C:\Users\PC\SCADE\olja\stend_vint\KCG\TarOboroti.c"
#include "C:\Users\PC\SCADE\olja\stend_vint\KCG\TarTenzo.c"
#include "C:\Users\PC\SCADE\olja\stend_vint\KCG\TarToka.c"
#include "C:\Users\PC\SCADE\olja\stend_vint\KCG\TarVoltage.c"
```

В блоке объявления глобальных переменных и входов-выходов операторов SCADE задается объект для управления двигателем, а также объявляются следующие переменные:

- массив для приема данных;
- переменная для записи количества ступеней циклограммы;
- строка для приема данных через COM-порт.

Также в этом блоке объявляются входы (.inC) и выходы (.outC) оператора SCADE. Они нужны для дальнейшего использования переменных, заданных в проекте SCADE:

Реализация блока объявления глобальных переменных и входов-выходов операторов SCADE в тексте программы (Листинг 2):

```
//Объект двигателя
Servo myservo;
int cmd = -1;
//Массив для приема данных
int Arr[500];
//Количество ступней циклограммы
int n=-1;
//Объявление входных и выходных переменных оператора SCADE
inC_Operator3 inC;
outC_Operator3 outC;
//Строка для приема данных
String inString = "";
```

В блоке распиновки подключений датчиков задаются переменные, определяющие пины платы ArduinoNano, к которым подключены датчики стенда испытаний винтомоторной группы БПЛА.

Реализация блока распиновки подключений датчиков в тексте программы (Листинг 3):

```
//Частота вращения винта
int PINrotorFreqIN = 2;
//Частота вращения винта желтый провод
//Ацп для тензодатчика HX711
int PINtenzoDT = 3; //Тензодатчик DT зеленый провод
int PINtenzoSCK = 4; //Тензодатчик SCK желтый
//Сервопривод
int PINservoSIG = 5; //Управление регулятором SIG желтый
int PINservoREZERV = 6; //Управление регулятором пусто
int PINservoGND = 7; //Управление регулятором GND коричневый
//Тензодатчик
Q2HX711 hx711(PINtenzoDT, PINtenzoSCK);
//Измерение калибровочного напряжения
int PINled = 13; //Калибровочное напряжение + LED коричневый
int PINcalibrAI = A0; //Калибровочное напряжение коричневый
//Измерение тока
int PINtokSIG = A6; //Измерезие тока SIG синий
```

В блоке инициализации переменных для работы с датчиками описываются функции для работы с датчиками, которым заданы пины платы Arduino, к которым они подключены.

Реализация блока инициализации переменных для работы с датчиками в тексте программы (Листинг 4):

```
//Частота вращения винта
pinMode(PINrotorFreqIN, INPUT);
//Сервопривод
pinMode(PINservoGND, OUTPUT);
digitalWrite(PINled, LOW);
pinMode(PINservoREZERV, INPUT_PULLUP);
myservo.attach(PINservoSIG);
//Инициализация сервопривода
myservo.writeMicroseconds(1000);
//Калибровочное напряжение
pinMode(PINled, OUTPUT);
digitalWrite(PINled, HIGH);
pinMode(PINcalibrAI, INPUT);
//Измерение напряжения и тока (большого)
pinMode(PINtokSIG, INPUT);
//Измерение напряжения и тока (малого)
pinMode(PINmeasureVT, INPUT);
//Измерение частоты
attachInterrupt(0, impuls, RISING);
```

В блоке функций для определения начального состояния переменных логики SCADE происходит инициализация оператора SCADE, который содержит конечный автомат. Функция init() задает начальные значения всех переменных логики SCADE, а также определяет начальное состояние конечного автомата. Функция reset() сбрасывает все значения до начальных при старте программы.

Реализация блока функций для определения начального состояния переменных логики SCADE в тексте программы (Листинг 5):

```
//Функция init() задает начальные значения всех переменных логики SCADE, а также определяет начальное состояние конечного автомата.
Operator3_init(&outC);
Operator3_reset(&outC);
```

Блок измерения частоты вращения винта состоит из функции, которая измеряет время между оборотами винта и блока вычисления частоты вращения в Герцах.

Функция измерения времени реализована следующим способом: запускается внешнее прерывание по перепаду на выводе платы, к которому подключен датчик оборотов, и одновременно фиксируется системное время функцией micros(). Как только происходит второе такое прерывание, вычисляем разницу во времени между прерываниями и таким образом находим период вращения винта.

Текст программы, реализующей эту функцию измерения времени (Листинг 6):

```
volatile unsigned long ttime = 0;
 //Время срабатывания датчика
volatile unsigned long ttime_old = 0;
//Предыдущее время
volatile uint8_t flag = 0;
void impuls() {
if (flag!=1) ttime = micros()time_old;
time_old = micros();}
Измерив период (время), можно вычислить и частоту в герцах.
Этот блок реализует следующий код:
flag = 1; //чтобы ttime изменилось в процессе вывода
float f;
if (ttime != 0) f = 1000000 / float(ttime);
//вычисляем частоту сигнала в Герцах
flag = 0;
```

В блоке присвоения значений входным переменным оператора SCADE задаются значения переменных, которые использует оператор SCADE. А именно, значения, считанные с аналоговых выходов платы ArduinoNano и преобразованные АЦП, присваиваются переменным оператора SCADE для дальнейшей тарировки параметров тока, напряжения, тяги и оборотов.

Реализация блока присвоения значений входным переменным оператора SCADE в тексте программы (Листинг 7):

```
//Датчик напряжения. Чтение аналогового входа
PINmeasureVT = A7;
inC.sensorValue_V = analogRead(PINmeasureVT);
//Датчик тока до 30А. Чтение аналогового входа
PINtokSIG = A6;
inC.sensorValue_A = analogRead(PINtokSIG);
//Чтение данных с АЦП тензодатчика
inC.sensorValue_Tenzo = hx711.read() / 100.0;
//Задание частоты для перевода Гц в обороты в минуту
inC.sensorValue_Hz = f;
```

В блоке чтения данных происходит проверка COM-порта на наличие данных. Если в порт поступило значение количества ступеней циклограммы, это значение присваивается входной переменной оператора SCADEinC. Steps, которая используется для запуска конечного аппарата и проведения дальнейших испытаний.

Реализация блока чтения данных в тексте программы (Листинг 8):

```
if (Serial.available() > 0) {
int n = Serial.read(); //Считываем заданное количество ступеней циклограммы
inC.Steps = n; }
```

Если программа считала данные с COM-порта, алгоритм переходит к блоку выполнения испытаний. Процесс проведения испытаний, а именно, переходы по ступеням циклограммы и тарировка параметров тока, напряжения, тяги и оборотов описан в проекте SCADE.

Для выполнения оператора SCADE, используется следующий код (Листинг 9):

```
Operator3(&inC, &outC); //Выполнение Operator3 SCADE
Данные о параметрах испытаний выводятся в COM-порт на каждой ступени циклограммы:
//Каждые три секунды (одна ступень), переменная WriteToCom = True
if (outC.WriteToCom==true) {
n++;
//Пишем в порт значения переменных через пробел
Serial.print(inC.currentMillis/1000);
Serial.print(" ");
Serial.print(outC.OutputOborotVmin);
Serial.print(" ");
Serial.print(outC.OutputWeight);
Serial.print(" ");
Serial.print(outC.OutputVoltage);
Serial.print(" ");
Serial.print(outC.OutputAmper);
Serial.print(" ");
//Переход на новую строку
Serial.println();}
```

Для запуска двигателя необходимо подать на него импульс. Запуск двигателя происходит внутри оператора SCADE, для этого прописываем функцию для управления сервоприводом со значением переменной SCADE, которая отвечает за поданный на двигатель импульс: myservo.writeMicroseconds(outC.Impuls).

Реализация блока запуска двигателя в тексте программы (Листинг 10):

```
//Подаем импульс для работы сервопрвода
if (outC.Impuls == 0 || outC.Impuls == 1)
digitalWrite(PINled, cmd);
if (outC.Impuls >= 1000 && outC.Impuls <= 2000) myservo.writeMicroseconds(outC.Impuls);
delay(500); //пауза между командами для стабильности
```

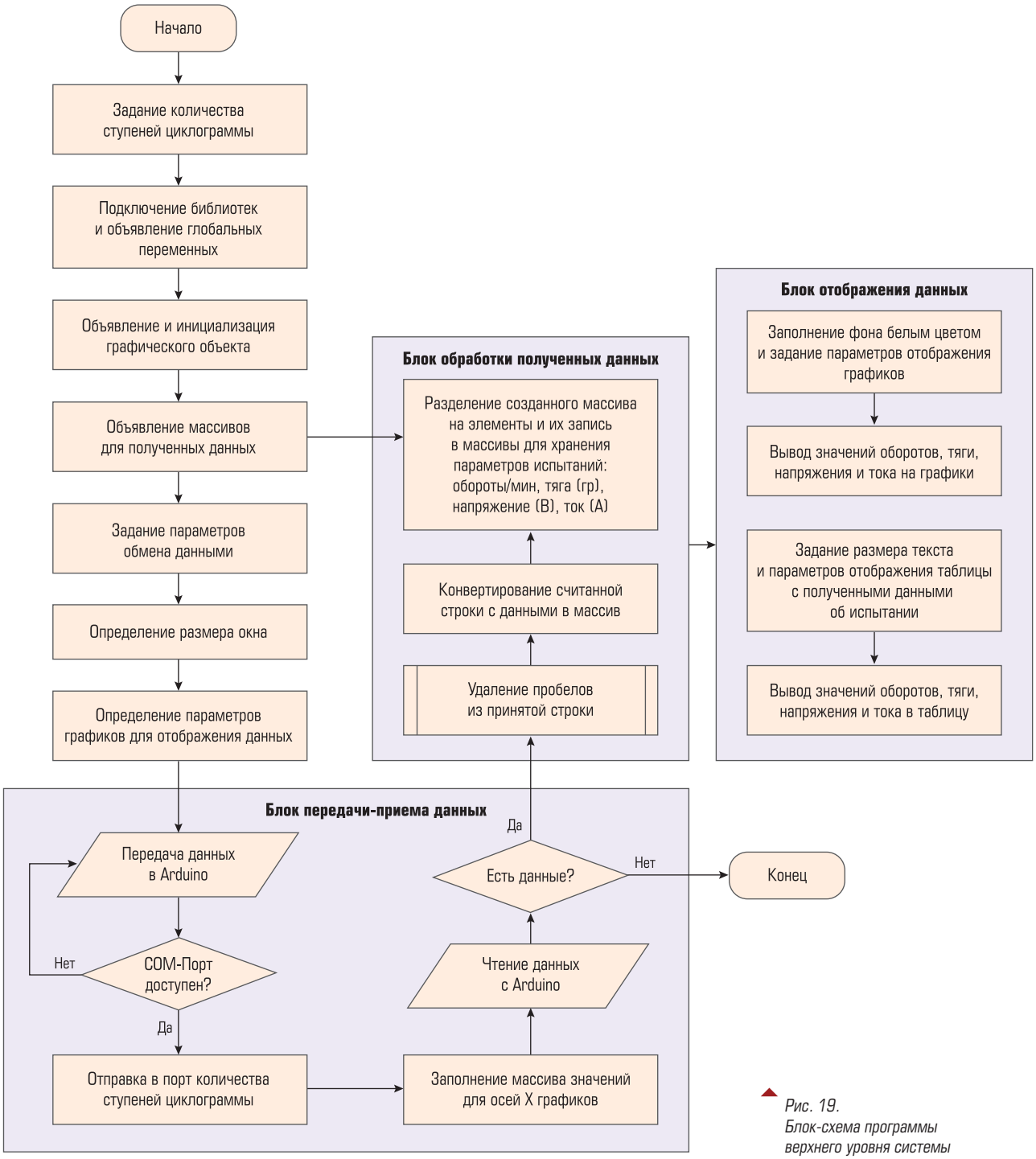


Рис. 19.
Блок-схема программы верхнего уровня системы

Разработка алгоритма и программы "верхнего уровня" системы. Блок-схема алгоритма "верхнего уровня" системы

ПО "верхнего" уровня системы работает согласно блок схеме, приведенной на рис. 19.

В блоке задания количества ступеней циклограммы задается количество ступеней циклограммы для испытания винтомоторной группы беспилотного летательного аппарата. Количество ступеней, задающее циклограмму испытания, определяет, сколько различных импульсов (от минимума до максимума) будет подано на двигатель. От количества ступеней циклограммы зависит количество циклов приема-передачи данных об испытании, которые совершит система, а, следовательно, количество отображенных

значений параметров и точность их зависимости от времени. Чем больше значение количества ступеней, тем больше значений параметров будет на графике. Чем больше точек показано на графике, тем проще оператору оценить динамику изменения того или иного параметра в зависимости от режима работы двигателя.

Реализация блока задания количества ступеней циклограммы в тексте программы (Листинг 11):

```
int n=20; //Задание количества ступеней
циклограммы
```

В блоке подключения библиотек и объявления переменных была подключена библиотека для передачи и приема данных от Arduino через COM-порт. Для работы с данными был создан объект последовательного порта и задана строка для приема данных.

Также была объявлена переменная, служащая для определения цикла работы системы, т.е. номера ступени циклограммы в процессе испытания. Отсчет ступеней начинается с нуля.

Для выполнения данного блока подключения библиотек и объявления переменных программы, используется код (Листинг 12):

```
//Библиотека для передачи и приема данных через
COM-порт
import processing.serial.*;
Serial serial; //Объект последовательного порта
String received; //Строка для данных, считываемых
с последовательного порта
int k=-1; //Номер ступеньки циклограммы
```

В блоке объявления и инициализации графического объекта происходит объявление и инициализация объекта, который используется в программе для рисовки графиков, на которых отображаются значения параметров испытаний винтомоторной группы БПЛА.

Для создания графиков применяется класс Graph, в котором прописаны все параметры графика, например, его размер и положение в окне вывода информации об испытании, цвет линий и фона, максимальные и минимальные значения по осям *X* и *Y*, параметры сглаживания графика.

В данном блоке задаются имена объектов, для графиков и определяются их цвет и положение в окне вывода информации.

Реализация блока объявления и инициализации графического объекта в тексте программы (Листинг 13):

```
//Graph(int x, int y, int w, int h,color k)
Graph MyFirstGraph = new Graph(150,
80,400,200,color (200,20,20));
Graph MySecondGraph = new Graph(150,
410,400,200,color (20,20,200));
Graph MyThirdGraph = new Graph(750,
80,400,200,color (200,20,20));
Graph MyFourthGraph = new Graph(750,
410,400,200,color (20,20,200));
```

Так как класс Graph, применяемый для создания графиков, использует для их построения массивы данных, необходимо, чтобы значения параметров испытания были записаны в соответствующие массивы. Для блока объявления массивов были объявлены массивы вещественного типа данных для записи каждого параметра: оборотов двигателя, тяги, напряжения и тока.

Реализация блока объявления массивов в тексте программы (Листинг 14):

```
//Массивы для параметров испытаний
float Array1 = new float[n+2];
float Array2 = new float[n];
float ArrayTest1 = new float[n+2]; //обороты
float ArrayTest2 = new float[n+2]; //тяга
float ArrayTest3 = new float[n+2]; //напряжение
float ArrayTest4 = new float[n+2]; //ток
```

В блоке задания параметров обмена данными выбирается COM-порт, к которому подключена плата Arduino, и задаются параметры связи с этим портом (Листинг 15):

```
//Выбираем из списка порт, к которому подключе-
на ArduinoNano
Stringport = Serial.list()[2];
//Задаем параметры связи с COM-портом
serial = new Serial(this, port, 9600);
```

Размер окна для отображения информации об испытании зависит от размера монитора рабочего места, на котором проводятся испытания, а также от количества отображаемых параметров. Размер окна был выбран с учетом всех элементов, которые будут отображены (Листинг 16):

```
//размер окна
size(1250, 900);
```

В блоке определения параметров графиков задаются название графиков, подписи осей *X* и *Y*, а также минимальное значение *X* и *Y*.

Реализация блока определения параметров графиков в тексте программы (Листинг 17):

```
MyFirstGraph.xLabel=" T, сек ";
MyFirstGraph.yLabel=" Обороты/мин ";
MyFirstGraph.Title=" Обороты двигателя ";
MyFirstGraph.yMin=0;
MyFirstGraph.xMin=0;
MySecondGraph.xLabel=" T, сек ";
MySecondGraph.yLabel="Тяга, граммы";
MySecondGraph.Title=" Тяга ";
MySecondGraph.yMin=0;
MyThirdGraph.xLabel=" T, сек ";
MyThirdGraph.yLabel="Напряжение, В";
MyThirdGraph.Title=" Напряжение ";
MyThirdGraph.yMin=0;
MyFourthGraph.xLabel=" T, сек ";
MyFourthGraph.yLabel="Ток, А";
MyFourthGraph.Title=" Ток ";
MyFourthGraph.yMin=0;
```

В блоке передачи-приема данных происходит обмен данными с платой Arduino Nano через COM-порт.

Если порт доступен, отправляем в него переменную n, заданную ранее, которая определяет количество ступеней циклограммы. Когда плата Arduino считает входящие данные, запустится процесс испытания, двигатель начнет работать, а в COM-порт будут поступать данные о параметрах работы стенда испытаний винтомоторной группы.

После отправки количества ступеней циклограммы, программа проверяет порт на наличие ответных данных. Если они есть, алгоритм переходит к их обработке и отображению.

Также в данном блоке происходит заполнение массива для оси *X*. В каждом из 4 графиков для параметров оборотов, тяги, напряжения и тока, по оси *X* будем указывать время. В процессе испытания двигатель работает в каждом режиме в течение 3-х секунд (3000 миллисекунд), соответственно, данные об оборотах, тяги, напряжении и токе также меняются каждые 3 секунды. Следовательно, для удобства построения графиков, по оси *X* укажем время, прошедшее от начала испытаний.

Реализация блока передачи-приема данных в тексте программы (Листинг18):

```
if (serial.available() > 0) {
//Отправляем в порт значение n типа char
serial.write((char) n);
//Заполняем массив 1 для оси X
for(int i = 0; i < Array1.length; i++) {Array1[i]
= i*3;} //Секунды
//Если есть данные,
//считываем их и записываем в переменную
received
received = serial.readStringUntil( \n);
```

После отправки количества ступеней циклограммы в COM-порт начнется испытание винтомоторной группы, и в COM-порт будут поступать данные от Arduino.

Программа считывает данные в виде строки, которая содержит записанные через пробел переменные, равные значениям параметров оборотов двигателя, тяги, тока и напряжения. Чтобы выделить эти переменные из строки, программа конвертирует ее в массив. Важно знать, что индекс последнего элемента в массиве – на единицу меньше, чем размер массива.

Далее, полученный массив разделяем на элементы и записываем их в отдельные массивы, созданные для каждого параметра. Например, на нулевой ступени циклограммы происходит заполнение нулевых элементов массивов параметров, на первой ступени – первых, на второй – второй и т.д.

Таким образом, в конце испытаний получаем 4 массива, каждый из которых содержит последовательно записанные значения об определенном параметре.

Реализация блока обработки данных в тексте программы (Листинг 19):

```
//Если данные не пустые
if (received != null) {
//Прибавляем 1 к номеру ступеньки циклограммы
k++;
//Убираем любые пробелы из считанной строки
trim(received);
//Конвертируем данные в массив float
float ArrayCOM = float(split(received, ));
//Записываем полученный массив в массив Array2
for(int i = 0; i < ArrayCOM.length-1; i++) {
Array2[i] = ArrayCOM[i];
//Для каждой ступеньки циклограммы делим
входящий массив данных
```

```
//на 4 массива для 4 параметров: обороты, тяга, напряжение и ток
if (k<n+2) {
//Нулевой элемент массива – время в микросекундах от запуска циклограммы
ArrayTest1[k] = Array2[1];
//Значения оборотов/мин
ArrayTest2[k] = Array2[2]; //Тяги
ArrayTest3[k] = Array2[3]; //Напряжения
ArrayTest4[k] = Array2[4]; //Тока
}
}
//Выводим в консоль строку с принятыми данными
println(received);
```

В блоке отображения данных выводятся на экран полученные в ходе испытания данные. Сначала необходимо заполнить фон экрана – белым цветом, как самым нейтральным.

Далее указываются основные параметры отображения данных. Для каждого графика определяется максимальное значение по оси *Y*. Для удобства построения графика приравняем максимальное значение по оси *Y* к максимально возможному значению отображаемого параметра. Тогда график будет выглядеть аккуратно и не выйдет за пределы оси.

Для построения каждого графика указываются массивы данных, определяющие координаты точек по осям *X* и *Y*. По оси *X* на всех графиках отображается время от начала испытания, следовательно, массив, содержащий значения времени; по оси *Y* – параметр работы двигателя: обороты в минуту, тяга, напряжение или ток, а значит, массив со значениями одного и этих параметров.

На каждой ступени испытания программа будет выводить на экран графики со значениями параметров, и рисовать их по мере проведения испытания в режиме реального времени.

Система предусматривает отображение данных не только в виде графиков, но и в виде таблицы со значениями считанных параметров. Для создания таблицы указываем размер текста, заголовки для строк и координаты их положения на экране. Каждая строка будет заполняться значениями в процессе испытания в режиме реального времени.

Данный блок программы отображения данных реализован в следующем коде (Листинг 20):

```
background(255); //Фон белый
//График 1 – Обороты двигателя
MyFirstGraph.yMax=int (max(ArrayTest1));
//Максимальное значение шкалы по оси Y
MyFourthGraph.xMax=int (max(Array1));
//Максимальное значение шкалы по оси X
MyFirstGraph.DrawAxis();
MyFirstGraph.LineGraph(Array1,ArrayTest1);
MyFirstGraph.GraphColor=color(40,40,200);
//График 2 – Тяга
MySecondGraph.yMax=int (max(ArrayTest2));
MyFourthGraph.xMax=int (max(Array1));
MySecondGraph.DrawAxis();
MySecondGraph.LineGraph(Array1,ArrayTest2);
//График 3 – Напряжение
MyThirdGraph.yMax=14;
MyFourthGraph.xMax=int (max(Array1));
MyThirdGraph.DrawAxis();
MyThirdGraph.LineGraph(Array1,ArrayTest3);
//График 4 – Ток
MyFourthGraph.xMax=int (max(Array1));
MyFourthGraph.DrawAxis();
MyFourthGraph.LineGraph(Array1,ArrayTest4);
//Рисовка таблицы со значением параметров
textSize(12); //Размер текста – 12
//Задаем название строк и их положение в окне
text("Ступень", 100,720);
text("Обороты/мин", 100,750);
text("Тяга, Гр", 100,780);
text("Напряжение, В", 100,810);
text("Ток, А", 100,840);
//Для каждой ступени записываем считанные данные в таблицу
for (int i=1; i<n+2; i++) {
int Otstup=100+i*50; //Отступ для каждого следующего значения
text(i, Otstup, 720); //Номер ступеньки циклограммы
text(int(ArrayTest1[i]), Otstup,750); //Обороты в минуту типа int
text(int(ArrayTest2[i]), Otstup,780); //Тяга типа int
text(ArrayTest3[i], Otstup,810); //Напряжение
text(ArrayTest4[i], Otstup,840); //Ток
}
```

ПРОВЕДЕНИЕ МОДЕЛИРОВАНИЯ И ИСПЫТАНИЙ СИСТЕМЫ

После подключения платы ArduinoNano к компьютеру и компиляции скетча, загружаем его в плату.

В скетче Processing устанавливаем количество ступеней циклограммы и запускаем скетч. После запуска скетча в COM-порт будет отправлено заданное значение ступеней, запустится двигатель и в COM-порт начнут поступать данные об испытании.

В окне Processing в режиме реального времени будут строиться графики и составляться таблица по полученным с датчиков параметрам.

На рис. 20, 21, 22 представлены данные об испытании двигателя по циклограмме в 21-х разных режимах работы.

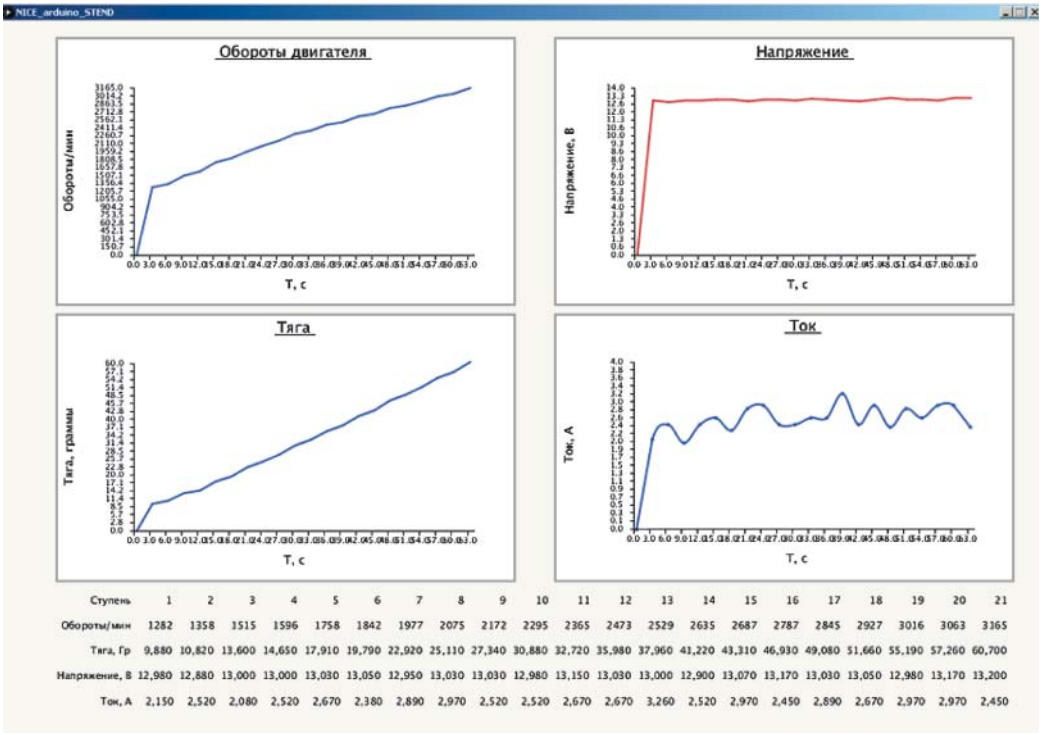


Рис. 20. Система автоматизированного управления стендом испытаний винтомоторной группы БПЛА

Ступень	1	2	3	4	5	6
Обороты/мин	1282	1358	1515	1596	1758	1842
Тяга, Гр	9,880	10,820	13,600	14,650	17,910	19,790
Напряжение, В	12,980	12,880	13,000	13,000	13,030	13,050
Ток, А	3,330	2,810	3,260	2,660	3,030	2,660

Рис. 21. Таблица с данными об испытании на каждой ступени циклограммы

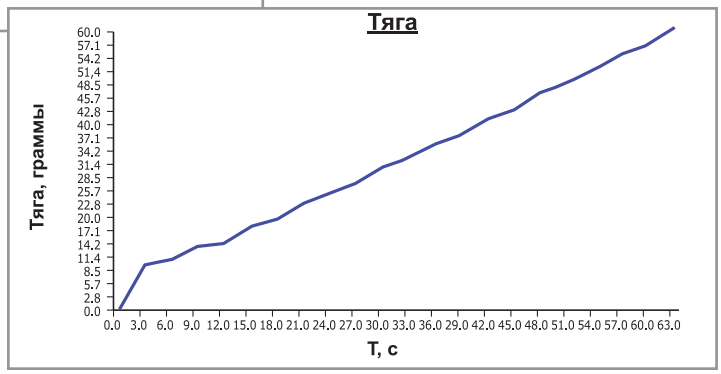


Рис. 22. Графики изменения тяги и оборотов двигателя в процессе испытания

Система автоматизированного управления предоставляет графические данные по следующим параметрам: обороты двигателя; тяга; напряжение; потребляемый ток.

Для проектировщика, выполняющего испытание, доступны данные в виде графиков – для наблюдения зависимости параметров от времени, и в виде таблицы – для определения зависимости параметров друг от друга [5].

Процесс испытания полностью автоматизирован и занимает небольшое количество времени – испытание по циклограмме на 20-ти режимах будет длиться в течение минуты. За это короткое время проектировщик получит все необходимые данные для выбора винтомоторной группы для создаваемого летательного аппарата.

ВЫВОДЫ

В данной статье представлены результаты разработаного программно-алгоритмического обеспечения системы автоматизированного управления для стенда испытаний винтомоторной группы квадрокоптера.

Алгоритм проведения испытания разработан с помощью передовых технологий SCADE, широко применяемых в авиационной промышленности в наши дни. Алгоритм проведения испытания винтомоторной группы, промоделированный в среде SCADE Suite, отвечает нормам авиационных стандартов.

Разработанное программно-алгоритмическое обеспечение позволяет проводить автоматизированные испытания винтомоторных групп БПЛА в режиме реального времени. Система позволяет пользователю задавать циклограмму на испытание и предоставляет данные о его проведении в виде таблиц и графиков со значениями оборотов двигателя, его тяги, напряжения и потребляемого тока.

Список литературы

1. *Егоров А.А.* “Проектирование комплексов аппаратуры систем управления испытаниями сложных объектов” // Автоматизация и IT в энергетике. Часть 2. 2020 г., № 6(131), стр. 38-47.
2. *Егоров А.А.* “Исследование и разработка измерительно-информационного и управляющего комплекса для полунатурного моделирования полета летательного аппарата” // Прикладная физика и математика”. 2018 г., № 10, стр. 60-63.

Егоров Александр Александрович *– канд. техн. наук, профессор АВН РФ, генеральный директор ООО “АВИАТЭКС”,*
Сурков Дмитрий Александрович *– канд. техн. наук, доцент технический директор ООО “АВИАТЭКС”,*
Кирпичев Константин Юрьевич *– ведущий инженер отдела автоматизации эксперимента (Институт систем управления, информатики и электроэнергетики МАИ).*