**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра САПР**

**ОТЧЕТ**

**по курсовой работе**

**по дисциплине «Проектирование цифровых устройств»**

**Тема: Проектирование и реализация "робота-диспесора"**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 9302 |  | Тарабурин А.П. |
|  |  | Кнауб К.В. |
|  |  | Ширнин К.В. |
|  |  | Давтян С.Д. |
| Преподаватель |  | Рыбин В.Г. |

Санкт-Петербург

2022

**АННОТАЦИЯ**

Данная работа включает проектирование и реализацию «робота-диспенсора». В работе были использованы методы проектирование корпусов и схем, а также написан программный код для цифрового устройства. Кроме того, были доработаны и реализованы различные детали по улучшению работы данного робота.

В итоге корпус цифрового устройства был напечатан на 3D принтере, запрограммирован микроконтроллер, распаяна схема и собран робот.

**ANNOTATION**

This work includes the design and implementation of a "dispenser robot". The methods of designing cases and circuits were used in the work, and the program code for a digital device was written. In addition, various details have been finalized and implemented to improve the operation of this robot.

As a result, the body of the digital device was printed on a 3D printer, the microcontroller was programmed, the circuit was soldered, and the robot was assembled.

**ОГЛАВЛЕНИЕ**

[Цель работы 4](#_Toc104844942)

[Исходное задание 4](#_Toc104844943)

[Ход выполнения работы 4](#_Toc104844944)

[Принцип работы цифрового устройства 4](#_Toc104844945)

[Описание схемы подключения 8](#_Toc104844946)

[Описание алгоритма работы устройства 9](#_Toc104844947)

[Блок-схема алгоритма 14](#_Toc104844948)

[ВЫВОД 15](#_Toc104844949)

[СПИСОК ЛИТЕРАТУРЫ 16](#_Toc104844950)

[ПРИЛОЖЕНИЕ 17](#_Toc104844951)

[Код программы 17](#_Toc104844952)

# Цель работы

Изучить методы проектирование корпусов и схем, и научиться писать программный код для цифровых устройств

# Исходное задание

Необходимо реализовать цифровое устройство «робот-диспенсора», который будет разливать жидкость по сосудам, поставленным на обозначенные слоты со световой индикацией статусов. В роботе должны быть обозначены несколько режимов работы: автоматический, при котором разлив происходит без подтверждения пользователя; ручной, в котором разлив происходит только по нажатию кнопки, в выставленные рюмки; режим сервиса для отладки. Обеспечить пользователя информацией, которая будет водиться на OLED экран.

# Ход выполнения работы

# Принцип работы цифрового устройства

Робот-диспенсор (Рисунок 1) состоит из 4 частей. Отдельно были спроектированы корпус, крышка, стойка и краник. Каждая часть имеет свои особенности. Робот был собран с помощью соединения этих деталей.

Корпус содержит в себе множество отверстий. Под каждым слотом для рюмки есть по два отверстия, которые предназначены для адресного светодиода и концевика. Также отдельно есть отверстие прямоугольной формы для экрана и два круглых отверстия для энкодера и кнопки. Вдоль границ корпуса есть множество отверстий, которые предназначены для крепления крышки. Также есть два смежных отверстия: одно из них для стойки, другое – для прохождения сервопривода и силиконовой трубки. Силиконовая трубка через данное отверстие будет проходить к кранику (Рисунок 2).

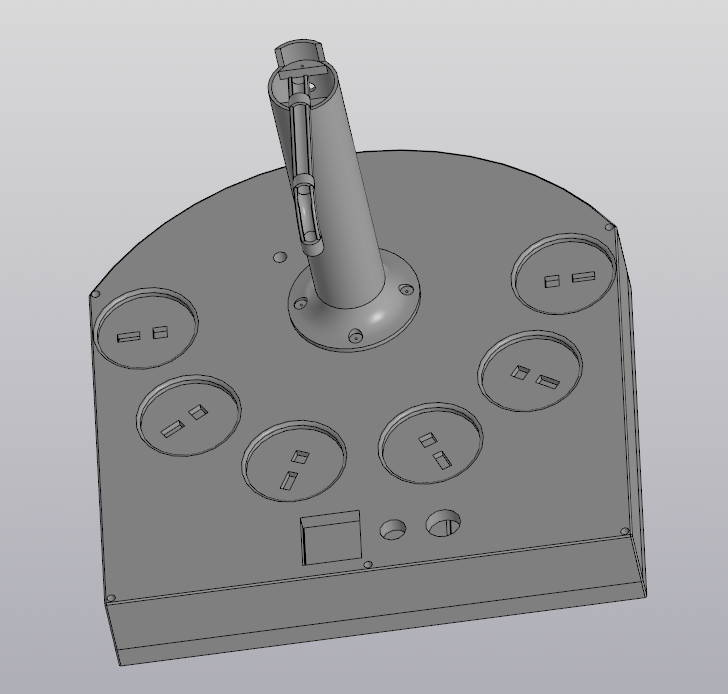


Рисунок 1. Робот-диспесор

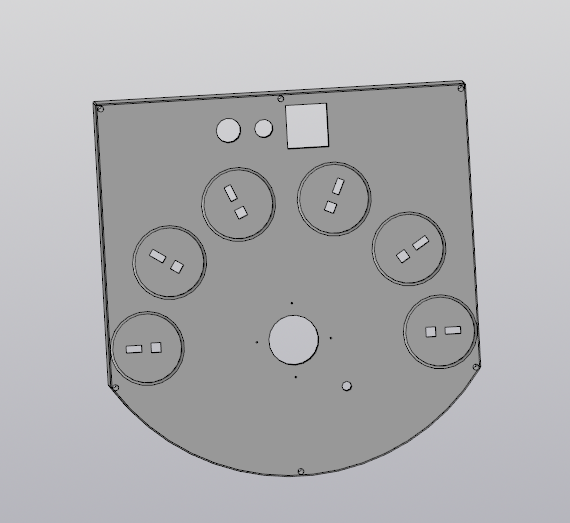


Рисунок 2. Корпус робота-диспесора

Стойка прикрепляется к корпусу. Для этого на ней предусмотрены четыре отверстия. Сама по себе она представляет из себя цилиндрический объект, через который проходит силиконовая трубка. Сверху есть крепление для сервопривода (Рисунок 3).

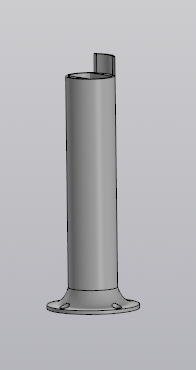


Рисунок 3. Стойка

Под корпусом находится крышка. Она крепится к корпусу с помощью саморезов. Для этого вдоль крышки сделаны отверстия. Также есть специальное крепление для водяной помпы (Рисунок 4).

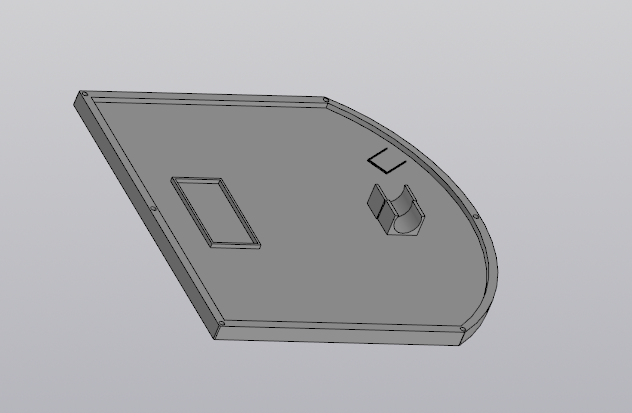


Рисунок 4. Крышка

Последняя часть робота-диспесора – краник. Он крепится к стойке. Для этого на нем сделаны отверстия. Через него будет поступать жидкость в рюмки (Рисунок 5).

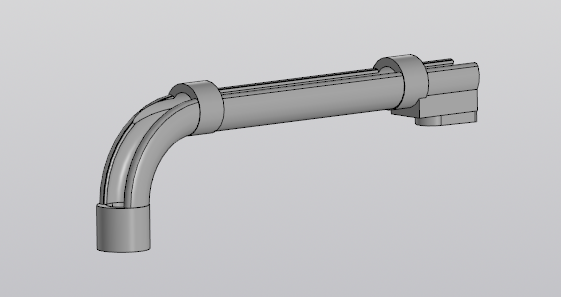


Рисунок 5. Краник

Описание схемы подключения

Схема подключения представлена ниже (Рисунок 7)

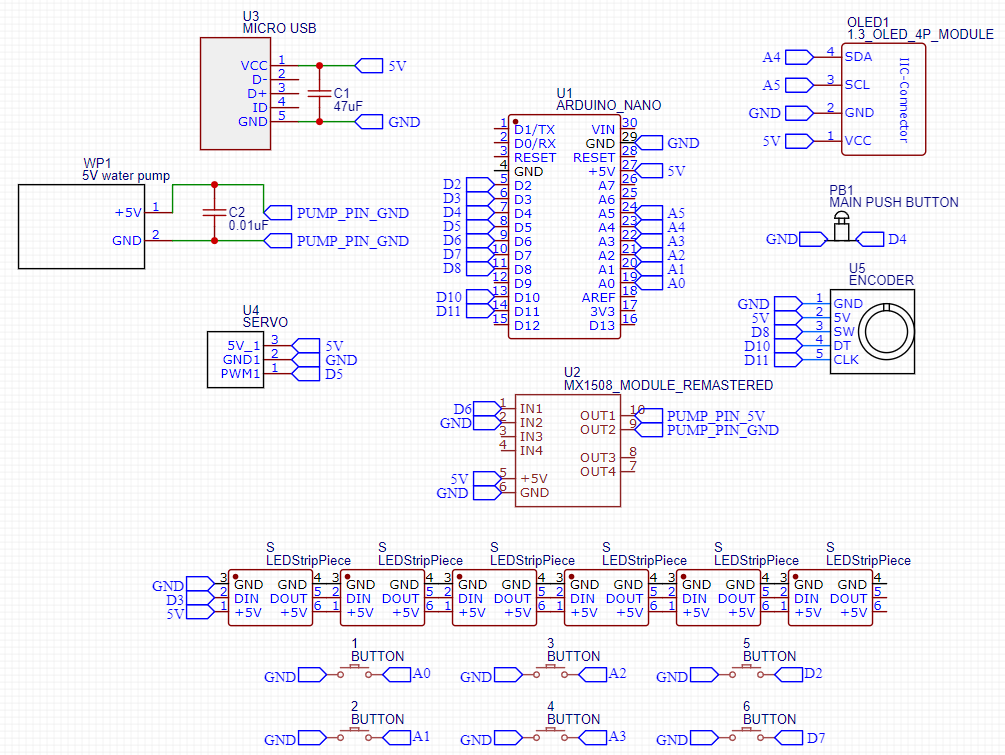


Рисунок 7

# Описание алгоритма работы устройства

**Class Timer**

Полями данного класса являются:

Интервал m\_interval, хранящий значение, после которого можно перевести таймер в другое состояние;

Таймер m\_timer показывает время начала работы в миллисекундах;

Состояние m\_status отображает рабочее состояние таймера – true/false

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| Timer(uint32\_t) | Конструктор от значения интервала |
| setInterval(uint32\_t) | Установка значения интервала с проверкой на ненулевой аргумент |
| start() | Начало работы таймера, фиксирование текущего таймера и перевод состояния в рабочее |
| stop() | Завершение работы таймера, перевод в нерабочее состояние |
| boolean isReady() | Проверка готовности обрабатывания внешней функции с имеющимся интервалом и смещение существующего таймера к текущему значению времени, в диапазоне таймера |
| reset() | Обновление значения таймера до актуального |
| getState() | Получение состояния работы таймера |

**Class LedStrip**

Полем класса ленты адресных светодиодов является массив leds класса CRGB, хранящий цвет подсветки имеющихся светодиодов

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| LedStrip () | Добавляем в массив LED\_COUNT позиций светодиодов, выставляем яркость и изначальный цвет |
| ledOn( uint8\_t pin, uint8\_t r, uint8\_t g, uint8\_t b) | Выставляем цвет подсветки светодиоду в положении pin из цветовой модели RGB, параметры которой получаем из аргументов |
| ledOff( uint8\_t) | Выставляем черный цвет подсветки светодиода в заданном положении, тем самым гася его |

**Class Pump**

Полями класса помпы являются:

Таймер первоначальной задержки startDelayTimer, хранящий время задержки перед работой помпы;

Таймер окончательной задержки finishDelayTimer, хранящий время задержки после работы помпы;

Таймер окончания работы помпы finishTimer, показывающий время прекращения работы;

Состояние m\_state, отображающее текущее рабочее состояние помпы BUSY/READY

Идентификатор пина m\_pompPin, показывающий куда присоединено питание помпы

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| Pump(int) | Конструктор класса помпы от пина питания аппарата |
| pumpStart(uint32\_t) | Старт работы помпы, перевод состояния в рабочее, установка заданного интервала для таймера окончания работы, запуск таймера первоначальной задержки |
| pumpStartWithoutTimer() | Начало работы помпы и перевод в рабочее состояние и запуск таймера первоначальной задержки |
| pumpStop( bool) | Остановка помпы, запуск таймера окончательной задержки и остановка остальных. Перевод в ожидающее состояние |
| getState() | Получение состояния работы помпы |
| isPouring() | Проверка накачивания жидкости в текущий момент при помощи проверки состояния таймера первоначальной задержки |
| isFinishedPoured () | Проверка завершения накачивания жидкости в текущий момент при помощи проверки состояния таймера окончания работы |
| pumpCheck() | Алгоритм приведения в работу помпы в зависимости от состояния одного из таймеров:  При готовности таймера первоначальной задержки начинается накачивание жидкости  При готовности таймера окончания работы останавливаем накачивание  При готовности таймера окончательной задержки переводим помпу в режим ожидания |

**Class ServoCrane**

Полями класса сервокраника являются:

Указатель на класс сервопривода Servo, хранящий методы работы с сервоприводами;

Значение m\_currentAngle, хранящее текущий угол поворота краника относительно начальной позиции

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| ServoCrane (uint8\_t) | Конструктор класса cервокраника с установкой пина питания сервопривода |
| rotate(uint8\_t) | Поворот сервокраника на заданный угол с проверкой углов, выходящих за плоскость работы сервопривода |
| getCurrentAngle() | Получение текущего угла поворота сервокраника |

**Class Display**

Поле класса дисплея является указатель на объект класса Adafruit\_SSD1306, являющегося библиотечным классом для работы с нашим дисплеем.

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| Display () | Конструктор класса дисплея с очисткой, переводом курсора и заданием цвета шрифта. Выводит первоначальное приветствие |
| printRandToast() | Вывод случайного тоста из базы данных файла mem.hpp |
| printInfo(uint16\_t volume, uint8\_t mode) | Вывод количества наливаемого объема жидкости volume и текущего режима работы Робота-Наливайки mode |

**Class Slot**

Полями класса слота для тары на корпусе робота являются:

Таймер работы кнопки подтверждения btnTimer, хранящий данные для ее работы;

Состояние кнопки btnState, обеспечивающее возможность ее нажатия;

Объекты ранее описанных классов сервокраника m\_servo, помпы p\_pump и ленты светодиодов p\_ledStrip;

Номер пина концевика m\_btnPin для обеспечения его питания;

Идентификатор слота m\_slotID для передачи данных о нём вглубь алгоритма;

Значение угла поворота сервокраника к слоту m\_slotAngle для поворота сервоприводом;

Состояние взаимодействия с тарой, размещенной на данном слоте, m\_state - NO\_GLASS, EMPTY, NEXT, PROCESS, READY;

Значение объема тары p\_glassVolume, для регулирования процесса работы помпы.

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| Slot( LedStrip \* ledStrip, Pump \* pump, ServoCrane \* servo, uint8\_t btnPin, uint8\_t slotID, uint8\_t slotAngle, uint16\_t \* glassVolume ) | Конструктор класса слота с присвоением светодиода, помпы, сервокраника, установкой заданного номера пина концевика, идентификатора слота, угла поворота и объема тары. Состояние m\_state изначально назначается NO\_GLASS. |
| slotFirstCheck() | Первая проверка нажатия концевика слота. Включаем красный светодиод если рюмку только что поставили. Выключаем светодиод, если рюмку убрали. Меняем состояния. |
| slotSecondCheck() | Вторая проверка наливания стопки. Если статус слота PROCESS, ждем завершения таймера и перекрашиваем светодиод в зеленый. Меняем состояние на READY. |
| slotThirdCheck() | Третья проверка пустоты текущей рюмки. Если статус слота EMPTY поворачиваем сервокраник, в этот момент красим светодиод в синий, и начинаем наливать, попутно устанавливаем желтый цвет светодиоду. Меняем состояние на PROCESS. |
| getState() | Получение состояния слота. |

**Class Brain**

Полями класса Brain, отвечающего за совмещение классов всех физических объектов и выполнение логики робота, являются:

Состояние робота m\_mode, хранящее текущий режим работы робота MANUAL - ручной, AUTO - автоматический, SERVICE – сервисный;

Объем тары m\_glassVolume, хранящий значение в миллилитрах;

Состояние m\_status, показывающее текущую занятость робота BUSY, READY;

Логическая переменная m\_toastFlag, отвечающее за возможность отображения тоста;

Логическая переменная isStarted, хранящий информацию о работе нового цикла робота;

Таймер начала работы p\_starterTimer, отмеряющий время от запуска робота;

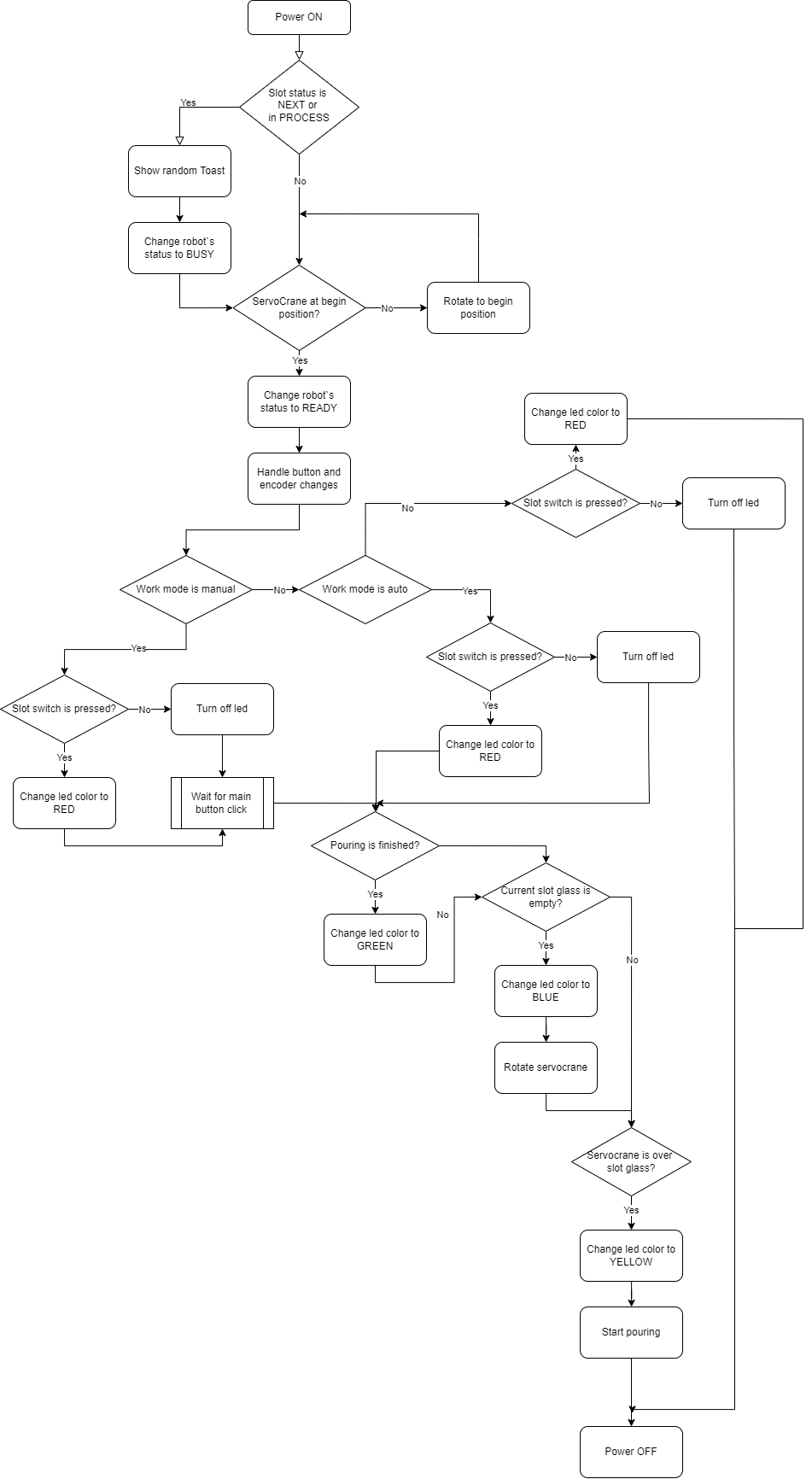
Значение текущего времени p\_serviceTime, применяющееся для обработки нажатия сервисной кнопки;

Объекты ранее описанных классов сервокраника p\_servo, помпы p\_pump, ленты светодиодов p\_ledStrip, дисплея p\_display и массива слотов p\_slots;

Указатели на библиотечные классы энкодера p\_encoder и кнопки p\_button;

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| Brain() | Конструктор по умолчанию с установкой ручного режима, объема тары 20 мл и всех прочих пинов питания |
| tick() | Срабатывает в loop .ino файла, здесь принимается решение о состоянии каждого слота, повороте сервокраника и отслеживании режима работы робота |
| changeAutoManualMode() | Изменение пользовательских режимов в зависимости от выбора |
| changeServiceMode() | Установка сервисного режима |
| incGlassVolume() | Увеличение объема тары от движения энкодера |
| decGlassVolume() | Уменьшение объема тары от движения энкодера |
| mainBtnPressed() | Обработчик длительного зажатия главной кнопки |
| mainBtnReleased() | Обработчик отпускания главной кнопки |
| mainBtnClick() | Обработчик нажатия главной кнопки |
| encClicked() | Обработчик нажатия энкодера, вызывающий отображение информации на главном экране |

# Блок-схема алгоритма



# ВЫВОД

В результате курсовой работы мы изучили методы проектирование корпусов и схем, и научились писать программный код для цифровых устройств**.**

А также мы реализовали цифровое устройство «робот-диспесора», который разливает жидкость в сосуды, поставленным на обозначенные слоты со световой индикацией статусов: красный, когда сосуд поставлен; синий, когда кран находиться над сосудом и скоро будет его наполнять; желтый, когда в сосуд наливается жидкость; зеленый, когда сосуд наполнили. В роботе обозначены несколько режимов работы: автоматический, при котором разлив происходит без подтверждения пользователя; ручной, в котором разлив происходит только по нажатию кнопки, в выставленные рюмки; режим сервиса для отладки. Еще мы обеспечили пользователя информацией, которая будет водиться на OLED экран, она включает в себя режим, угол движения крана, и тосты.

# СПИСОК ЛИТЕРАТУРЫ

1. Подбельский В. В. Язык C++ / В. В. Подбельский. — 3-е издание. — Москва : Финансы и статистика, 1998. — 560 c.
2. Макс Шлее Qt 5.10. Профессиональное программирование на C++ / Шлее Макс. — 1-е издание. — СПб : БХВ-Петербург, 2020. — 1072 c.
3. Герасимов, А. А. Самоучитель КОМПАС-3D V19 / А. А. Герасимов. — СПб : BHV, 2021. — 624 c. — Текст : непосредственный.

# ПРИЛОЖЕНИЕ

# Код программы

Timer.hpp

1. #ifndef TIMER\_H
2. #define TIMER\_H
4. */\**
5. ***\* Класс таймера***
6. *\* Использован улучшенный алгоритм таймера на millis*
7. *\* Обеспечивает кратные интервалы и защиту от пропусков и переполнений*
8. *\**
9. *\*/*
10. **class Timer**
11. {
12. public:
13. Timer( uint32\_t interval ); //!< Конструктор
14. void setInterval( uint32\_t interval ); //!< Выставляет интервал срабатывания таймера
15. **boolean isReady(); //!< Возвращает true, когда пришло время**
16. void reset(); //!< Ручной сброс таймера
17. void stop(); //!< Остановка таймера
18. void start(); //!< Запуск таймера
19. bool getState();
21. private:
22. uint32\_t m\_timer = 0;
23. uint32\_t m\_interval = 0;
24. bool m\_status = true;
25. **};**
27. Timer::Timer(uint32\_t interval)
28. {
29. m\_interval = interval;
30. **m\_timer = millis();**
31. }
33. void Timer::setInterval(uint32\_t interval)
34. {
35. **m\_interval = (interval == 0) ? 10 : interval;**
36. }
38. void Timer::start()
39. {
40. **m\_status = true;**
41. m\_timer = millis();
42. }
44. void Timer::stop()
45. **{**
46. m\_status = false;
47. }
49. boolean Timer::isReady()
50. **{**
51. uint32\_t thisMls = millis();
52. if (m\_status && thisMls - m\_timer >= m\_interval)
53. {
54. do
55. **{**
56. m\_timer += m\_interval;
57. if (m\_timer < m\_interval) break;
58. } while (m\_timer < thisMls - m\_interval);
59. return true;
60. **}**
61. else
62. {
63. return false;
64. }
65. **}**
67. void Timer::reset() {
68. m\_timer = millis();
69. }
71. bool Timer::getState()
72. {
73. return m\_status;
74. }
76. #endif

ledStrip.hpp

1. #ifndef LED\_STRIP\_H
2. #define LED\_STRIP\_H
4. #include "settings.hpp"
5. **#include <FastLED.h>**
7. */\**
8. *\* Класс ленты адресных светодиодов*
9. *\**
10. ***\*/***

13. class LedStrip
14. {
15. **public:**
16. LedStrip(); //!< Конструктор
17. void ledOn( uint8\_t pin, uint8\_t r, uint8\_t g, uint8\_t b ); //!< Включить светодиод. pin - позиция светодиода. r/g/b - цвет (0-255)
18. void ledOff( uint8\_t pin ); //!< Выключить светодиод. pin - позиция светодиода
20. **private:**
21. CRGB leds[LED\_COUNT];
22. };
24. LedStrip::LedStrip()
25. **{**
26. FastLED.addLeds <WS2812, LED\_PIN, GRB>(leds, LED\_COUNT).setCorrection(TypicalLEDStrip);
27. FastLED.setBrightness( BRIGHTNESS );
28. for( int i = 0 ; i < LED\_COUNT; ++i )
29. leds[i] = CRGB(0, 0, 0);
30. **FastLED.show();**
31. Serial.println(F("LedStrip init"));
32. }
34. void LedStrip::ledOn( uint8\_t pin, uint8\_t r, uint8\_t g, uint8\_t b )
35. **{**
36. //Serial.print( "Led ON (pin / R;G;B): " );
37. //Serial.print( pin );
38. //Serial.print( " / " );
39. //Serial.print( r );
40. **//Serial.print( ";" );**
41. //Serial.print( g );
42. //Serial.print( ";" );
43. //Serial.println( b );
45. **leds[pin] = CRGB(r, g, b);**
46. FastLED.show();
47. }
49. void LedStrip::ledOff( uint8\_t pin )
50. **{**
51. //Serial.print( "Led OFF (pin): " );
52. //Serial.println( pin );
53. leds[pin] = CRGB(0, 0, 0);
54. FastLED.show();
55. **}**
57. #endif

Pump.hpp

1. #ifndef PUMP\_H
2. #define PUMP\_H
4. #include "settings.hpp"
5. **#include "timer.hpp"**
7. */\**
8. *\* class Pump - Класс помпы*
9. *\*/*
11. class Pump
12. {
13. public:
14. Pump( int pin ); //!< Конструктор. pin - Пин питания на помпу
15. **void pumpStart( uint32\_t milli = DEFAULT\_FINISH\_TIMER ); //!< Включить помпу на milli времени (мс)**
16. void pumpStartWithoutTimer();
17. void pumpStop(bool justNow=false); //!< Выключить помпу
18. uint8\_t getState(); //!< Получить состояние помпы (BUSY, READY)
19. bool isPouring();
20. **bool isFinishedPoured();**
22. void pumpCheck(); //!< Метод, который следует положить в loop(). Нужен для своевременного отключения по таймеру
24. private:
25. **Timer \* startDelayTimer;**
26. Timer \* finishTimer;
27. Timer \* finishDelayTimer;
28. int m\_pompPin;
29. enum{ BUSY = 0, READY = 1 } m\_state;
30. **};**
32. Pump::Pump( int pin )
33. {
34. startDelayTimer = new Timer( DEFAULT\_DELAY\_TIMER );
35. **finishTimer = new Timer( DEFAULT\_FINISH\_TIMER );**
36. finishDelayTimer = new Timer( DEFAULT\_DELAY\_TIMER );
37. m\_state = READY;
38. m\_pompPin = pin;
39. Serial.println(F("Pump init"));
41. finishTimer->stop();
42. startDelayTimer->stop();
43. finishDelayTimer->stop();
44. }
46. void Pump::pumpStart( uint32\_t milli )
47. {
48. if ( m\_state == BUSY )
49. {
50. **Serial.println(F("Pump BUSY"));**
51. return;
52. }
53. m\_state = BUSY;
54. startDelayTimer->start();
55. **finishTimer->stop();**
56. finishDelayTimer->stop();
57. finishTimer->setInterval( milli );
58. Serial.println(F("Pump PRE-START"));
59. }
61. void Pump::pumpStartWithoutTimer()
62. {
63. digitalWrite( m\_pompPin, HIGH );
64. m\_state = BUSY;
65. **}**
67. void Pump::pumpStop( bool justNow )
68. {
69. digitalWrite( m\_pompPin, LOW );
70. **Serial.println(F("Pump stoped"));**
72. finishTimer->stop();
73. startDelayTimer->stop();
74. finishDelayTimer->start();
76. if ( justNow )
77. {
78. finishDelayTimer->stop();
79. m\_state = READY;
80. **}**
81. }
83. uint8\_t Pump::getState()
84. {
85. **return m\_state;**
86. }
88. bool Pump::isPouring()
89. {
90. **if ( startDelayTimer->getState() == false )**
91. return true;
92. return false;
93. }
95. **bool Pump::isFinishedPoured()**
96. {
97. if ( finishTimer->getState() == false )
98. return true;
99. return false;
100. **}**
102. void Pump::pumpCheck()
103. {
104. if ( startDelayTimer->isReady() )
105. **{**
106. startDelayTimer->stop();
107. finishTimer->start();
108. finishDelayTimer->stop();
110. **digitalWrite( m\_pompPin, HIGH );**
111. Serial.println(F("Pump started"));
112. }
114. if ( finishTimer->isReady() )
115. **pumpStop();**
117. if ( finishDelayTimer->isReady() )
118. {
119. startDelayTimer->stop();
120. **finishTimer->stop();**
121. finishDelayTimer->stop();
122. m\_state = READY;
123. }
124. }
126. #endif

servoCrane.hpp

1. #ifndef SERVO\_CRANE\_H
2. #define SERVO\_CRANE\_H
4. #include <Servo.h>
6. class ServoCrane
7. {
8. public:
9. ServoCrane( uint8\_t pin );
10. **void rotate( uint8\_t angle );**
11. uint8\_t getCurrentAngle();
12. private:
13. Servo \* p\_servo;
14. uint8\_t m\_currentAngle;
15. **};**
17. ServoCrane::ServoCrane( uint8\_t pin )
18. {
19. p\_servo = new Servo();
20. **m\_currentAngle = 180;**
21. p\_servo->attach( pin );
22. p\_servo->write( m\_currentAngle );
24. Serial.println(F("ServoCrane init"));
25. **}**
27. void ServoCrane::rotate( uint8\_t angle )
28. {
29. if ( angle > 180 || angle < 0 )
30. **return;**
31. p\_servo->write( angle );
32. m\_currentAngle = angle;
33. Serial.println("ServoCrane rotate to " + angle);
34. }
36. uint8\_t ServoCrane::getCurrentAngle()
37. {
38. return m\_currentAngle;
39. }


43. #endif

Display.hpp

1. #ifndef DISPLAY\_H
2. #define DISPLAY\_H
4. #include "mem.hpp"
5. **#include <Adafruit\_GFX.h>**
6. #include <Adafruit\_SSD1306.h>
8. String utf8rus(String source);
9. //String printPGM(byte idx);
11. class Display
12. {
13. public:
14. Display();
15. **void printRandToast();**
16. void printInfo( uint16\_t volume, uint8\_t mode );
17. void printInfo( uint32\_t volume );
18. private:
19. Adafruit\_SSD1306 \* display;
20. **};**
22. Display::Display()
23. {
24. display = new Adafruit\_SSD1306(128, 32, &Wire);
25. **display->begin(SSD1306\_SWITCHCAPVCC, 0x3C);**
26. display->clearDisplay();
27. display->display();
29. Serial.println(F("Display init"));
31. display->setTextSize(2);
32. display->setTextColor(SSD1306\_WHITE);
33. display->setCursor(0,0);
34. display->println(utf8rus("Робот**\n**Разливайка"));
35. **display->display();**
36. }
38. void Display::printRandToast()
39. {
40. **display->setTextSize(2);**
41. display->clearDisplay();
42. display->display();
43. display->setCursor(0,0);
44. display->println(utf8rus(printPGM(random(0, 34))));
45. **display->display();**
46. }
48. void Display::printInfo( uint16\_t volume, uint8\_t mode )
49. {
50. **display->setTextSize(4);**
51. display->clearDisplay();
52. display->display();
53. display->setCursor(0,0);
54. display->println( volume );
55. **display->setCursor(100,0);**
57. if ( mode == 0 )
58. display->println( "M" );
59. else if ( mode == 1 )
60. **display->println( "A" );**
61. else if ( mode == 2 )
62. display->println( "S" );
64. display->display();
65. **}**
67. void Display::printInfo( uint32\_t volume )
68. {
69. display->setTextSize(4);
70. **display->clearDisplay();**
71. display->display();
72. display->setCursor(0,0);
73. display->println( volume );
74. display->display();
75. **}**

78. String utf8rus(String source)
79. {
80. **int i,k;**
81. String target;
82. unsigned char n;
83. char m[2] = { '0', '**\0**' };
85. **k = source.length(); i = 0;**
87. while (i < k) {
88. n = source[i]; i++;
90. **if (n >= 0xC0) {**
91. switch (n) {
92. case 0xD0: {
93. n = source[i]; i++;
94. if (n == 0x81) { n = 0xA8; break; }
95. **if (n >= 0x90 && n <= 0xBF) n = n + 0x30;**
96. break;
97. }
98. case 0xD1: {
99. n = source[i]; i++;
100. **if (n == 0x91) { n = 0xB8; break; }**
101. if (n >= 0x80 && n <= 0x8F) n = n + 0x70;
102. break;
103. }
104. }
105. **}**
106. m[0] = n; target = target + String(m);
107. }
108. return target;
109. }
111. #endif

Display\_debug.hpp

1. #ifndef DISPLAY\_H
2. #define DISPLAY\_H
4. #include "mem.hpp"
6. String utf8rus(String source);
7. String printPGM(byte idx);
9. class Display
10. **{**
11. public:
12. Display();
13. void printRandToast();
14. void printInfo( uint16\_t volume, uint8\_t mode );
15. **};**
17. Display::Display()
18. {
19. Serial.println(F("Display init"));
20. **Serial.println(F("Робот\nРазливайка"));**
21. }
23. void Display::printRandToast()
24. {
25. **Serial.println( printPGM(random( 0, 20 ) ) );**
26. }
28. void Display::printInfo( uint16\_t volume, uint8\_t mode )
29. {
30. **String res;**
31. res = volume;
32. res += " ";
34. if ( mode == 0 )
35. **res += "M";**
36. else if ( mode == 1 )
37. res += "A";
38. else if ( mode == 2 )
39. res += "S";
41. Serial.println( res );
42. }

45. **void Display::printInfo( uint32\_t volume )**
46. {
47. Serial.println( volume );
48. }

51. String utf8rus(String source)
52. {
53. int i,k;
54. String target;
55. **unsigned char n;**
56. char m[2] = { '0', '**\0**' };
58. k = source.length(); i = 0;
60. **while (i < k) {**
61. n = source[i]; i++;
63. if (n >= 0xC0) {
64. switch (n) {
65. **case 0xD0: {**
66. n = source[i]; i++;
67. if (n == 0x81) { n = 0xA8; break; }
68. if (n >= 0x90 && n <= 0xBF) n = n + 0x30;
69. break;
70. **}**
71. case 0xD1: {
72. n = source[i]; i++;
73. if (n == 0x91) { n = 0xB8; break; }
74. if (n >= 0x80 && n <= 0x8F) n = n + 0x70;
75. **break;**
76. }
77. }
78. }
79. m[0] = n; target = target + String(m);
80. **}**
81. return target;
82. }
84. #endif

Slot.hpp

1. #ifndef SLOT\_H
2. #define SLOT\_H
4. #include "timer.hpp"
5. **#include "servocrane.hpp"**
6. #include "pump.hpp"
7. #include "ledstrip.hpp"
9. class Slot
10. **{**
11. public:
12. Slot( LedStrip \* ledStrip, Pump \* pump, ServoCrane \* servo, uint8\_t btnPin, uint8\_t slotID, uint8\_t slotAngle, uint16\_t \* glassVolume );
13. void slotFirstCheck();
14. void slotSecondCheck();
15. **void slotThirdCheck();**
16. uint8\_t getState();
18. private:
19. Timer \* btnTimer;
20. **bool btnState;**
21. ServoCrane \* m\_servo;
22. Pump \* p\_pump;
23. LedStrip \* p\_ledStrip; //!< Указатель на сервис ленты светодиодов
24. uint8\_t m\_btnPin; //!< Номер пина концевика
25. **uint8\_t m\_slotID; //!< ID слота. Должен совпадать с индексом светодиода**
26. uint8\_t m\_slotAngle; //!< Угол расположения слота (0-180)
27. enum{ NO\_GLASS, EMPTY, NEXT, PROCESS, READY } m\_state; //!< Статус слота
28. uint16\_t \* p\_glassVolume;
29. };
31. Slot::Slot( LedStrip \* ledStrip, Pump \* pump, ServoCrane \* servo, uint8\_t btnPin, uint8\_t slotID, uint8\_t slotAngle, uint16\_t \* glassVolume )
32. {
33. btnTimer = new Timer( 100 );
34. btnState = false;
35. **p\_ledStrip = ledStrip;**
36. p\_pump = pump;
37. m\_servo = servo;
38. m\_btnPin = btnPin;
39. m\_slotID = slotID;
40. **m\_slotAngle = slotAngle;**
41. m\_state = NO\_GLASS;
42. p\_glassVolume = glassVolume;
43. }
45. **void Slot::slotFirstCheck()**
46. {
47. */\**
48. *\* Этап #1. Проверяем нажатие концевика. Включаем красный светодиод если рюмку только*
49. *\* что поставили. Выключаем светодиод, если рюмку убрали. В ручном режиме этот режим нужно включить в loop отдельно,*
50. ***\* чтобы светодиоды закагарились красным до начала разлива***
51. *\**
52. *\*/*
53. // Если концевик НЕ нажат
55. **if ( btnTimer->isReady() )**
56. btnState = digitalRead( m\_btnPin );
58. if ( btnState )
59. {
60. **// У сервопривода будет поле текущего угла. Нужно проверить - если угол серво равен углу слота и статус слота = PROCESS, значит стопаем помпу**
61. if ( m\_servo->getCurrentAngle() == m\_slotAngle && ( m\_state == PROCESS || m\_state == NEXT ) )
62. {
63. p\_pump->pumpStop( false );
64. }
66. if ( m\_state != NO\_GLASS )
67. {
68. m\_state = NO\_GLASS;
69. p\_ledStrip->ledOff( m\_slotID );
70. **}**
71. }
72. // Если концевик нажат
73. else if ( !btnState )
74. {
75. **if ( m\_state == NO\_GLASS )**
76. {
77. m\_state = EMPTY;
78. p\_ledStrip->ledOn( m\_slotID, 255, 0, 0 );
79. }
80. **}**
81. }
83. void Slot::slotSecondCheck()
84. {
85. ***/\****
86. *\* Этап #2. После первой проверки проверяем статус. Если рюмка налита красим светодиод в зеленый.*
87. *\**
88. *\*/*
90. **if ( m\_state == PROCESS && p\_pump->isFinishedPoured() )**
91. {
92. m\_state = READY;
93. p\_ledStrip->ledOn( m\_slotID, 0, 255, 0 );
94. }
96. }
98. void Slot::slotThirdCheck()
99. {
100. ***/\****
101. *\* Этап #3. После второй проверки проверяем статус. Если рюмка пустует - пытаемся запросить серво и помпу налить в нее.*
102. *\**
103. *\*/*
104. if ( m\_state == EMPTY && p\_pump->getState() != 0 */\* m\_servo->getState() != BUSY \*/* )
105. **{**
106. m\_state = NEXT;
107. m\_servo->rotate( m\_slotAngle );
108. p\_pump->pumpStart( ONE\_HUNDRED\_PER\_SECOND / 100 \* (\*p\_glassVolume) );
109. p\_ledStrip->ledOn( m\_slotID, 0, 0, 255 );
110. **}**
112. if ( m\_state == NEXT && p\_pump->isPouring() )
113. {
114. m\_state = PROCESS;
115. **p\_ledStrip->ledOn( m\_slotID, 255, 255, 0 );**
116. }
118. }
120. **uint8\_t Slot::getState()**
121. {
122. return m\_state;
123. }
125. **#endif**

Brain.hpp

1. #ifndef BRAIN\_H
2. #define BRAIN\_H
4. #include "timer.hpp"
5. **#include "settings.hpp"**
6. #include <EncButton.h>
8. #include "slot.hpp"
9. #include "servocrane.hpp"
10. **#include "pump.hpp"**
11. #include "ledstrip.hpp"
12. #include "display.hpp"


16. const uint8\_t BTN\_SLOT\_PINS[SLOT\_COUNT] { A0, A1, A2, A3, 2, 7 };
17. const uint8\_t SLOT\_IDS[SLOT\_COUNT] { 0, 1, 2, 3, 4, 5 };
18. const uint8\_t SLOT\_ANGLES[SLOT\_COUNT] { 175, 143, 109, 71, 35, 5 };
20. **class Brain**
21. {
22. public:
23. Brain();
24. void tick();
25. **private:**
26. void changeAutoManualMode();
27. void changeServiceMode();
28. void incGlassVolume();
29. void decGlassVolume();
30. **void mainBtnPressed();**
31. void mainBtnReleased();
32. void mainBtnClick();
33. void encClicked();
35. **enum{MANUAL, AUTO, SERVICE} m\_mode;**
36. uint16\_t m\_glassVolume;
37. enum{ BUSY, READY } m\_status;
38. enum{ STARTED, FINISHED } m\_manualModeFlag;
39. bool m\_toastFlag;
41. bool isStarted;
42. Timer \* p\_starterTimer;
43. uint32\_t p\_serviceTime;
44. ServoCrane \* p\_servo;
45. **Pump \* p\_pump;**
46. LedStrip \* p\_ledStrip;
47. Slot \*\* p\_slots;
48. Display \* p\_display;
49. EncButton<EB\_TICK, ENC\_A\_PIN, ENC\_B\_PIN, ENC\_BTN\_PIN> \* p\_encoder;
50. **EncButton<EB\_TICK, MAIN\_BTN\_PIN> \* p\_button;**
51. };
53. Brain::Brain()
54. {
55. **p\_starterTimer = new Timer( 3000 );**
56. isStarted = false;
58. m\_mode = MANUAL;
59. m\_glassVolume = 20;
60. **m\_toastFlag = false;**
61. p\_servo = new ServoCrane( SERVO\_PIN );
62. p\_pump = new Pump( PUMP\_PIN );
63. p\_ledStrip = new LedStrip();
64. p\_slots = new Slot \* [SLOT\_COUNT];
66. for ( int i = 0; i < SLOT\_COUNT; ++i )
67. p\_slots[i] = new Slot( p\_ledStrip, p\_pump, p\_servo, BTN\_SLOT\_PINS[i], SLOT\_IDS[i], SLOT\_ANGLES[i], &m\_glassVolume );
69. p\_display = new Display();
71. p\_encoder = new EncButton<EB\_TICK, ENC\_A\_PIN, ENC\_B\_PIN, ENC\_BTN\_PIN>;
72. p\_button = new EncButton<EB\_TICK, MAIN\_BTN\_PIN>;
74. m\_manualModeFlag = FINISHED;
75. **}**
77. void Brain::tick()
78. {
79. if ( !isStarted )
80. **{**
81. isStarted = p\_starterTimer->isReady();
82. return;
83. }
85. **m\_status = READY;**
86. for ( int i = 0; i < SLOT\_COUNT; ++i )
87. if ( p\_slots[i]->getState() == 2 || p\_slots[i]->getState() == 3 )
88. {
89. m\_status = BUSY;
90. **m\_manualModeFlag = STARTED;**
91. if ( !m\_toastFlag )
92. p\_display->printRandToast();
93. m\_toastFlag = true;
94. break;
95. **}**
97. if ( m\_status == READY && p\_servo->getCurrentAngle() != 180 && m\_mode != SERVICE && p\_pump->getState() != 0 )
98. p\_servo->rotate( 180 );
100. **if ( m\_status == READY && p\_servo->getCurrentAngle() == 180 && m\_mode != SERVICE && p\_pump->getState() != 0 )**
101. {
102. m\_manualModeFlag = FINISHED;
103. m\_toastFlag = false;
104. }
106. p\_encoder->tick();
107. p\_button->tick();
108. p\_pump->pumpCheck();
110. **if ( m\_status == READY )**
111. {
112. if ( p\_encoder->rightH() || p\_encoder->leftH() )
113. changeAutoManualMode();
114. else if ( p\_encoder->held() )
115. **changeServiceMode();**
116. else if ( p\_encoder->right() )
117. incGlassVolume();
118. else if ( p\_encoder->left() )
119. decGlassVolume();
120. **else if ( p\_encoder->click() )**
121. encClicked();
122. else if ( p\_button->press() )
123. mainBtnPressed();
124. else if ( p\_button->release() )
125. **mainBtnReleased();**
126. else if ( p\_button->click() )
127. mainBtnClick();
128. }
130. **if ( m\_mode == MANUAL )**
131. {
132. for ( int i = 0; i < SLOT\_COUNT; ++i )
133. p\_slots[i]->slotFirstCheck();
134. if ( m\_manualModeFlag == STARTED )
135. **{**
136. for ( int i = 0; i < SLOT\_COUNT; ++i )
137. p\_slots[i]->slotSecondCheck();
138. for ( int i = 0; i < SLOT\_COUNT; ++i )
139. p\_slots[i]->slotThirdCheck();
140. **}**
141. }
142. else if ( m\_mode == AUTO )
143. {
144. for ( int i = 0; i < SLOT\_COUNT; ++i )
145. **p\_slots[i]->slotFirstCheck();**
146. for ( int i = 0; i < SLOT\_COUNT; ++i )
147. p\_slots[i]->slotSecondCheck();
148. for ( int i = 0; i < SLOT\_COUNT; ++i )
149. p\_slots[i]->slotThirdCheck();
150. **}**
151. else if ( m\_mode == SERVICE )
152. {
153. for ( int i = 0; i < SLOT\_COUNT; ++i )
154. p\_slots[i]->slotFirstCheck();
155. **}**
156. }
158. void Brain::changeAutoManualMode()
159. {
160. **Serial.println(F("Mode changed"));**
161. if ( m\_mode != SERVICE )
162. {
163. if ( m\_mode == AUTO )
164. m\_mode = MANUAL;
165. **else if ( m\_mode == MANUAL )**
166. m\_mode = AUTO;
168. p\_display->printInfo( m\_glassVolume, m\_mode );
169. }
170. **}**
172. void Brain::changeServiceMode()
173. {
174. Serial.println(F("Mode changed"));
175. **if ( m\_mode == AUTO || m\_mode == MANUAL)**
176. {
177. m\_mode = SERVICE;
178. p\_display->printInfo( p\_servo->getCurrentAngle(), m\_mode );
179. }
180. **else if ( m\_mode == SERVICE && !p\_button->hold() )**
181. {
182. m\_mode = MANUAL;
183. p\_display->printInfo( m\_glassVolume, m\_mode );
184. }
185. **}**
187. void Brain::incGlassVolume()
188. {
189. if ( m\_mode == AUTO || m\_mode == MANUAL )
190. **{**
191. m\_glassVolume = m\_glassVolume >= 250 ? 250 : m\_glassVolume + 1;
192. p\_display->printInfo( m\_glassVolume, m\_mode );
193. }
195. **else if ( m\_mode == SERVICE )**
196. {
197. p\_servo->rotate( p\_servo->getCurrentAngle() + 1 );
198. p\_display->printInfo( p\_servo->getCurrentAngle(), m\_mode );
199. }
200. **}**
202. void Brain::decGlassVolume()
203. {
204. if ( m\_mode == AUTO || m\_mode == MANUAL )
205. **{**
206. m\_glassVolume = m\_glassVolume <= 5 ? 5 : m\_glassVolume - 1;
207. p\_display->printInfo( m\_glassVolume, m\_mode );
208. }
210. **else if ( m\_mode == SERVICE )**
211. {
212. p\_servo->rotate( p\_servo->getCurrentAngle() - 1 );
213. p\_display->printInfo( p\_servo->getCurrentAngle(), m\_mode );
214. }
215. **}**
217. void Brain::mainBtnPressed()
218. {
219. Serial.println(F( "MainBTN PRESSED" ) );
220. **if ( m\_mode == AUTO )**
221. return;
222. else if ( m\_mode == MANUAL )
223. return;
224. else if ( m\_mode == SERVICE )
225. **{**
226. p\_serviceTime = millis();
227. p\_pump->pumpStartWithoutTimer();
228. }
229. }
231. void Brain::mainBtnReleased()
232. {
233. Serial.println(F("MainBTN RELEASED"));
234. if ( m\_mode == SERVICE )
235. **{**
236. p\_display->printInfo( ( millis() - p\_serviceTime ) % 100000 );
237. p\_pump->pumpStop();
238. }
239. }
241. void Brain::mainBtnClick()
242. {
243. if ( m\_mode == MANUAL )
244. {
245. **m\_manualModeFlag = STARTED;**
246. }
247. }
249. void Brain::encClicked()
250. **{**
251. if ( ( m\_mode == AUTO || m\_mode == MANUAL ) && m\_manualModeFlag == FINISHED )
252. p\_display->printInfo( m\_glassVolume, m\_mode );
254. else if ( m\_mode == SERVICE && m\_manualModeFlag == FINISHED )
255. **p\_display->printInfo( p\_servo->getCurrentAngle(), m\_mode );**
256. }
258. #endif

Setting.hpp

1. #ifndef SETTINGS\_H
2. #define SETTINGS\_H

5. **//Общая настройка**
6. #define SLOT\_COUNT 6
7. #define SERVO\_PIN 5
8. #define PUMP\_PIN 6

11. //Настройка элементов управления
12. #define MAIN\_BTN\_PIN 4
13. #define ENC\_BTN\_PIN 11
14. #define ENC\_A\_PIN 10
15. **#define ENC\_B\_PIN 8**

18. //Настройка светодиодной ленты
19. #define BRIGHTNESS 255 //!< Яркость светодиодов
20. **#define LED\_PIN 3 //!< Пин подключения светодиодной ленты**
21. #define LED\_COUNT 6 //!< Кол-во светодиодов в ленте

24. //Настройка помпы
25. **#define DEFAULT\_DELAY\_TIMER 1500**
26. #define DEFAULT\_FINISH\_TIMER 2000 //!< Время по умолчанию, которое будет отрабатывать мотор при pumpStart( uint32\_t seconds )
27. #define ONE\_HUNDRED\_PER\_SECOND 12900

30. **#endif**

Mem.hpp

1. #ifndef MEM\_H
2. #define MEM\_H
4. const char str1[] PROGMEM = " ЗА**\n** ЛЮБОВЬ";
5. **const char str2[] PROGMEM = " ЧТОБЫ\n ВСЕ";**
6. const char str3[] PROGMEM = " ЗА ВСЁ**\n** ХОРОШЕЕ";
7. const char str4[] PROGMEM = " ЗА**\n** ЗДОРОВЬЕ";
8. const char str5[] PROGMEM = " ЗА**\n** ДЕТЕЙ";
9. const char str6[] PROGMEM = "ЗА**\n**РОДИТЕЛЕЙ";
10. **const char str7[] PROGMEM = " ЗА\n ДРУЖБУ";**
11. const char str8[] PROGMEM = " БЫТЬ**\n** ДОБРУ";
12. const char str9[] PROGMEM = " ЗА МИР";
13. const char str10[] PROGMEM = " НЕ**\n** ЧОКАЯСЬ";
14. const char str11[] PROGMEM = " ЗА**\n** ЖИЗНЬ";
15. **const char str12[] PROGMEM = " ЗА ДАМ";**
16. const char str13[] PROGMEM = " ЗА**\n** РОССИЮ";
17. const char str14[] PROGMEM = " ЗА**\n** МУЖЧИН";
18. const char str15[] PROGMEM = " ЗА**\n** УДАЧУ";
19. const char str16[] PROGMEM = " ЗА НАС";
20. **const char str17[] PROGMEM = " ЗА\n КРАСОТУ";**
21. const char str18[] PROGMEM = " ЗА**\n** ДОБРОТУ";
22. const char str19[] PROGMEM = " ЗА**\n** ПОГОДУ";
23. const char str20[] PROGMEM = " ЗА**\n** ПРИРОДУ";
24. const char str21[] PROGMEM = " ЗА**\n** ХОЗЯИНА";
25. **const char str22[] PROGMEM = " ЗА\n БУДУЩЕЕ";**
26. const char str23[] PROGMEM = " ЗА**\n** ПАМЯТЬ";
27. const char str24[] PROGMEM = " ЗА**\n** ОТДЫХ";
28. const char str25[] PROGMEM = "ЗА**\n**НАСТРОЕНИЕ";
29. const char str26[] PROGMEM = " ЧТОБ**\n** СТОЯЛ";
30. **const char str27[] PROGMEM = " ЗА\n ДОСТАТОК";**
31. const char str28[] PROGMEM = " ЗА**\n** КОСМОС";
32. const char str29[] PROGMEM = " ЗА**\n** ТЕРПЕНИЕ";
33. const char str30[] PROGMEM = " ЗА**\n** МЕДВЕДЕЙ";
34. const char str31[] PROGMEM = " ЗА**\n** ЛОГИКУ";
35. **const char str32[] PROGMEM = "ЗА\n ТРЕЗВОСТЬ";**
36. const char str33[] PROGMEM = " ЗА**\n** УСПЕХ";
37. const char str34[] PROGMEM = " ЗА**\n** ПОБЕДУ";
39. const char\* const names[] PROGMEM = {
40. **str1, str2, str3, str4, str5,**
41. str6, str7, str8, str9, str10,
42. str11, str12, str13, str14, str15,
43. str16, str17, str18, str19, str20,
44. str21, str22, str23, str24, str25,
45. **str26, str27, str28, str29, str30,**
46. str31, str32, str33, str34
47. };
49. String printPGM(byte idx) {
50. **PGM\_P p = pgm\_read\_word(names + idx);**
51. char buf[strlen\_P(p)];
52. strcpy\_P(buf, p);
53. return String(buf);
54. }
56. #endif

Main.ino

1. #include "brain.hpp"
3. Brain \* brain;
5. **void setup() {**
6. pinMode(A0, INPUT\_PULLUP);
7. pinMode(A1, INPUT\_PULLUP);
8. pinMode(A2, INPUT\_PULLUP);
9. pinMode(A3, INPUT\_PULLUP);
10. **pinMode(2, INPUT\_PULLUP);**
11. pinMode(7, INPUT\_PULLUP);
12. pinMode(4, INPUT\_PULLUP);
13. pinMode(6, OUTPUT);
14. pinMode(3, OUTPUT);
15. **pinMode(5, OUTPUT);**
16. //digitalWrite(6, LOW);
18. Serial.begin(9600);
19. brain = new Brain();
20. **}**
22. void loop() {
23. brain->tick();
24. }
26. extern int \_\_bss\_end;
27. extern void \*\_\_brkval;
29. int memoryFree()
30. **{**
31. int freeValue;
32. if((int)\_\_brkval == 0)
33. freeValue = ((int)&freeValue) - ((int)&\_\_bss\_end);
34. else
35. **freeValue = ((int)&freeValue) - ((int)\_\_brkval);**
36. return freeValue;
37. }