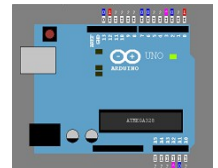


UnoArduSimV2.7 Полный Помогите



Оглавление

Обзор

Код Панель, Предпочтения и Изменить/Изучить

Код Панель

Предпочтения

Изменить/Изучить

Переменные Панель и Изменить/Монитор Переменная окно

Лабораторная Скамья Панель

'Uno'

'I/O' Устройства

Serial Монитор ('SERIAL')

Программное Serial ('SFTSER')

SD-Диск ('SD_DRV')

TFT дисплей ('TFT')

Конфигурируемый SPI Ведомый ('SPISLV')

Двухпроводная I2C Ведомый ('I2CSLV')

Текстовый LCD I2C ('LCDI2C')

Текстовый LCD SPI ('LCDSPI')

Текстовый LCD D4 ('LCD_D4')

Мультиплексор LED I2C ('MUXI2C')

Мультиплексор LED SPI ('MUXSPI')

Порт Расширения SPI ('EXPSPi')

Порт Расширения I2C ('EXPI2C')

'1-Wire' Ведомый ('OWIISLV')

Генератор с Один-Выстрел ('1SHOT')

Регистр Сдвига Ведомый ('SRSLV')

Программируемый 'I/O' Устройство ('PROGIO')

Цифровой Генератор Импульсов ('PULSER')

Аналоговый Генератор Функций ('FUNCGEN')

Шаговый Мотор ('STEPR')

Пульсирующий Шаговый Мотор ('PSTEPR')

DC Мотор ('MOTOR')

Сервомотор ('SERVO')

Пьезоэлектрический Динамик ('PIEZO')

Скольжение Резистор ('R=1K')

Толкать-Кнопка ('PUSH')

Цветной LED ('LED')

4-LED ряд ('LED4')

7-сегментный LED Цифра ('7SEG')

Аналоговый Слайдер

Pin Перемычка ('JUMP')

Меню

Файл:

Нагрузка INO или PDE Prog (Ctrl-L)

Изменить/Изучить (Ctrl-E)

Сохранить

Сохранить как следующий ('#include')

предыдущий

Выход

Найти:

Восходить стек вызовов

Спуститесь стек вызовов

[Установить текст Установить текст \(Ctrl-F\)](#)

[Найти Следующий текст](#)

[Найти Предыдущий текст](#)

Выполнить:

[Шаг Внутри \(F4\)](#)

[Шаг Вокруг \(F5\)](#)

[Шаг Вне \(F6\)](#)

[Выполнить До \(F7\)](#)

[Выполнить Пока \(F8\)](#)

[Выполнить \(F9\)](#)

[Стой \(F10\)](#)

[Сброс](#)

[Анимация](#)

[Замедленная съемка](#)

Опции:

[Шаг Вокруг Structors/ Операторы](#)

[Регистр-Allocation](#)

[Ошибка при неинициализированном](#)

[добавленной 'loop\(\)' задержка](#)

[Разрешить вложенные прерывания](#)

Конфигурировать:

['I/O' Устройства](#)

[Предпочтения](#)

ПеремОбновить:

[Разрешить авто \(-\) Сокращаться](#)

[минимальная](#)

[Основной момент изменения](#)

Окна:

[Serial Монитор](#)

[Восстановить все](#)

[Pin Цифровые Осциллограммы](#)

[Pin Аналоговый Осциллограммо](#)

Помогите:

[Быстрый Помогите Файл](#)

[Полный Помогите Файл](#)

[Исправления Ошибка](#)

[Изменение / Улучшение](#)

[Около](#)

['Uno' Версия для Uno и 'I/O' Устройства](#)

[Синхронизация](#)

['I/O' Устройство Синхронизация](#)

[Звуки](#)

Ограничения и неподдерживаемые элементы

[Включено Файлы](#)

[Динамическое распределение памяти и оперативная память](#)

['Flash' Распределение памяти](#)

['String' Переменные](#)

[Библиотеки Arduino](#)

[указатели](#)

['class' а также 'struct' Объектов](#)

[Сфера](#)

[Отборочные 'unsigned', 'const', 'volatile', 'static'](#)

[Директивы Компилятор](#)

[Ардуино-языковые элементы](#)

[C / C ++ - языковые элементы](#)

[Функциональный модуль Шаблоны](#)

[Эмуляция в реальном времени](#)

Примечания к выпуску

[Исправления Ошибка](#)

[V2.7.0- март 2020](#)

[V2.6.0- Jan 2020](#)

[V2.5.0- октября 2019](#)

[V2.4- май 2019](#)

[V2.3- декабрь 2018](#)

[V2.2– июнь 2018](#)
[V2.1.1– март 2018](#)
[V2.1– март 2018](#)
[V2.0.2 февраль 2018](#)
[V2.0.1– январь 2018](#)
[V2.0– декабрь 2017](#)
[V1.7.2– февраль 2017](#)
[V1.7.1– февраль 2017](#)
[V1.7.0– декабрь 2016](#)
[V1.6.3– сентябрь 2016](#)
[V1.6.2– сентябрь 2016](#)
[V1.6.1– август 2016](#)
[V1.6– июнь 2016](#)
[V1.5.1– июнь 2016](#)
[V1.5 - май 2016](#)
[V1.4.3 - апрель 2016](#)
[V1.4.2 - март 2016](#)
[V1.4.1 - январь 2016](#)
[V1.4 - декабрь 2015](#)
[V1.3 - октябрь 2015](#)
[V1.2 - июнь 2015](#)
[V1.1 - март 2015](#)
[V1.0.2 - август 2014](#)
[V1.0.1 - июнь 2014](#)
[V1.0 - первый выпуск май 2014](#)
[ЧангES / Улучшения](#)
[V2.7.0- март 2020](#)
[V2.6.0 января 2020](#)
[V2.5.0 октября 2019](#)
[V2.4 мая 2019 г.](#)
[V2.3 дек. 2018](#)
[V2.2 июнь 2018](#)
[V2.1 марта 2018](#)
[V2.0.1 январь 2018](#)
[V2.0 сентябрь 2017](#)
[V1.7.2– февраль 2017](#)
[V1.7.1– февраль 2017](#)
[V1.7.0– декабрь 2016](#)
[V1.6.3– сент. 2016](#)
[V1.6.2– сент. 2016](#)
[V1.6.1– август 2016](#)
[V1.6 - июнь 2016](#)
[V1.5.1 - июнь 2016](#)
[V1.5 - май 2016](#)
[V1.4.2 - март 2016](#)
[V1.4 - декабрь 2015](#)
[V1.3 - октябрь 2015](#)
[V1.2 июнь 2015](#)
[V1.1 - март 2015](#)
[V1.0.1 - июнь 2014](#)
[V1.0 - первый выпуск май 2014](#)

Обзор

UnoArduSim является бесплатной **реальное время** (увидеть для Синхронизация **ограничения**) инструмент симулятор, который я разработал для студента и энтузиаста Arduino. Он разработан для того, чтобы вы могли экспериментировать и легко отлаживать Arduino программы **без необходимости какого-либо фактического оборудования** , Он нацелен на **Arduino 'Uno'** Версия для Uno, и позволяет выбирать из набора виртуальных 'I/O' Устройства, а также настраивать и подключать эти Устройства к вашему виртуальному 'Uno' в **Лабораторная Скамья Панель** , - вам не нужно беспокоиться об ошибках проводки, обрыве / разрыве соединений или неисправности Устройства, которые мешают разработке и тестированию программа.

UnoArduSim предоставляет простые сообщения об ошибках для любых ошибок разобрать или выполнение, с которыми он сталкивается, и позволяет выполнять отладку с помощью **Сброс** , **Выполнить** , **Выполнить До** , **Выполнить Пока** , **Стоп** и гибкий **Шаг** операции в **Код Панель** с одновременным просмотром всех глобальных и в настоящее время активных локальных переменные, массивы и объектов в **Переменные Панель** , Выполнить-временная проверка границ массива обеспечена, и переполнение памяти ATmega будет обнаружено (и преступник программа выделит строку!). Любое электрическое устройство конфликтует с, подключенное к 'I/O' Устройства, помечается и сообщается о его возникновении.

Когда INO или PDE программа файл открыт, он загружается в программа **Код Панель** , программа затем дается Анализировать, чтобы преобразовать его в исполняемый файл, который затем готов к **смоделированный выполнение** (в отличие от Arduino.exe, автономный исполняемый файл двоичном *не* создано) Любая ошибка разобрать обнаружена и помечена путем выделения строки, которая не достигла разобрать, и сообщения об ошибке на **Статус бар** в самом низу приложения UnoArduSim окно. An **Изменить/Изучить** окно может быть открыт, чтобы вы могли видеть и редактировать выделенную синтаксисом версию вашего пользователя программа. Ошибки во время смоделированного выполнение (такие как несоответствующий скорость передачи) сообщаются в строке состояния и через всплывающее окно сообщения.

UnoArduSim V2.7 является практически полной реализацией **Язык программирования Arduino V1.8.8 как документально подтверждено на arduino.cc** , Веб-страница со справочником по языку, а также с дополнениями, указанными на странице версии Загрузка. Примечания к выпуску. Хотя UnoArduSim не поддерживает полную реализацию C ++, которую поддерживает Arduino.exe, лежащий в основе GNU компилятор, вполне вероятно, что только самые продвинутые программисты обнаружат, что какой-то элемент C / C ++, который они хотят использовать, отсутствует (и, конечно, всегда есть простые кодирование обходных путей для таких отсутствующих функций). В общем, я поддерживал только то, что, по моему мнению, является наиболее полезным C / C ++ для любителей и студентов Arduino - например, 'enum' а также '#define' поддерживаются, но указатели функциональный модуль - нет. Даже если пользовательский объектов ('class' а также 'struct') и (большинство) перегрузок операторов поддерживаются, *множественное наследование не* ,

Поскольку UnoArduSim является языковым симулятором высокого уровня, **поддерживаются только операторы C / C ++** , *заявления на ассемблере не* , Точно так же, потому что это не машинная симуляция низкого уровня, **Регистры ATmega328 не доступны для вашего программа** для чтения или записи, хотя распределение, передача и возврат регистров эмулируются (если вы выбираете это в меню **Опции**).

По версии 2.6, UnoArduSim имеет встроенной автоматической поддержки для ограниченного подмножества Arduino при условии, библиотек, они являются: 'Stepper.h' , **LL1** , 'SoftwareSerial.h' , 'SPI.h' , 'Wire.h' , 'OneWire.h' , 'SD.h' , 'TFT.h' и 'EEPROM.h' (Версия 2). V2.6 вводит механизм 3^й поддержка библиотеки стороны через файлы предоставляется в 'include_3rdParty' папка которые могут быть найдены внутри UnoArduSim директории установки. Для любого '#include' из пользовательских библиотек, UnoArduSim будет **не** найдите обычную структуру каталогов установки Arduino, чтобы найти библиотеку; вместо тебя **необходимость** скопировать соответствующий заголовок (".h") и исходный код (".cpp") файл в тот же каталог, что и программа файл, над которым вы работаете (конечно, при условии ограничения содержимого любого '#include' файл должен быть полностью понятен UnoArduSim синтаксический анализатор).

Я разработал UnoArduSimV2.0 в QtCreator с поддержкой нескольких языков, и в настоящее время он доступен только для Окна™, Портирование на Linux или MacOS, это проект на будущее! UnoArduSim выросла из симуляторов, которые я разрабатывал на протяжении многих лет для курсов, которые я преподавал в Университете Королевы, и он был протестирован достаточно обширно, но там наверняка будет несколько ошибки, которые все еще прячутся там. Если вы хотите сообщить о ошибка, опишите его (кратко) по электронной почте unoArduSim@gmail.com а также **обязательно приложите полный исходный код программа Arduino, вызывающий ошибка** так что я могу скопировать ошибка и исправить его. Я не буду отвечать на отдельные отчеты ошибка, и у меня нет гарантированных сроков исправлений в следующем выпуске (помните, что обходные пути почти всегда есть!).

Ура,

Стэн Симмонс, PhD, P.Eng.

Доцент (в отставке)

Кафедра электротехники и вычислительной техники

Королевский университет

Кингстон, Онтарио, Канада







Код Панель, Предпочтения и Изменить/Изучить


(В сторону: образец окна, показанный ниже, все под выбранной пользователем Окна-OS цветовая тема, имеющая темно-синий цвет фона окна).

Код Панель



The **Код Панель** отображает ваш пользователь программа, а выделение отслеживает его выполнение. После того, как загруженный программа имеет успешный Анализировать, первая строка в 'main()' выделен, и программа готов к выполнение. Обратите внимание, что 'main()' неявно добавляется Arduino (и UnoArduSim), и вы делаете **не** включите его как часть вашего пользователя программа файл. Выполнение находится под контролем меню **Выполнить** и связанные с ним **Инструмент-Var** кнопки и комбинации клавиш функциональный модуль.






После активизации выполнение одним (или более) инструкций (вы можете использовать **Инструмент-Var** кнопки , , , или ), Линия программа что будет выполнил следующий затем выделены зеленым цветом - зеленый, выделенная строка всегда следующая строка **готов быть выполнил**,

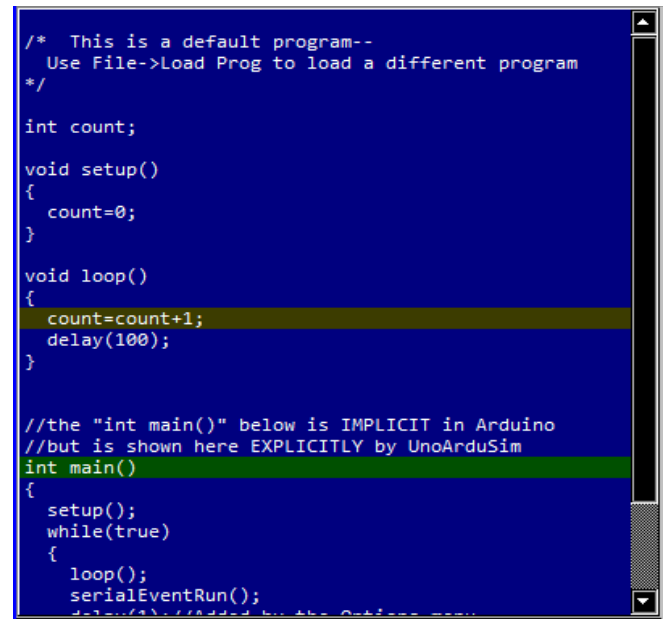
Точно так же, когда бегущий программа поражает (временный **Выполнить До**) точка остановки, выполнение останавливается и линия точка остановки подсвечивается (и затем готова к выполнение).

Если программа выполнение в настоящее время остановлен, и нажмите в **Код Панель** окно, линия вы нажимали подсвечивается в темно-оливковый (как показано на рисунке) - ближайшие к-быть-выполнил линии всегда остается подсвеченной зеленым цветом (по состоянию на V2.7). Но вы можете вызвать выполнение *прогрессировать до* линия, которую вы только что выделили, нажав затем на **Выполнить До** .

Инструмент-Var кнопка. Эта функция позволяет вам быстро и легко достичь определенных линий в программа, чтобы впоследствии вы могли шаг за шагом проходить интересующую вас часть программа.

Если ваш загруженный программа имеет какие-либо '#include' файлы, вы можете перемещаться между ними с помощью **Файл | предыдущий** а также **Файл | следующий** (с **Инструмент-Var** кнопки  а также ).

Действиях **Найти Меню** позволит Вам **ИЛИ** найти текст в **Код Панель** или **Переменные Панель** (**Инструмент-Var** кнопки  и  или сочетания клавиш **стрелка вверх** и Кнопка "Стрелка вниз") *после первого использования* **Найти | Набор Установить** текст текст или **Инструмент-Var** , **ИЛИ АЛЬТЕРНАТИВЫ** в *перемещаться по стек вызовов* в **Код Панель** (**Инструмент-Var** кнопки  и  или сочетания клавиш **стрелка вверх** и Кнопка "Стрелка вниз"). Ключи **вниз** на страницу и **вверх** на страницу прыгать выбор к следующему / предыдущему функциональный модуль .



```
/* This is a default program--
   Use File->Load Prog to load a different program
*/

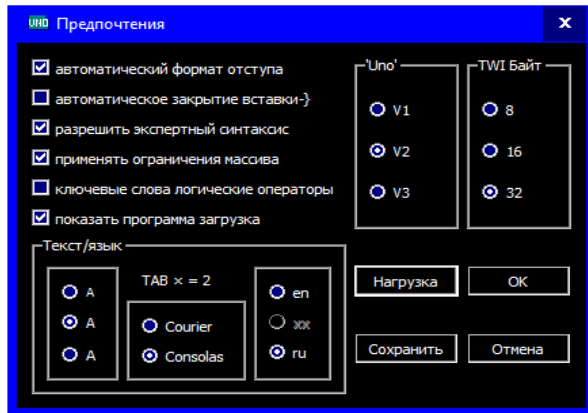
int count;

void setup()
{
  count=0;
}

void loop()
{
  count=count+1;
  delay(100);
}

//the "int main()" below is IMPLICIT in Arduino
//but is shown here EXPLICITLY by UnoArduSim
int main()
{
  setup();
  while(true)
  {
    loop();
    serialEventRun();
    delay(1); //added by the Outpost team
  }
}
```

Предпочтения



Конфигурировать | Предпочтения позволяет пользователям установить программа и предпочтения просмотра (которые пользователь обычно желает принять на он следующий сеанс). Поэтому их можно сохранить и загрузить из **'myArduPrefs.txt'** файл, который находится в том же каталоге, что и загруженный 'Uno' программа (**'myArduPrefs.txt'** автоматически загружается, если он существует).

Это диалоговое окно позволяет выбирать между двумя моноширинными шрифтами и тремя размерами шрифта, а также другими разными предпочтениями. Начиная с версии 2.0, выбор языка теперь включен. - это всегда включают английский (**ан**), плюс один или два других

языка локали пользователя (где они существуют), и одно переопределение на основе двухбуквенного кода языка ISO-639 на самой первой линии из **'myArduPrefs.txt'** файл (если таковой имеется). Выбор появляется только если перевод ".qm" файл существует в папке переводов (внутри домашний каталог UnoArduSim.exe).

Изменить/Изучить

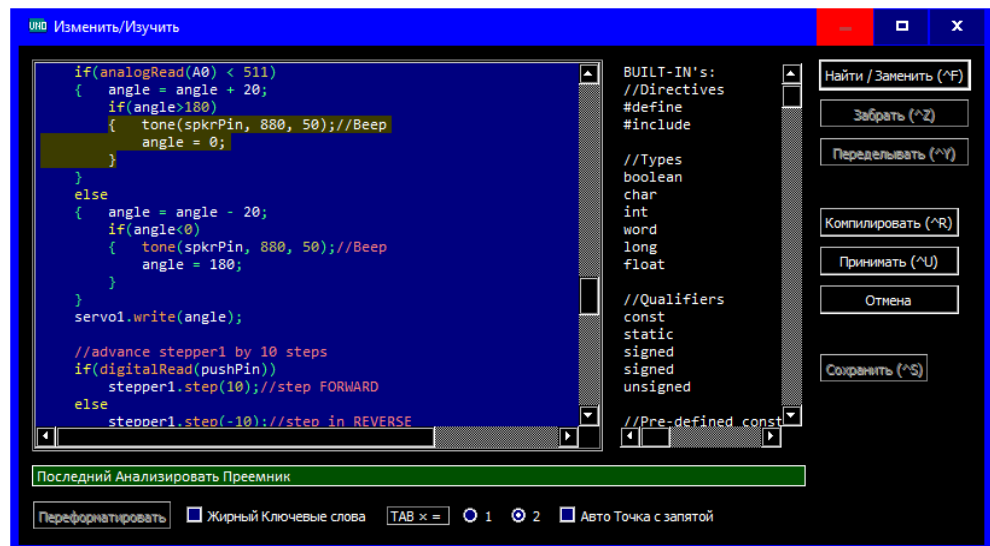
Двойным щелчком по любой строке в **Код Панель** (или используя меню **Файл**), **Изменить/Изучить** окно открывается для внесения изменений в ваш программа файл, с **выбранная строка** в **Код Панель** выделено

Это окно имеет возможность полного редактирования с динамической подсветкой синтаксиса (различные цвета основной момент используются для ключевых слов C ++, комментариев и т. Д.). Существует дополнительная подсветка синтаксиса жирный и автоматическое форматирование на уровне отступа (при условии, что вы выбрали это с помощью

Конфигурировать | Предпочтения). Вы

также можете удобно выбрать встроенный функциональный модуль звонки (или встроенный **'#define'** константы) для добавления в ваш программа из предоставленного списка - просто дважды щелкните по нужному элементу списка, чтобы добавить его к вашему программа в текущей позиции каретки (функциональный модуль-вызов переменная **типы** только для информации и лишены возможности оставлять фиктивные заполнители при добавлении в ваш программа).

окно имеет **Найти** (использование **Ctrl-F**) а также **Найти / Заменить** возможность (использовать **Ctrl-H**), The **Изменить/Изучить** окно имеет **Забрать** (**Ctrl-Z**), а также **Переделывать** (**Ctrl-Y**) кнопки (которые появляются автоматически).



Использование ALT-стрелка вправо для запроса выбора Автозаполнение встроенный **глобальный переменные**, и для **член переменные** и **функциональные модули**.

Отказаться **все изменения** Вы сделали, так как вы впервые открыли программа для редактирования, нажмите **Отмена** кнопка. Принять текущее состояние, нажмите **Принимать** Кнопка и программа автоматически получает другой Анализировать (и загружается в 'Uno', если ошибок не обнаружено), и в главном UnoArduSim окно появляется новый статус **Статус бар** ,

А **Компилировать (Ctrl-R)** кнопка (плюс связанный **Анализировать Статус** было добавлено окно сообщения, как показано на рисунке выше), чтобы можно было проверять правки без необходимости сначала закрывать окно. А **Сохранить (Ctrl-S)** также была добавлена в качестве ярлыка (эквивалент **Принимать** плюс позже отдельный **Сохранить** от основного окно).

Либо на **Отмена** или же **Принимать** без внесенных изменений **Код Панель** текущая строка меняется, чтобы стать **последняя позиция каретки Изменить/Изучить** и вы можете использовать эту функцию, чтобы перейти **Код Панель** к определенной линии (возможно, чтобы подготовиться к **Выполнить До**), Вы также можете использовать **Ctrl-PgDn** а также **Ctrl-PgUp** перейти к следующему (или предыдущему) разрыву пустой строки в программа - это полезно для быстрой навигации вверх или вниз по значимым местам (например, пустые строки между функциональные модули) Вы также можете использовать **Ctrl-Home** а также **Ctrl-End** перейти к началу и концу программа, соответственно.

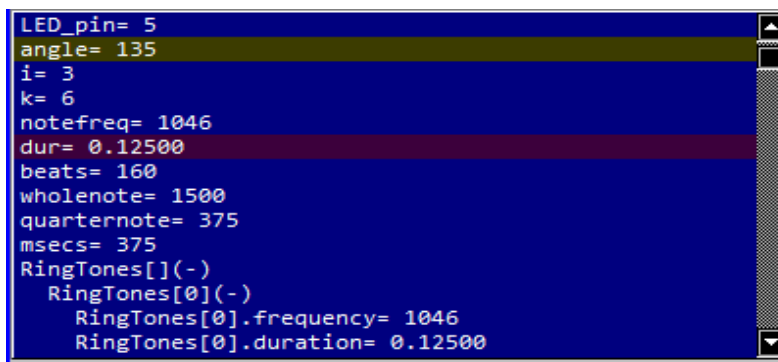
'**Tab**' уровня автоматического форматирования отступов делается при открытии окно, если эта опция была установлена в соответствии с **Конфигурировать | Предпочтения**, Можно повторить, что при форматировании в любое время, нажав на кнопку **Переформатировать** Кнопка (она включена, только если ранее выбрана **автоматический отступ Предпочтение**). Вы можете также добавлять или удалять язычки себя к группе предварительно выбранных последовательных линий с помощью клавиатуры **правая стрелка** или **стрелка влево** ключи - но **автоматический отступ Предпочтение должно быть выключено** чтобы не потерять свои собственные уровни вкладки.

когда **Авто точка с запятой** проверяется нажатием **Войти** для завершения строки автоматически вставляется точка с запятой.

И чтобы помочь вам лучше отслеживать ваши контексты и изогнутые скобки, нажав на ' { ' или же ' } ' изогнутая скобка **выделяет весь текст между этим изогнутая скобка и его соответствующим партнером** ,

Переменные Панель и Изменить/Монитор Переменная окно

The **Переменные Панель** расположен чуть ниже **Код Панель**, Он показывает текущие значения для каждого пользователя глобального и активного (в сфера) локального переменная / массива / объект в загруженном программа. Когда ваш программа выполнение перемещается между функциональные модули, **содержимое изменяется, чтобы отразить только те локальные переменные, которые доступны для текущего функциональный модуль / сфера, плюс любые объявленные пользователем глобальные переменные**, Любой переменные объявлен как 'const' или как 'PROGMEM' (выделено для 'Flash' память) имеют значения, которые не могут быть изменены, и поэтому для экономии места они **не отображаются**, 'Servo' а также 'SoftwareSerial' Экземпляры объект не содержат полезных значений, поэтому также не отображаются.



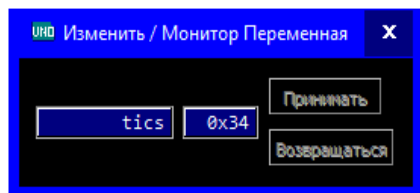
Вы можете **находить** указанный **текст** с его текстом поиск команд (с **Инструмент-Var** кнопки  **text** и  **text** или сочетания клавиш **стрелка вверх** и Кнопка **"Стрелка вниз**), После того, как сначала с помощью **Найти | Набор Установить текст** текст или  ,

Массивы а также **объектов** показаны либо в **ун-расширенный** или же **расширенный** формат, либо с завершающим плюсом ' (+) ' или минус ' (-) ' подписать соответственно. Символ для массива **Икс** показывает как ' **x** [] ' , Чтобы расширять это показать все элементы массива, просто нажмите один раз на

'x[] (+)' в **Переменные Панель**, Чтобы сокращаться вернуться к представлению un-расширенный, нажмите на 'x[] (-)', Un-расширенный по умолчанию для объект 'p1' показывает как 'p1 (+)' К расширять это показать всем членам этого 'class' или же 'struct' Например, нажмите один раз на 'p1 (+)' в **Переменные Панель**, Чтобы сокращаться вернуться к представлению un-расширенный, нажмите один раз на 'p1 (-)'. Если ты **одного щелчка по любой линии основной момент его в темно-оливковый** (Это может быть просто переменная, или совокупность для или '(-)' линия с массива или объект, или одного элемента массива или объект-членов), то делать **Выполнить Пока** заставит выполнение возобновить и заморозить на следующей **записи доступа** в любом месте внутри этого выбранного агрегата, или что выбран единый переменная местоположение.

Когда используешь **Шаг** или же **Выполнить** Обновление отображаемых значений переменная производится в соответствии с настройками пользователя, выполненными в меню. **ПеремОбновить** - это допускает полный диапазон поведения от минимальных периодических обновлений до полных немедленных обновлений. Уменьшенные или минимальные обновления полезны для уменьшения нагрузки на процессор и могут быть необходимы, чтобы выполнение не отставал в реальном времени от того, что в противном случае было бы чрезмерным **Переменные Панель** Обновление окно загружает. когда **Анимация** действует или если **Основной момент Изменения** опция меню, изменяется на значение переменная во время **Выполнить** приведет к обновлению отображаемого значения **немедленно**, и он становится выделенным - это приведет к **Переменные Панель** прокрутить (если необходимо) до строки, которая содержит переменная, и выполнение больше не будет в режиме реального времени !.

Когда выполнение замерзает после **Шаг**, **Выполнить До**, **Выполнить Пока**, или же **Выполнить** -затем-**Стоит**, **Переменные Панель** выдвигает на первый план переменная, соответствующий **адрес (а) адреса, которые были изменены** (если есть) **самая последняя инструкция** во время этого выполнение (включая инициализацию объявления переменная). Если эта инструкция **полностью** заполнены **объект или массива, родительская (+) или (-) строка** для этого агрегата становится выделенным. Если вместо этого инструкция изменила место нахождения то, что в данный момент видно, затем становится подсвеченным. Но если измененное местоположение (я) в настоящее время скрывается внутри UN-расширенный массива или объект, которые совокупные **родительская линия** получает **выделение курсивом** в качестве визуальной подсказки, что что-то внутри этого было записано - щелчок по расширять вызовет **прошлой** измененный элемент или член, чтобы стать выделенным.



The **Изменить/Монитор** окно дает вам **способность следовать любому значению переменная в течение выполнение или изменить его значение в середине (остановлено) программа выполнение** (так что вы можете проверить, каков будет эффект продолжения работы с этим новым значением). **Стоит** Сначала выполнение, потом **левый двойной щелчок** на переменная, значение которого вы хотите отслеживать или изменить. Чтобы просто контролировать значение во время программа

выполнение, **оставьте диалоговое окно открытым** а затем один из **Выполнить** или же **Шаг** команды - его значение будет обновлено в **Изменить/Монитор** в соответствии с теми же правилами, которые регулируют обновления в **Переменные Панель**, **Чтобы изменить значение переменная**, заполните значение поля ввода и **Принимать**, Продолжить выполнение (используя любой из **Шаг** или же **Выполнить** команды), чтобы использовать это новое значение с этого момента вперед (или вы можете **Возвращаться** к предыдущему значению).

На программа Нагрузка или Сброс обратите внимание, что все **неинициализированное значение переменные сбрасывается до значения 0**, а все **неинициализированный указатель переменные сбрасывается до 0x0000**.

Лабораторная Скамья Панель

Лабораторная Скамья Панель показывает 5-вольтовый 'Uno' Версия для Uno, который окружен набором 'I/O' Устройства, который вы можете выбрать / настроить и подключить к желаемому 'Uno' pins.

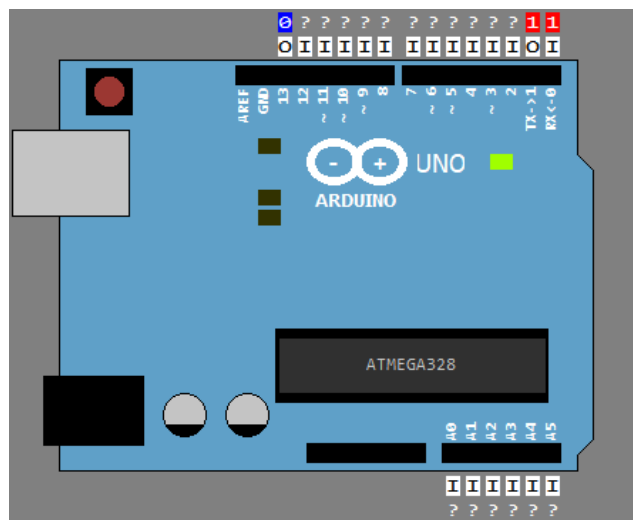
'Uno'

Это изображение 'Uno' Версия для Uno и его встроенных светодиодов. Когда вы загружаете новый программа в UnoArduSim, если он успешно анализирует, он подвергается "моделированию загрузка" в 'Uno', который имитирует поведение фактического 'Uno' Версия для Uno - вы увидите мигание серийного RX и TX LED (наряду с активностью в pins и 0, которые *встроенный для последовательной связи с главным компьютером*). За ним сразу следует вспышка pin 13 LED, которая обозначает сброс Версия для Uno и (и автоматическую остановку UnoArduSim в) начало загруженной программа выполнение. Вы можете избежать этого отображения и связанной с ним задержки загрузки, отменив выбор **Показать Загрузка** от **Конфигурировать | Предпочтения**,

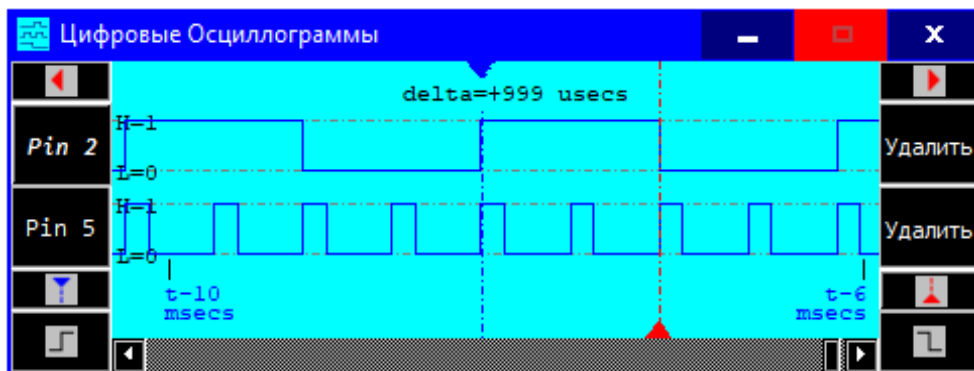
окно позволяет визуализировать логические уровни цифровой на всех 20 'Uno' pins ('1' на красный для 'HIGH', '0' на синем для 'LOW', а также '?' на сером для неопределенного неопределенного напряжения) и направления запрограммированный ('I' за 'INPUT', или же 'O' за 'OUTPUT'). Для pins, которые пульсируют с использованием ШИМ через 'analogWrite()' или 'tone()' или 'Servo.write()', цвет меняется на фиолетовый и отображаемый символ становится '^',

Обратите внимание, что **Цифровой pins 0 и 1 подключены через резисторы 1 кОм к USB-чипу для последовательная связь с хост-компьютером.**

В стороне: Цифровой pins 0-13 отображаются как симулятор pins 0-13, а аналоговый pins 0-5 отображаются как A0-A5. Чтобы получить доступ к аналоговый pin в вашей программа, вы можете обратиться к номеру pin одним из двух эквивалентных наборов чисел: 14-19; или A0-A5 (A0-A5 являются встроенный 'const' переменные, имеющий значения 14-19). И только при использовании 'analogRead()' доступен третий вариант - для этой инструкции вы можете удалить 'A' префикс из числа pin и просто используйте 0-5. Для доступа к pins 14-19 в вашей программа с помощью 'digitalRead()' или же 'digitalWrite()', вы можете просто обратиться к этому номеру pin или вместо этого использовать псевдонимы A0-A5.













Левой кнопкой мыши на любом 'Uno' pin откроет **Pin Цифровые Осциллограммы** окно, который отображает прошлое **стоимость одной секунды** из **Активность уровня цифровой** на этом pin. Вы можете нажать на другой pins, чтобы добавить их на дисплей Pin Цифровые Осциллограммы (до 4 сигналов одновременно).



Нажмите для просмотра страницы влево или вправо или используйте клавиши Home, PgUp, PgDn, End

Один из отображаемых сигналов будет **активный pin** осциллограмма На это указывает ее кнопка "Pin", отображаемая как нажатая (как на приведенном выше снимке экрана Pin Цифровые Осциллограммы). Вы

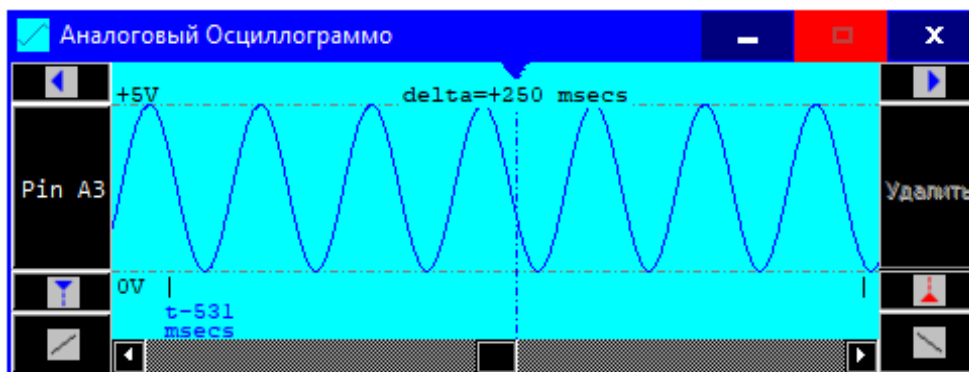
можете выбрать осциллограмма, нажав его цифровую кнопку Pin, а затем выбрать интересующую полярность кромки, нажав соответствующую кнопку выбора полярности восходящего / спадающего края,  , или же  или с помощью сочетаний клавиш **стрелка вверх** а также **стрелка вниз** , Вы можете тогда **Прыгать** активный курсор (синие или красные линии курсора с указанным их дельта-временем) назад или вперед к краю выбранной полярности цифровой **этого активного pin** осциллограмма с помощью кнопок курсора ( ,  или же  ,  (в зависимости от того, какой курсор был активирован ранее с  или же ), или просто используйте клавиши клавиатуры \leftarrow а также \rightarrow ,

Чтобы активировать курсор, нажмите его цветную кнопку активации ( или же  показано выше) - *это также прокручивает представление к текущему местоположению этот курсор* , Кроме того, вы можете быстро чередовать активацию между курсорами (с их соответственно центрированными видами), используя ярлык **'Tab'** ключ.







Вы можете **Прыгать** текущий активированный курсор **щелкнув левой кнопкой мыши в любом месте** в осциллограмма на экране просмотра региона. Кроме того, вы можете выбрать красную или синюю линию курсора, щелкнув справа сверху (чтобы активировать ее), затем **перетащите его в новое место** и отпустите. Когда нужный курсор находится где-то вне экрана, вы можете **щелкните правой кнопкой мыши в любом месте** чтобы перейти к этому новому экранному местоположению. Если оба курсора уже находятся на экране, щелчок правой кнопкой мыши просто чередует активированный курсор.

Чтобы ZOOM IN и ZOOM OUT (масштаб всегда центрирован на АКТИВНОМ курсоре), используйте колесо мыши или сочетания клавиш CTRL-стрелка вверх и CTRL-стрелка вниз,

Делать вместо **щелкните правой кнопкой мыши на любом 'Uno' pin** открывает **Pin Аналоговый Осциллограммо** окно, который отображает **за одну секунду стоит** из **Активность уровня аналоговый** на этом pin. В отличие от Pin Цифровые Осциллограммы окно, вы можете отображать активность аналоговый только на одном pin одновременно.



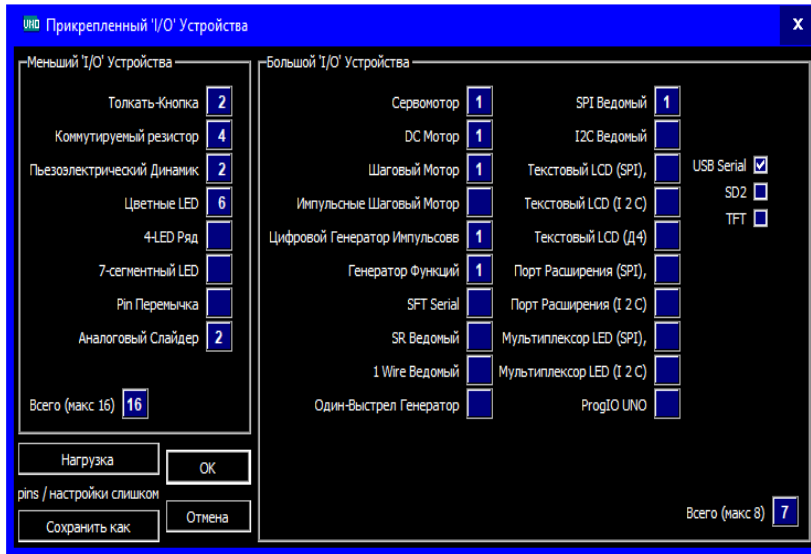
Нажмите для просмотра страницы влево или вправо или используйте клавиши Home, PgUp, PgDn, End

Вы можете **Прыгать** the синие или красные линии курсора к следующей поднимающейся или опускающейся "точке наклона" с помощью кнопок со стрелками вперед или назад ( ,  или же  ,  , снова в зависимости от активированного курсора, или используйте \leftarrow а также \rightarrow клавиш) в сочетании с кнопками выбора подъема / спада  ,  ("Точка наклона" возникает, когда напряжение аналоговый проходит через высокий порог логического уровня цифровой ATmega pin). Кроме того, вы можете снова щелкнуть, чтобы прыгать, или перетащить эти линии курсора, аналогичные их поведению в Pin Цифровые Осциллограммы окно

прессование **'Ctrl-S'** внутри **любой окно позволяет сохранить осциллограмма (X, Y) данные** на текст файл по вашему выбору, где **Икс** в микросекундах с левой стороны, и **Y** в болтах.

'I/O' Устройства

Ряд различных Устройства окружают 'Uno' по периметру **Лабораторная Скамья Панель**, "Маленькие" 'I/O' Устройства (из которых вам разрешено до 16) находятся вдоль левой и правой сторон Панель. "Большой" 'I/O' Устройства (из которых вам разрешено до 8) имеют "активные" элементы и расположены вдоль верхней и нижней части **Лабораторная Скамья Панель**, Желаемое количество каждого типа доступных 'I/O' устройство можно установить с помощью меню **Конфигурировать | 'I/O' Устройства**,



Каждый 'I/O' устройство имеет одно или несколько вложений pin, показанных как **два-цифра** Номер pin (00, 01, 02,... 10,11,12, 13 и либо A0-A5, или 14-19, после этого) в соответствующем поле ввода. Для номеров pin со 2 по 9 вы можете просто ввести один цифра - ведущий 0 будет предоставлен автоматически, но для pins 0 и 1 вы должны сначала ввести ведущий 0. Входы *обычно* на левой стороне 'I/O' устройство, и выходы *обычно* справа (если позволят). Все 'I/O' Устройства будут реагировать непосредственно на уровни pin и изменения уровня pin, поэтому будут реагировать либо на библиотеку функциональные модули, нацеленную на их подключенный pins, либо на запрограммированный 'digitalWrite()' (для "битовой" операции).

Вы можете подключить несколько Устройства к одному ATmega pin до тех пор, пока это не создаст **электрический конфликт**, Такой конфликт может быть создан либо 'Uno' pin как 'OUTPUT' вождение с подключением устройство с сильной проводимостью (низким сопротивлением) (например, с выходом 'FUNCGEN' или 'MOTOR' **Енс** выход) или двумя подключенными Устройства, конкурирующими друг с другом (например, и кнопка 'PULSER' и 'PUSH' -, прикрепленные к одному и тому же pin). Любой такой конфликт будет иметь катастрофические последствия в реальной аппаратной реализации и поэтому будет запрещен, и будет помечен для пользователя через всплывающее окно сообщения).

Диалоговое окно может использоваться, чтобы позволить пользователю выбирать типы и номера требуемого 'I/O' Устройства. Из этого диалогового окна вы также можете **Сохранить** 'I/O' Устройства для текста файл и / или **Нагрузка** 'I/O' Устройства из ранее сохраненного (или отредактированного) текста файл (**включая все соединения pin, интерактивные настройки и любые введенные значения в поле ввода**).

Обратите внимание, что начиная с версии 1.6 значениям в полях редактирования периода, задержки и ширины импульса в соответствующем IO Устройства может быть присвоен суффикс 'S' (или 's'). Это указывает на то, что они должны быть **масштабируется** в соответствии с позицией глобального 'I/O ____S' ползунок, который появляется в главном окне **Инструмент-Bar**, С этим ползунком вправо масштабный коэффициент равен 1,0 (единица), и с ползунком полностью влево масштабный коэффициент равен 0,0 (с учетом минимальных значений, установленных каждым конкретным 'I/O' устройство). Вы можете масштабировать более одного значения поля ввода **одновременно** используя этот слайдер. Эта функция позволяет перетаскивать ползунок во время выполнения легко эмулировать изменение длительности импульсов, периодов и задержек для подключенных 'I/O' Устройства.

В оставшейся части этого раздела приведены описания для каждого типа устройство.

Несколько из этих Устройства **поддержка масштабирования их введенных значений** с помощью ползунка на основной окно **Инструмент-Bar**, Если значение устройство имеет букву 'S' в качестве суффикса, его значение будет умножено на коэффициент масштабирования (между 0,0 и 1,0), который определяется положением ползунка, с учетом ограничения минимального значения устройство (1.0 полностью вправо, 0.0 полностью влево) - смотрите 'I/O ____S' под каждым шлангом Устройства подробно описан ниже.



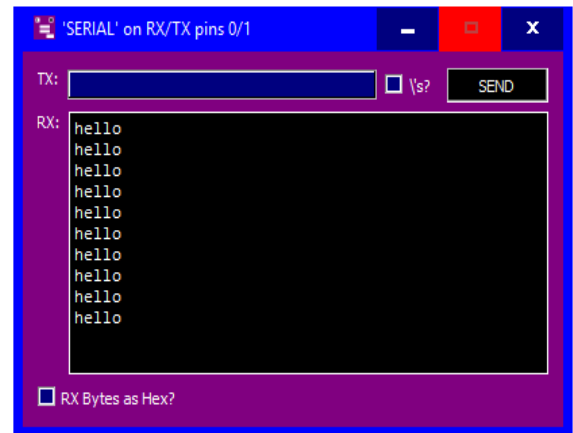
Serial Монитор ('SERIAL')



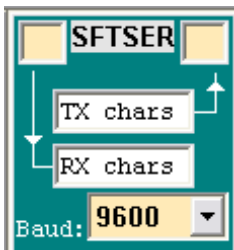
Этот 'I/O' устройство позволяет аппаратно-опосредованный последовательный ввод и вывод ATmega (через USB-чип 'Uno') на 'Uno' pins 0 и 1. скорость передачи устанавливается с помощью раскрывающегося списка внизу - выбранный скорость передачи **должен соответствовать** значению, которое ваш программа передает '`Serial.begin()`' функциональный модуль для правильной передачи / приема. *Последовательная связь фиксируется на 8 битах данных, 1 стоповом бите и без битов четности.* Вам разрешено **Отключить** (Пусто) **но не заменить** TX pin 00, но не RX pin 01.

Чтобы отправить ввод с клавиатуры на программа, введите один или несколько символов в верхнем (символы TX), отредактируйте окно, а затем ударь **'Enter'** клавиша на клавиатуре, (символы выделяются курсивом, чтобы указать, что передачи начались) - или, если они уже выполняются, добавленные печатные символы будут выделены курсивом. Затем вы можете использовать '`Serial.available()`' а также '`Serial.read()`' функциональные модули для чтения символов в порядке их поступления в буфер pin 0 (крайний левый набранный символ будет отправлен первым). Отформатированные текстовые и числовые распечатки или неформатированные байтовые значения могут быть отправлены на нижний вывод консоли (RX-символы) окно, вызвав Arduino. '`print()`', '`println()`', или же '`write()`' функциональные модули.

Дополнительно, **окно большего размера для установки / просмотра символов TX и RX можно открыть, дважды щелкнув (или щелкнув правой кнопкой мыши) на этом 'SERIAL' устройстве**, Этот новый окно имеет увеличенное поле редактирования символов TX и отдельную кнопку 'Send', на которую можно нажать, чтобы отправить символы TX на 'Uno' (на pin 0). Существует также опция флажка для повторной интерпретации последовательностей символов с обратной косой чертой, таких как '`\n`' или же '`\t`' для необработанного отображения.



Программное Serial ('SFTSER')

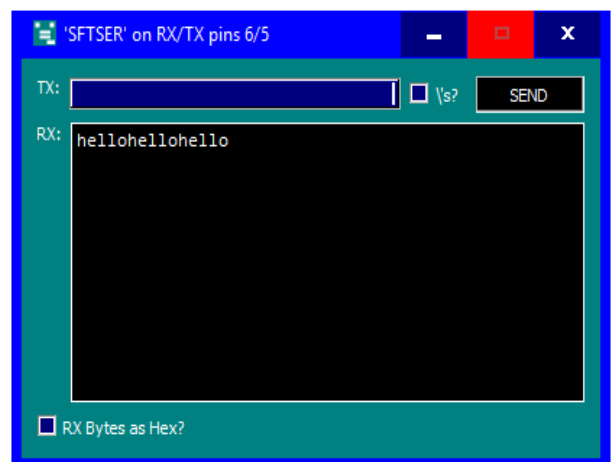


Этот 'I/O' устройство позволяет использовать библиотеку с программным обеспечением или, в качестве альтернативы, последовательный ввод и вывод пользователя с "побитовой последовательностью" на любой паре 'Uno' pins, которую вы хотите заполнить **кроме** pins 0 и 1, которые предназначены для аппаратного обеспечения 'Serial' связи). Ваш программа должен иметь '`#include <SoftwareSerial.h>`' строка рядом с верхом, если вы хотите использовать функциональные возможности этой библиотеки. Как и в случае аппаратного 'SERIAL' устройство, скорость передачи для 'SFTSER' устанавливается с помощью раскрывающегося списка внизу - выбранный скорость передачи должен соответствовать значению, которое программа передает в

'`begin()`' функциональный модуль для правильной передачи / приема. *Последовательная связь фиксируется на 8 битах данных, 1 стоповом бите и без битов четности.*

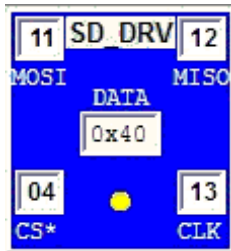
Кроме того, как с аппаратным 'SERIAL', **окно большего размера для настройки и просмотра TX и RX можно открыть, дважды щелкнув (или щелкнув правой кнопкой мыши) на SFTSER устройство**,

Обратите внимание, что в отличие от аппаратной реализации 'Serial' нет предоставленного TX-буфер поддерживается внутренними операциями прерывания ATmega (только RX-буфер), поэтому тот '`write()`' (или же '`print()`') вызовы блокируются (т. е. ваш программа не будет продолжаться, пока они не будут завершены).



SD-Диск ('SD_DRV')

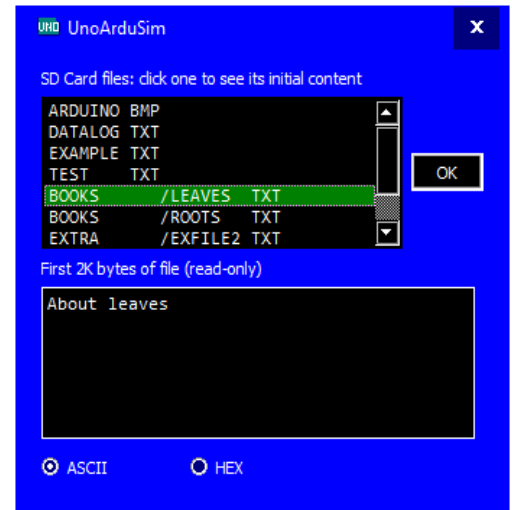
Этот 'I/O' устройство позволяет для библиотеки программного обеспечения (но **не** "побитый бит") файл операции ввода и вывода на 'Uno' SPI pins (вы можете выбрать, какой **CS *** pin вы будете использовать). Ваш программа может просто `#include <SD.h>` линия около вершины, и вы можете использовать `<SD.h>` функциональные модули ИЛИ позвоните напрямую `SdFile` функциональные модули самостоятельно.



окно большего размера с каталогами и файлы (и содержимым) можно открыть, дважды щелкнув (или щелкнув правой кнопкой мыши) на 'SD_DRV' устройство. Все содержимое диска *загружен из* an SD подкаталог в загруженном каталоге программа (если он существует) в `SdVolume::init()`, *и отражается в* то же самое SD подкаталог на файл `close()`, `remove()` и на

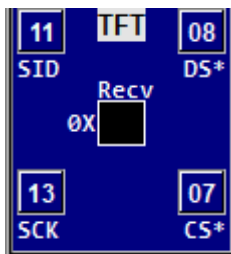
`makeDir()` а также `rmDir()`,

Желтый LED мигает во время передачи SPI, и 'DATA' показывает последний 'SD_DRV' *ответ* байт. Все сигналы SPI точны и могут быть просмотрены в **Осциллограмма окно**,



TFT дисплей ('TFT')

Это 'I/O' устройство эмулирует Adafruit™ TFT-дисплей размера 1280by-160 пикселей (в своем нативном вращении = 0, но при использовании библиотеки 'TFT.h', `TFT.begin()` Наборы для инициализации вращения = 1, который дает "пейзаж" вид 160-по-128 пикселей). Вы можете толкнул эту устройство, вызвав функциональные модули из `TFT.h` библиотека (которая первой требует `#include <TFT.h>`), или вы можете использовать систему SPI для отправки вам собственную последовательность байтов толкнул его.

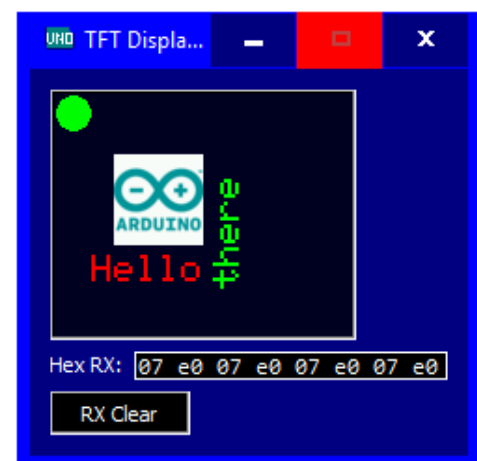


TFT всегда подключен в 'SPI' pins 'MOSI' (для 'SID') и 'SCK' (для 'SCK') - те, которые не могут быть изменены. 'DS*' pin для Данные / команды выбора ('LOW' выбирает режим передачи данных), и 'CS*' pin является активным низким микросхема-выбор

Существует нет Сброс pin не предусмотрено, так что вы не можете сделать аппаратный сброс это устройство при движении с низким pin (как `TFT::begin()` функциональный

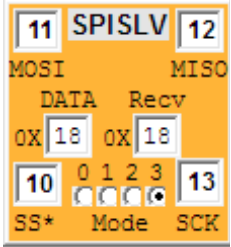
модуль пытается сделать, когда вы прошли действительный номер 'reset' pin в качестве третьего параметра к `TFT(int cs, int ds, int rst)` конструктор). устройство тем не менее есть скрытое подключение к системе Сброс линии, поэтому она сбрасывает себя каждый раз, когда вы нажимаете главный UnoArduSim Сброс значок на панели инструментов, или кнопку 'Uno' Версия для Uno сброса ..

По **двойной щелчок** (или **щелкнув правой кнопкой мыши**) На этом устройство, большой окно открывается, чтобы показать, что полный 160-на-128 пикселей, ЖК-дисплей, наряду с наиболее недавно получили 8 байт (как показано ниже)



Конфигурируемый SPI Ведомый ('SPISLV')

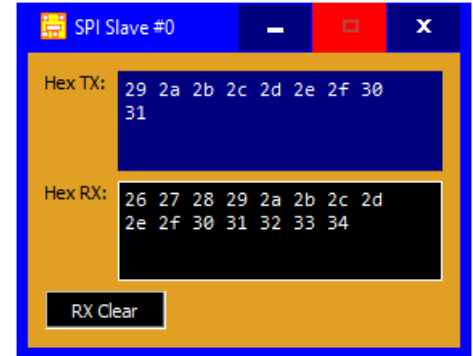
Этот 'I/O' устройство эмулирует ведомый SPI выбранного режима с активным низким **SS *** ("выбор ведомого") pin управляет **MISO** выход pin (когда **SS *** в приоритете, **MISO** это не толкнул). Ваш программа должен иметь '#include <SPI.h>' линии, если вы хотите использовать функциональность встроенный SPI Arduino объект и библиотеки. В качестве альтернативы, вы можете создать свой собственный "битовый удар" **MOSI** а также **SCK** сигналы к толкнул это устройство.



устройство ощущает краевые переходы на своем **CLK** ввод в соответствии с выбранным режимом ('MODE0' , 'MODE1' , 'MODE2' , или же 'MODE3'), который должен быть выбран в соответствии с режимом SPI запрограммированный вашего программа.

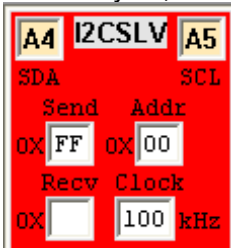
Двойной щелчок (или щелчок правой кнопкой мыши) на устройство позволяет открыть более крупный спутник окно что вместо позволяет вам Вы должны заполнить 32-байтовый

максимальный буфер (чтобы эмулировать SPI Устройства, который автоматически возвращает свои данные), и увидеть последние 32 полученных байта (все как шестнадцатеричные пары). **Обратите внимание, что** следующий байт буфера TX автоматически отправляется только на 'DATA' после полный 'SPI.transfer()' ' завершено!



Двухпроводная I2C Ведомый ('I2CSLV')

Этот 'I/O' устройство только эмулирует *ведомый режим* устройство. устройство может быть назначен адрес шины I2C, используя запись в два шестнадцатеричных цифра в поле ввода 'Addr' (он будет отвечать только на I2C автобусные операции с использованием назначенного ему адреса). устройство отправляет и получает данные на своем открытом канале (только для раскрывающегося списка) **SDA** pin, и отвечает на сигнал тактовой частоты шины на своем открытом канале (только для раскрывающегося списка) **SCL** pin. Хотя 'Uno' будет хозяином шины, ответственным за генерацию **SCL** сигнал, этот раб устройство также будет тянуть **SCL** низкий во время своей низкой фазы, чтобы увеличить (если это необходимо) низкое время шины до значения, соответствующего ее внутренней скорости (которую можно установить в поле редактирования 'Clock').

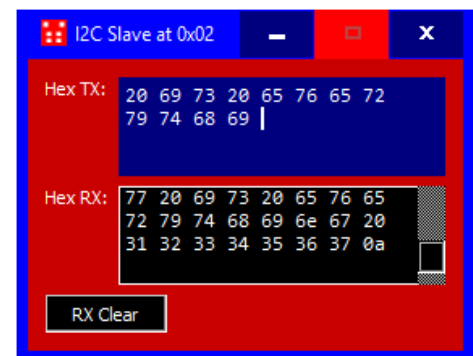


Ваш программа должен иметь '#include <Wire.h>' линии, если вы хотите использовать функциональность 'TwoWire' библиотека для взаимодействия с этим устройство. В качестве альтернативы, вы можете создать свои собственные данные с синхронизацией битов и тактовые сигналы для толкнул этого подчиненного устройства устройство.

Один байт для передачи обратно на ведущее устройство 'Uno' может быть установлен в поле редактирования 'Send', а один (самый последний полученный) байт может быть просмотрен в его (только для чтения) окне редактирования 'Recv'. **Обратите внимание, что** значение поля редактирования 'Send' всегда отражает следующий байт для передачи из этого внутреннего буфера данных устройство.

Двойной щелчок (или щелчок правой кнопкой мыши) на устройство позволяет открыть более крупный спутник окно

вместо этого это позволяет вам заполнить буфер FIFO с максимальным размером 32 байта (чтобы эмулировать TWI Устройства с такой функциональностью) и просматривать (максимум до 32) байтов самых последних полученных данных (как два hex-цифра отображает 8 байтов на строку). Количество строк в этих двух полях редактирования соответствует выбранному размеру буфера TWI (который можно выбрать с помощью **Конфигурировать | Предпочтения**). Это было добавлено в качестве опции, так как Arduino 'Wire.h' библиотека использует **5** такие буферы ОЗУ в своем коде реализации, который является дорогой оперативной памяти. Отредактировав установку Arduino 'Wire.h' файл для изменения определенной константы 'BUFFER_LENGTH' (а также редактирование компаньона 'utility/twi.h' файл поменять Длина буфера TWI) оба должны быть вместо 16 или 8, пользователь *мог* значительно сократить объем оперативной памяти 'Uno' в целевом **аппаратная реализация** - поэтому UnoArduSim отражает эту реальную возможность через **Конфигурировать | Предпочтения** ,

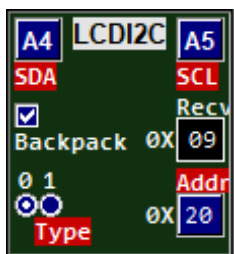


Текстовый LCD I2C ('LCDI2C')

Это 'I/O' устройство эмулирует 1,2, 04 4-строчный символьный LCD, в одном из трех режимов:

- 1) тип 0 (Adafruit стиль расширителя порта с аппаратным обеспечением, имеющим I2C адрес шины 0x20-0x27)
- 2) тип рюкзака 1 (DFRobot стиль порт расширитель с аппаратным, имеющими адрес шины I2C 0x20-0x27)
- 3) нет рюкзака (Собственного режим интегрированного интерфейса I2C, имеющего 2C адрес шины 0x3C-0x3F)

Поддержка библиотеки коды для каждого режима устройство была предусмотрена внутри 'include_3rdParty' папки UnoArduSIm каталога установки: 'Adafruit_LiquidCrystal.h', 'DFRobot_LiquidCrystal.h', и 'Native_LiquidCrystal.h' Соответственно.



устройство может быть назначен любой адрес I2C шины, используя вход два-гекс-цифра в его 'Addr' редактировать ящик (он будет отвечать только на I2C Операции с участием автобусов назначенного адреса). устройство принимает шину адреса и данных (и отвечает ACK = 0 или NAK = 1) на его открытом стоке (выпадающий только) **SDA** pin. Вы можете только записи ЖК-команды и данные DDRAM - вы **не можешь** считаны данные из письменных мест

DDRAM.

Двойной клик или **щелкните правой кнопкой мыши** чтобы открыть ЖК-экран монитора окно, из которого вы можете также установить размер экрана и набор символов.



Текстовый LCD SPI ('LCSPI')

Это 'I/O' устройство эмулирует 1,2, 04 4 строки символьный LCD, в одном из двух режимов:

- 1) (Adafruit стиль SPI порт расширитель)
- 2) нет рюкзака (Родного режим не интегрирована SPI интерфейс - как показано ниже)

Поддержка библиотеки коды для каждого режима устройство была предусмотрена внутри 'include_3rdParty' папки UnoArduSIm каталога установки: 'Adafruit_LiquidCrystal.h', и 'Native_LiquidCrystal.h' Соответственно.



Pin 'SID' является последовательной передачи данных в, 'SS' является активным низким устройство-выбрать, 'SCK' это часы pin и 'RS' это данные / команда pin. Вы можете только записи ОК-команды и данные DDRAM (все SPI транзакция запись) - вы **не можешь** считаны данные из письменных мест DDRAM.

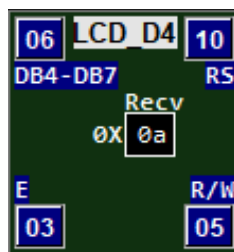
Двойной клик или **щелкните правой кнопкой мыши** чтобы открыть ЖК-экран монитора окно, из

которого вы можете также установить размер экрана и набор символов.



Текстовый LCD D4 ('LCD_D4')

Это 'I/O' устройство эмулирует 1,2, 04 4 строки символьные КИ, имеющие 4-битовый параллельный интерфейс шины. байтов данных записываются / читать **две половины** по его данным 4 pins 'DB4-DB7' (где окно редактирования содержит с **наименьшим номером из его 4 последовательных чисел pin**), - данные синхронизируются по заднему фронту на 'E' (включить) pin, с направлением данных под контролем 'R/W' pin и ЖК данных / команд режима по 'RS' pin.



Поддержка библиотеки кода было предусмотрено внутри 'include_3rdParty' папка вашего UnoArduSIm каталога установки:

'Adafruit_LiquidCrystal.h', и

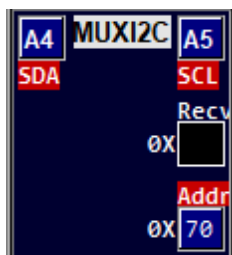
'Native_LiquidCrystal.h' обе работы.

Двойной клик или **щелкните правой кнопкой мыши** чтобы открыть ЖК-экран монитора окно, из которого вы можете также установить размер экрана и набор символов.



Мультиплексор LED I2C ('MUXI2C')

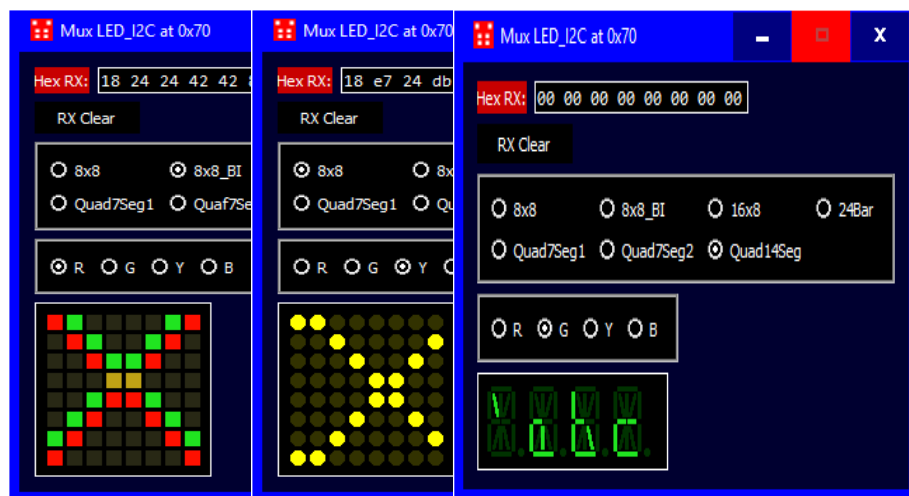
Это 'I/O' устройство эмулирует I2C-интерфейсом контроллера HT16K33 (ч адрес шины I2C AVING 0x70-0x77) которому один из нескольких различных типов мультиплексированных LED дисплеев могут быть присоединено:



- а) 8x8 или 16x8 LED массива
- б) 8x8 двухцветные LED массива
- в) 24-двухцветный-LED бар
- г) два стиля 4-цифра дисплеев 7-сегментных
- е) один 4-цифра 14-сегментный буквенно-цифровой дисплей

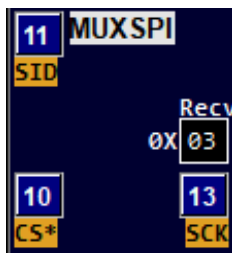
Все поддерживаются 'Adafruit_LEDBackpack.h' код, предусмотренный внутри 'include_3rdParty' папка:

Двойной клик (Или правая кнопка мыши) чтобы открыть увеличенную окно выбирать и просматривать один из нескольких цветных LED дисплеи.

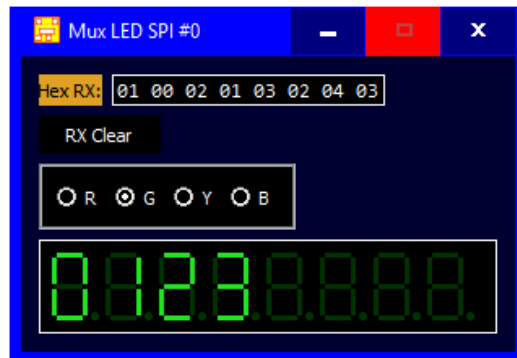


Мультиплексор LED SPI ('MUXSPI')

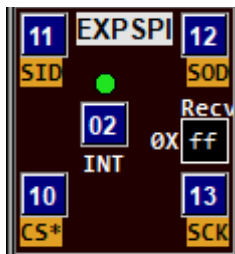
Мультиплексированный-LED контроллер на основе MAX6219, с поддержкой 'MAX7219.h' код, предоставленный в папке 'include_3rdParty' для толкнул до восьми 7-сегментных цифр.



Двойной клик (Или правая кнопка мыши) чтобы открыть увеличенную окно смотреть цветной 8-цифра 7-сегментные- дисплей.



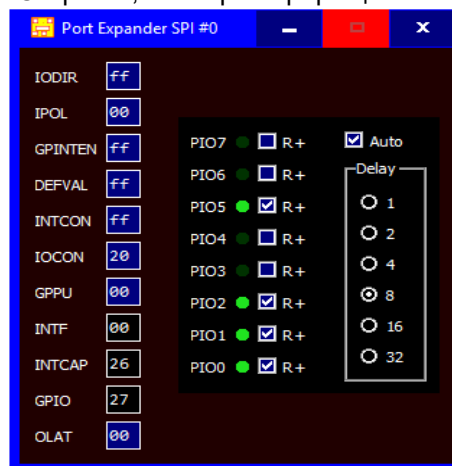
Порт Расширения SPI ('EXPSPi')



8-разрядный порт расширитель на основе MCP23008, с поддержкой 'MCP23008.h' код при условии, внутри 'include_3rdParty' папки. Вы можете написать MCP23008 регистры, и читать обратно GPIO pin уровни. Прерывания могут быть включены при каждом изменении GPIO pin - сработавшей прерывание будет толкнул в 'INT' pin.

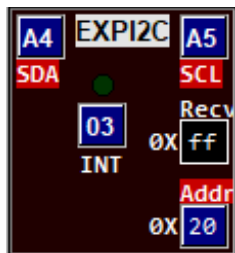
Двойной клик (Или правая кнопка мыши) открыть **большая окно чтобы увидеть 8** GPIO порта линии, и приложенные подтягивающие резисторы. Вы можете изменить подтягивания вручную, нажав кнопку, или прикрепить счетчик, который будет периодически изменять их в до подсчета способом. Скорость, с которой приращение

счета определяется масштаб вниз задержками фактор, выбранный пользователем (а 1x множитель соответствует один шаг примерно каждые 30 миллисекунд, более высокие коэффициенты задержки дают более медленный до подсчета скорости)

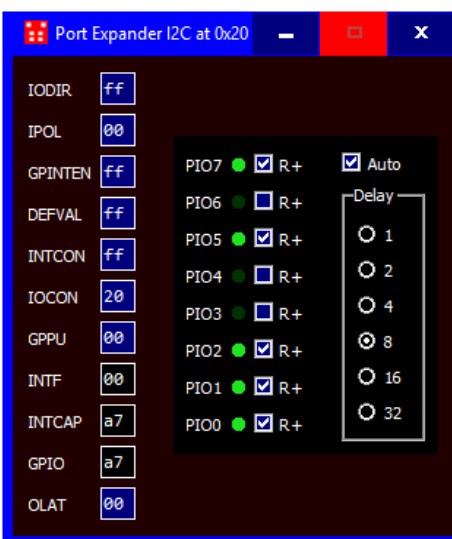


Порт Расширения I2C ('EXPI2C')

8-разрядный порт расширитель на основе MCP23008, с поддержкой 'MCP23008.h' код, предусмотренный внутри 'include_3rdParty' папки. Возможности соответствовать 'EXPSPi' устройство.



Двойной клик (Или правая кнопка мыши) чтобы открыть увеличенную окно а сюда в 'EXPSPi' устройство.



'1-Wire' Вedomый ('OWIISLV')

Этот 'I/O' устройство эмулирует один из небольшого набора шины '1-Wire' Устройства, подключенного к pin OWIO. Вы можете создать шину '1-Wire' (с одним или несколькими из этих ведомых '1-Wire' Устройства) на 'Uno' pin по вашему выбору. Эту кабину устройство можно использовать, позвонив 'OneWire.h' библиотека функциональные модули после размещения '#include <OneWire.h>' линия в верхней части вашего программа. В качестве альтернативы вы также можете использовать побитовые сигналы на OWIO для этого устройство (хотя это очень сложно сделать правильно, не вызывая электрический конфликт - такой конфликт все еще возможен даже при использовании 'OneWire.h' функциональные модули, но такие конфликты сообщаются в UnoArduSim).



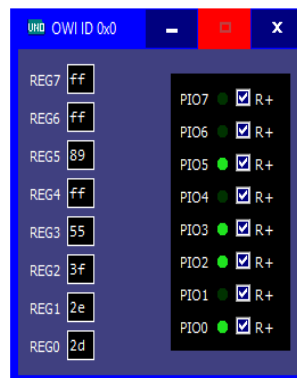
Каждый реальный OWISLV устройство должен иметь уникальный 8-байтовый i (64-битный!) внутренний серийный номер - в UnoArduSim это упрощается пользователем, предоставляя короткий 1-байтовый шестнадцатеричный 'ID' значение (которое назначается последовательно по умолчанию при загрузке / добавлении устройства), плюс 'Fam' Семейный код для этого устройство. UnoArduSim распознает небольшой набор кодов семейства с V2.3 (0x28, 0x29, 0x3A, 0x42), покрывающий датчик температуры, и параллельный ввод-вывод (PIO) Устройство (нераспознанный код семейства делает устройство универсальным 8-байтовым блоком устройство с универсальным датчиком.

датчиком.

Если у семейства устройство нет регистров PIO, регистры **D0** а также **D1** представлять первые два байта блокнота, иначе они представляют PIO Регистр "status" (фактические уровни pin) и регистр данных фиксатора PIO pin соответственно.

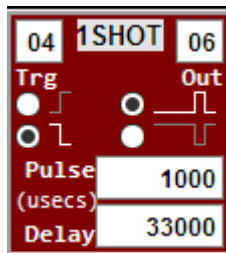
От **двойной щелчок** (или же **щелкнув правой кнопкой мыши**) на устройство, больше **OWIMonitor** окно открыт. Из этого большего окно вы можете проверить все регистры устройство, изменить местоположения блокнота SCR0 и SCR1, используя правки и ползунок (опять же, SCR0 и SCR1 соответствуют только **D0** а также **D1** если PIO отсутствует), или установите внешние подтягивания pin PIO. Когда SCR0 и SCR1 редактируются, UnoArduSim запоминает эти отредактированные значения как пользовательское "предпочтение", представляющее начальное (начиная с Сброс) значение, представляющее значение со знаком, выводимое из устройство; датчик s - ползунок сбрасывается на 100% (масштабный коэффициент 1,0) во время редактирования. Когда ползунок впоследствии перемещается, 'signed' значение в SCR1 уменьшается в соответствии с положением ползунка (масштабный коэффициент от 1,0 до 0,0) - его функция позволяет вам легко проверить реакцию вашего программа на плавно изменяющиеся значения датчика. ,

Для устройство с PIO pins, когда вы устанавливаете флажки уровня pin, UnoArduSim запоминает эти проверенные значения как текущие подтягивания применяются к pins. Эти внешние значения подтягивания затем используются вместе с данными защелки pin (регистр **D1**), чтобы определить окончательные фактические уровни pin и зажечь или погасить зеленый LED, прикрепленный к PIO pin (только pin 'HIGH' если применяется внешнее подтягивание, **а также** соответствующий **D1** защелка немного '1').



Генератор с Один-Выстрел ('1SHOT')

Этот 'I/O' устройство эмулирует однократный выстрел цифровой, который может генерировать импульс выбранной полярности и ширины импульса на своем 'Out' pin, возникающий после указанной задержки от фронта запуска, полученного на его **Trg** (триггерный) вход pin. Как только указанный фронт запуска получен, начинается синхронизация, а затем новый импульс запуска не будет распознан, пока 'Out' пульт был произведен (и полностью закончен).



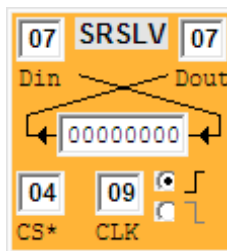
Одним из возможных применений этого устройство является моделирование датчики ультразвукового диапазона, которые генерируют импульс дальности в ответ на импульс запуска. Его также можно использовать везде, где вы хотите сгенерировать входной сигнал pin, синхронизированный (после выбранной вами задержки) с выходным сигналом pin, созданным вашим программа.

'Pulse' и значения 'Delay' могут быть масштабированы из основного окно **Инструмент-Bar** Управление ползунком 'I/O ____S' с добавлением суффикса 'S' (или 's') или один (или оба).

Другое использование этого устройство - тестирование программа, использующего прерывания, и вы хотели бы посмотреть, что произойдет, если **специальная инструкция программа** прерывается. Временно отключите 'I/O' Устройство, который вы подключили к pin 2 (или pin 3), и замените его на '1SHOT' устройство, к которому 'Out' pin подключен к 'Uno' pin 2 (или pin3, соответственно), затем вы можете запустить его вход 'Trg' (при условии установки чувствительности переднего фронта), вставив пара инструкций { `'digitalWrite(LOW)'` , `'digitalWrite(HIGH)'` } **только до** инструкции, внутри которой вы хотите, чтобы прерывание произошло. Установите 1SHOT; с 'Delay' рассчитывать время импульса, генерируемого на 'Out', внутри инструкции программа, которая следует за этой парой инструкций запуска. Обратите внимание, что некоторые инструкции маскируют прерывания (например, `'SoftwareSerial.write(byte)'` , аи так не может быть прервано.

Регистр Сдвига Ведомый ('SRSLV')

Этот 'I/O' устройство эмулирует простой регистр сдвига устройство с активным низким **SS *** ("выбор ведомого") pin управляет '**Dout**' выход pin (когда **SS *** в приоритете, '**Dout**' это не толкнул). Ваш программа может использовать функциональные возможности встроенный SPI Arduino объект и библиотеки. В качестве альтернативы, вы можете создать свой собственный "битовый удар" '**Din**' а также **CLK** сигналы к толкнул это устройство.



устройство ощущает краевые переходы на своем **CLK** вход, который запускает сдвиг своего регистра - полярность воспринимается **CLK** край может быть выбран с помощью радио-кнопки управления. На каждом **CLK** край (воспринимаемой полярности), регистр захватывает его **шум** уровень в позицию младшего значащего бита (LSB) регистра сдвига, поскольку оставшиеся биты одновременно сдвигаются влево на одну позицию в направлении позиции MSB. Всякий раз, когда **SS *** низкое, текущее значение в позиции MSB регистра сдвига толкнул на '**Dout**' ,

Программируемый 'I/O' Устройство ('PROGIO')



Этот 'I/O' устройство на самом деле является чистым 'Uno' Версия для Uno, который вы можете программа (с отдельным программа), чтобы эмулировать 'I/O' устройство чье поведение вы можете полностью определить. Вы можете выбрать до четырех pins (IO1, IO2, IO3 и IO4), которые этот подчиненный 'Uno' будет совместно использовать с главным (основным) Uno, который появляется в середине вашего **Лабораторная Скамья Панель**, Как и в случае с другим Устройства, любой электрический конфликт между этим ведомым 'Uno' и ведущим 'Uno' будет обнаружен и помечен. Обратите внимание, что все общие pins являются **непосредственно** связано, **кроме pin 13** (где между двумя pins предполагается

последовательный резистор R-1K, чтобы предотвратить электрический конфликт на Сброс, у этого ведомого 'Uno' не может быть '**I/O**' Устройства своего. На рисунке слева показаны 4 соединения pin, которые будут 4 pins системы SPI (**SS ***, **MISO**, **MOSI**, **SCK**) - это позволит вам программа это ведомое устройство как универсальное ведомое устройство (или ведущее устройство) SPI, чье поведение вы определить.

От **двойной щелчок** (или же **щелкнув правой кнопкой мыши**) на этом устройстве открывается большой экран, чтобы показать, что у этого ведомого 'Uno' есть свой **Код Панель** а также связанный **Переменные Панель**. Так же, как мастер 'Uno' имеет. У него также есть свой **Инструмент-бар**, который вы можете использовать для **нагрузка** а также **управление выполнению** подчиненного программа - действия пиктограмм имеют те же сочетания клавиш, что и в главном окне. (**Нагрузка** является **Ctrl-L**, **Сохранить** является **Ctrl-S** так далее.). После загрузки вы можете Выполнить из **или** Основной UnoArduSim окно или изнутри этого монитора Ведомый экран - в любом случае Main 'Uno' программа и Ведомый 'Uno' программа остаются заблокированными в синхронизации с прохождением в реальном времени, так как их исполнения продвигаются вперед. **Чтобы выбрать Код Панель, который будет иметь загрузка, поиск и фокус выполнение**, щелчок на его родительская строка заголовка окна - нефокусированный Код Панель имеет свою панель инструментов действия затенены ,

Некоторые возможные идеи для ведомого Устройства, который может быть запрограммированный в этот 'PROGIO' устройство, перечислены ниже. Для последовательной эмуляции I2C или SPI устройство можно использовать соответствующее кодирование программа с массивы для буферов отправки и получения, в порядке для имитации сложного поведения устройство:

а) Ведущий или ведомый SPI устройство. UnoArduSimV2.4. Продлил 'SPI.h' библиотека, чтобы разрешить режим Slave SPI через необязательный 'mode' параметр в 'SPI.begin(int mode = SPI_MSTR)', Явно передать 'SPI_SLAVE' выбрать подчиненный режим (вместо использования основного режима по умолчанию). Теперь вы также можете определить пользовательское прерывание функциональный модуль (давайте назовем это 'onSPI') в любом 'Uno' для передачи байтов путем вызова другое добавленное расширение 'SPI.attachInterrupt(user_onSPI)', Сейчас с Alling 'rxbyte=SPI.transfer(tx_byte)' изнутри вашего 'user_onSPI' функциональный модуль очистит флаг прерывания и вернуть немедленно с только что полученным байтом в вашей переменной 'rxbyte', Кроме того, вы можете избежать прикрепления прерывания SPI и вместо этого просто позвоните 'rxbyte=SPI.transfer(tx_byte)' изнутри вашего основного программа - этот вызов будет блок выполнение пока байт SPI не был передан, и будет затем вернуть с недавно полученным байтом внутри 'rxbyte',

б) общий последовательный 'I/O' устройство. Вам нужно будет общаться между одним 'SoftwareSerial' определенный в вашем главном 'Uno' программа и другой, созданный в ведомом 'Uno' программа - они должны использовать противоположно определенный 'txpin' а также 'rxpin' значения, так что раб 'Uno' 'SoftwareSerial' получает на pin, на котором мастер 'SoftwareSerial' передает (вы не можете установить аналогичное соединение, используя the 'Serial' порт на pins 0 и 1 на обеих платах, так как два 'Serial' потоки было бы толкнул их TX сигналы друг против друга).

в) Общий ведущий или подчиненный 'I2C' устройство. Операция Ведомый была добавлена для завершения UnoArduSim; реализация 'Wire.h' библиотека (функциональные модули 'begin(address)', 'onReceive()' а также 'onRequest' теперь были реализованы для поддержки операций в подчиненном режиме).

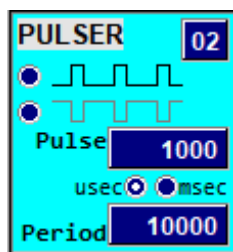
г) универсальный цифровой пульсатор, С помощью 'delayMicroseconds()' а также 'digitalWrite()' звонки внутри 'loop()' в вашем 'PROGIO' программа, вы можете определить 'HIGH' а также 'LOW' интервалы пульса. Добавляя отдельный 'delay()' позвоните в свой 'setup()' функциональный модуль, вы можете отложить запуск этой последовательности импульсов. Вы даже можете изменить ширину импульса как время прогрессирует с помощью счетчика переменная. Вы также можете использовать отдельный 'IOx' pin как триггер для запуска синхронизация эмулируемого '1Shot' (или двойного, тройного и т. Д.) устройство, и вы можете контролировать производимую ширину импульса, чтобы она изменялась любым желаемым образом с течением времени.

д) случайный сигнализатор. Это вариация на цифровой пульсатор который также использует звонки 'random()' а также 'delayMicroseconds()' генерировать случайные моменты времени, в которые 'digitalWrite()' сигнал на любом выбранном pin, совместно используемом с ведущим 'Uno'. Используя все четыре 'IOx' pins допускает четыре одновременных (и уникальных) сигнала.

д) Универсальный 'bit-banging' устройство. Создайте любые битовые или тактовые комбинации, которые вы хотите, чтобы сигналы толкнул поступали в любую подсистему на главном устройстве 'Uno'. И запомни, **Вы можете использовать любого раба 'Uno' программа инструкции или подсистемы, которые вы хотите**,

Цифровой Генератор Импульсов ('PULSER')

Это 'I/O' устройство эмулирует простой цифровой импульсного генератор осциллограмма, который производит периодический сигнал, который может быть применен к любым выбранным 'Uno' pin.



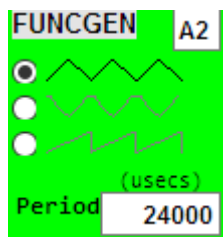
Ширины период и импульсов (в микросекундах) можно установить с помощью Edit-боксы-минимальный допустимый период составляет 50 микросекунд, а минимальная ширина импульса составляет 10 мкс. Вы можете выбрать между значениями синхронизация в микросекундах ('usec') и миллисекундах ('msec'), и этот выбор будет сохранен вместе с другими значениями, когда вы 'Save' от Конфигурировать | I / O диалог Устройства.

Полярности также могут быть выбраны: как положительные, передовыми импульсы (от 0 до 5 В) или отрицательных передовые импульсов (5В 0В).

Значения 'Pulse' и 'Period' можно масштабировать от основного **Инструмент-Bar** 'I/O ____ S' масштабно-фактор слайдер управления, добавляя в качестве суффикса 'S' (или 's') к одному (или оба). Это позволяет затем изменить 'Pulse' или значение 'Period' **динамично** во время выполнение.

Аналоговый Генератор Функций ('FUNCGEN')

Этот 'I/O' устройство эмулирует простой генератор аналоговый осциллограмма, который генерирует периодический сигнал, который может быть применен к любому выбранному 'Uno' pin.



Период (в микросекундах) можно установить с помощью поля редактирования - минимально допустимый период составляет 100 микросекунд. осциллограмма, который он создает, может быть выбран синусоидальным, треугольным или пилообразным (для создания прямоугольной волны используйте вместо этого 'PULSER'). В меньшие периоды для моделирования произведенного осциллограмма используется меньшее количество образцов за цикл (только 4 образца за цикл в период = 100 микросекунд).

'Period' значение можно масштабировать из основного окно **Инструмент-Bar** Управление ползунком масштабного коэффициента 'I/O ____ S' путем добавления в качестве суффикса буквы 'S' (или 's').

Шаговый Мотор ('STEPR')

Этот 'I/O' устройство эмулирует биполярный или униполярный шаговый двигатель 6 В со встроенным контроллером толкатель толкнул. **либо два** (на P1 , P2) **или четыре** (на P1 , P2 , P3 , P4) контрольные сигналы. Количество шагов на оборот также может быть установлено. Вы можете использовать 'Stepper.h' функциональные модули 'setSpeed()' а также 'step()' до толкнул 'STEPR'. Кроме того, 'STEPR' будет *также ответить* к себе 'digitalWrite()' " битовые"сигналы толкнул.



Мотор точно смоделирован как механически, так и электрически. Мотор-толкатель падения напряжения и изменяющееся сопротивление и индуктивность моделируются наряду с реалистичным моментом инерции относительно удерживающего момента. Обмотка ротора двигателя имеет смоделированное сопротивление R = 6 Ом и индуктивность L = 6 милли-Генри, которая создает электрическую постоянную времени 1,0 миллисекунды. Из-за реалистичного моделирования вы заметите, что очень узкие импульсы управления pin *не получают* шаг двигателя - как из-за конечного времени нарастания тока, так и из-за влияния инерции ротора. Это согласуется с тем, что

наблюдается при управлении реальным шаговым двигателем от 'Uno' с, конечно, **и требуется**) чип двигателя толкатель между проводами двигателя и 'Uno'!

Несчастный ошибка в Ардуино 'Stepper.h' Код библиотеки означает, что при сбросе шаговый двигатель не будет в положении Шаг 1 (из четырех шагов). Чтобы преодолеть это, пользователь должен использовать 'digitalWrite()' в его /ее 'setup()' рутина для инициализации контрольных уровней pin до 'step(1)' уровни, соответствующие 2-pin (0,1) или 4-pin (1,0,1,0), и позволяют двигателю за 10 миллисекунд перейти в исходное заданное положение двигателя в 12 часов.

АС версии 2.6, это устройство теперь включает в себя 'sync' LED (зеленый для синхронизации или красный при

выключении одного или нескольких шагов). Также, в дополнение к числу шагов на оборот, два дополнительных (скрытых) значения могут необязательно быть указаны в IODevs.txt файл, чтобы определить механическую load-, например, значение 20, 50, 30 определяют, 20 шагов на оборот, момент инерции нагрузки в 50 раз, что сам ротор двигателя, и крутящий момент нагрузки на 30 процентов полной фиксации момента двигателя.

Обратите внимание, что **редуктор не поддерживается напрямую** из-за недостатка места, но вы можете эмулировать его в своей программе, внедрив счетчик по модулю N переменная и вызывая только 'step()' когда этот счетчик достигает 0 (для уменьшения передачи на коэффициент N).

Пульсирующий Шаговый Мотор ('PSTEPR')

Это 'I/O' устройство эмулирует 6V **микро-шаговый** биполярный Шаговый Мотор со встроенным контроллером толкатель толкнул по **импульсный 'Step'** pin, активные низкий **'EN*'** (Включить) pin, и **'DIR'** (направление) pin, Количество полных шагов на оборот также может быть установлено непосредственно, вместе с количеством микро-шагов на полный шаг (1,2,4,8 или 16). В дополнение к этим установкам, два дополнительных (скрытых) значения могут необязательно быть указаны в IODevs.txt файл, чтобы определить механическую load-, например, значение 20, 4, 50, 30 определяют, 20 шагов на оборот, 4 микро-шагов на полный шаг, момент инерции нагрузки в 50 раз, что электродвигатель ротор сам по себе, и крутящий момент нагрузки в 30 процентов от полной фиксации момента двигателя.

Вы должны написать код для толкнул управления pins соответствующим образом.

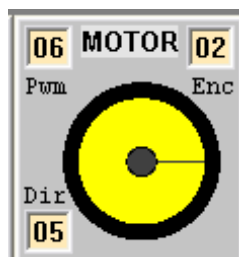


Двигатель точно смоделированы как механически, так и электрически. Мотор-толкатель напряжение падает и варьируя нежелание и индуктивность моделируются вместе с реалистическим моментом инерции относительно удерживающего момента. Двигателя обмотка ротора имеет смоделированную сопротивление $R = 6$ Ом и индуктивность $L = 6$ милли-генри, который создает электрическую постоянную времени, равную 1,0 мс.

Это устройство включает в себя желтый 'STEP' активность LED, и 'sync' LED (зеленый для синхронизации или красного цвета при отключении одного или нескольких шагов).

DC Мотор ('MOTOR')

Этот 'I/O' устройство имитирует 6-вольтовый двигатель постоянного тока 100: 1 с встроенным контроллером толкатель толкнул сигналом широтно-импульсной модуляции (на его **широтно-импульсная модуляция** вход) и управляющий сигнал направления (на его **Dir** вход). Двигатель также имеет выход энкодера колеса, который толкает на его **Enc** выход pin. Ты можешь использовать 'analogWrite()' к толкнул **широтно-импульсная модуляция** pin с 490 Гц (на pins 3,9,10,11) или 980 Гц (на pins 5,6) с широтно-импульсной модуляцией осциллограмма с рабочим циклом от 0,0 до 1,0 ('analogWrite()' значения от 0 до 255). Кроме того, 'MOTOR' будет *также ответить* к себе 'digitalWrite()' "битовые" сигналы толкнул.



Мотор точно смоделирован как механически, так и электрически. Учет падений напряжения на транзисторе двигателя толкатель и реалистичного крутящего момента зубчатого колеса без нагрузки дает полную скорость примерно 2 оборота в секунду, а крутящий момент составляет чуть более 5 кг-см (при постоянном рабочем цикле ШИМ 1,0) с суммарный момент инерции двигателя плюс нагрузка 2,5 кг-см. Обмотка ротора двигателя имеет смоделированное сопротивление $R = 2$ Ом и индуктивность $L = 300$ мкГн, что создает электрическую постоянную времени 150 микросекунд. Из-за реалистичного моделирования вы заметите, что очень узкие импульсы ШИМ *не получают* вращение двигателя - как из-за конечного времени нарастания тока, так и из-

за значительного времени простоя после каждого узкого импульса. Они объединяются, чтобы вызвать недостаточный импульс ротора, чтобы преодолеть пружинную защелку редуктора при статическом трении. Следствие при использовании 'analogWrite()' рабочий цикл ниже примерно 0,125 не заставит двигатель сдвинуться - это согласуется с тем, что наблюдается при движении реального редукторного двигателя от 'Uno' с, конечно, соответствующим (**и требуется**) Модуль двигателя толкатель между двигателем и 'Uno'!

Датчик эмуляции двигателя - это датчик оптического прерывания, установленный на валу, который вырабатывает 50% -ный рабочий цикл осциллограмма, имеющий 8 полных периодов высокого-низкого за один оборот колеса (поэтому ваш программа может воспринимать изменения вращения колеса с разрешением 22,5 градуса).

Сервомотор ('SERVO')

Этот 'I/O' устройство имитирует 6-вольтовый сервопривод постоянного тока PWM-толкнул с управлением положением. Механические и электрические параметры моделирования для работы сервопривода будут точно соответствовать параметрам стандартного сервопривода HS-422. Сервопривод имеет максимальную скорость вращения около 60 градусов за 180 миллисекунд, Если левый нижний флажок установлен, сервопривод становится **непрерывное вращение** сервопривод с той же максимальной скоростью, но теперь ширина импульса ШИМ устанавливает **скорость** а не угол



Ваш программа должен иметь '#include <Servo.h>' линия, прежде чем объявить 'Servo' экземпляр (ы) *если вы решите использовать функциональность библиотеки 'Servo.h'* например, 'Servo.write()', 'Servo.writeMicroseconds()' Кроме того, 'SERVO' также реагирует на 'digitalWrite()' "Битовые" сигналы. Из-за внутренней реализации UnoArduSim, вы ограничены 6 'SERVO' Устройства.

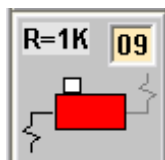
Пьезоэлектрический Динамик ('PIEZO')



Этот устройство позволяет вам "прослушивать" сигналы на любом выбранном 'Uno' pin и может быть полезным дополнением к светодиодам для отладки вашей операции программа. Вы также можете повеселиться, играя соответствующие мелодии 'tone()' а также 'delay()' звонки (хотя и нет прямоугольной фильтрации осциллограмма, поэтому вы не услышите "чистых" заметок).

Вы также можете прослушивать подключенный 'PULSER' или 'FUNCEN' устройство, подключив 'PIEZO' к pin, который устройство толкает на.

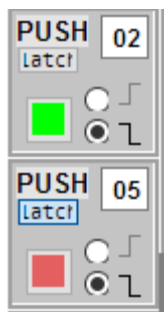
Скользящее Резистор ('R=1K')



Этот устройство позволяет пользователю подключаться к 'Uno' pin либо с повышающим сопротивлением 1 кОм до + 5 В, либо с понижающим резистором 1 кОм с массой. Это позволяет вам моделировать электрические нагрузки, добавленные к реальному оборудованию устройство. Щелкнув левой кнопкой мыши по ползунку **тело** Вы можете переключать желаемый выбор подтягивания или выпадающего. Использование одного или нескольких из этих Устройства позволит вам установить один (или несколько) -битный "код" для вашего программа

для чтения и ответа.

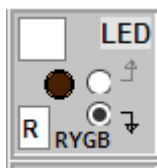
Толкать-Кнопка ('PUSH')



Этот 'I/O' устройство эмулирует нормально открытый **мгновенное ИЛИ с фиксацией** однополюсная, однополюсная (SPST) кнопка с нагрузочным (или понижающим) резистором 10 кОм. Если для устройство выбран переходный режим с передним фронтом, кнопочные контакты будут подключены между устройство и pin до + 5 В с опусканием 10 кОм на землю. Если для устройство выбран переход по фронту, кнопочные контакты будут подключены между устройство и pin и землей с напряжением 10 кОм до + 5В.

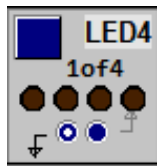
Щелкнув левой кнопкой мыши по кнопке или нажав любую клавишу, вы закрываете контакт кнопки. В **моментальный** режим, он остается закрытым, пока вы удерживаете кнопку мыши или клавишу, и в **защелка** режим (включается нажатием на 'latch' кнопка) он остается закрытым (и другого цвета), пока вы не нажмете кнопку еще раз. Отскок контактов (в течение 1 миллисекунды) будет производиться каждый раз, когда вы использовать **клавиша для интервалов** нажать на кнопку.

Цветной LED ('LED')



Вы можете подключить LED между выбранным 'Uno' pin (через токоограничивающий резистор с сопротивлением 1 кОм скрытой серии встроенный) либо к земле, либо к + 5В - это дает вам возможность включить подсветку LED, когда подключенный 'Uno' pin 'HIGH' или вместо когда он является 'LOW',
Цвет LED может быть выбран как красный ('R'), желтый ('Y'), зеленый ('G') или синий ('B'), используя его поле редактирования.

4-LED ряд ('LED4')



Вы можете подключить этот ряд из 4 цветных светодиодов между выбранным набором 'Uno' pins (каждый из которых имеет токоограничивающий резистор 1 кОм со скрытой серией встроенный) к заземлению или к + 5В - это дает вам возможность выбора светодиодов. когда подключен 'Uno' pin 'HIGH' или вместо когда он является 'LOW',

The '1of4' Поле ввода pin принимает одно число pin, которое будет означать **первый из четырех подряд** 'Uno' pins, который подключится к 4 светодиодам.

Цвет LED ('R', 'Y', 'G' или 'B') является **скрытая опция** это может быть **быть выбранным только редактирование IODevices.txt файл** (который Вы можете создать с помощью **Сохранить** от **Конфигурировать | I/O Устройства** диалоговое окно).

7-сегментный LED Цифра ('7SEG')



Вы можете подключить этот 7-сегментный дисплей Цифра LED к выбранный набор **четыре последовательных 'Uno' pins, которые дают код шестнадцатеричный** для требуемого отображаемого цифра (от '0' до 'F') и включите или выключите этот цифра с помощью CS * pin (active-LOW для ON).

Это устройство включает в себя декодер встроенный, который использует **активный ВЫСОКИЙ** уровни на четырех последовательных '1of4' pins для определения запрашиваемого шестнадцатеричный цифра для отображения. Уровень Те на самом низком номере pin (тот, который отображается в '1of4' поле редактирования) представляет младший бит 4-битного кода шестнадцатеричный.

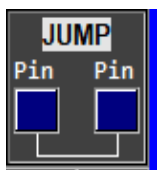
Цвет сегментов LED ('R', 'Y', 'G' или 'B') является **скрытая опция** это может быть **быть выбранным только редактирование IODevices.txt файл** Вы можете создать с помощью **Сохранить** от **Конфигурировать | I/O Устройства** диалоговое окно.

Аналоговый Слайдер

Ползунковый потенциометр 0-5 В может быть подключен к любому выбранному 'Uno' pin для получения статического (или медленно меняющегося) уровня напряжения аналоговый, который будет считываться 'analogRead()' как значение от 0 до 1023. Используйте мышь, чтобы перетащить, или нажмите, чтобы прыгать, слайдер аналоговый.



Pin Перемычка ('JUMP')



Вы можете подключить два 'Uno' pins вместе с помощью этого устройство (если какой-либо электрический конфликт обнаружен при заполнении второго номера pin, выбранное соединение не разрешается, и pin отключается).

Эта перемычка устройство имеет ограниченную полезность и наиболее полезна в сочетании с прерываниями для тестирования программа, экспериментов и учебные цели. **Начиная с UnoArduSim V2.4, вы можете обнаружить, что использование 'PROGIO' устройство предлагает большую гибкость, чем методы прерывания толкнул, приведенные ниже.**

Вот три возможных варианта использования этого устройство:

- 1) Вы можете **создать вход цифровой для тестирования вашего программа** который имеет более

сложной синхронизации, чем может быть произведен с использованием любого из набора Стандарт 'I/O' Устройства, а именно:

Определить прерывание функциональный модуль (назовем его `'myIntr'`) и делать `'attachInterrupt(0, myIntr, RISING)'` внутри вашего `'setup()'`. Подключите **пульсатор** От устройство до pin2 - сейчас `'myIntr()'` будет Выполнить каждый раз **пульсатор** возникает нарастающий фронт. Ваш `'myIntr()'` функциональный модуль может быть алгоритм у вас есть запрограммированный (используя глобальный счетчик переменные и, возможно, даже `'random()'`) изготовить осциллограмма вашего собственного дизайна на любой доступной `'OUTPUT'` pin (допустим, это pin 9). Сейчас **ПРЫГАТЬ** pin 9 по вашему желанию 'Uno' 'INPUT'pin, чтобы применить сгенерированный цифровой осциллограмма к этому входу pin (чтобы проверить ваш программа; ответ этого конкретного осциллограмма). , Вы можете генерировать последовательность импульсов, или последовательные символы, или просто граничные переходы, любой произвольной сложности и с различными интервалами. Обратите внимание, что если ваш основной программа звонит `'micros()'` (или вызывает любой функциональный модуль, который полагается на него), его `'return'` значение **будет увеличено** по времени, проведенному внутри вашего `'myIntr()'` функциональный модуль каждый раз, когда прерывание срабатывает. Вы можете произвести быстрый всплеск точно рассчитанных граней, используя вызовы `'delayMicroseconds()'` от внутри `'myIntr()'` (возможно, чтобы создать целый **байт** из высокая передача скорость передачи), или просто сгенерировать один переход за прерывание (возможно, для генерации **один бит** передачи с низким уровнем скорость передачи) с **пульсатор** устройство **'Period'** выбран в соответствии с вашими потребностями синхронизации (напомним, что **пульсатор** ограничивает его минимум **'Period'** до 50 микросекунд).

2) Вы можете **Эксперимент с подсистемой обратной связи:**

Например, отключите 'SERIAL' 'I/O' устройство TX '00' pin (отредактируйте его пустым), а затем **ПРЫГАТЬ** 'Uno' pin '01' вернуться к 'Uno' pin '00' эмулировать аппаратную петлю ATmega 'Serial' подсистема. Теперь в вашем тесте программа, внутри `'setup()'` сделать **не замужем** `'Serial.print()'` из слово или символ, и внутри вашего `'loop()'` вернуть все полученные символы (когда `'Serial.available()'`) сделав `'Serial.read()'` с последующим `'Serial.write()'`, а затем посмотреть, что происходит. Вы могли заметить, что похожий 'SoftwareSerial' петля-обратно **не удастся** (как это было бы в реальной жизни - программное обеспечение не может делать две вещи одновременно).

Вы также можете попробовать **SPI** заикливание с помощью **ПРЫГАТЬ** подключить pin 11 (MOSI) обратно к pin 12 (MISO).

3) Вы можете **подсчитать количество и / или измерить интервал между переходами**





определенного уровня на любом 'Uno' выход pin X которые происходят в результате сложного Инструкция Arduino или библиотека функциональный модуль (как примеры: `'analogWrite()'`, или же `'OneWire::reset()'`, или же `'Servo::write()'`), следующее:

ПРЫГАТЬ pin **Икс** прервать pin 2 и внутри вашего `'myIntr()'` использовать `'digitalRead()'` и `'micros()'` вызов, и сравнить с сохраненными уровнями и временем (от предыдущих прерываний). При необходимости вы можете изменить чувствительность к краям для следующего прерывания, с помощью `'detachInterrupt()'` а также `'attachInterrupt()'` от **внутри** ваш `'myIntr()'`, Обратите внимание, что вы не сможете отслеживать pin переходы, которые происходят слишком близко друг к другу (ближе, чем общее время выполнение ваш `'myIntr()'` функциональный модуль), например, те, которые происходят с передачами I2C или SPI, или с высоким скорость передачи 'Serial' переводы (даже если ваше прерывание функциональный модуль не будет мешать переходу синхронизации этих аппаратных передач). Также обратите внимание, что программно-опосредованные переводы (например, `'OneWire::write()'` а также `'SoftwareSerial::write()'`) являются Преднамеренно защищен от прерываний (их библиотечный код временно отключает все прерывания, чтобы предотвратить сбои синхронизации), поэтому вы не можете проводить измерения внутри тех, кто использует этот метод.





Хотя вы можете вместо этого сделать те же измерения расстояния между краями **визуально** в Цифровой **формы волны** окно, если вас интересует минимальный или максимальный интервал для большого количества переходов или подсчет переходов, сделайте это с помощью этого `'myIntr()'` -plus- **ПРЫГАТЬ** Техника удобнее. И вы можете измерьте, например, вариации расстояния между основными переходами, созданными программа (из-за того, что ваше программное обеспечение использует разные пути выполнение в разное время выполнение), сделать вид программа "Профилирование".

Меню

Файл:

<u>Нагрузка INO или PDE Prog (Ctrl-L)</u> 	Позволяет пользователю выбрать программа файл с выбранным расширением. программа немедленно получает Анализировать
<u>Изменить/Изучить (Ctrl-E)</u>	Открывает загруженный программа для просмотра / редактирования.
<u>Сохранить</u> 	Сохранить отредактированное содержание программа возвращается к оригинальному программа файл.
<u>Сохранить как</u>	Сохранить отредактированное содержимое программа под другим именем файл.
<u>следующий ('#include')</u> 	Продвигает Код Панель отображать следующий '#include' файл
<u>предыдущий</u> 	Возвращает Код Панель показать на предыдущий файл
<u>Выход</u>	Выход из UnoArduSim после напоминания пользователю сохранить любые измененные файл (s).

Найти:

<u>Восходить стек вызовов</u> 	Перейти к предыдущей функции вызывающего абонента в стеке вызовов - Панель Переменных будет адаптирована к этой функции
<u>Спуститесь стек вызовов</u> 	Перейти к следующей вызываемой функции в стеке вызовов - Панель Переменных будет адаптирована к этой функции
<u>Установить текст</u> <u>Установить текст (Ctrl-F)</u> 	Активировать Инструмент-Bar Найти поле ввода для определения текста, который необходимо найти следующим (и добавляет первое слово из выделенной в данный момент строки в Код Панель или же Переменные Панель если один из них имеет фокус).
<u>Найти Следующий текст</u> 	Перейти к следующему вхождению текста в Код Панель (если он имеет активный фокус), или к следующему вхождению текста в Переменные Панель (если вместо этого он имеет активный фокус).
<u>Найти Предыдущий текст</u> 	Перейти к предыдущему вхождению текста в Код Панель (если он имеет активный фокус), или к предыдущему вхождению текста в Переменные Панель (если вместо этого он имеет активный фокус).

Выполнить:

<u>Шаг Внутри (F4)</u>		Шаги выполнение вперед на одну инструкцию, или <i>в так называемый функциональный модуль</i> ,
<u>Шаг Вокруг (F5)</u>		Шаги выполнение вперед на одну инструкцию, или <i>одним полным звонком функциональный модуль</i> ,
<u>Шаг Вне (F6)</u>		Авансы выполнение по <i>Достаточно, чтобы оставить текущий функциональный модуль</i> ,
<u>Выполнить До (F7)</u>		Работает программа, <i>остановка на нужной линии программа</i> - прежде чем использовать Выполнить До, вы должны сначала щелкнуть основной момент на нужную строку программа.
<u>Выполнить Пока (F8)</u>		Запускает программа до тех пор, пока не произойдет запись в переменная с текущим основной момент в Переменные Панель (нажмите на один, чтобы установить начальный основной момент).
<u>Выполнить (F9)</u>		Работает программа.
<u>Стой (F10)</u>		Остановки программа выполнение (<i>и замораживает время</i>).
<u>Сброс</u>		Сбрасывает программа (все значения переменные сбрасываются в значение 0, а все указатели переменные сбрасываются в 0x0000).
<u>Анимация</u>		Автоматически шаги последовательных строк программа с <i>добавленной искусственной задержкой</i> и выделение текущей строки кода. Операции в реальном времени и звуки теряются.
<u>Замедленная съемка</u>		Замедляет время в 10 раз.

Опции:

<u>Шаг Вокруг Structors/ Операторы</u>	Пролетите сквозь конструкторы, деструкторы и перегрузку оператора функциональные модули во время любого шага (т.е. он не остановится внутри этих функциональные модули).
<u>Регистр-Allocation</u>	Назначьте локальные функциональный модуль для регистров ATmega свободно вместо стека (генерирует несколько уменьшенное использование RAM).
<u>Ошибка при неинициализированном</u>	Пометить как ошибку Анализировать везде, где ваш программа пытается использовать переменная без предварительной инициализации его значения (или хотя бы одного значения внутри массива).
<u>добавленной 'loop()' задержка</u>	Добавляет 1000 микросекунд задержки каждый раз 'loop()' вызывается (в случае, если нет других вызовов программа на 'delay()' где угодно) - полезно избегать слишком большого отставания от реального времени.
<u>Разрешить вложенные прерывания</u>	Разрешить повторное включение с 'interrupts()' изнутри подпрограммы обслуживания прерываний пользователя.

Конфигурировать:

<u>'I/O' Устройства</u>	Открывает диалоговое окно, позволяющее пользователю выбрать тип (ы) и номера требуемого 'I/O' Устройства. Из этого диалогового окна вы также можете Сохранить 'I/O' Устройства в текст файл и / или Нагрузка 'I/O' Устройства из ранее сохраненного (или отредактированного) текста файл (включая все соединения pin, интерактивные настройки и введенные значения).
<u>Предпочтения</u>	Открывает диалоговое окно, позволяющее пользователю устанавливать предпочтения, в том числе автоматический отступ исходного программа строк, разрешать синтаксис Expert, выбирать шрифт гарнитура, выбирать больший размер шрифта, применять границы массива, разрешать ключевые слова логического оператора, показывая программа загрузка , выбор версии 'Uno' Версия для Uno и длины буфера TWI (для I2C Устройства).

ПеремОбновить:

<u>Разрешить авто (-) Сокращаться</u>	Разрешить UnoArduSim для сокращаться отображается расширенный массивы / объектов при отставании в режиме реального времени.
<u>минимальная</u>	Только освежить Переменные Панель отображать 4 раза в секунду.
<u>Основной момент изменения</u>	Основной момент изменил значения переменная при работе (может вызвать замедление).

Окна:

<u>Serial Монитор</u>	Подключите последовательный ввод / вывод устройство к pins 0 и 1 (если его нет) и потяните больший 'Serial' монитор TX / RX с текстом окно.
<u>Восстановить все</u>	Восстановите все свернутые дочерние элементы окна.
<u>Pin Цифровые Осциллограммы</u>	Восстановить свернутый Pin Цифровые Осциллограммы окно.
<u>Pin Аналоговый Осциллограммо</u>	Восстановить свернутый Pin Аналоговый Осциллограммо окно.

Помогите:

<u>Быстрый Помогите Файл</u>	Открывает UnoArduSim_QuickHelp PDF файл.
<u>Полный Помогите Файл</u>	Открывает UnoArduSim_FullHelp PDF файл.
<u>Исправления Ошибка</u>	Просмотреть важные исправления ошибка с момента предыдущего выпуска.
<u>Изменение / Улучшение</u>	Просмотр значительных изменений и улучшений по сравнению с предыдущим выпуском.
<u>Около</u>	Отображает версию, авторское право.

'Uno' Версия для Uno и 'I/O' Устройства

'Uno' и подключенный 'I/O' Устройства все точно смоделированы электрически, и вы сможете получить хорошее представление о том, как ваш программы будет вести себя с фактическим оборудованием, и все электрические pin конфликты будут помечены.

Синхронизация

UnoArduSim выполняет достаточно быстро на ПК или планшете, что может (в большинстве случаев) действия модели программа в режиме реального времени, **но только если ваш программа включает в себя** хоть какой то маленький 'delay()' звонки или другие звонки, которые, естественно, синхронизируют его с реальным временем (см. ниже).

Для этого UnoArduSim использует таймер обратного вызова Окна функциональный модуль, который позволяет ему точно отслеживать реальное время. выполнение из ряда инструкций программа моделируется в течение одного таймера, а инструкции, для которых требуется больше выполнение (например, вызовы 'delay()') может потребоваться использовать несколько таймеров. Каждая итерация таймера обратного вызова функциональный модуль корректирует системное время, используя системные аппаратные часы, так что программа выполнение постоянно регулируется, чтобы не отставать от режима реального времени.

Единственный раз выполнение ставка должен отставать от реального времени когда пользователь создал плотные петли **без дополнительной задержки** или 'I/O' Устройства сконфигурированы для работы с очень высокими частотами 'I/O' устройство (и / или скорость передачи), которые будут генерировать чрезмерное количество событий изменения уровня pin и связанной с этим перегрузки обработки. UnoArduSim справляется с этой перегрузкой, пропуская некоторые интервалы таймера для компенсации, а затем замедляет прогрессирование программа до **ниже реального времени** ,

Кроме того, программы с большим массивы отображается или снова с плотными петлями **без дополнительной задержки** может вызвать высокую частоту вызовов функциональный модуль и генерировать высокую **Переменные Панель** загрузка обновлений дисплея, приводящая к отставанию в режиме реального времени - UnoArduSim автоматически снижает частоту обновления переменная, чтобы попытаться не отставать, но когда требуется еще большее уменьшение, выберите **Минимальный**, от **ПеремОбновить** меню указать только четыре обновления в секунду.

Точное моделирование времени выполнение с точностью до миллисекунды для каждой инструкции или операции программа **не сделано** - только очень приблизительные оценки для большинства были приняты для целей моделирования. Тем не менее, синхронизация из 'delay()' , а также 'delayMicroseconds()' функциональные модули и функциональные модули 'millis()' а также 'micros()' все совершенно точно, **и до тех пор, пока вы используете хотя бы одну задержку функциональные модули** в петле где-то в вашей программа, **или же** Вы используете функциональный модуль, который естественным образом привязан к работе в режиме реального времени (например, 'print()' который привязан к выбранному скорости передачи), то смоделированная производительность вашей программа будет очень близка к реальному времени (опять же, за исключением явного чрезмерного высокочастотного события изменения уровня pin или чрезмерных разрешенных пользователем обновлений Переменные, которые могут замедлить его).

Чтобы увидеть эффект отдельных инструкций программа в *курица бежит* может быть желательно иметь возможность замедлить ход событий. Коэффициент замедления времени 10 может быть установлен пользователем в меню **Выполнить** ,

'I/O' Устройство Синхронизация

Эти виртуальные Устройства принимают в реальном времени сигналы об изменениях, которые происходят на их входе pins, и вырабатывают соответствующие выходы на своих выходах pins, которые затем могут распознаваться 'Uno' - поэтому они по своей сути синхронизируются с программа выполнение. Внутренний 'I/O' устройство синхронизация устанавливается пользователем (например, с помощью выбора скорости передачи или тактовой частоты), а события симулятора устанавливаются для отслеживания внутренней работы в реальном времени.

Звуки

Каждый 'PIEZO' устройство издает звук, соответствующий изменениям электрического уровня, происходящим на подключенном pin, независимо от источника таких изменений. Чтобы синхронизировать звуки с программа выполнения, UnoArduSim запускает и останавливает воспроизведение соответствующего звукового буфера, когда выполнение запускается / останавливается.

Начиная с версии 2.0 звук теперь был изменен для использования аудио API Qt - к сожалению, его QAudioOutput 'class' не поддерживает зацикливание звукового буфера, чтобы избежать исчерпания сэмплов (как это может случиться во время более длительных операционных задержек). Поэтому, чтобы избежать подавляющего большинства раздражающих щелчков звука и прерывания звука во время задержек ОС, звук теперь отключается в соответствии со следующим правилом:

Звук отключается до тех пор, пока UnoArduSim не является "активным" окно (кроме случаев, когда новый дочерний объект окно только что был создан и активирован), **и даже** когда UnoArduSim является "активным" основным окно, но указатель мыши **вне** из его основная клиентская зона окно.

Обратите внимание, что это означает, что звук будет временно приглушен как король при наведении курсора мыши **над ребенком окно**, и будет отключен **если этот ребенок окно нажал, чтобы активировать его** (до тех пор, пока основной UnoArduSim окно не будет нажат снова, чтобы повторно активировать его) ,

Звук всегда можно отключить, щелкнув в любом месте внутри клиентской области главного KO193 UnoArduSim.

Из-за буферизации, с В реальном времени задержка составляет до 250 миллисекунд с соответствующего времени события на pin подключенного 'PIEZO'.

Ограничения и неподдерживаемые элементы

Включено Файлы

A '<>' - в скобках '#include' из '<Servo.h>', '<Wire.h>', '<OneWire.h>', '<SoftwareSerial.h>', '<SPI.h>', '<EEPROM.h>' а также '<SD.h>' **является** поддерживается, но они только эмулируются - фактический файлы не ищется; вместо этого их функциональные возможности непосредственно "встроены" в UnoArduSim и действительны для фиксированной поддерживаемой версии Arduino.

Любой цитируемый '#include' (например, "supp.ino", "Myutil.cpp", или же "Mylib.h") поддерживается, но все такие файлы должны **проживать в тот же каталог в качестве родителя программа файл** тот содержит их '#include' (нет поиска в других каталогах). The '#include' функция может быть полезна для минимизации количества кода программа, показанного в **Код Панель** в любое время. Жатка файлы с '#include' (т.е. те, которые имеют ".h" расширение) дополнительно заставит симулятор попытаться включить одноименный файл, имеющий ".cpp" расширение (если оно также существует в каталоге родительского программа).

Динамическое распределение памяти и оперативная память

операторы 'new' а также 'delete' поддерживаются, как и родные Arduino 'String' объектов, **но не прямые звонки** 'malloc()', 'realloc()' а также 'free()' что они полагаются

Чрезмерное использование ОЗУ для объявлений переменная помечается во время Анализировать, а переполнение памяти ОЗУ отмечается во время программа выполнение. An пункт в меню **Опции** позволяет вам эмулировать обычное распределение регистров ATmega, как это будет сделано AVR компилятор, или моделировать альтернативную схему компиляции, которая использует только стек (в качестве опции безопасности в случае, если ошибка появляется в моем моделировании распределения регистров). Если бы вы использовали указатель для просмотра содержимого стека, он должен точно отражать то, что появилось бы в реальной аппаратной реализации.

'Flash' Распределение памяти

Память 'Flash' 'byte', 'int' а также 'float' переменные / массивы и соответствующий им доступ для чтения функциональные модули поддерживаются. любой 'F()' Звонок функциональный модуль ('Flash' Макро) из любая буквенная строка **является** поддерживается, но единственной поддерживаемой строкой памяти 'Flash' прямого доступа функциональные модули являются 'strcpy_P()' а также 'memcpy_P()', поэтому для использования другого функциональные модули вам нужно сначала скопировать строку 'Flash' в обычную оперативную память 'String' переменная, а затем работать с этой оперативной памятью 'String', Когда вы используете 'PROGMEM' Ключевое слово-модификатор переменная, оно должно появляться **перед** имя переменная, и это переменная **также должен быть объявлен** как 'const',

'String' Переменные

Родной 'String' библиотека почти полностью поддерживается с несколькими очень (и незначительными) исключениями.

The 'String' операторы поддерживаются +, +=, <, <=, >, >=, ==, !=, **знак равно**, а также [], Обратите внимание, что: 'concat()' занимает **не запомним** аргумент, который является 'String', или же 'char', или же 'int' быть добавленным к оригиналу 'String' объект, **не** два аргумента, как ошибочно указано на веб-страницах Arduino Reference).

Библиотеки Arduino

Только 'Servo.h', 'SoftwareSerial.h', 'SPI.h', 'Wire.h', 'OneWire.h', 'Stepper.h', 'SD.h', 'TFT.h' и 'EEPROM.h' для **Arduino v1.8.8** релиз в настоящее время поддерживаются в UnoArduSim. V2.6 вводит механизм 3^й поддержка библиотеки стороны через файлы предоставляется в 'include_3rdParty' папка которые могут быть найдены внутри UnoArduSim директории установки. Попытка '#include' ".cpp" и ".час" файлы из других пока еще не поддерживается библиотеки будут **не работает** как они будут содержать инструкции по сборке низкоуровневых и неподдерживаемые директивы и непризнанный файлы!

указатели

Поддерживаются указатели на простые типы, массивы или объектов. Указатель может быть приравнен к массива того же типа (например, 'iptr = intarray'), но тогда бы **нет последующей проверки границ массивы** на выражение как 'iptr[index]',

Функциональные модули может возвращать указатели, или 'const' указатели, но любой последующий уровень 'const' на возвращенном указателе игнорируется.

Есть **никакой поддержки** для функциональный модуль звонки через **объявленные пользователем указатели функциональный модуль**,

'class' а также 'struct' Объектов

Хотя поддерживается полиморфизм и наследование (на любую глубину), 'class' или же 'struct' можно определить только на максимум **один** база 'class' (т.е. **Многолучевая** наследование не поддерживается). Поддерживаются вызовы инициализации конструктора Base-'class' (через двоеточие) в строках объявления конструктора, но **не** инициализация членов с использованием той же записи двоеточия. Это означает, что объектов, которые содержат 'const' не-'static' переменные, или ссылочный тип переменные, не поддерживаются (это возможно только с указанными инициализациями элементов во время строительства)

Перегрузки оператора назначения копирования поддерживаются вместе с конструкторами перемещения и назначениями перемещения, но пользовательское преобразование объект ("переменная") функциональные модули не поддерживается.

Сфера

Там нет поддержки для 'using' ключевое слово, или для 'namespace' или для 'file' сфера. Все нелокальные объявления по реализации предполагаются глобальными.

любой 'typedef', 'struct', или же 'class' определение (т.е. это может быть использовано для

будущих объявлений), должно быть сделано **Глобальный** сфера (**местный** определения таких предметов внутри функциональный модуль не поддерживаются).

Отборочные 'unsigned', 'const', 'volatile', 'static'

The 'unsigned' Префикс работает во всех нормальных юридических контекстах. The 'const' ключевое слово, когда используется, должен *предшествовать* имя переменная или имя функциональный модуль или 'typedef' имя, которое объявляется - размещение его после имени приведет к ошибке Анализировать. За Объявления функциональный модуль, только функциональные модули с возвратом указателя может иметь 'const' появляются в их декларации.

Все UnoArduSim переменные являются 'volatile' реализацией, поэтому 'volatile' Ключевое слово просто игнорируется во всех объявлениях переменная. Функциональные модули не разрешается объявлять 'volatile' также не являются аргументами вызова функциональный модуль.

The 'static' ключевое слово разрешено для обычного переменные, а также для членов объект и member-функциональные модули, но явно запрещено для самих экземпляров объект ('class' / 'struct'), для не члена функциональные модули и для всех аргументов функциональный модуль.

Директивы Компилятор

'#include' и регулярно '#define' оба поддерживаются, но **не макрос** '#define', The '#pragma' директивы и директивы условного включения ('#ifdef', '#ifndef', '#if', '#endif', '#else' а также '#elif') являются также **не поддерживается**, The '#line', '#error' и предопределенные макросы (например, '_LINE_', '_FILE_', '_DATE_', а также '_TIME_') являются также **не поддерживается**.

Ардуино-языковые элементы

Все родные элементы языка Arduino поддерживаются, за исключением сомнительных 'goto' инструкция (единственное разумное использование, которое я могу себе представить, это переход (к бесконечному циклу аварийного и безопасного отключения) в случае возникновения ошибки, с которой ваш программа не может иначе справиться)

С / С ++ - языковые элементы

Сохраняющие бит "квалификаторы битовых полей" для членов в определениях структуры **не поддерживается**,

'union' является **не поддерживается**.

Странный "оператор запятой" **не поддерживается** (потому вы не можете выполнить несколько выражений, разделенных запятыми, когда обычно ожидается только одно выражение, например, в 'while()' а также 'for(; ;)' конструкции).

Функциональный модуль Шаблоны

Определяемый пользователем функциональные модули, который использует ключевое слово "template", чтобы позволить ему принимать аргументы "универсального" типа, **не поддерживается**,

Эмуляция в реальном времени

Как отмечалось выше, выполнение раз из множества отдельных возможных инструкций Arduino программа **не** смоделированы точно, так что для того, чтобы работать в режиме реального времени, ваш программа будет нуждаться в некотором доминировании 'delay()' инструкция (по крайней мере, один раз в 'loop()') или инструкция, которая естественным образом синхронизируется с изменениями уровня pin в реальном времени (например, 'pulseIn()', 'shiftIn()', 'Serial.read()', 'Serial.print()', 'Serial.flush()' так далее.).

Увидеть **Синхронизация** а также **Звуки** выше для более подробной информации об ограничениях.

Примечания к выпуску

Исправления Ошибка

V2.7.0- март 2020

- 1) Когда (Окна по умолчанию) была принята свет тема OS, то **Код Панель** не показывает цвет, выделяя введенный в версии 2.6 (вместо того, чтобы только серый основной момент в результате чего из системы переопределения).
- 2) Версия 2.6 неосторожно мели автоматического отступ вкладок форматирование на первом `'switch()'` построить.
- 3) Новая стек вызовов функция навигации введена в версии 2.6 показал **неправильные значения** для местных переменные когда не внутри исполняемой в данный момент функциональный модуль, так и не удалось с вложенными членов функциональный модуль вызовов.
- 4) `'TFT::text()'` работал, но функции `'TFT::print()'` не работали (они просто блокировались навсегда). Кроме того, `'TFT::loadImage()'` не удалось, если `'Serial.begin()'` был выполнен ранее (что является нормальным случаем и теперь требуется).
- 5) Версия 2.6 введена ошибка, что отображается неверное значение для прошлого и настоящего 'RX' байты для 'I2CSLV', 'SPISLV', 'TFT', 'LCDI2C' and 'LCDSPI' Устройства (и их Monitor окна).
- 6) Изменение, внесенное в V2.4 вызвало Выполнить | Анимация подсветку пропустить много выполнил кодовых строк.
- 7) Так V2.4 де утверждающий 'SS*' или 'CS*' на 'I/O' устройство в инструкции сразу после `'SPI.transfer()'` будет причина того, что устройство чтобы не получить переданные байты данных. Кроме того, байты прием в `'SPI_MODE1'` и `'SPI_MODE3'` не был помечен до начала следующего байта, посланного мастер (и байты были полностью потеряны, если устройство 'CS*' был выбран де-до этого).
- 8) В новом `'SPI_SLV'` режим недопустим, поскольку V2.4, `'bval = SPI.transfer()'` только правильное значение для `'bval'` если передача байта была уже завершена и ждет, когда `'transfer()'` назывался.
- 9) 'DATA' окно редактирования на 'SPISLV' Устройства теперь получает по умолчанию 0xFF, когда нет больше байт отвечать.
- 10) Синхронизации LED состояние было неправильным для 'PSTEPR' Устройства, имеющими более 1 микро-шага за полный шаг.
- 11) Электрические конфликты вызванные 'I/O' Устройства реагируют на переходы на 'SPI' часы, сигнал 'PWM' или `'tone'` сигнал, не сообщался, и может привести к необъяснимому (повреждено) прием данных.
- 12) Когда интервал между прерываниями был слишком мал (менее 250 мкс), а (неисправность) изменение V2.4 изменило синхронизация из встроенный функциональные модули, которые используют либо системные таймеры, или петли команд, чтобы генерировать задержки (примеры каждого из них `'delay()'` и `'delayMicroseconds()'`). Последующее изменение в V2.5 вызвало неправильное выравнивание `'shiftOut()'` данные и сигналы синхронизации, когда прерывание произошло между битами.
- 13) Принимая встроенный функциональный модуль текст автозавершения с помощью клавиши Enter не удалось вырезать типы параметров из текста вставленного вызова функциональный модуль.
- 14) Загрузка новый (пользовательское прерывание толкнул) программа при предыдущем запуске программа еще прерывание ожидающего может вызвать сбой во время загрузки (из-за неисправную попытку выполнение новой программы обработки прерывания).
- 15) Объект-члены авто-пополнение (доступны через 'ALT'-стрелку вправо) для объектов внутри `'#include'` файлы теперь доступны, как только их `'#include'` файл успешно разобранный.
- 16) функциональный модуль декларация, имеющая непреднамеренное пространство внутри имени параметра функциональный модуль вызвала непонятное сообщение об ошибке Анализировать.

17) Когда выполнение остановился в модуле отличается от основного программа, в Файл | Предыдущее действие не удалось стать включено.

18) Single цитируемый изогнутые скобки (' { ' и ' } ') Все еще подсчитывают (неправильно), как сфера в скобках Анализировать, а также путать авто-табуляцией отступа форматирования ..

19) 'OneWire::readBytes(byte* buf, int count)' были не в состоянии немедленно обновить отображаемыми 'buf' содержание в Переменные Панель.

20) Восьмиштырьковая-защелка 'OWISLV' Устройства показала OUTPUT pin уровней, которые отстают от одной защелки-записи в регистре.

V2.6.0- Jan 2020

- 1) ошибка введена в V2.3 привело к аварии, когда добавляется **близко** кнопка была использована в **Найти / Заменить** диалог (а не его Выход Кнопка название-бар).
- 2) Если пользователь сделал программа '#include' другого пользователя файлы, то **Сохранить** кнопка внутри **Изменить/Изучить** не смогли бы реально сэкономить модифицированном файл если есть существующий Анализировать или Выполнение ошибка попадает в другой файл.
- 3) **Отмена** после **Сохранить** также может сбивать с толку - по этим причинам **Сохранить** и **Отмена** Кнопка функциональные возможности были изменены (см **Изменения и улучшения**).
- 4) Несбалансированные скобки внутри 'class' определение может привести к зависанию, так как V2.5.
- 5) Прямое логическое тестирование на 'long' возвращаемые значения 'false' если ни один из 16 самых низких битов не были установлены.
- 6) UnoArduSim были пометки об ошибке, когда указатель переменная был объявлен как петли переменная внутри скобок а 'for()' заявление.
- 7) UnoArduSim были запрещая логические тесты, которые по сравнению указателей 'NULL' или '0',
- 8) UnoArduSim был запрещая арифметику указателей с участием целого переменная (только целое число констант были разрешены).
- 9) Прерывания устанавливается с помощью 'attachInterrupt(pin, name_func, LOW)' было только воспринимаемые на а **переход** в 'LOW',
- 10) При использовании более одного устройства 'I2CSLV' неадресуемый ведомый может интерпретировать более поздние данные шины как соответствующие его шине (или глобальному вызову 0x00), и, таким образом, ложно сигнализировать АСК, повреждая уровень АСК шины и вешая 'requestFrom()'.
- 11) Передача числового значения '0' (или 'NULL') В качестве аргумента функциональный модуль к указателю в вызове функциональный модуль теперь разрешено.
- 12) Табулируя отступ уровня после вложенности 'switch()' конструкции были слишком мелкими, когда выбор 'auto-indent formatting' из **Конфигурировать | Предпочтения** был использован.
- 13) Вычитание двух совместимых указателей в настоящее время приводит к типу 'int',
- 14) UnoArduSim ожидал определенный пользователем конструктора по умолчанию для объект элемента, даже если он не был объявлен как 'const',
- 15) При разрыве выполнение, втянутый положение 'STEPR', 'SERVO' или 'MOTOR' двигатель может запаздывать до 30 миллисекунд движения за его фактическое последнее вычисленное положение.
- 16) 'Stepper::setSpeed(0)' был причиной аварии из-за деления на ноль.
- 17) Одна линия 'if()', 'for()', и 'else' не строит больше не вызывает слишком много вкладок автоотступа.

V2.5.0- октября 2019

- 1) ошибка введен в V2.4 мели инициализации 'SD' карты (причина аварии).
- 2) Использование подсистемы 'SPI' в новом 'SPI_SLV' Режим работал неправильно в 'SPI_MODE1' и

`'SPI_MODE3'`,

3) Авто-завершение всплывающие окна (в соответствии с просьбой 'ALT-right=arrow') были установлены для функциональные модули, имеющих объект параметры; список всплывающих окон теперь также включает наследуется (base-) члены класса, и авто-пополнения теперь также появляться `'Serial'`,

4) Так как V2.4 возвращаемого значения для `'SPI.transfer()'` и `'SPI.transfer16()'` был неправильным, если подпрограмма прерывания пользователя уполена во время такой передачи.

5) В V2.4, быстрые периодические сигналы были показаны как имеющие более длительный срок, чем их фактической продолжительности очевидной, если смотреть на более высоком увеличении.

6) конструктор `'File::File(SdFile &sdf, char *fname)'` не работает, так `'File::openNextFile()'` (Который зависит от этого конструктора) также не работает.

7) UnoArduSim был неправильно объявлять ошибку Анализировать на объект-переменные и объект возвращающих функциональные модули, объявлен `'static'`,

8) Операторы присваивания с `'Servo'`, `'Stepper'`, или `'OneWire'` объект переменная на LHS, и объект возвращающих функциональный модуль или конструктор на PIT, вызвало внутренний MISCOUNT из числа ассоциированных объектов, что приводит к возможному врезаться.

9) Логические тесты на объектов типа `'File'` всегда возвращаются `'true'` даже если файл не был открыт.

V2.4– май 2019

1) Точки наблюдения Выполнить Пока могут быть ложно инициированы записью в соседний переменная (адрес на 1 байт ниже).

2) Выбор скорости передачи данных может быть ощутим когда мышь была внутри `'SERIAL'` или `'SFTSER'` устройство бод раскрывающийся список (даже когда скорость передачи данных не была нажата).

3) Начиная с V2.2, ошибки последовательного приема происходили со скоростью 38400 бод.

4) Внесенные изменения в V2.3 **'SoftwareSerial'** иногда ошибочно сообщать об отключенном прерывании (и, таким образом, отказывать при первом приеме RX).

5) Изменение, внесенное в V2.3, заставило SPISLV Устройства неправильно интерпретировать шестнадцатеричные значения, непосредственно введенные в их поле ввода `'DATA'`.

6) Внесенные изменения в V2.3 вызвали SRSLV Устройства иногда ложно и молча обнаруживать электрический конфликт на их `'Dout'` pin и, таким образом, запрещать назначение pin там. Повторные попытки присоединить pin к `'Dout'` могут привести к возможному сбою после удаления устройство.

7) Попытка присоединить `'LED4'` устройство за pin 16 может привести к сбою.

8) ошибка, вызванный быстрыми повторными вызовами `'analogWrite(255)'` за **'MOTOR'** контроль у пользователя программа вызвал результирующий **'MOTOR'** скорости быть неправильными (слишком медленными).

9) Начиная с версии 2.3, SRSLV Устройства не может быть назначен `'Dout'` pin из-за неисправного электрического обнаружения конфликт (и, таким образом, запрещается назначение pin).

10) Ведомые SPI теперь сбрасывают свою логику передатчика и приемника, когда их **'SS*'** pin идет `'HIGH'`,

11) призывание `'Wire.h'` функциональные модули `'endTransmission()'` или же `'requestFrom()'` когда прерывания в настоящее время \ отключены, теперь генерирует ошибку выполнение (`'Wire.h'` для работы нужны прерывания).

12) `'Ctrl-Home'` и `'Ctrl-End'` теперь работают как положено в Изменить/Изучить.

13) The `'OneWire'` автобусная команда `0x33 ('ROM_READ')` не работал и повесил автобус.

V2.3– декабрь 2018

1) ошибка, представленный в V2.2, сделал невозможным редактирование значения элемента массива внутри **Изменить/Монитор**.

- 2) Начиная с версии 2.0, текст в '**RAM free**' Панель инструментов управления была видна только при использовании темной темы Окна-OS.
- 3) На **Файл | Нагрузкаи** на ввод / вывод устройство файл **Нагрузка**, фильтры файл (типа '*.ino' а также '*.txt') не работали - вместо этого были показаны файлы всех типов.
- 4) "Модифицированное" состояние файл было потеряно после выполнения **Принимать** или же **Компилировать** в последующем **Файл | Изменить/Изучить если дальнейшие правки не были внесены (Сохранить** стал отключен, и не было автоматического запроса **Сохранить** на программа **Выход**).
- 5) операторы '**/=**' а также '**%=**' дал только правильные результаты для '**unsigned**' левая сторона переменные
- 6) Троичный условный '**(bval) ? s1:s2**' дал неверный результат, когда '**s1**' или же '**s2**' было локальное выражение вместо переменная.
- 7) Функциональный модуль '**noTone()**' был исправлен, чтобы стать '**noTone(uint8_t pin)**',
- 8) Давний ошибка вызвал сбой после выхода из **Сброс** когда этот Сброс был выполнен в середине функциональный модуль, который был вызван с отсутствующим одним из его параметров (и таким образом получил значение инициализатора по умолчанию).
- 9) Выражения членов (например, '**myobj.var**' или же '**myobj.func()**' не наследовали '**unsigned**' свойство их правой стороны ('**var**' или же '**func()**') и поэтому не может быть непосредственно сопоставлено или объединено с другими '**unsigned**' типы - промежуточное присвоение '**unsigned**' переменная был первым обязательным.
- 10) UnoArduSim настаивал на том, что если в определении функциональный модуль есть какой-либо параметр с инициализатором по умолчанию, что функциональный модуль имеет ранее объявленный прототип.
- 11) Звонки в '**print(byte bvar, base)**' ошибочно повышен '**bvar**' для '**unsigned long**', и так распечатано слишком много цифр.
- 12) '**String(unsigned var)**' а также '**concat(unsigned var)**' и операторы '**+=(unsigned)**' а также '**+(unsigned)**' неправильно создан '**signed**' Строки вместо.
- 13) 'R=1K' устройство загружен из **IODevices.txt** файл с положением '**U**' был по ошибке нарисован с его ползунком (всегда) в **противоположная позиция** от его истинного электрического положения.
- 14) Попытка положиться на дефолт '**inverted=false**' аргумент при объявлении '**SoftwareSerial()**' объект вызвал сбой и прохождение '**inverted=true**' работал только если пользователь программа сделал последующий '**digitalWrite(txpin, LOW)**' для того, чтобы сначала установить необходимый холостой '**LOW**' уровень на **Техас pin**.
- 15) 'I2CSLV' Устройства не реагировал на изменения в своих полях редактирования pin (значения по умолчанию A4 и A5 оставались в силе).
- 16) 'I2CSLV' и 'SPISLV' Устройства не обнаружили и не исправили частичные изменения, когда мышь покинула свои границы
- 17) Значения Pin для Устройства, которые следовали за SPISLV или SRSLV, были неправильно сохранены в **IODevs.txt** файл в шестнадцатеричном формате.
- 18) Попытка подключить более одного SPISLV устройство MISO к pin 12 всегда приводила к электрической ошибке Конфликт.
- 19) Переключение pin от '**OUTPUT**' вернуться к **Режим** '**INPUT**' не удалось сбросить уровень фиксации данных pin до '**LOW**',
- 20) С помощью '**sendStop=false**' в звонках '**Wire.endTransmission()**' или же '**Wire.requestFrom()**' вызвал сбой.
- 21) UnoArduSim неправильно допустил '**SoftwareSerial**' прием должен происходить одновременно с '**SoftwareSerial**' коробка передач.
- 22) Переменные объявлен с '**enum**' типу не может быть присвоено новое значение после их строки объявления, и UnoArduSim не распознает элементы 'enum', когда на них ссылается (Legal) '**enumname::**' префикс.

V2.2– июнь 2018

- 1) Вызов функциональный модуль с меньшим количеством аргументов, чем требовалось его определением (когда этот функциональный модуль был "определен вперед", то есть, когда у него не было более ранней строки объявления прототип), вызвал нарушение памяти и сбой.
- 2) Начиная с V2.1, **Волновые** не обновлялся во время **Выполнить** (только в **Стойили** после **Шаг**) - к тому же, **Переменные Панель** значения не обновлялись в течение длительного **Шаг** операции.
- 3) Какой-то несовершеннолетний **Осциллограмма** проблемы с прокруткой и масштабированием, которые существовали с версии 2.0, теперь исправлены.
- 4) Даже в ранних версиях Сброс в момент времени $t = 0$ с PULSER или FUNCGEN, чей период совершил бы самый последний цикл перед $t = 0$ быть только частичный цикл, в результате чего его **Осциллограмма** после $t = 0$ смещается из своего истинного положения на эту (или оставшуюся) величину дробного цикла, либо вправо, либо влево (соответственно).
- 5) Исправлены некоторые проблемы с подсветкой синтаксиса в цвете **Изменить/Изучить** ,
- 5) Начиная с V2.0, нажатие на расширять одного объект в массива объектов не работало должным образом.
- 6) `'delay(long)'` был исправлен, чтобы быть `'delay(unsigned long)'` а также `'delayMicroseconds(long)'` был исправлен, чтобы быть `'delayMicroseconds(unsigned int)'` ,
- 7) Начиная с версии 2.0, функциональные модули прикреплен с использованием `'attachInterrupt()'` не были проверены как действительные функциональные модули для этой цели (т.е. `'void'` вернуть, и не имея параметров вызова).
- 8) Влияние `'noInterrupts()'` на функциональные модули `'micros()'`, `'mills()'`, `'delay()'` , `'pulseInLong()'` ; его влияние на `'Stepper::step()'` и после `'read()'` а также `'peek()'` таймауты; на всех RX последовательный прием, и на `'Serial'` передача, теперь точно воспроизведена.
- 9) Время, потраченное внутри пользовательских процедур прерываний, теперь учитывается в значении, возвращаемом `'pulseIn()'` задержка, вызванная `'delayMicroseconds()'` и в положении ребер на дисплее **Pin Цифровые Осциллограммы** ,
- 10) Звонки в **объект-член** функциональные модули, которые были частью больших сложных выражений или сами были внутри вызовов функциональный модуль, имеющих несколько сложных аргументов, е, g, `'myobj.memberfunc1() + count/2'` или же `'myfunc(myobj.func1(), count/3)'` , будет иметь неверные значения, вычисленные / переданные во время выполнение из-за неправильного распределения стекового пространства.
- 11) Указатель Массивы переменные работал правильно, но отображались неверные значения в **Переменные Панель**.
- 12) Когда динамический массивы простого типа был создан с `'new'` , только первый элемент получал инициализацию (по умолчанию) к значению 0 - теперь все элементы делают.
- 13) `'noTone()'` , или конец конечного тона, больше не сбрасывает pin (остается `'OUTPUT'` и идет `'LOW'`).
- 14) 'SERVO' Устройства с непрерывным вращением теперь идеально неподвижны при длительности импульса 1500 микросекунд.
- 15) Вызов `SdFile::ls()` (список каталогов SD-карты) работал правильно, но неправильно показывал некоторые дублированные передачи SPI блоков в Waveforms окно.

V2.1.1– март 2018

- 1) Исправлены несоответствия в неанглийских локалях с языком, сохраненным в `'myArduPrefs.txt'`, с отображаемыми кнопками языка радио в **Предпочтения** диалоговое окно, и с соответствием переведенным строкам в `'myArduPrefs.txt'`.
- 2) Распределение с `'new'` Теперь примите целочисленное измерение массива, которое не является константой.
- 3) Нажав на **Переменные Панель** к расширять многомерный массива покажет лишнюю пустую `'[]'` консольно-пара ,

4) Ссылки на элементы Массива с завершающими лишними символами (например, `'y[2]12'`) не были обнаружены как ошибки во время Анализировать (дополнительные символы просто игнорировались).

V2.1– март 2018

- 1) ошибка в новых версиях V2.0.x вызывал рост кучи Окна с каждым обновлением в **Переменные Панель** -- после миллионы обновлений (много минут стоит выполнение), может произойти сбой.
- 2) Звонки на 'static'член функциональные модули с использованием двойной двоеточия `'::'` обозначение не удалось Анализировать, когда внутри `'if()'`, `'while()'`, `'for()'`, а также `'switch()'` в скобках и когда внутри выражений используются в качестве аргументов вызова функциональный модуль или индексов массива.

V2.0.2 февраль 2018

- 1) ошибка, введенный в V2.0, вызвал **Файл | Нагрузка** сбой, если `'#include'` упоминается как отсутствующий или пустой файл
- 2) Внутри **IOdevs.txt** файл, он `'I/O'` название 'One-Shot' ожидалось вместо более старого 'Oneshot'; оба теперь приняты.

V2.0.1– январь 2018

- 3) В неанглийских языковых локалях, `'en'` был неправильно показан как выбранный в **Предпочтения**, делая возврат к английскому языку неловким (требующим отмены выбора, а затем повторного выбора).
- 4) Пользователь мог оставить значение поля редактирования Устройство pin в неполном состоянии (например, `'A_'`) и оставить биты `'DATA'` `'SRS:V'` неполными.
- 5) Максимальное количество слайдеров Аналоговый было ограничено 4 (теперь исправлено 6).
- 6) UnoArduSim больше не настаивает на `'='` появляется в агрегатной инициализации массива.
- 7) UnoArduSim настоял, чтобы аргумент `"inverted_logic"` был предоставлен `'SoftwareSerial()'`,
- 8) Операции сдвига битов теперь позволяют сдвиги длиннее, чем размер сдвинутого переменная.

V2.0– декабрь 2017

- 1) Все функциональные модули, которые были объявлены как `'unsigned'` тем не менее возвращал значения, как если бы они были `'signed'`, Это не имело никакого эффекта, если `'return'` значение был назначен на `'unsigned'` переменная, но вызвало бы неправильное отрицательное толкование, если оно имеет `MSB == 1`, а затем он был назначен `'signed'` переменная, или проверено в неравенстве.
- 2) Аналоговый Слайдеры только достигли максимума `'analogRead()'` значение 1022, не правильное 1023.
- 3) ошибка случайно введен обратно в V1.7.0 в логику, используемую для ускорения обработки системы SPI. SCK pin вызвал передачу SPI для `'SPI_MODE1'` а также `'SPI_MODE3'` потерпеть неудачу после первого переданного байта (ложная дополнительная Переход SCK следовал за каждым байтом). Также были отложены обновления в поле `'SPISLV'` для редактирования `'DATA'` для переданных байтов,
- 4) Цветной LED устройство не включал `'B'` (для синего цвета) в качестве цветного варианта (хотя это было принято).
- 5) Настройки для `'SPISLV'` и `'I2CSLV'` Устройства не были сохранены пользователю **'I/O' Устройства** файл.
- 6) Копирование `'Servo'` экземпляры не удалось из-за неисправного `'Servo::Servo(Servo ©)'` реализация конструктора копирования.
- 7) Вне диапазона `'Servo.writeMicroseconds()'` значения были правильно обнаружены как ошибка, но указанные предельные значения, сопровождающие текст сообщения об ошибке, были неверными.
- 8) Законный скорость передачи 115200 не был принят при загрузке из **'I/O' Устройства** текст файл.
- 9) Электрические pin конфликты, вызванные подключенным Аналоговый Слайдер устройство, не всегда обнаруживались.

- 10) В редких случаях передача неверного указателя строки (при отсутствии строки 0-терминатор) в `'String'` функциональный модуль может привести к сбою UnoArduSim.
- 11) **Код Панель** может основной момент текущая строка ошибки Анализировать в **неправильно** Модуль программа (когда `'#include'` использовался).
- 12) Загрузка 'I/O' Устройства файл с устройство, который (ненадлежащим образом) толкнул против 'Uno' pin 13 привел к зависанию программа при появлении всплывающего сообщения об ошибке.
- 13) УноАрдуСим по ошибке позволил пользователь может вставить не шестнадцатеричные символы в TX-буфер расширенный окна для SPISLV и I2CSLV.
- 14) Инициализация строки объявления не удалось, когда значение с правой стороны было `'return'` значение от члена объект-функциональный модуль (как в `'int angle = myservo1.read();'`).
- 15) `'static'` член переменные имея явный `'ClassName::'` префиксы не распознаются, если они появляются в самом начале строки (например, при назначении на базу - `'class'` переменная),
- 16) призвание `'delete'` на указателе, созданном `'new'` был распознан, только если использовалась нотация функциональный модуль круглая скобка, как в `'delete(pptr)'`,
- 17) UnoArduSim реализация `'noTone()'` неправильно настаивал на предоставлении аргумента pin.
- 18) Изменения, которые увеличили глобальные байты 'RAM' в программа, который использовал `'String'` переменные (через **Изменить/Изучить** или же **Файл | Нагрузка**), может привести к повреждению в этом глобальном пространстве 'Uno' из-за удаления `'String'` объектов, принадлежащий старому программа, при использовании (неправильно) кучи, принадлежащей новому программа. В некоторых случаях это может привести к крушению программа. Хотя второй Нагрузка или Анализировать решил проблему, этот ошибка наконец исправлен.
- 19) Возвращаемые значения для `'Wire.endTransmission()'` а также `'Wire.requestFrom()'` оба они застряли на 0 - теперь они исправлены.

V1.7.2– февраль 2017

- 1) Прерывания на pin 2 также были (непреднамеренно) вызваны активностью сигнала на pin 3 (и наоборот).

V1.7.1– февраль 2017

- 1) Функциональный модуль `'delayMicroseconds()'` производил задержку в **Милли**-секунд (в 1000 раз больше).
- 2) Явный переменная из `'unsigned'` переменная с более длинным целым типом дал неправильный (`'signed'`) результат.
- 3) Шестнадцатеричные литералы больше `0x7FFF` есть сейчас `'long'` по определению и так теперь будет генерировать `'long'` результирующие арифметические выражения, в которые они вовлекаются.
- 4) ошибка, случайно введенный V1.7.0, предотвратил альтернативный стиль C++ переменная числовых литералов (например, `'(long) 1000*3000'` не был принят).
- 5) `'Serial'` больше не занимает много байтов в оперативной памяти 'Uno', если она никогда не нужна пользователю программа.
- 6) Объявленный пользователем глобальный переменные больше не занимает место в оперативной памяти 'Uno', если они фактически никогда не используются.
- 7) Сингл переменные объявлен как `'const'`, `'enum'` Участники, и указатели на строковые литералы, больше не занимающие места в оперативной памяти 'Uno' (по согласованию с компиляцией Arduino),
- 8) ОЗУ байтов, необходимых для `'#include'` встроенные библиотеки теперь точно соответствуют результатам условной компиляции Arduino.
- 9) С помощью `'new'` на указателе фактическая строка объявления потерпела неудачу (только позже `'new'` присвоение указателю сработало).
- 10) Исправлен ошибка, из-за которого "ожидаящий" показ каталога на SD-диске мог вызвать зависание

программа.

V1.7.0– декабрь 2016

0) Ряд проблем с обработкой пользовательских прерываний теперь исправлен:

а) Прерывает 0 и 1 ребра, которые произошли во время Arduino функциональный модуль, что блоки во время ожидания (как `'pulseIn()'`, `'shiftIn()'`, `'SPI.transfer()'`, `'flush()'`, а также `'write()'`) вызвал ошибку в потоке выполнение при возврате прерывания

б) Несколько копий локального переменные любого прерванного функциональный модуль имели появлялся в **Переменные Панель** (одна копия на возврат прерывания), и это было исправлено в V1.6.3, но другие проблемы с прерываниями остались).

в) Функциональный модуль `'delayMicroseconds()'` не создавал никакой задержки, если вызывался из подпрограммы прерывания пользователя.

г) звонки на блокировку функциональные модули вроде `'pulseIn()'` от **внутри** прерывание рутина не работала.

1) ошибка, введенный в V1.6.3, вызвал потерю обновления значения в **Переменные Панель** во время выполнения, когда значения действительно менялись (это происходило только после двух или более **Стой** или меню **ПеремОбновить** действия пользователя). Кроме того, когда Выполнить До был сделан после **Разрешить сокращение** был вызван, **Переменные Панель** изредка не перерисовывался (поэтому старые значения и local-переменные могли появляться там до следующего Шаг).

2) The **Код Панель** выделяя поведение **Шаг Вокруг** команда может показаться вводящей в заблуждение в `'if() -else'` цепи - это теперь исправлено (хотя реальная функциональность степпинга была правильной).

3) Функциональный модуль `'pulseIn()'` неправильно установил тайм-аут в миллисекундах вместо микросекунд - он также неправильно перезапускал тайм-аут, когда переходы на неактивный и активный уровни были впервые замечены.

4) Использование литералов HEX между `0x8000` а также `0xFFFF` в заданиях или арифметике с `'long'` целое число переменные дало неправильные результаты из-за непроверенный знак-расширение.

5) Передача или возвращение `'float'` из любого `'unsigned'` целочисленный тип со значением MSB = 1 дал неверные результаты из-за неисправности `'signed'` интерпретация.

6) Все `'bit_()'` функциональные модули теперь также принимает операции на `'long'` -размер переменные и тесты UnoArduSim на недопустимые битовые позиции (которые выходят за пределы размера переменная).

7) Неверный ввод в поле редактирования 'Pulse' (ширина) на 'PULSER' Устройство вызвал повреждение значения 'Period' (до тех пор, пока не будет исправлено следующей записью редактирования пользователем 'Period').

8) Удаление 'PULSER' или 'FUNCGEN' устройство с помощью Меню Конфигурировать не удаляло свой периодический сигнал из pin, который он вел (Сброс больше не требуется).

9) Возможность инициализации 1-D `'char'` массива со строкой в кавычках отсутствует, (например, `'char strg[] = "hello";'`)

10) Шестнадцатеричный дисплей в расширенный 'SERIAL' или 'SFTSER' Monitor окна показывал неверный старший значащий символ для значений байтов больше 127.

11) Осциллограмма окна не отражали программные изменения пользователя, сделанные `'analogWrite()'` когда новое значение было либо 0%, либо 100% рабочего цикла.

12) Реализация `'Serial.end()'` сейчас исправлено.

13) А `'myArduPrefs.txt'` файл с более чем 3 словами в одной строке (или пробелами в **'I/O' Устройства Имя** файл) может вызвать сбой из-за неисправного внутреннего указателя.

14) Финальная строка **'I/O' Устройства** файл не был принят, если это не так **заканчивается переводом строки**,

- 15) Добавление более четырех слайдеров Аналоговый вызвало молчаливый ошибка, который перезаписал LED 'I/O' устройство указатели
- 16) Начиная с V1.6.0, аналоговый осциллограмма образцов для **первая половина** каждого **треугольник** осциллограмма были все **нуль** (из-за ошибка в вычислении таблицы осциллограмма).
- 17) Делать повторный **Выполнить До** когда на линии точка остановки больше не требуется несколько кликов за аванс.
- 18) Передача адресных выражений в параметр функциональный модуль функциональный модуль не был принят Синтаксический анализатор.
- 19) Рекурсивный функциональные модули, который возвращал выражения, содержащие указатель или разыменованные ссылки массива, дал неверные результаты из-за сброса флагов "готовности" в этих выражениях компонентов.
- 20) призвание 'class' член-функциональные модули через **любой указатель объект переменная или выражение указателя** не работал
- 21) Пользователь функциональные модули, который возвратил значение объектов, только успешно возвратил свое значение при самом первом вызове функциональный модуль **если** вернули безымянную постройку объект (вроде 'String("dog")' - при последующих вызовах возврат был пропущен из-за застрявшего флага "готово".
- 22) Там не было никакой защиты, чтобы предотвратить команду **Окна | Serial Монитор** eРом добавляя новый 'SERIAL' устройство, когда на самом деле не было места для этого.
- 23) Если добавление фиксированного pin устройство (например, 'SPISLV') вызвало всплывающее сообщение pin конфликт, **Лабораторная Скамья Панель** перерисовка может показать дубликата "призрак" устройство наложение самого правого 'I/O' устройство (до следующей перерисовки).
- 24) Исправлены некоторые проблемы с ненадежными звуками 'PIEZO' для непериодических сигналов pin.
- 25) 'PROGMEM' переменные теперь также должен быть явно объявлен как 'const' согласиться с Arduino.
- 26) "Нет места в куче" был неправильно помечен как ошибка выполнение, когда 'SD.open()' не удалось найти названный файл или 'openNextFile()' достиг последнего файл в каталоге.
- 27) Синтаксический анализатор ошибка неправильно принял закрытие изогнутая скобка '}',
- 28) ошибка с **Переменные Панель** Исправлено удаление после возврата конструктора member-объект (ошибка применяется только для объектов, которые сами содержат другие объектов в качестве элементов).

V1.6.3– сентябрь 2016

- 1) Местный переменные любого прерванного функциональный модуль не удалялся из **Переменные Панель** при прерывании записи функциональный модуль, что приводит к нескольким копиям появляется там при возврате прерывания-функциональный модуль (и возможной возможной ошибке выполнение или сбое).
- 2) Осциллограмма окна не отражали программные изменения в 'analogWrite()' к новому рабочему циклу 0% или 100%.
- 3) Шестнадцатеричный дисплей в расширенный 'SERIAL' или 'SFTSER' Monitor окно показывал неправильный символ MSB для байтовых значений больше 127.

V1.6.2– сентябрь 2016

- 1) Функциональный модуль вызовы, сделанные с неправильным номером или типом аргументов, не сгенерировали соответствующее сообщение об ошибке Анализировать (только появилось общее сообщение "Недействительный идентификатор").
- 2) The **Инструмент-Bar** Кнопка сброса теперь работает так же, как и кнопка сброса 'Uno' Версия для Uno.
- 3) Текст ошибки Анализировать больше не обрезается после 16 символов без многоточия.

V1.6.1– август 2016

- 1) В V1.6 версия 'Uno' Версия для Uno в 'myArduPrefs.txt' файл, который отличался от значения версии 2 по умолчанию, вызвал исключение при запуске (из-за неинициализированного события pin 13).

- 2) Изменение значения переменная двойным щелчком мыши в **Переменные Панель** может вызвать ошибочные всплывающие окна с сообщениями об отсутствии выделения памяти (для программы с любым определенным пользователем 'class').
- 3) 'SoftwareSerial' не разрешил доступ к 'write(char* ptr)' а также 'write(byte* ptr, int size)' функциональные модули из-за неисправного обнаружения перегрузки функциональный модуль.
- 4) Исправлена проблема с автоматическим включением соответствующего ".cpp" файл для изолированной ".h" библиотеки '#include',

V1.6– июнь 2016

- 1) В V1.5 автоматический отступ на ключе 'Enter' в **Изменить/Изучить** (при вводе новой строки) был потерян.
- 2) Теперь добавлено обнаружение pin конфликтует с прилагается внешний проводник с сильной проводимостью 'I/O' Устройства 'Serial' pin 1, на SPI pins SS, MOSI и SCK, на I2C pins SCL и SDA (все, когда соответствующие 'begin()' называется), а по любому объявлен 'SoftwareSerial' TX pin.

V1.5.1– июнь 2016

- 1) В V1.5 новые адаптируемые к темам цвета синтаксиса Основной момент не сбрасывались каждый раз должным образом **Изменить/Изучить** был открыт, и поэтому (с темой на белом фоне) были правильными только каждый второй раз.
- 2) Прерывание 'RISING' а также 'FALLING' Чувствительность была напротив к фактической полярности края запуска.

V1.5 - май 2016

- 1) ошибка, представленный в V1.4.1, предотвращает передачу открытых строковых литералов в член функциональные модули, который ожидал 'String' объект, как в 'mystring1.startsWith("Hey")' ,
- 2) А ошибка в оригинале SD реализация UnoArduSim разрешена только SD доступ с помощью звонков 'read()' а также 'write()' (доступ через 'Stream' функциональные модули был предотвращен).
- 3) 'R=1K' Ползунковые переключатели не перерисовывались должным образом при перемещении ползунка.
- 4) **Отмена** в диалоговом окне Confirm-Сохранить файл должен был предотвратить выход из приложения.
- 5) Отсутствует закрывающая цитата или закрытие '>' -круглая скобка на пользователя файл '#include' приведет к зависанию
- 6) Исправлен ошибка в подсветке синтаксиса 'String' и пользователь 'class' или же 'struct' и расширенное выделение, чтобы включить конструктор функциональный модуль вызывает.
- 7) Исправлены незначительные проблемы в **Изменить/Изучить** с изменениями текста / выделения и **Забрать** кнопка.

V1.4.3 - апрель 2016

- 1) С помощью **Конфигурировать | 'I/O' Устройства** Добавление нового Устройства, а затем последующее удаление одного из этих недавно добавленных Устройства может привести к сбою при перезагрузке или к прекращению работы другого устройство.
- 2) Модификация 'String' переменная двойным щелчком мыши в **Переменные Панель** не удалось (новый 'String' было прочитано неправильно).
- 3) Изменения Pin на 'FUNCEN' и 'PULSER' Устройства не распознавались до тех пор, пока не был сделан первый сброс.

V1.4.2 - март 2016

- 1) V1.4.1 имел **введены** неудачный Анализировать ошибка, который препятствовал выполнению заданий, связанных с любым 'class' объектов (в том числе 'String').
- 2) Неполное исправление ошибка, сделанное в V1.4.1, вызвало 'unsigned' значения типа 'char' печатать как символы ASCII, а не как их целочисленные значения.
- 3) Сложные аргументы вызова функциональный модуль в выражениях-членах не всегда распознавались как допустимые совпадения параметров функциональный модуль.
- 4) **Все** целочисленные литералы и выражения были слишком велик 'long') и, следовательно, выполнение не отражает **фактический** переполнения (до отрицания), которые могут возникнуть в Arduino при операциях сложения / умножения, включающих 'int' размерные значения.
- 5) Выражения, включающие смесь 'signed' а также 'unsigned' целочисленные типы не всегда обрабатывались должным образом ('signed' значение будет неправильно рассматриваться как 'unsigned').
- 6) В случаях pin-конфликт сообщения об ошибках "value =" могут показывать устаревшие значения pin даже после Сброс из предыдущего конфликт, который пользователь уже очистил.

V1.4.1 - январь 2016

- 1) Звонки на 'print(char)' теперь печатайте правильно как символы ASCII (а не числовые значения).
- 2) Ответ на прерывание теперь включен по умолчанию, когда 'attachInterrupt()' называется, так что больше нет необходимости в вашем 'setup()' позвонить в разрешающий функциональный модуль 'interrupts()' ,
- 3) Несколько '#include' экземпляры пользователя файлы из внутри одного файл теперь обрабатываются правильно.

V1.4 - декабрь 2015

- 1) А **давнишний** ошибка неправильно помечен **деления на ноль** условие при делении на дробное значение меньше единицы.
- 2) Исправлена 'SoftwareSerial' (который был случайно сломан добавленным 'class' -проверка валидации члена в V1.3 релизов).
- 3) Конечные вызовы функциональный модуль с пропущенной точкой с запятой не были перехвачены, и Синтаксический анализатор пропустил следующую строку.
- 4) Плохо отформатированный **'/O' Устройства** Текст файл дал неправильное сообщение об ошибке.
- 5) Анализировать Исправлена ошибка подсветки неправильной (смежной) строки в многострочных выражениях и выражениях
- 6) Логическое тестирование указателей с использованием 'not' (! Оператор был перевернут.

V1.3 - октябрь 2015

- 1) Неправильная внутренняя обработка блокнота переменные, вызванная случайным **превышена максимальная глубина вложения блокнота** "Ошибки Анализировать.
- 2) Скобки *внутри одинарных кавычек* изогнутые скобки, точка с запятой, parentheses внутри заключенных в кавычки строк и экранированные символы обрабатывались неправильно.
- 3) Массива с пустым измерением и без списка инициализации вызвал зависание RESET, а массивы только с одним элементом не был запрещен (и вызвал их неверную интерпретацию как неверно инициализированный указатель).
- 4) Иногда ошибки Анализировать иногда приводят к неправильной (смежной) строке основной момент.
- 5) Передача указателя на не-'const' к функциональный модуль принимает указатель на 'const' был запрещен (а не наоборот).
- 6) Выражения инициализации неправильно наследуются 'PROGMEM' классификаторы из переменная инициализируются

- 7) '**PROGMEM**' объявил, что переменные неверно сосчитал их размер в байтах **дважды** против их выделения памяти 'Flash' во время процесса Анализировать.
- 8) Ввод в поле ввода 'Send' 'I2CSLV' иногда вызывает сбой из-за '**sscanf**' ошибка.
- 9) Загрузка нового программа с новым 'I/O' **Устройства** файл в своем каталоге может вызвать неактуальные pin конфликтует с **старый** pin направления.
- 10) Экранированная последовательная обработка символов была неправильно применена к полученным, а не переданным, последовательностям символов в (большем) **Serial Монитор** буферы окно.
- 11) '**while()**' а также '**for()**' петли с совершенно пустыми телами, такими как '**while(true);**' или же '**for(int k=1;k<=100;k++);**' передал Синтаксический анализатор (с предупреждением), но потерпел неудачу во время выполнение.

V1.2 - июнь 2015

- 1) Самый простой пользователь функциональные модули, который звонил либо '**digitalRead()**' или '**analogRead()**' или же '**bit()**' мог испортить их (самый первый) объявленный локальный переменная (если есть) из-за недостаточно выделенного пространства блокнота функциональный модуль (если бы только два байта блокнота были выделены в самом начале стека функциональный модуль) Любого числового выражения внутри функциональный модуль достаточно, чтобы вызвать 4-байтовое выделение блокнота, что позволяет избежать этой проблемы. Этот неудачный ошибка существует с момента первоначального выпуска V1.0.
- 2) Функциональные модули, которые являются '**void**' с ранним явным '**return**' и не '**void**' функциональные модули с более чем одним '**return**' заявление, будет видеть провал выполнение на **закрытие изогнутая скобка** (если это было достигнуто).
- 3) любой '**return**' заявления внутри '**if()**' отсутствующие контексты изогнутые скобки привели к ошибочной цели возврата к вызывающей стороне.
- 4) '**PULSER**' и длительность импульса 'FUNCGEN' или периоды значения 0 могут вызвать сбой (0 теперь не разрешен).
- 5) Там, где не было изогнутые скобки, '**else**' продолжения после '**if()**' не сработало, если они следовали за '**break**', '**continue**', или же '**return**',
- 6) когда **множественный 'enum' user- были сделаны заявления** константы определены во всех, кроме самого первого '**enum**' порожденный неисправный '**enum**' несоответствие" Ошибки Анализировать (этот ошибка был введен в V1.1).
- 7) Нулевой идентификатор самого последнего параметра функциональный модуль прототип вызвал ошибку Анализировать.
- 8) **Выполнить До** Точки останова, установленные на сложных линиях, не всегда обрабатывались должным образом (и поэтому их можно было пропустить).
- 9) '**HardwareSerial**' а также '**SoftwareSerial**' использовался закрытый TX-ожидающий буфер реализации, который не был очищен на Сброс (чтобы могли появиться оставшиеся символы последнего времени).
- 10) Синтаксический анализатор не удалось проверить на незаконное переключение битов '**float**' и арифметика указателя предпринята с недопустимыми операторами.

V1.1 - март 2015

- 1) Индексы Массива, которые были '**byte**' или же '**char**' размер переменные вызвал неправильные смещения массива (если соседний переменная содержал старший байт, отличный от 0).
- 2) При логическом тестировании указателей проверяется значение-указатель на ненулевое значение, а не само значение указателя.
- 3) любой '**return**' заявления встроены внутри '**for()**' или же '**while()**' петли были неправильно

обработаны.

- 4) Списки агрегированной инициализации для массивы из объектов или объектов, содержащие другие объектов / массивы, или полностью пустые списки инициализации, обрабатывались неправильно.
- 5) Доступ к `'enum'` значения членов с использованием `'enumname::'` префикс не поддерживается.
- 6) Инициализация строки объявления `'char[]'` массива с заключенным в кавычки строковым литералом не работал.
- 7) массива, передаваемый в функциональный модуль без предварительной инициализации, был неправильно помечен с ошибкой "используется, но не инициализирован".
- 8) Выражения указателя с именами массива были неправильно обработаны.
- 9) Параметры Функциональный модуль объявлены как `'const'` не были приняты.
- 10) Pin Аналоговый Осциллограммо окно не отображал сигналы ШИМ (`'servo.write()'` а также `'analogWrite()'`).
- 11) Элемент функциональные модули, доступ к которому осуществляется через указатель объект, дал неверный доступ к элементу.
- 12) Сигналы не обновлялись, когда **Выполнить До** точка остановки был достигнут.
- 13) Моделирование распределения регистров может завершиться ошибкой, когда параметр функциональный модуль был использован непосредственно в качестве аргумента для другого вызова функциональный модуль

V1.0.2 - август 2014

Исправлено упорядочение A0-A5 pins по периметру 'Uno' Версия для Uno,

V1.0.1 - июнь 2014

Исправлен ошибка, который урезал монтажные пасты, которые были в три раза больше количества байтов в оригинальном программа.

V1.0 - первый выпуск май 2014

ЧангES / Улучшения

V2.7.0- март 2020

- 1) В дополнение к текущей кодовой линии (зеленый, если готов к работе, красный, если ошибка), UnoArduSim Теперь поддерживает для каждого модуля, последний пользователь щелкнул или стек навигации кодовую строку (выделенную с темно-оливковым фоном), что делает это проще установить и найти линии временных точек остановки (один для каждого модуля теперь разрешено, но только один в отображаемой в данный момент модуль действует на 'Run-To').
- 2) Добавлены новые 'I/O' Устройства (и с поддержкой третьих сторон код библиотеки), в том числе 'SPI' и 'I2C' **Порт расширителя** 'SPI' и 'I2C' **Мультиплексор LED** Контроллеры и дисплеи (LED массивы, 4 буквенно-цифровые и 4-цифра или 8-цифра отображает 7-сегмент).
- 3) 'Wire' операции не больше не запрещены от пользователя внутри процедуры обработки прерываний (это поддерживает внешние прерывания от порта к 'I2C' расширителя).
- 4) Цифровой осциллограмм теперь показывают промежуточный уровень (между 'HIGH' и 'LOW'), Когда pin не будучи толкнул.
- 5) Чтобы избежать путаницы при пошаговом через более чем одного 'SPI.transfer()' инструкции, UnoArduSim теперь делает уверен, что придает 'I/O' Устройства теперь получают их (логику задержки) окончательное 'SCK' часы края до функциональный модуль возвращается.
- 6) Когда авто-элементный форматирование **предпочтение** включен, набрав закрывающую изогнутая скобка '}' в **Изменить/Изучить** Теперь вызывает переход к отступу позиции табуляции его соответствие открытие-изогнутая скобка '{' партнер.
- 7) **Переформатировать** Кнопка была добавлена **Изменить/Изучить** (Чтобы вызвать немедленные автоматические закладки отступа переформатирования) - эта кнопка включена только при автоматических закладках отступа Предпочтителен включен.
- 8) Четкое сообщение об ошибке в настоящее время происходит, когда ключевое слово префикса (например, 'const', 'unsigned', или 'PROGMEM') следует идентификатор в объявлении (оно должно предшествовать идентификатор).
- 9) Инициализирован глобальный переменные, даже если никогда не использовал позже, теперь всегда присваивается адрес памяти, и так будет появляться видно.

V2.6.0 января 2020

- 1) Добавлен символ-LCD дисплей Устройства имеющий 'SPI', 'I2C' и 4-би-параллельный интерфейс. Поддержка исходного кода библиотеки была добавлена в папку 'include_3rdParty' новой установки (и могут быть доступны с помощью нормального '#include' Директива) - пользователи могут альтернативно выбрать вместо писать свои собственные функциональные модули в толкнул ЖК устройство.
- 2) **Код Панель** Подсветка была улучшена, с отдельными основной момент цветов для готового кода строки, для кода ошибки линии, и для любого другого кода строки.
- 3) **Найти** меню и инструменты бар 'func' действия (предыдущие и следующие вниз) больше не перехода к предыдущему / следующему функциональный модуль начиная линию, и вместо этого в настоящее время взойти (или спуск) вызов стека, выделив соответствующий код строки в вызывающем (или называется) функциональный модуль, соответственно, где **Переменные Панель** содержание корректируется, чтобы показать переменные для функциональный модуль, содержащего текущую выделенную кодовую строку.
- 4) Чтобы избежать путаницы, а, **Сохранить** сделано в **Изменить/Изучить** вызывает немедленные ПЕРЕУСТАНОВКИ **Компилировать** И если Сохранить был успешным, используя последующее Отмена или Выход теперь только вернется текст к этому последнему сохраненному тексту.
- 5) Добавлен вход импульсного Шаговый Мотор ('PSTEPR') с 'STEP' (пульс), 'EN*' (включить) и 'DIR' (направление) входов и установку микро-шаги-за-стадии (1,2,4,8 или 16) ,

- 6) Оба 'STEPR' и 'PSTEPR' Устройства теперь имеют 'sync' LED (зеленый для синхронизированы или красный цвет при отключении от один или более шагов.)
- 7) 'PULSER' Устройства теперь есть выбор между микросекунд и миллисекунд для 'Period' и 'Pulse'.
- 8) Встроенный-функциональный модуль авто-пополнения больше не сохраняют типа параметра перед именем параметра.
- 9) При переключении обратно к предыдущей **Код Панель**, Его ранее выделенная строка теперь повторно выделен.
- 10) В качестве вспомогательного средства для установления временного брейк-пойнта, используя Отмена или Выход из **Изменить/Изучить** оставляет основной момент в **Код Панель** на линии последнего посещения курсора в **Изменить/Изучить**,
- 11) Определенный пользователь (или третья сторона) **'class'** теперь разрешено использовать **'Print'** или **'Stream'** в качестве базового класса. В подтверждении этого, новая папка 'include_Sys' была добавлена (в UnoArduSim папки установки), которая предоставляет исходный код для каждой базы **'class'**, В этом случае вызовы такой корпус-**'class'** функциональные модули будут обрабатываться одинаково для пользователя Код (который) может быть вошел в), вместо того, чтобы как встроенный функциональный модуль, который не может быть ступенчатыми в (такие, как **'Serial.print()'**).
- 12) Член-функциональный модуль авто-доработки теперь включают имя параметра **вместо того** его тип.
- 13) UnoArduSim Анализировать позволяет теперь имя объект в объявлении переменная быть предисловием его факультативным (и согласования) **'struct'** или **'class'** ключевое слово, за которым следует **'struct'** или **'class'** имя.

V2.5.0 октября 2019

- 1) Добавлена поддержка **'TFT.h'** библиотека (за исключением **'drawBitmap()'**), И добавил ассоциированный 'TFT' I/O Устройство (128 на 160 пикселей). Обратите внимание, что для того, чтобы избежать чрезмерных лагов в режиме реального времени во время большого **'fillXXX()'** передача, са часть переводов **'SPI'** **в середине заполнения** будет отсутствовать в **'SPI'** автобусе.
- 2) Во время больших файл переводов через **'SD'**, а часть трансфертов **'SPI'** в середине последовательности байтов аналогичным образом отсутствует в **'SPI'** шине,
- 3) Снижение **'Stream'**-usage накладных байт, так что **'RAM free'** значение более точно соответствует Arduino компиляции.
- 4) UnoArduSim Теперь предупреждает пользователя, когда **'class'** имеет несколько членов объявили на одной строке объявления.
- 5) Использование 'File | Save As'теперь устанавливает текущий каталог, который спас-в каталог.
- 6) Два пропавших без вести **'remove()'** членом функциональные модули, которые были добавлены к **'String'** учебный класс.
- 7) UnoArduSim Теперь Запрещает конструктор базовых вызовов в конструкторе функциональный модуль прототип если полное определение тела функциональный модуль сразу не вытекает (так, чтобы согласиться с Arduino компилятор).
- 8) изданиеGE время перехода цифровой осциллограмм была уменьшена для поддержки визуализации быстрых сигналов **'SPI'** при высоком увеличении.
- 9) ООН oArduSim позволяет теперь некоторые конструкторы должны быть объявлены **'private'** или **'protected'** (Для внутреннего использования класса).

V2.4 мая 2019 г.

- 1) Все файлы устройств 'I/O' теперь сохраняются в переведенной на язык форме, и вместе с файлом **Предпочтения** теперь все они сохраняются в кодировке текста UTF-8, чтобы избежать ошибок совпадения при последующем чтении.
- 2) Добавлен новый '**PROGIO**' Устройство, который является голым программируемым ведомым 'Uno' Версия для Uno, который разделяет до 4 pins вместе с **Лабораторная Скамья Панель** мастер 'Uno', - раб 'Uno' может иметь нет 'I/O' Устройства собственной.
- 3) Теперь вы можете удалить любой 'I/O' устройство, нажав на него, одновременно нажимая клавишу 'Ctrl'.
- 4) В **Изменить/Изучить** текстовое автозаполнение было добавлено для глобальных, встроенных модулей и членов переменные и функциональные модули (используйте ALT-стрелка вправо, чтобы запросить завершение, или **Войти** если список встроенных элементов в настоящее время выделяет соответствующий выбор).
- 5) В **Предпочтения** , новый выбор позволяет автоматически вставлять точку с запятой в конце строки на п **Войти** нажатие клавиши (если текущая строка является исполняемым оператором, который кажется автономным и завершенным).
- 6) прессование '**Ctrl-S**' из **Осциллограмма** окно позволяет сохранить на файл все (X, Y) указывает вдоль отображаемого участка каждого осциллограмма (где X - микросекунды от самого левого осциллограмма точка, а Y вольт).
- 7) А 'SFTSER' 'устройство теперь имеет скрытый (опционально) '**inverted**' значение (*относится как к TX, так и к RX*) который может быть добавлен после его значения скорости передачи в конце своей строки в **IODevs.txt** файл.
- 8) Добавил '**SPISettings**' класс, функциональные модули '**SPI.transfer16()**' , '**SPI.transfer(byte* buf, int count)**' , '**SPI.beginTransaction()**' , а также '**SPI.endTransaction()**' , так же как '**SPI.usingInterrupt()**' а также '**SPI.notUsingInterrupt()**' ,
- 9) Добавлена библиотека SPI функциональные модули '**SPI.detachInterrupt()**' вместе с расширением библиотеки SPI '**SPI.attachInterrupt(void myISRfunc)**' (вместо фактической библиотеки функциональный модуль '**SPI.attachInterrupt(void)**' (чтобы избежать необходимости распознавать '**ISR(int vector)**' низкоуровневые декларации прерываний функциональный модуль).
- 10) Система SPI теперь может использоваться в подчиненном режиме, либо путем '**SS**' pin (pin 10) и '**INPUT**' pin и за рулем '**LOW**' после '**SPI.begin()**' или указав '**SPI_SIV**' как дополнительный '**mode**' параметр в '**SPI.begin(int mode=SPI_MSTR)**' (еще одно расширение UnoArduSim для '**SPI.h**'). Полученные байты могут быть собраны с помощью '**rxbyte = SPI.transfer(tx_byte)**' либо внутри функциональный модуль без SPI-прерывания, либо внутри службы прерываний пользователя функциональный модуль, ранее присоединенной '**SPI.attachInterrupt(myISRfunc)**' , В подчиненном режиме, '**transfer()**' ждет, пока байт данных не будет готов в SPDR (поэтому обычно блокирует ожидание полного приема байта, но в процедуре прерывания это будет **вернуть** немедленно, потому что полученный байт SPI уже существует). В любом случае, '**tx_byte**' помещается в SPDR, поэтому будет получен прикрепленным главным SPI на следующем '**transfer()**' ,
- 11) Поддержка режима Ведомый была добавлена в реализацию UnoArduSim 'Wire.h'. Функциональный модуль '**begin(uint8_t slave_address)**' теперь доступно, как и '**onReceive(void*)**' а также '**onRequest(void*)**' ,
- 12) '**Wire.end()**' а также '**Wire.setClock(freq)**' теперь можно позвонить; последний, чтобы установить частоту SCL с '**freq**' значение 100 000 (частота SCL в стандартном режиме по умолчанию) или 400 000 (быстрый режим).

13) 'I2CSLV' Устройства теперь все отвечают на 0x00 адрес общего вызова и т. д. 0x00 больше не может быть выбран в качестве уникального адреса шины I2C для одного из этих ведомых.

14) Смоделированное выполнение задержки основных целочисленных операций и операций присваивания и массива и операции с указателями были уменьшены, и теперь 4 микросекунды добавляются для каждой операции с плавающей запятой.

V2.3 дек. 2018

1) Отслеживание теперь включено на **Инструмент-Bar** 'I/O ____S' **ползунок** для непрерывного и гладкого масштабирования значений 'I/O' устройство, к которым пользователь добавил суффикс 'S'.

2) Новый '**LED4**' 'I/O' устройство (ряд из 4 светодиодов на **4 последовательных номера pin**) был добавлен.

3) Новый '**7SEG**' 'I/O' устройство (7-сегментный LED цифра с кодом шестнадцатеричный на **4 последовательных номера pin** и с активно-низким **CS *** выберите вход), был добавлен.

4) Новый '**JUMP**' Добавлен 'I/O' устройство, который действует как переключатель между двумя 'Uno' pins. Это позволяет 'OUTPUT' pin должен быть подключен к 'INPUT' pin (см. Выше устройство для возможного использования этой новой функции).

5) Новый '**OWISLV**' 'I/O' устройство был добавлен и сторонний '<OneWire.h>' библиотека теперь может быть использована с '#include' так что пользователь программы может проверить взаимодействие с небольшим подмножеством шины '1-Wire' Устройства.

6) The **Выполнить** меню **Сброс** Теперь команда подключена к **Сброс** кнопка.

7) Для большей ясности, когда **Искусственная задержка 'loop()'** выбран под **Опции** меню, явное '**delay(1)**' вызов добавляется в конец цикла внутри '**main()**' - теперь это реальная задержка, которая может быть прервана пользовательскими прерываниями на 'Uno' pins 2 и 3.

8) Электрические pin конфликтует с с открытым стоком или CS-'I/O' Устройства (например, I2CLV или SPISLV) теперь объявлены ***только когда истинный конфликт происходит во время выполнения*** вместо немедленной ошибки при первом подключении устройство.

9) Функциональный модуль '**pulseInLong()**' теперь с точностью до 4-8 микросекунд можно согласиться с Arduino (предыдущая точность была 250 микросекунд).

10) Ошибки, помеченные во время инициализации глобальной переменной, теперь основной момент, что переменная в **Код Панель**,

V2.2 июнь 2018

1) На **Сохранить** либо из **Предпочтения** диалоговое окно, или из **Конфигурировать | 'I/O' Устройства**, the '**myArduPrefs.txt**' файл теперь сохраняется в каталоге загруженного в данный момент программа - каждый последующий **Файл | Нагрузка** затем автоматически загружает файл вместе с указанными IODevs файл из того же каталога программа.

2) Функциональный модуль '**pulseInLong()**' **был** отсутствует, но теперь был добавлен (полагается на '**micros()**' для его измерений).

3) Когда пользователь программа делает '**#include**' из '***.h**' файл, UnoArduSim теперь также автоматически пытается загрузить соответствующий '***.c**' файл ***если*** соответствующий '***.cpp**' файл не был найден.

4) Автоматическая вставка close-изогнутая скобка '}' (после каждого открытия-изогнутая скобка '{') был добавлен в **Предпочтения**,

5) Новый **Опции** выбор меню теперь позволяет '**interrupts()**' вызываться изнутри подпрограммы прерывания пользователя - это только для образовательных целей, поскольку на практике следует избегать вложения прерываний.

6) Переменная указателей на '**int**' значение теперь поддерживается (но появится всплывающее сообщение с предупреждением).

7) UnoArduSim теперь поддерживает помеченные строки программа (например, `'LabelName: count++;'` для удобства пользователя (но `'goto'` все еще *запрещено*)

8) Выполнение предупреждения теперь появляются, когда при вызове `'tone()'` может помешать активному ШИМ на pins 3 или 11, когда `'analogWrite()'` будет мешать сервоприводу, уже активному на том же pin, когда пропуск серийного символа пропущен, потому что прерывания в настоящее время отключены, и когда прерывания будут приходить так быстро, что UnoArduSim пропустит некоторые из них.

V2.1 марта 2018

1) Отображаемое **Переменные Панель** Значения теперь обновляются только каждые 30 миллисекунд (и опция Minimal может все еще уменьшить эту частоту обновления), но **ПеремОбновить** Пункт меню, запрещающий уменьшение обновления, был удален.

2) Операции, которые нацелены только на часть байтов значения переменная (например, те, которые сделаны с помощью указателей) в настоящее время вызвать изменение к тому, что значение переменная будет отражено в **Переменные Панель** дисплей.

V2.0.1 январь 2018

1) Недокументированный Arduino функциональные модули `'exp()'` а также `'log()'` были добавлены

2) 'SERVO' Устройства теперь можно выполнять непрерывным вращением (поэтому ширина импульса контролирует скорость, а не угол).

3) В **Изменить/Изучить**Закрытие изогнутая скобка `'}'` теперь автоматически добавляется при вводе открытия изогнутая скобка `'{'` если вы выбрали это **Предпочтение** .

4) Если вы нажмете **Изменить/Изучить** Строка заголовка окно `'x'` чтобы выйти, теперь у вас есть возможность прервать, если вы изменили, но не сохранили, отображаемую программа файл.

V2.0 сентябрь 2017

1) Реализация была портирована на QtCreator, поэтому графический интерфейс имеет некоторые незначительные визуальные различия, но никаких функциональных отличий, кроме некоторых улучшений:

а) Статусная строка сообщений внизу основного окна и внутри **Изменить/Изучить** было улучшено диалоговое окно и добавлено цветовое кодирование основной момент.

б) вертикальное пространство, выделенное между **Код Панель** а также **Переменные Панель** теперь настраивается через перетаскиваемый (но не видимый) разделитель на их общей границе.

с) Значения поля редактирования 'I/O' устройство теперь проверяются только после того, как пользователь переместил указатель мыши за пределы устройство - это позволяет избежать неуклюжих автоматических изменений для обеспечения допустимых значений, когда пользователь печатает.

2) UnoArduSim теперь поддерживает несколько языков через **Конфигурировать | Предпочтения**, Английский язык всегда можно выбрать, в дополнение к языку для языкового стандарта пользователя (если в папке UnoArduSim 'translations' присутствует пользовательский перевод * .qm файл для этого языка).

3) Звук теперь был изменен для использования аудио API Qt - для этого требовалось приглушить звук при определенных обстоятельствах, чтобы избежать раздражающего прерывания звука и щелчков во время более длительных оконных операционных задержек, вызванных обычными щелчками мыши пользователем - для получения более подробной информации см. раздел "Звуки". на этом.

4) Для удобства пользователя теперь используются пробелы для представления значения 0 в полях редактирования устройство-count в **Конфигурировать | 'I/O' Устройства** (так что теперь вы можете использовать пробел, чтобы удалить Устройства).

5) Не масштабированный (U) квалификатор теперь не обязателен для 'PULSER', 'FUNCGEN' и '1SHOT' Устройства (это предполагаемое значение по умолчанию).

6) UnoArduSim теперь позволяет (в дополнение к буквальным числовым значениям) `'const'` целочисленные переменные, и `'enum'`

V1.7.2– февраль 2017

1) Выбор цвета синий (B) был добавлен для LED Устройства.

V1.7.1– февраль 2017

1) Суффиксы 'L' и / или 'U' теперь принимаются в конце числовых литеральных констант (чтобы определить их как 'long' и / или 'unsigned'), а также ('0b' или же '0B' с префиксом) Константы двоичном теперь также принимаются. Любая десятичная числовая константа **начиная с '0'** в настоящее время считается **восьмеричный** значение. (согласиться с Arduino).

2) При выполнении в тесной петле, из которой нет выхода (например, 'while (x) ; x++;' где *Икс* всегда верно), нажав **Стоп** второй раз теперь гарантирует, что программа выполнение фактически останавливается (и на этой неисправной линии программа).

V1.7.0– декабрь 2016

1) Новый **Инструмент-Вар** добавлена функция, которая показывает байты оперативной памяти свободно во время программа выполнение (учитывая общее количество байтов, используемых глобальным переменные, распределением кучи и локальным стеком переменные).

2) Пользовательское прерывание функциональные модули теперь может и сами вызывать блокировку Arduino функциональные модули вроде 'pulseIn()' (но это следует использовать только с осторожностью, поскольку прерывание функциональный модуль не вернется, пока не будет завершена блокировка функциональный модуль).

3) Пользовательские прерывания больше не отключаются во время заблокированных операций потокового чтения, поэтому поведение теперь соответствует фактической операции потокового чтения Arduino.

4) Теперь вы можете входить и выходить из блокировки Arduino функциональные модули, которая может быть прервана (например, 'delay()' а также 'pulseIn()'), и сообщения в строке состояния были дополнены, чтобы показать, когда вы нажали прерывание точка остановки внутри такого функциональный модуль (или когда вы нажмете Стоп, когда выполнение в настоящее время находится внутри такого функциональный модуль).

5) Новый **Выполнить Пока** команда (и **Инструмент-Вар** пункт) был добавлен - одним щелчком мыши на любой **Переменные Панель** переменная (это может быть просто, совокупность массива или объект, или элемент массива или член объект) к основной момент это, затем сделать **Выполнить Пока** - выполнение замерзнет на следующем **записи доступа** внутри этого агрегата переменная или в это единственное место.

6) Когда выполнение зависает после **Шаг, Выполнить До, Выполнить Пока**, или же **Выполнить-затем-Стоп** действие, **Переменные Панель** Теперь выдвигает на первый план переменная, который соответствует **адрес (а) адреса, которые были изменены** (если есть) **самый последний инструкция во время этого выполнение** - если это местоположение в настоящее время скрыто внутри ип-расширенный массива или объект, щелчок по расширять приведет к тому, что последний измененный элемент или элемент будет выделен.

7) Теперь пользователь может следить за значением определенного **Переменная Панель** переменная / member / element во время выполнения - дважды щелкните по этой строке в **Переменные Панель** открыть **Изменить/Монитор Значение Переменная** окно, затем сделайте один из **Выполнить** или же **Шаг** команды - Показанное значение будет обновлено в течение выполнения в соответствии с теми же правилами, которые регулируют обновления в **Переменные Панель**, После остановки выполнения, вам разрешается вводить новое значение и **Принимать** это, прежде чем возобновить выполнение (и может **Возвращаться** до **Принимать** значение, если вы передумаете до того).

8) Клавиши ускорения F4-F10 установлены в соответствии с меню Выполнить **Инструмент-Bar** команды (слева направо).

9) В дополнение к двойному щелчку по ним, щелкните правой кнопкой мыши по 'SERIAL', 'SFTSER', 'SPISLV', 'I2CSLV' Устройства теперь будет также отображать байты / символы TX / RX большего размера окно (и на 'SD_DRV', файлы-мониторинг окно).

10) Поле редактирования TX в 'SERIAL' или 'SFTSER' больше не отключается во время активной передачи символов (так что теперь вы можете добавлять или заменять то, что там есть), но возврат каретки (или нажатие кнопки 'Send' в соответствующем "Последовательном порядке; контролировать дочерний элемент" окно) будет игнорироваться до тех пор, пока передача не вернется в состояние ожидания снова (теперь символы выделены курсивом, когда передача готова к началу, она активна). Кроме того, пользователь теперь предупрежден в последовательном потоке 'begin()' если они уже начали ранее прикрепленный устройство (в настоящее время) передачи, поскольку тогда не было бы синхронизации кадров, что приводило к ошибкам приема.

11) По умолчанию добавлено 'loop()' задержка была увеличена с 250 микросекунд до одной миллисекунды, чтобы не отставать от реального времени, когда пользователь не учитывает некоторые 'delay()' (явный или естественный) где-то внутри 'loop()' или внутри функциональный модуль, который он вызывает.

12) Массивы и простые типы теперь добавлены в поддержку выделения кучи 'new' инструкция.

13) Более широкие проверки (и связанные сообщения об ошибках) были добавлены для доступа пользователя за пределы программа (т.е. вне оперативной памяти 'Uno' или вне 'Flash' для 'PROGMEM' доступ).

14) Значения указателя в **Переменные Панель** теперь более близко напоминают фактические значения указателя Arduino.

15) Пользователь 'myArduPrefs.txt' файл теперь загружается на каждом **Файл | Нагрузка**, а не только при запуске UnoArduSim.

16) Ошибка Анализировать теперь помечается при попытке 'attachInterrupt()' для пользователя функциональный модуль, который не 'void' возврат, или который имеет параметры функциональный модуль, или который не был объявлен где-то раньше 'attachInterrupt()',

17) 'static' member-переменные теперь отображаются в верхней части **Переменные Панель** как глобальные, а не появляющиеся внутри каждого экземпляра (расширенный) объект.

18) Функциональный модуль 'availableForWrite()' был добавлен к реализации 'Serial',

19) Все специальные 'PROGMEM', 'typedef' лайк 'prog_char' а также 'prog_int16' теперь были удалены (они были объявлены устаревшими в Arduino).

20) Улучшены сообщения об ошибках для ошибок Анализировать, вызванных неверно написанными или недопустимыми типами объявлений.

21) Максимально допустимый размер программа был увеличен.

V1.6.3- сент. 2016

- 1) Добавил улучшено сообщение об ошибке разобрать, когда `'attachInterrupt()'` относится к прерыванию функциональный модуль, который не был *прототип ранее* ,
- 2) Добавлено улучшенное сообщение об ошибке Анализировать для многомерных списков инициализации массива.

V1.6.2– сент. 2016

- 1) Добавил **Найти-Text** редактировать управление в **Инструмент-Bar** упростить поиск текста (в **Код Панель** а также **Переменные Панель**).
- 2) The **Инструмент-Bar** Кнопка Сброс теперь работает идентично кнопке 'Uno' Версия для Uno Сброс.

V1.6.1– август 2016

Добавлена проверка, чтобы избежать дублирования загрузки и разбора уже предыдущего `'#include'` файлы, ,

V1.6 - июнь 2016

- 1) Добавлен новый '1SHOT' (однократный) 'I/O' Устройство, который генерирует импульс после выбранной задержки от фронта триггерного сигнала выбранной полярности.
- 2) Добавлена новая функция, которая позволяет легко изменять значения в редакторе 'I/O' устройство *масштабируется* во время выполнение, перетаскивая глобальный слайдер 'I/O_____S' Scale на основной **Инструмент-Bar** (просто введите одну букву 's' или 'S' после значения, чтобы указать масштабирование).

V1.5.1 - июнь 2016

- 1) Добавлена поддержка библиотеки EEPROM функциональные модули `'update()'` , `'put()'` а также `'get()'` , и для байтового доступа через нотацию массива, например `'EEPROM[k]'` ,
- 2) **Разрешить авто (-) Сокращаться** был добавлен в меню **ПеремОбновить** чтобы разрешить явный контроль того, будет ли расширенный массивы / объектов автоматически сокращаться, когда выполнение отстает от реального времени.
- 3) Персонажи из `'String'` переменная теперь также доступен через обозначение массива, например `'mystring[k]'` ,

V1.5 - май 2016

- 1) **Изменить/Изучить** теперь есть комбинация клавиш Ctrl-E и новая кнопка для **Компилировать** (ctrl-R), а также окно с ошибкой встроенный Анализировать, чтобы разрешить тестирование изменений без необходимости закрытия окно.
- 2) **Изменить/Изучить** сейчас сейчас также поддерживает **Переделывать** и имеет новый **Сохранить** (Ctrl-S) кнопка (эквивалентно **Принимать** плюс позже основной-окно **Сохранить**), а теперь дает выбор **'Tab'** размер (новое предпочтение, которое можно сохранить с помощью **Конфигурировать | Предпочтения**).
- 3) Все доступные для редактирования поля редактирования теперь соответствуют выбранным цветам темы ОС Окна, и для контраста, все доступные только для чтения поля редактирования 'RECV' используют белый текст на черном фоне. The **Изменить/Изучить** цвета фона и синтаксиса основной момент теперь также адаптируются к выбранной теме.
- 4) UnoArduSim теперь позволяет выбирать шрифт - этот выбор и его размер были перемещены в **Конфигурировать | Предпочтения** (так можно сохранить в `'myArduPrefs.txt'` файл).
- 5) Arduino предопределенные буквенные значения двоичном (например, `'B01011011'`) теперь разрешено.
- 6) Экранированные шестнадцатеричные, восьмеричные и 4-цифра последовательности символов в кавычках Юникода теперь могут использоваться в качестве числовых литералов.

- 7) После первоначального щелчка мышью на кнопочной панели 'PUSH' устройство пользователь может вместо этого использовать нажатие клавиши (любую клавишу) для нажатия на кнопочные контакты.
- 8) **Изменить/Изучить** теперь освобождает свое временное начальное состояние только для чтения (и удаляет выделение исходной выбранной строки) после краткой визуальной вспышки.
- 9) UnoArduSim теперь проверяет несколько 'Stepper' а также 'Servo' pin конфликты, т.е. неисправный пользователь программа пытается подключиться к pins, уже подключенному к ранее 'Stepper' или же 'Servo' переменные.
- 10) Ошибка Анализировать, вызванная отсутствием левой или правой стороны оператора (пропуская выражение LHS или RHS или переменная), теперь генерирует четкое сообщение об ошибке.
- 11) Неиспользованный 'String' учебный класс 'flags' член переменная был удален, чтобы согласиться с Arduino V1.6.6. А 'String' объект теперь занимает 6 байтов (плюс его куча символов).

V1.4.2 - март 2016

- 1) Определяемый вперед функциональные модули (то есть те, у которых нет объявления прототип до их первого вызова) теперь генерирует предупреждения (не ошибки разобрать), когда более поздний тип возвращаемого определения функциональный модуль не соответствует типу, выведенному из их первого использования.
- 2) Массивы, имеющий размерность, равную 1, больше не отклоняется (чтобы соответствовать стандартным правилам C ++).
- 3) поля редактирования больше не устанавливаются черными на белом фоне - теперь они используют палитру, установленную используемой темой ОС Окна.
- 4) 'SERIAL', 'SFTSER', 'SPISLV' и 'I2CSLV' устройство расширенный Монитор окна (открывается двойным щелчком мыши) теперь принимает цвет фона своего родителя 'I/O' Устройство.

V1.4 - декабрь 2015

- 1) 'Stepper.h' функциональность библиотеки и связанные с ней 'I/O' Устройства были добавлены.
- 2) **Все настройки и значения 'I/O' Устройство** (в дополнение к выбранному pins) теперь также сохраняются как часть выбранного пользователя 'I/O' Устройства текст файл для последующей перезагрузки.
- 3) **LED Цвет 'I/O' устройство** теперь можно установить как красный, желтый или зеленый, используя поле редактирования на устройство.
- 4) Инициализаторам объявлений Переменная теперь разрешено занимать несколько строк.
- 5) Индексы Массива теперь могут быть элементами массива.
- 6) **Конфигурировать | Предпочтения** теперь включает флажок, чтобы разрешить 'and', 'or', 'not' ключевые слова, которые будут использоваться вместо стандарта C '&&', '||', а также '!' логические операторы.
- 7) "Шоу Программа Загрузка" было перенесено на **Конфигурировать | Предпочтения**

V1.3 - октябрь 2015

- 1) The '**PUSH**' устройство теперь имеет "push-like" флажок, помеченный 'latch', чтобы сделать их "защелкивающимися" (а не "мгновенными"), то есть они будут фиксироваться в закрытом положении (и менять цвет) при нажатии, пока они не будут нажаты снова освободить контакты.
- 2) Добавлена полная возможность 'SPISLV' Устройства с выбором узла ('MODE0', 'MODE1', 'MODE2', или

же 'МОДЕЗ'). Двойной щелчок открывает буферы TX / RX окно, в которых могут быть определены предстоящие байты REPLY (TX), и для просмотра прошлых принятых (RX) байтов. Простое ведомое устройство сдвигового регистра устройство предыдущей версии было переименовано в 'SRSLV' устройство.

3) **Жирный** гарнитура теперь можно выбрать для **Код Панель** а также **Переменные Панель** (из меню **Опции**), а также **жирный подсветка ключевых слов и операторов** теперь можно включить / выключить в **Изменить/Изучить** ,

4) UnoArduSim теперь позволяет 'bool' как синоним 'boolean' ,

5) Для ясности в сообщениях об ошибках объявлению переменная больше не разрешается занимать несколько строк (кроме массивы, имеющего списки инициализаторов).

6) Скорость окраски синтаксиса в **Изменить/Изучить** был улучшен (это будет заметно при увеличении программы).

7) Дополнительные 200 микросекундные издержки (в меню **Опции**) был добавлен к каждому вызову 'loop()' - это делается для того, чтобы не отставать слишком далеко от реального времени в случае, когда Пользователь программа не добавил 'delay()' в любом месте (см. обсуждение Синхронизация ниже).

V1.2 июнь 2015

1) Библиотека SD теперь полностью реализована, и был добавлен (небольшой) 8-мегабайтный SD-диск 'I/O' устройство ('SD_DRV') (и функциональность протестирована на всех образцах Arduino SD программы).

2) Как и Arduino, UnoArduSim теперь будет автоматически преобразовывать аргумент функциональный модуль в свой адрес при вызове функциональный модуль, ожидающего передачи указателя.

3) Сообщения об ошибках Анализировать теперь более уместны, когда пропущены точки с запятой и после нераспознанных объявлений.

4) несвежий **Переменные Панель** основные линии теперь удаляются при вызове / возврате функциональный модуль.

V1.1 - март 2015

1) Основной окно теперь может быть увеличен или изменен, чтобы сделать **Код Панель** а также **Переменные Панель** шире (для больших экранов).

2) Новое меню Найти (с **Кнопки панели инструментов**) были добавлены для более быстрой навигации в **Код Панель** а также **Переменные Панель** (PgUp и PgDown или текстовый поиск со стрелкой вверх, стрелкой вниз).

3) The **Изменить/Изучить** окно теперь позволяет переходы навигации ctrl-PgUp и ctrl-PgDn (к следующей пустой строке), и был расширен **Найти / Заменить** функциональность.

4) А новый пункт был добавлен в меню **ПеремОбновить** чтобы позволить пользователю выбрать подход к экономии вычислений под тяжелым **Переменные Панель** обновить загрузки.

5) 'Uno' pins и прикрепленный LED теперь отражают любые изменения, внесенные в 'I/O' Устройства, даже когда время заморожено (то есть даже когда выполнение остановлен).

6) Другой пользователь функциональные модули теперь может вызываться изнутри пользовательского прерывания функциональный модуль (в соответствии с обновлением до Arduino 1.06).

7) А **большой шрифт** теперь можно выбрать из меню **Опции**,

V1.0.1 - июнь 2014

Осциллограмма окна теперь маркирует аналоговый pins как A0-A5 вместо 14-19.

V1.0 - первый выпуск май 2014