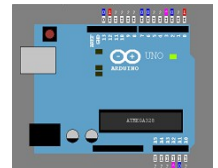


UnoArduSimV2.7 Ajuda Completa



Índice

[Visão Global](#)

[Painel de Códigos, Preferências e Editar/Examinar](#)

[Painel de Códigos](#)

[Preferências](#)

[Editar/Examinar](#)

[Painel de Variáveis e Editar/Monitorar Variável janela](#)

[Painel de Banco de Laboratório](#)

[o 'Uno'](#)

['I/O' Dispositivos](#)

['Serial' Monitor \('SERIAL'\)](#)

[Software Serial \('SFTSER'\)](#)

[Unidade de Disco SD \('SD_DRV'\)](#)

[Tela TFT \('TFT'\)](#)

[Configurável SPI Escravo \('SPISLV'\)](#)

[Clicando duas vezes \(ou clicando com o botão direito do mouse\) no dispositivo você pode abrir um companheiro maior janela que ao invés permite y ou para preencher o buffer máximo de 32 bytes \(para emular o SPI dispositivos que retorna automaticamente os dados\) e para ver os últimos 32 bytes recebidos \(todos como pares hexadecimais\). Observe que o próximo byte buffer TX é enviado automaticamente para 'DATA' apenas depois de um cheio 'SPI.transfer\(\)'](#)
[Completo!](#)

[Dois fios I2C Escravo \('I2CSLV'\)](#)

[LCD de Texto I2C \('LCDI2C'\)](#)

[LCD de Texto SPI \('LCDSPI'\)](#)

[LCD de Texto D4 \('LCD'\)D4'\)](#)

[Multiplexador DEL I2C \('MUXI2C'\)](#)

[Multiplexador DEL SPI \('MUXSPI'\)](#)

[Porta de Expansão SPI \('EXPSPi'\)](#)

[Porta de Expansão I2C \('EXPI2C'\)](#)

['1-Wire' Escravo \('OWIISLV'\)](#)

[Registro de Deslocamento Escravo \('SRSLV'\)](#)

[Gerador Único-Tiro \('1SHOT'\)](#)

[Programável 'I/O' Dispositivo \('PROGIO'\)](#)

[Pulsador Digital \('PULSER'\)](#)

[Analogico Gerador de Funções \('FUNCGEN'\)](#)

[Motor Passo a Passo \('STEPR'\)](#)

[Pulsou Motor Passo a Passo \('PSTEPR'\)](#)

[Motor DC \('MOTOR'\)](#)

[Servo Motor \('SERVO'\)](#)

[Orador Piezo \('PIEZO'\)](#)

[Resistor de slide \('R=1K'\)](#)

[Botão de Pressão \('PUSH'\)](#)

[DEL colorido \('LED'\)](#)

[Linha 4-DEL \('LED4'\)](#)

[7-Segmento DEL Dígito \('7SEG'\)](#)

[Deslizante Analógico](#)

[Pin Jumper \('JUMP'\)](#)

[Menus](#)

[Arquivo:](#)

[Carregar INO ou PDE Prog \(ctrl-L\)](#)

[Editar/Examinar \(ctrl-E\)](#)

[Salvar](#)

[Salvar Como](#)

[Próximo \('#include'\)](#)

[Anterior](#)

[Saída](#)

[Buscar:](#)

[Escalar pilha de chamadas](#)
[Descer pilha de chamadas](#)
[Definir texto Buscar \(ctrl-f\)](#)
[Buscar Próximo texto](#)
[Buscar Texto anterior](#)

[Executar:](#)

[Passo Dentro \(F4\)](#)
[Passo Acima \(F5\)](#)
[Passo Fora \(F6\)](#)
[Executar Para \(F7\)](#)
[Executar Até \(F8\)](#)
[Executar \(F9\)](#)
[Parar \(F10\)](#)
[Reinicializar](#)
[Animar](#)
[Câmera Lenta](#)

[Opções:](#)

[Passo Acima Structors/ Operadores](#)
[Alocação de Registros](#)
[Erro no Uninitialized](#)
[Adicionado 'loop\(\)' Demora](#)
[Permitir interrupções aninhadas](#)

[Configurar:](#)

['I/O' Dispositivos](#)
[Preferências](#)

[VarAtualizar:](#)

[Permitir Auto \(-\) Contrair](#)
[Mínimo](#)
[Realçar Alterar](#)

[Janelas:](#)

['Serial' Monitor](#)
[Restaurar tudo](#)
[Formas de Onda Digitais](#)
[Fomra de Onda Analógica](#)

[Socorro:](#)

[Socorro rápido Arquivo](#)
[Socorro Completo Arquivo](#)
[Erro Correções](#)
[Mudança / Melhorias](#)
[Sobre](#)

['Uno' Placa de circuito e 'I/O' Dispositivos](#)

[Cronometragens](#)

['I/O' Dispositivo Cronometragens](#)

[Sons](#)

[Limitações e Elementos não Suportados](#)

[Incluído Arquivos](#)

[Alocações Dinâmicas de Memória e RAM](#)

[Alocações de Memória 'Flash'](#)

['String' Variáveis](#)

[Bibliotecas Arduino](#)

[Ponteiros](#)

['class' e 'struct' Objetos](#)

[Escopo](#)

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

[Diretrizes Compilador](#)

[Elementos de Linguagem Arduino](#)

[C / C ++ - Elementos de Linguagem](#)

[Modelos Módulo funcional](#)

[Emulação em Tempo Real](#)

[Notas de Lançamento](#)

[Erro Correções](#)

[V2.7.0- Março 2020](#)

[V2.6.0- Janeiro 2020](#)
[V2.5.0- outubro 2019](#)
[V2.4 - maio de 2019](#)
[V2.3 - dez. 2018](#)
[V2.2 - jun. 2018](#)
[V2.1.1 - Mar. 2018](#)
[V2.1 - mar. 2018](#)
[V2.0.2 Fev. 2018](#)
[V2.0.1 - jan. 2018](#)
[V2.0 - dez. 2017](#)
[V1.7.2 - fev. 2017](#)
[V1.7.1 - fev. 2017](#)
[V1.7.0 - dez. 2016](#)
[V1.6.3- setembro de 2016](#)
[V1.6.2- setembro de 2016](#)
[V1.6.1 - ago. 2016](#)
[V1.6 - Jun. 2016](#)
[V1.5.1 - Jun. 2016](#)
[V1.5 - maio de 2016](#)
[V1.4.3 - abr 2016](#)
[V1.4.2 - mar. 2016](#)
[V1.4.1 - jan. 2016](#)
[V1.4 - dez. 2015](#)
[V1.3 - outubro de 2015](#)
[V1.2 - Jun. 2015](#)
[V1.1 - mar. 2015](#)
[V1.0.2 - ago. 2014](#)
[V1.0.1 - jun. 2014](#)

[V1.0 - primeiro lançamento maio 2014](#)

[Mudanças / Melhorias](#)

[V2.7.0- Março 2020](#)
[V2.6.0 janeiro 2020](#)
[V2.5.0 outubro 2019](#)
[V2.4 de maio de 2019](#)
[V2.3 dez. 2018](#)
[V2.2 de junho de 2018](#)
[V2.1 Mar. 2018](#)
[V2.0.1 jan. 2018](#)
[V2.0 de setembro de 2017](#)
[V1.7.2 - fev. 2017](#)
[V1.7.1 - fev. 2017](#)
[V1.7.0 - dez. 2016](#)
[V1.6.3 - set. 2016](#)
[V1.6.2 - set. 2016](#)
[V1.6.1 - ago. 2016](#)
[V1.6 - Jun. 2016](#)
[V1.5.1 - Jun. 2016](#)
[V1.5 - maio de 2016](#)
[V1.4.2 - mar. 2016](#)
[V1.4 - dez. 2015](#)
[V1.3 - outubro de 2015](#)
[V1.2 de junho de 2015](#)
[V1.1 - mar. 2015](#)
[V1.0.1 - jun. 2014](#)

[V1.0 - primeiro lançamento maio 2014](#)

Visão Global

UnoArduSim é um freeware **tempo real** (Veja para Cronometragens **restrições**) ferramenta de simulador que desenvolvi para o aluno e entusiasta do Arduino. Ele é projetado para permitir que você experimente e depure facilmente, Arduino programas **sem a necessidade de qualquer hardware real**. Ele é direcionado para o **Arduino 'Uno'** placa de circuito, e permite que você escolha de um conjunto de 'I/O' dispositivos virtual, e para configurar e conectar esses dispositivos ao seu 'Uno' virtual no **Painel de Banco de Laboratório**. - você não precisa se preocupar com erros de fiação, conexões quebradas / soltas ou dispositivos defeituoso bagunçando seu desenvolvimento e teste programa.

O UnoArduSim fornece mensagens de erro simples para qualquer erro analisar ou execução encontrado e permite a depuração com **Reinicializar**, **Executar**, **Executar Para**, **Executar Até**, **Parar** e flexível **Passo** operações no **Painel de Códigos**, com uma visão simultânea de todos os variáveis, matrizes e objetos locais globais e atualmente ativos no **Painel de Variáveis**. Executar-time matriz verificação de limites é fornecida, e estouro ATmega RAM será detectado (e a linha programa culpado destacada!). Qualquer LK42 dispositivos ligado ao entra em conflito com elétrico é sinalizado e reportado à medida que ocorrem.

Quando um INO ou PDE programa arquivo é aberto, ele é carregado no programa **Painel de Códigos**. O programa é então dado um Analisar, para transformá-lo em um executável tokenizado que está pronto para **simulado execução** (ao contrário do Arduino.exe, um executável binário autônomo é *não* criado) Qualquer erro analisar é detectado e sinalizado, destacando a linha que falhou no analisar e reportando o erro no **Barra de status** na parte inferior do aplicativo KO193 do UnoArduSim. A **Editar/Examinar** O janela pode ser aberto para permitir que você veja e edite uma versão realçada por sintaxe do seu usuário programa. Erros durante o execução simulado (como um taxa de baud com correspondência incorreta) são relatados na barra de status e por meio de uma caixa de mensagem pop-up.

O UnoArduSim V2.7 é uma implementação substancialmente completa do **Linguagem de programação do Arduino V1.8.8 conforme documentado no arduino.cc**. Página da Web Language Reference e com acréscimos, como indicado na página Descarregando, Notas da versão. Embora o UnoArduSim não suporte a implementação completa de C ++ que o Arduino.exe subjacente ao GNU compilador faz, é provável que apenas os programadores mais avançados encontrem algum elemento C / C ++ que desejem usar esteja faltando (e, claro, sempre haverá codificação de soluções para esses recursos ausentes). Em geral, suportei apenas o que eu sinto que são os recursos C / C ++ mais úteis para entusiastas e estudantes do Arduino - por exemplo, '**enum**' e '**#define**' são suportados, mas módulo funcional-ponteiros não são. Mesmo que o objetos definido pelo usuário ('**class**' e '**struct**') e (a maioria) sobrecargas do operador são suportadas, *herança múltipla não é*.

Porque UnoArduSim é um simulador de linguagem de alto nível, **somente instruções C / C ++ são suportadas**, *instruções de linguagem assembly não são*. Da mesma forma, porque não é uma simulação de máquina de baixo nível, **Os registros ATmega328 não estão acessíveis ao seu programa** para ler ou escrever, embora a alocação de registro, a passagem e o retorno sejam emulados (você escolhe isso no menu **Opções**).

Como de V2.6, UnoArduSim tem construídas-em suporte automático para um subconjunto limitado de Arduino fornecida bibliotecas, sendo estes: '**Stepper.h**', '**Servo.h**', '**SoftwareSerial.h**', '**SPI.h**', '**Wire.h**', '**OneWire.h**', '**SD.h**', '**TFT.h**' e '**EEPROM.h**' (versão 2). V2.6 introduz um mecanismo para trêsrd festa de apoio à biblioteca via arquivos fornecido no '**include_3rdParty**' pasta que pode ser encontrado no interior do UnoArduSim diretório de instalação. Para qualquer '**#include**' de bibliotecas criadas pelo usuário, o UnoArduSim **não** pesquisar a estrutura de diretórios de instalação usual do Arduino para localizar a biblioteca; ao invés disso você **precisar** para copiar o cabeçalho correspondente (".h") e a fonte (".cpp") arquivo para o mesmo diretório que o programa arquivo em que você está trabalhando (sujeito, é claro, à limitação de que o conteúdo de qualquer '**#include**' arquivo deve ser totalmente compreensível para o UnoArduSim analisador).

Eu desenvolvi o UnoArduSimV2.0 no QtCreator com suporte multilíngüe e atualmente só está disponível para o Janelas TM. Portanto para Linux ou MacOS, é um projeto para o futuro! O UnoArduSim surgiu de simuladores que desenvolvi ao longo dos anos para cursos que ensinei na Queen's University, e foi testado razoavelmente extensivamente, mas alguns erros ainda estão escondidos lá. Se você gostaria de relatar um erro, por favor, descreva-o (brevemente) em um e-mail para unoArduSim@gmail.com e **não se esqueça de anexar o seu código fonte programa Arduino erro-inducing completo** então eu posso replicar o erro e consertá-lo. Eu não estarei respondendo a relatórios individuais do erro, e não tenho prazos garantidos para correções em uma versão subsequente (lembre-se que quase sempre existem soluções alternativas!).

Felicidades,

Stan Simmons, PhD, P.Eng.
Professor Associado (aposentado)
Departamento de Engenharia Elétrica e de Computação
Universidade da Rainha
Kingston, Ontário, Canadá







Painel de Códigos, Preferências e Editar/Examinar

(Aparte: A amostra janelas mostrada abaixo está sob um Janelas-OS escolhido pelo usuário cor tema que tem uma cor de fundo azul escuro janela).


Painel de Códigos

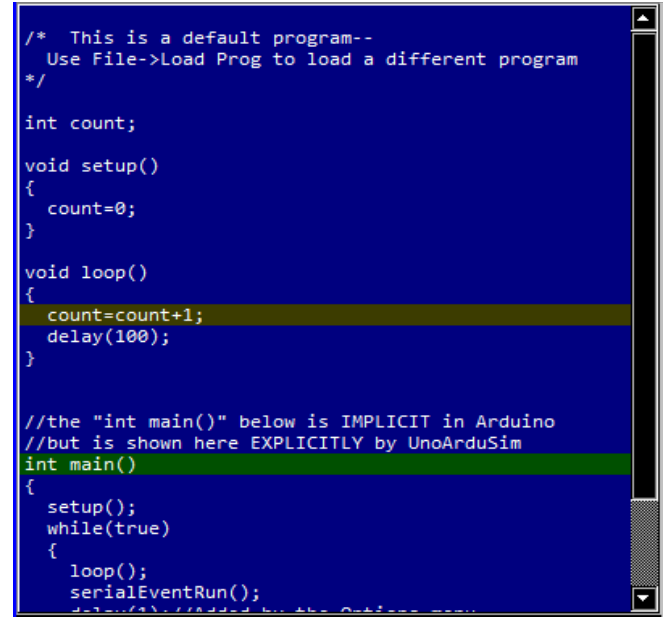
o **Painel de Códigos** exibe o seu usuário programa, e destaca as faixas do seu execução.

Depois de um programa carregado ter um Analisar de sucesso, a primeira linha 'main()' está realçado e o programa está pronto para o execução. Observe que 'main()' é adicionado implicitamente pelo Arduino (e pelo UnoArduSim) e você faz **não** incluí-lo como parte de seu usuário programa arquivo. Execução está sob controle do menu **Executar** e seus associados **Barra de ferramentas** botões e atalhos de teclas módulo funcional.

Depois de pisar execução por um (ou mais) instruções (você pode usar **Tool-Bar** botões , ,  ou ), A linha programa que será executado próxima é então destacado em verde - a linha verde-realçado é sempre a próxima linha **pronto para ser executado**.

Da mesma forma, quando um programa em execução atinge um **Executar Para)** ponto de parada, execução é interrompido e a linha ponto de parada é destacada (e fica pronta para execução).

Se programa execução está interrompida, e você clicar no **Painel de Códigos** janela, a linha que você acabou de clicar fica em destaque na verde-oliva escuro (como mostrado na imagem) - nos próximos-a-ser-executado linha fica sempre destacado em verde (a partir de V2.7). Mas você pode causar execução *progridir até* a linha que você acabou de destacar clicando em **Executar Para**  **Barra de ferramentas** botão. Esse recurso permite que você alcance linhas específicas em um programa de maneira rápida e fácil, de modo que você possa subseqüentemente percorrer linha por linha sobre uma porção de programa de interesse.





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






int count;

void setup()
{
  count=0;
}

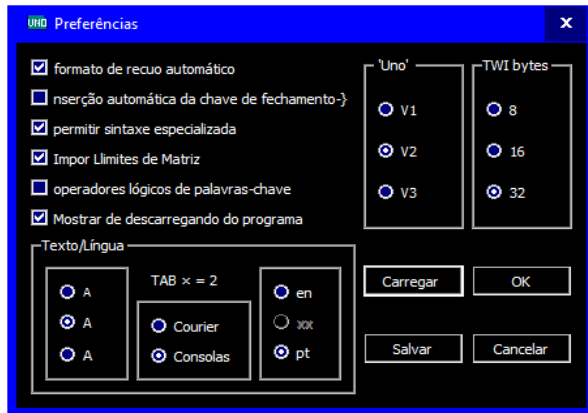
void loop()
{
  count=count+1;
  delay(100);
}

//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 Settings menu
  }
}
```

Se o seu programa carregado tiver algum '#include' arquivos, você pode se mover entre eles usando **Arquivo | Anterior** e **Arquivo | Próximo** (com **Barra de ferramentas** botões  e ).

As ações do **Buscar** menu de permitir que você **OU** encontrar texto no **Painel de Códigos** ou **Painel de Variáveis** (**Tool-Bar** botões  e , ou atalhos de teclado -se seta e seta para baixo) **após a primeira utilização** **Buscar | O texto apresentado** **Buscar** ou **Tool-Bar** , **OU ALTERNATIVAMENTE** para **navegar no call-pilha** no **Painel de Códigos** (**Tool-Bar** botões  e , ou atalhos de teclado -se seta e seta para baixo). Chaves PgDn e PgUp pular a seleção para a próxima / módulo funcional anterior.

Preferências



Configurar | Preferências permite aos usuários para definir programa e preferências de visualização (que um usuário normalmente deseja adotar em a próxima sessão). Estes podem, portanto, ser salvos e carregados de um **'myArduPrefs.txt'** arquivo que reside no mesmo diretório que o 'Uno' programa carregado (**'myArduPrefs.txt'** é carregado automaticamente se existir).

Essa caixa de diálogo permite escolher entre duas fontes mono espaçadas e três tamanhos de tipos, além de outras diversas preferências. A partir da V2.0, a escolha da linguagem agora está incluída. - isso sempre inclui inglês (**en**), mais um ou dois outros idiomas de idioma do usuário (se existirem) e uma substituição baseada no código de

idioma ISO-639 de duas letras na primeira linha do **'myArduPrefs.txt'** arquivo (se um for fornecido lá). As escolhas só aparecem se existe uma tradução ".qm" arquivo na pasta de traduções (dentro o diretório inicial do UnoArduSim.exe).

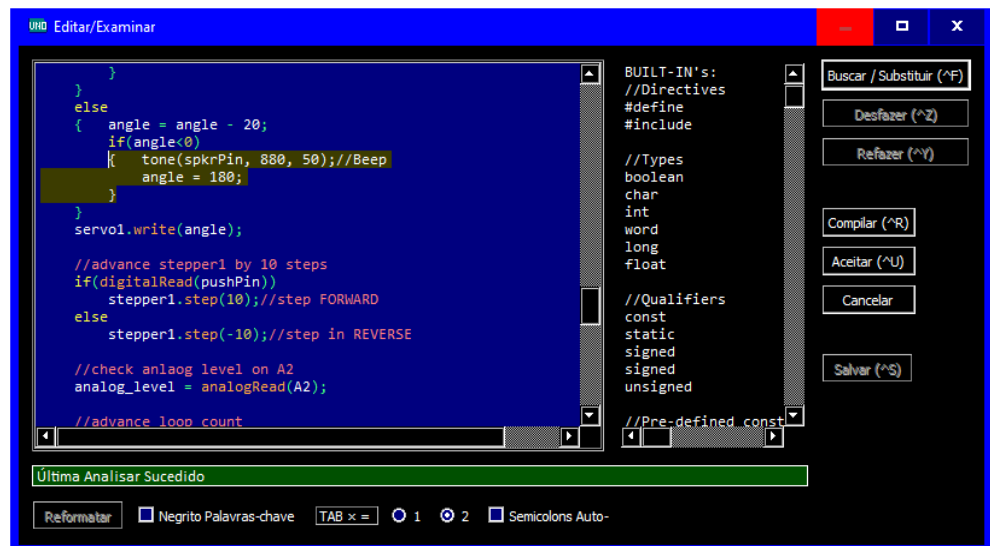
Editar/Examinar

Clicando duas vezes em qualquer linha no **Painel de Códigos** (ou usando o menu **Arquivo**), a **Editar/Examinar** janela é aberto para permitir alterações no seu programa arquivo, com o **linha atualmente selecionada** no **Painel de Códigos** está destacado.

Esta janela possui capacidade de edição completa com realce dinâmico de sintaxe (diferentes cores realçar são usadas para palavras-chave, comentários, etc.) do C ++. Há destaque de sintaxe negrito opcional e formatação automática de nível de recuo (supondo que você tenha selecionado **Configurar | Preferências**). Você também pode

selecionar convenientemente chamadas construídas-em módulo funcional (ou

construídas-em **'#define'** constantes) para serem adicionados ao seu programa a partir da lista de - basta clicar duas vezes no item desejado da lista para adicioná-lo ao seu programa na posição atual do cursor (módulo funcional-call variável **tipos** são apenas para informação e são retirados para deixar espaços reservados dummy quando adicionados ao seu programa).



O janela tem **Buscar** (usar **ctrl-f**) e **Buscar / Substituir** capacidade (uso **ctrl-H**) . o **Editar/Examinar** janela tem **Desfazer** (**ctrl-z**) e **Refazer** (**ctrl-y**) botões (que aparecem automaticamente).

Use ALT-right-arrow a solicitação escolhas auto-conclusão para construídas-em **mundo variáveis**, e para **membro variáveis** e **módulos funcionais**.

Descartar **todas as mudanças** você fez desde que abriu o programa para edição, clique no botão **Cancelar** botão. Aceitar o estado atual, clique no **Aceitar** botão e o programa recebe automaticamente outro Analisar (e é baixado

para o 'Uno' se nenhum erro for encontrado) e o novo status aparece no KO193 principal do UnoArduSim **Barra de status** .

UMA **Compilar** (**ctrl-r**) (mais um associado **Analisar Status** caixa de mensagem como visto na imagem acima) foi adicionado para permitir o teste de edições sem precisar primeiro fechar o janela. UMA **Salvar** (**ctrl-s**) também foi adicionado como um atalho (equivalente a um **Aceitar** mais um posterior separado **Salvar** do janela principal).

Em ambos **Cancelar** ou **Aceitar** sem edições feitas, o **Painel de Códigos** mudanças de linha atuais para se tornar o **última posição do Editar/Examinar** , e você pode usar esse recurso para pular o **Painel de Códigos** para uma linha específica (possivelmente para preparar uma **Executar Para**), Você também pode usar **ctrl-PgDn** e **ctrl-PgUp** para pular para a próxima (ou anterior) quebra de linha vazia em seu programa - isso é útil para navegar rapidamente para cima ou para baixo para locais significativos (como linhas vazias entre módulos funcionais). Você também pode usar **ctrl-Home** e **ctrl-end** para pular para o programa start e end, respectivamente.




'**Tab**' de nível **formatação de avanço automático** é feito quando o janela abre, se essa opção foi definido em **Configurar | Preferências**. Você pode refazer que a formatação, a qualquer momento, clicando no **Reformatar** botão (ele só é habilitado se você tiver selecionado anteriormente a **Preferências recuo automático**). Você também pode adicionar ou excluir guias-se a um grupo de linhas consecutivas pré-selecionados usando o teclado **seta direita** ou **seta esquerda** chaves - mas **Preferências recuo automático deve estar desligado** para evitar perder seus próprios níveis de tabulação personalizada.

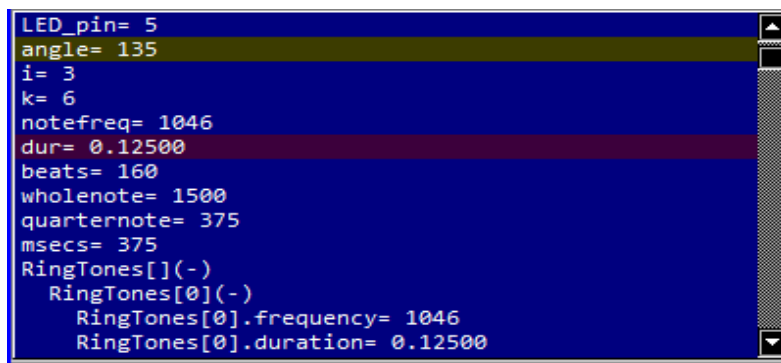
Quando **Auto ponto e vírgula** está marcada, pressionando **Entrar** terminar uma linha insere automaticamente o ponto e vírgula terminador de linha.

E para ajudá-lo a melhor acompanhar seus contextos e parênteses, clicando em um ' { ' ou ' } ' chavetas **destaca todo o texto entre esse chavetas e seu parceiro correspondente** .

Painel de Variáveis e Editar/Monitorar Variável janela

o **Painel de Variáveis** está localizado logo abaixo do **Painel de Códigos**. Ele mostra os valores atuais para cada usuário variável / matriz / Buscar local global e ativo (no escopo) no programa carregado. Como o seu programa execução se move entre módulos funcionais, **o conteúdo muda para refletir apenas os variáveis locais acessíveis ao módulo funcional / escopo atual, além de quaisquer globals declarados pelo usuário**. Qualquer variáveis declarado como 'const' ou como 'PROGMEM' (alocado para 'Flash' memória) tem valores que não podem mudar, e para economizar espaço são, portanto, *não exibido*. 'Servo' e 'SoftwareSerial' As instâncias do objeto não contêm valores úteis, portanto, também não são exibidas.

Você pode **encontrar** Especificadas **texto** com seus comandos de texto de busca (com **Tool-Bar** botões  e , ou atalhos de teclado **-se seta e seta para baixo**), Após a primeira utilização **Buscar | Set Buscar** texto ou  .

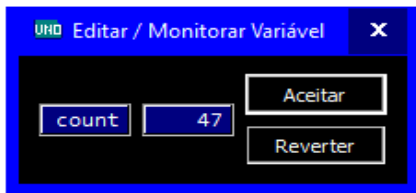


```
LED_pin= 5
angle= 135
i= 3
k= 6
notefreq= 1046
dur= 0.12500
beats= 160
wholenote= 1500
quarternote= 375
msec= 375
RingTones[0](-)
  RingTones[0](-)
    RingTones[0].frequency= 1046
    RingTones[0].duration= 0.12500
```

Matrizes e **objetos** são mostrados em qualquer **un-expandidos** ou **expandidos** formato, com um trailing plus ' (+) ' ou menos ' (-) ' assinar, respectivamente. O símbolo de um matriz **x** mostra como ' **x[]** ' . Para expandir para mostrar todos os elementos do matriz, apenas um clique sobre ' **x[] (+)** ' no **Painel de Variáveis** . Para o contrair voltar para uma visualização não-expandidos, clique no ' **x[] (-)** ' . O padrão não expandidos para um objeto ' **p1** ' mostra como ' **p1 (+)** ' . Para expandir para mostrar todos os membros desse 'class' ou 'struct' exemplo, clique uma vez em ' **p1 (+)** ' no **Painel de Variáveis** . Para o contrair voltar para uma visualização não-expandidos, clique uma vez em ' **p1 (-)** ' . Se você *clique com o único em qualquer linha para realçar-lo em oliva escuro* (Que pode ser simples variável, ou o agregado ' (+) ' ou ' (-) ' A linha de um matriz ou objeto, ou um único elemento matriz ou objeto membros), em seguida, fazendo um **Executar Até** causará execução para retomar e congelar no próximo **acesso de gravação** em qualquer lugar dentro que agregado seleccionados, ou para o seleccionado único local variável.

Ao usar **Passo** ou **Executar**, as atualizações para valores exibidos variável são feitas de acordo com as configurações do usuário feitas no menu **VarAtualizar**. Isso permite uma gama completa de comportamentos, desde atualizações periódicas mínimas até atualizações imediatas completas. Atualizações reduzidas ou mínimas são úteis para reduzir a carga da CPU e podem ser necessárias para evitar que a execução fique para trás em tempo real sob o que seria excessivo **Painel de Variáveis**. Carregamentos de atualização da janela. Quando **Animar** está em vigor, ou se a opção **Alterações no Realçar** de menu é selecionada, muda para o valor de uma variável durante **Executar** resultará em seu valor exibido sendo atualizado **imediatamente**, e torna-se destaque - isso fará com que o **Painel de Variáveis** para rolar (se necessário) para a linha que contém a variável e a execução não será mais em tempo real !.

Quando execução congela depois de **Passo**, **Executar Para**, **Executar Até** ou **Executar** -então- **Parar**, a **Painel de Variáveis** realça a variável correspondente ao **endereço local (s) que foi modificado** (se houver) pelo **última instrução** durante essa execução (incluindo por inicializações de declaração variável). Se essa instrução **completamente** encheu um **objeto ou matriz**, a **linha pai (+) ou (-)** para esse agregado fica realçada. Se, em vez disso, a instrução modificou uma localização que está atualmente visível, então fica realçada. Mas se a (s) localização (ões) modificada (s) estiver (m) atualmente escondida (s) dentro de uma matriz matriz matriz ou objeto, esse agregado **linha parental** recebe um **destaque de fonte em itálico** como uma sugestão visual de que alguma coisa dentro dele foi gravada - ao clicar para expandir, isso fará com que o **último** elemento modificado ou membro para ficar realçado.



o **Editar/Monitorar** janela dá-lhe a **capacidade de acompanhar qualquer valor de variável durante a execução**, ou para **mudar seu valor no meio de (parado) programa execução** (assim você pode testar qual seria o efeito de continuar adiante com esse novo valor). **Parar** execução primeiro, depois **esquerda, clique duas vezes** na variável cujo valor você deseja rastrear ou alterar. Para simplesmente monitorar o valor durante a execução do programa, **deixe a caixa de diálogo aberta** e então um dos **Executar** ou **Passo** comandos - seu

valor será atualizado em **Editar/Monitorar** de acordo com as mesmas regras que regem as atualizações no **Painel de Variáveis**. **Para alterar o valor de variável**, preencha o valor da caixa de edição e **Aceitar**. Continue execução (usando qualquer um dos **Passo** ou **Executar** comandos) para usar esse novo valor a partir desse ponto (ou você pode **Reverter** para o valor anterior).

Em programa Carregar ou Reinicializar note que todos os **valor não inicializado-variáveis** são redefinidos para o valor **0** e todos os ponteiros não inicializados variáveis são redefinidos para **0x0000**.

Painel de Banco de Laboratório

O Painel de Banco de Laboratório mostra um 'Uno' placa de circuito de 5 volts que é rodeado por um conjunto de 'I/O' dispositivos que você pode selecionar / personalizar e conectar ao seu 'Uno' pins desejado.

o 'Uno'

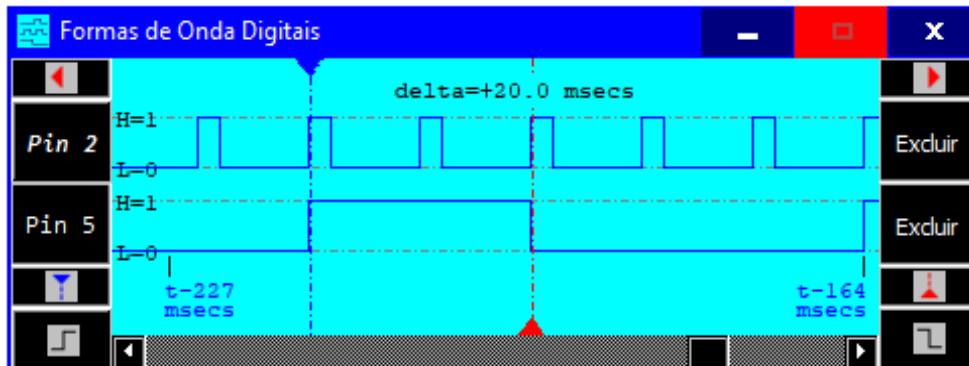
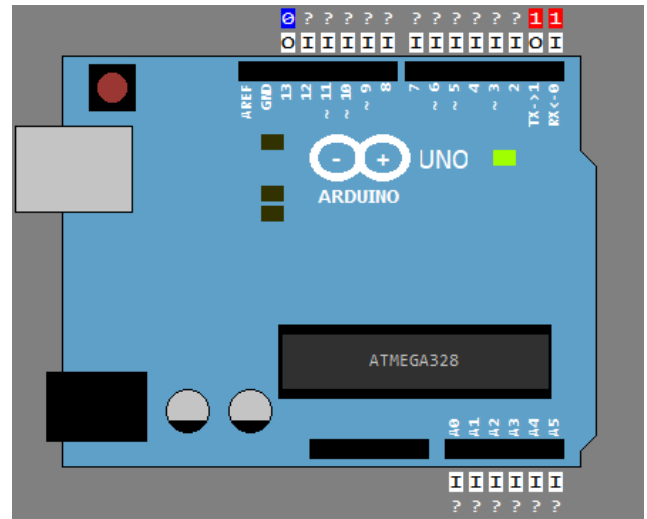
Esta é uma representação do 'Uno' placa de circuito e seus LEDs onboard. Quando você carrega um novo programa no UnoArduSim, se ele for analisado com sucesso ele passa por um "descarregando simulado" para o 'Uno' que imita a maneira como um 'Uno' placa de circuito real se comporta - você verá o RX serial e TX DEL piscando (junto com a atividade no pins 1 e 0 que são *hard-wired para comunicação serial com um computador host*). Isso é imediatamente seguido por um flash pin 13 DEL que significa placa de circuito reset e (e UnoArduSim para automática em) o início do programa execução carregado. Você pode evitar essa exibição e o atraso de carregamento associado desmarcando **Mostrar Descarregando de Configurar | Preferências**.

O janela permite visualizar os níveis lógicos digital em todos os 20 'Uno' pins ('1' no vermelho para 'HIGH', '0' em azul para 'LOW' e '?' em cinza para uma tensão indeterminada indefinida) e as direções programado ('I' para 'INPUT' ou 'O' para 'OUTPUT'). Para pins que estão sendo pulsados usando PWM via 'analogWrite()' , ou por 'tone()' , ou por 'Servo.write()' , a cor muda para roxo e o símbolo exibido torna-se '^' .

Observe que **Digital pins 0 e 1 são ligados através de resistores de 1 kOhm ao chip USB para comunicação serial com um computador host.**









Aparte: Digital pins 0-13 aparecem como simulador pins 0-13, e analógico pins 0-5 aparecem como A0-A5. Para acessar um analógico pin no seu programa, você pode consultar o número pin por um dos dois conjuntos equivalentes de números: 14-19; ou A0-A5 (A0-A5 são construídas em 'const' variáveis com valores 14-19). E somente quando usando 'analogRead()' , uma terceira opção é disponibilizada - você pode, para esta instrução, 'A' prefixo do número pin e simplesmente use 0-5. Para acessar o pins 14-19 no seu programa usando 'digitalRead()' ou 'digitalWrite()' , você pode simplesmente se referir a esse número pin, ou você pode usar os aliases A0-A5.



Clicando com o botão esquerdo em qualquer 'Uno' pin irá abrir um **Formas de Onda Digitais** janela que exibe o passado **um segundo vale a pena do Atividade no nível do digital** naquele pin. Você pode clicar em outro pins para adicioná-los ao display do Formas de Onda Digitais (até um máximo de 4 formas de onda a qualquer momento).



Clique para visualizar a página para a esquerda ou direita, ou use as teclas Home, PgUp, PgDn, End

Uma das formas de onda exibidas será a **pin ativo** forma de onda , indicado pelo botão "Pin" exibido como pressionado (como na captura de tela Formas de Onda Digitais acima). Você pode selecionar um forma de onda clicando no botão numérico Pin e, em seguida, selecionar a polaridade de borda de interesse clicando no botão

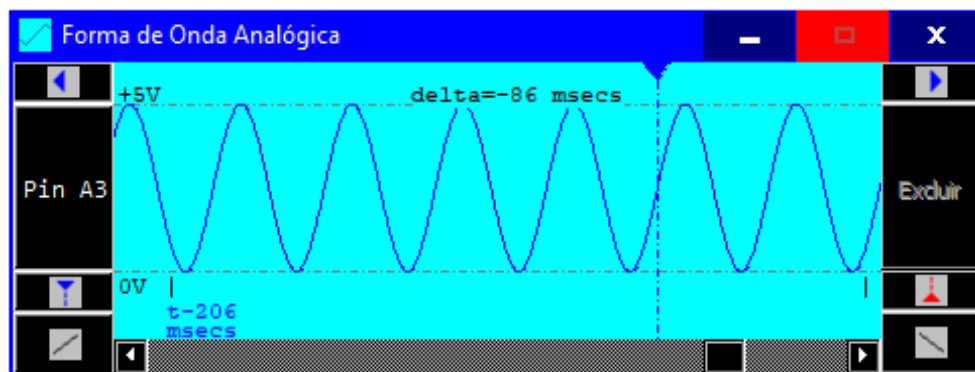
apropriado de seleção de polaridade de borda crescente / decrescente,  ou  ou usando as teclas de atalho **seta para cima** e **seta para baixo**. Você pode então **saltar** o cursor ativo (as linhas do cursor azul ou vermelho com o tempo delta mostrado) para trás ou para frente para a borda digital de polaridade escolhida *deste pin ativo* forma de onda usando os botões do cursor (,  ou ,  (dependendo de qual cursor foi ativado anteriormente com  ou ), ou simplesmente use as teclas do teclado ← e →.

Para ativar um cursor, clique no botão de ativação colorido ( ou  Mostrado acima) - *isso também rola a vista para a localização atual do aquele cursor*. Como alternativa, você pode alternar rapidamente a ativação entre os cursores (com suas visualizações centradas respectivamente) usando o atalho **'Tab'** chave.







Você pode **saltar** o cursor atualmente ativado por **clique com o botão esquerdo em qualquer lugar** na região de exibição forma de onda na tela. Alternativamente, você pode selecionar a linha de cursor vermelha ou azul clicando diretamente em cima dela (para ativá-la), então **arraste-o para um novo local** e solte. Quando um cursor desejado está atualmente em algum lugar fora da tela, você pode **clique com o botão direito em qualquer lugar** na perspectiva de pular para aquele novo local na tela. Se ambos os cursores já estiverem na tela, clicar com o botão direito simplesmente alterna entre o cursor ativado.

Para ZOOM IN e ZOOM OUT (o zoom está sempre centralizado no cursor ACTIVE), use a roda do mouse ou os atalhos de teclado CTRL-up-arrow e CTRL-down-arrow.

Fazendo um pouco **clique com o botão direito em qualquer 'Uno' pin** abre um **Forma de Onda Analógica** janela que exibe o **passado um segundo vale a pena do Atividade no nível analógico** naquele pin. Ao contrário do Formas de Onda Digitais janela, você pode exibir atividade analógico em apenas um pin por vez.



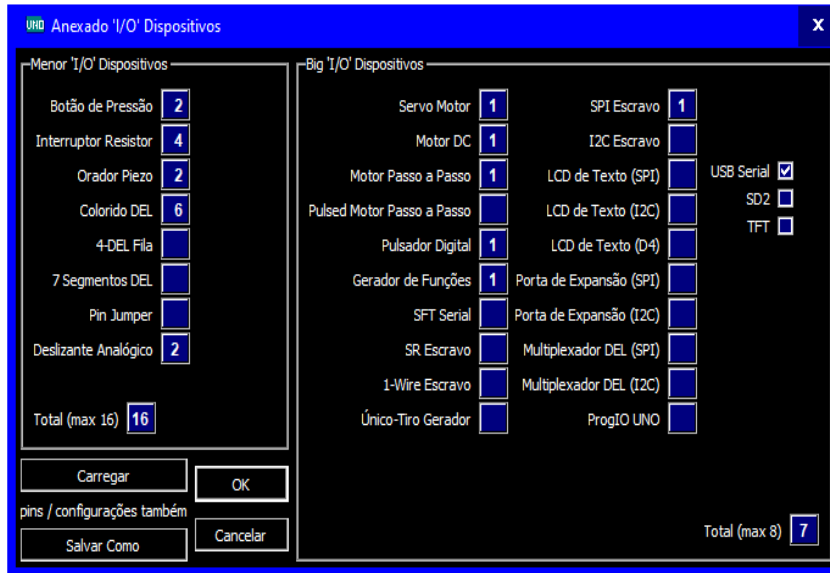
Clique para visualizar a página para a esquerda ou direita, ou use as teclas Home, PgUp, PgDn, End

Você pode **saltar** a linhas de cursor azuis ou vermelhas até o próximo "ponto de declive" crescente ou decrescente, usando os botões de seta para frente ou para trás (,  ou , , novamente dependendo do cursor ativado, ou use o ← e → chaves) em conjunto com os botões de seleção de subida / descida ,  (o "ponto de declividade" ocorre onde a tensão analógico passa pelo limiar de nível lógico digital alto do ATmega pin). Alternativamente, você pode clicar novamente para pular, ou arrastar estas linhas de cursor similares ao seu comportamento no Formas de Onda Digitais janela

Pressionando **'Ctrl-S'** dentro **ou janela permite que você salve o forma de onda Dados (X, Y)** para um texto arquivo de sua escolha, onde **X** está em microssegundos do lado esquerdo e **Y** está em parafusos.

'I/O' Dispositivos

Um número de diferentes dispositivos cercam o 'Uno' no perímetro do **Painel de Banco de Laboratório**. "Small" 'I/O' dispositivos (dos quais você tem permissão até 16 no total) reside ao longo dos lados esquerdo e direito do Painel. 'I/O' dispositivos "grande" (dos quais você tem permissão para até 8 no total) tem elementos "ativos" e residem ao longo da parte superior e inferior do **Painel de Banco de Laboratório**. O número desejado de cada tipo de 'I/O' dispositivo disponível pode ser definido usando o menu **Configurar | 'I/O' Dispositivos**.



Cada 'I/O' dispositivo tem um ou mais acessórios pin mostrados como **dois dígito** Número pin (00, 01, 02,... 10,11,12, 13 e A0-A5, ou 14-19, depois disso) em uma caixa de edição correspondente. Para números pin 2 a 9 você pode simplesmente digitar o único dígito - o 0 inicial será automaticamente fornecido, mas para pins 0 e 1 você deve primeiro inserir 0 inicial. Entradas são *normalmente* no lado esquerdo de um 'I/O' dispositivo, e as saídas são *normalmente* a direita (*espaço permitindo*). Todos os 'I/O' dispositivos responderão diretamente aos níveis de pin e alterações no nível de pin, portanto responderão à biblioteca módulos funcionais direcionada ao pins conectado ou ao programado '`digitalWrite()`' (para operação "bit-banged").

Você pode conectar vários dispositivos ao mesmo ATmega pin, desde que isso não crie um **conflito elétrico**. Tal conflito pode ser criado por um 'Uno' pin como '**OUTPUT**' condução contra um dispositivo conectado de condução forte (baixa impedância) (por exemplo, dirigindo contra uma saída 'FUNCGEN' ou um 'MOTOR' **Enc** saída), ou por dois dispositivos conectados competindo entre si (por exemplo, um botão 'PULSER' e um 'PUSH' - conectados ao mesmo pin). Qualquer conflito desse tipo seria desastroso em uma implementação de hardware real e, portanto, não é permitido e será sinalizado para o usuário por meio de uma caixa de mensagem pop-up).

A caixa de diálogo pode ser usada para permitir que o usuário escolha os tipos e números do 'I/O' dispositivos desejado. Nesta caixa de diálogo, você também pode **Salvar** 'I/O' dispositivos para um texto arquivo e / ou **Carregar** 'I/O' dispositivos de um texto previamente salvo (ou editado) arquivo (**incluindo todas as conexões pin, configurações clicáveis e qualquer valor de caixa de edição digitado**).

Observe que, a partir da versão 1.6, os valores nas caixas de edição de período, atraso e largura de pulso no IO dispositivos relevante podem receber o sufixo 'S' (ou 's'). Isso indica que eles deveriam ser dimensionado de acordo com a posição de um global '**I/O ____S**' controle deslizante que aparece no principal janela **Barra de ferramentas**. Com esse controle deslizante totalmente para a direita, o fator de escala é 1,0 (unidade) e com o controle deslizante totalmente para a esquerda, o fator de escala é 0.0 (sujeito a valores mínimos impostos por cada 'I/O' dispositivo). Você pode dimensionar mais de um valor de caixa de edição **simultaneamente** usando este controle deslizante. Este recurso permite que você arraste o controle deslizante durante a execução para facilmente emular a mudança de largura de pulso, períodos e atrasos para aqueles ligados 'I/O' dispositivos.

O restante desta seção fornece descrições para cada tipo de dispositivo.

Vários desses dispositivos **suporte escalonamento de seus valores digitados** usando o controle deslizante na janela principal **Barra de ferramentas**. Se o valor dispositivo tiver a letra 'S' como sufixo, seu valor será multiplicado

por um fator de escala (entre 0,0 e 1,0) que é determinado pela posição do ponteiro, sujeito à restrição de valor mínimo dispositivo (1.0 é totalmente para a direita, 0.0 é totalmente para a esquerda) - veja '**I/O ____S**' sob cada uma das mangueiras dispositivos



detalhadas abaixo.

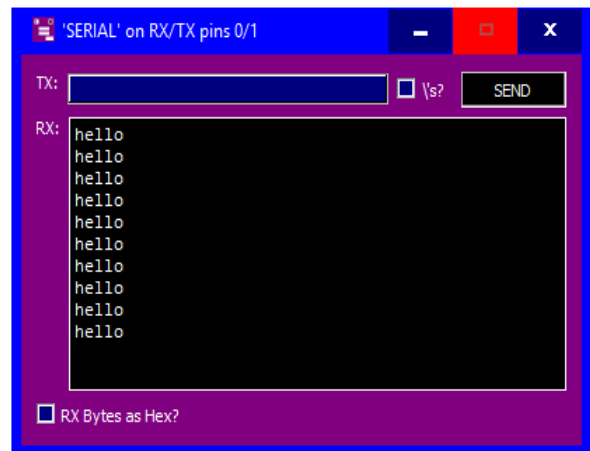
'Serial' Monitor ('SERIAL')



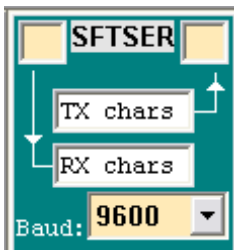
Este 'I/O' dispositivo permite entrada e saída serial mediada por hardware ATmega (através do chip USB 'Uno') em 'Uno' pins 0 e 1. O taxa de baud é definido usando a lista suspensa na parte inferior - o taxa de baud selecionado **deve combinar** o valor que seu programa passa para o '`Serial.begin()`' módulo funcional para transmissão / recepção adequada. A comunicação serial é fixada em 8 bits de dados, 1 bit de parada e nenhum bit de paridade. Você tem permissão para **desconectar** (em branco) **mas não substitua** TX pin 00, mas não RX pin 01.

Para enviar a entrada do teclado para o seu programa, digite um ou mais caracteres no topo (TX chars), edite janela e depois Acerte o '**Enter**' tecla do teclado. (os caracteres ficam em *itálico* para indicar que as transmissões começaram) - ou se já estiverem em andamento, os caracteres digitados adicionados estarão em *itálico*. Você pode então usar o '`Serial.available()`' e '`Serial.read()`' módulos funcionais para ler os caracteres na ordem em que foram recebidos no buffer pin 0 (o caractere digitado mais à esquerda será enviado primeiro). Impressões textuais e numéricas formatadas, ou valores de byte não formatados, podem ser enviadas para a saída inferior do console (caracteres RX) janela chamando o Arduino '`print()`', '`println()`' ou '`write()`' módulos funcionais

Além disso, **um janela maior para definir / visualizar os caracteres TX e RX pode ser aberto clicando duas vezes (ou clicando com o botão direito) neste 'SERIAL' dispositivo**. Este novo janela tem uma caixa de edição de caracteres TX maior e um botão 'Send' separado que pode ser clicado para enviar os caracteres TX para o 'Uno' (no pin 0). Há também uma opção de caixa de seleção para reinterpretar sequências de caracteres com escape de barra invertida, como '`\n`' ou '`\t`' para exibição não bruta.



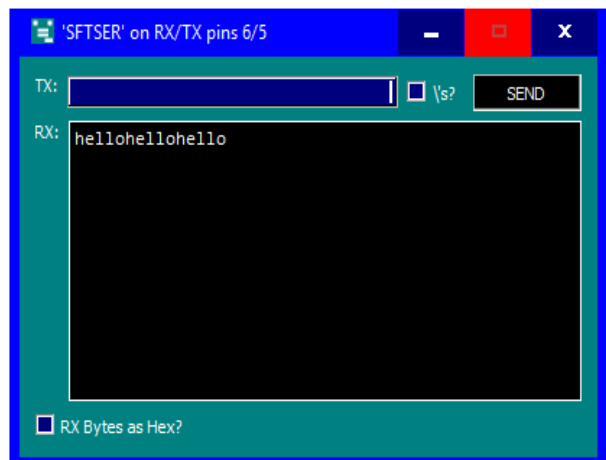
Software Serial ('SFTSER')



Este 'I/O' dispositivo permite entrada e saída serial mediada por software, ou, alternativamente, usuário "bit-banged", em qualquer par de 'Uno' pins que você escolher preencher (**exceto por** pins 0 e 1 que são dedicados ao hardware '`Serial`' comunicação). O seu programa deve ter um '`#include <SoftwareSerial.h>`' linha perto do topo, se você deseja usar a funcionalidade dessa biblioteca. Tal como acontece com o 'SERIAL' dispositivo baseado em hardware, o taxa de baud para 'SFTSER' é definido usando a lista suspensa na parte inferior - o taxa de baud selecionado deve corresponder ao valor que seu programa passa para o '`begin()`' módulo funcional para transmissão / recepção adequada. A comunicação serial é fixada em 8 bits de dados, 1 bit de parada e nenhum bit de paridade.

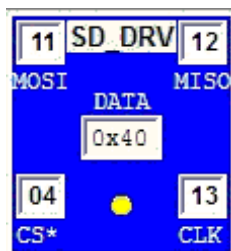
Além disso, como acontece com o hardware '`SERIAL`', **um janela maior para configuração / visualização de TX e RX pode ser aberto clicando-se duas vezes (ou clicando com o botão direito) no SFTSER dispositivo**.

Observe que, diferentemente da implementação de hardware '`Serial`', não é fornecido TX buffer suportado por operações internas de interrupção ATmega (apenas um buffer RX), então naquela '`write()`' (ou '`print`') as chamadas estão bloqueando (ou seja, o seu programa não continuará até que estejam concluídas).



Unidade de Disco SD ('SD_DRV')

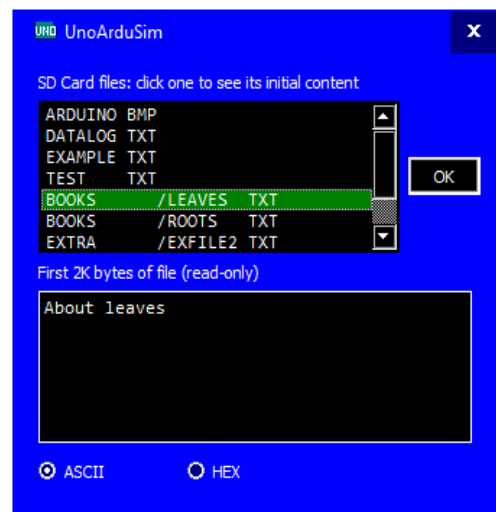
Este 'I/O' dispositivo permite mediação de software de biblioteca (mas **não** "bit-banged") arquivo operações de entrada e saída no 'Uno' SPI pins (você pode escolher qual **CS** * pin você usará). O seu programa pode simplesmente `'#include <SD.h>'` linha perto do topo, e você pode usar `'<SD.h>'` módulos funcionais OU ligar diretamente `'SdFile'` módulos funcionais você mesmo.



*Um janela maior exibindo diretórios e arquivos (e conteúdo) pode ser aberto clicando duas vezes (ou clicando com o botão direito) no 'SD_DRV' dispositivo . Todo o conteúdo do disco é **carregado de a SD** subdiretório no diretório programa carregado (se existir) em `'SdVolume::init()'` , e é **espelhado para** o mesmo SD subdiretório no arquivo `'close()'` , `'remove()'` e em*

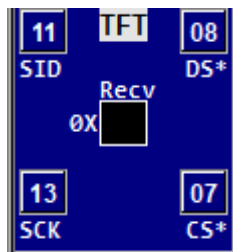
`'makeDir()'` e `'rmDir()'` .

Um DEL amarelo pisca durante as transferências SPI, e o 'DATA' mostra o último 'SD_DRV' **resposta** byte. Todos os sinais SPI são precisos e podem ser visualizados em um **Forma de onda janela**.



Tela TFT ('TFT')

Este dispositivo 'I/O' emula um Adafruit™ TFT de tamanho 1280by-160 pixels (na sua rotação nativa = 0, mas quando se utiliza a biblioteca 'TFT.h', `'TFT.begin()'` conjuntos de inicialização para rotação = 1, que dá uma visão "paisagem" de 160-por-128 pixels). Você pode impelir este dispositivo chamando o módulos funcionais do `'TFT.h'` biblioteca (que requer primeiro `'#include <TFT.h>'`), ou você pode usar o sistema SPI para enviar-lhe própria seqüência de bytes a impelir-lo.

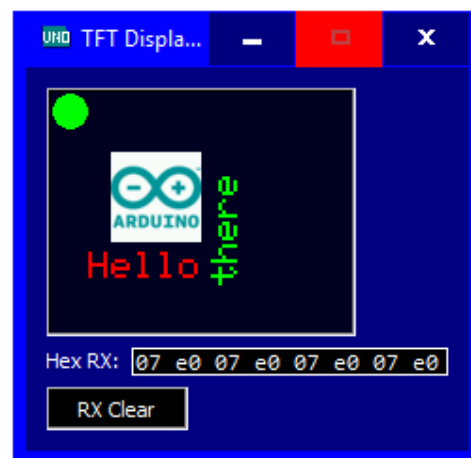


O TFT está sempre conectado para 'SPI' pins 'MOSI' (para 'SID') e 'SCK' (para 'SCK') - aqueles que não podem ser alterados. O 'DS*' pin é para dados / comando selecionar (modo de dados selecciona 'LOW'), e o 'CS*' pin é o chip-select activo de baixa

Não há Reinicializar pin fornecido para que você não pode fazer um reset de hardware este dispositivo por condução de um baixo pin (como o `'TFT::begin()'` módulo funcional tenta

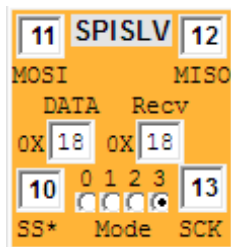
fazer quando você passou um número 'reset' pin válida como o terceiro parâmetro para o `'TFT(int cs, int ds, int rst)'` construtor). faz o dispositivo no entanto, ter uma conexão escondida à linha Reinicializar sistema para que ele repõe-se a cada vez que você clicar o principal UnoArduSim ícone da barra de ferramentas Reinicializar, ou o botão de reset 'Uno' placa de circuito ..

Por **duplo-clique** (ou **Botão direito do mouse**) Nesta dispositivo, um janela maior está aberto para mostrar que visor LCD completo 160-por-pixel de 128, juntamente com os recebido mais recentemente 8 bytes (como mostrado abaixo)



Configurável SPI Escravo ('SPISLV')

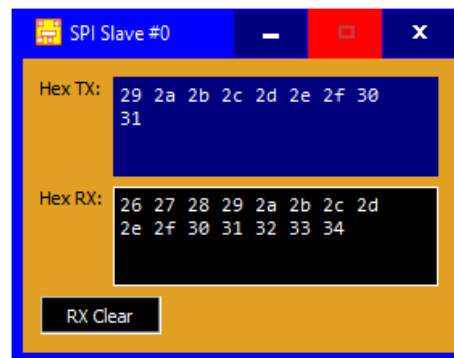
Este 'I/O' dispositivo emula um escravo SPI de modo selecionado com um **SS *** ("slave-select") pin controlando o **MISSÔ** saída pin (quando **SS *** é alto, **MISSÔ** não é impulsionado). O seu programa deve ter um `'#include <SPI.h>'` linha se você deseja usar a funcionalidade do construídas-em SPI Arduino objeto e biblioteca. Alternativamente, você pode escolher criar seu próprio "bit-banged" **MOSI** e **SCK** sinais para impelir este dispositivo.



O dispositivo detecta transições de borda em sua **CLK** entrada de acordo com o modo selecionado (`'MODE0'`, `'MODE1'`, `'MODE2'` ou `'MODE3'`), que deve ser escolhido para corresponder ao modo programado SPI do seu programa.

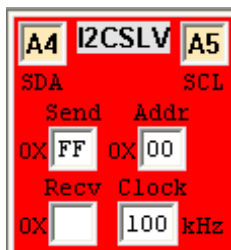
Clicando duas vezes (ou clicando com o botão direito do mouse) no dispositivo você pode abrir um companheiro maior janela que ao invés permite y

ou para preencher o buffer máximo de 32 bytes (para emular o SPI dispositivos que retorna automaticamente os dados) e para ver os últimos 32 bytes recebidos (todos como pares hexadecimais). *Observe que o próximo byte buffer TX é enviado automaticamente para 'DATA' apenas depois de um cheio 'SPI.transfer()' Completo!*



Dois fios I2C Escravo ('I2CSLV')

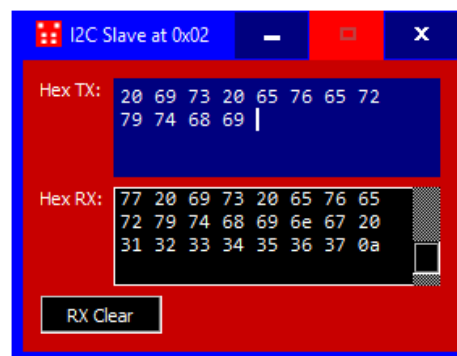
Este 'I/O' dispositivo apenas emula um *modo escravo* dispositivo. O dispositivo pode receber um endereço de barramento I2C usando uma entrada de dois hex-dígito em sua caixa de edição 'Addr' (ele só responderá a I2C transações de ônibus envolvendo seu endereço atribuído). O dispositivo envia e recebe dados em seu dreno aberto (somente pull-down) **SDA** pin, e responde ao sinal de clock do barramento em seu dreno aberto (somente pull-down) **SCL** pin. Embora o 'Uno' seja o mestre de ônibus responsável por gerar o **SCL** sinal, este escravo dispositivo também vai puxar **SCL** baixo durante sua fase baixa, a fim de estender (se necessário) o tempo baixo do barramento para um apropriado para sua velocidade interna (que pode ser ajustada em sua caixa de edição 'Clock').



O seu programa deve ter um `'#include <Wire.h>'` linha se você deseja usar a funcionalidade do `'TwoWire'` biblioteca para interagir com este dispositivo. Alternativamente, você pode optar por criar seus próprios dados bit-banged e sinais de clock para impelir este escravo dispositivo.

Um único byte para transmissão de volta ao mestre 'Uno' pode ser definido na caixa de edição 'Send', e um único byte (mais recentemente recebido) pode ser visto em sua caixa de edição (somente leitura) 'Recv'. *Observe que* o valor da caixa de edição 'Send' sempre reflete Próximo byte para transmissão deste buffer de dados interno dispositivo.

Clicando duas vezes (ou clicando com o botão direito do mouse) no dispositivo você pode abrir um companheiro maior janela que em vez disso permite que você preencha um buffer FIFO máximo de 32 bytes (de modo a emular o TWI dispositivos com tal funcionalidade) e visualize (até um máximo de 32) bytes dos dados recebidos mais recentemente (como um exibição hex-dígito de 8 bytes por linha). O número de linhas nessas duas caixas de edição corresponde ao tamanho do buffer TWI escolhido (que pode ser selecionado **Configurar | Preferências**). Isto foi adicionado como uma opção desde o Arduino `'Wire.h'` usos de biblioteca **cinco** tais buffers de RAM em seu código de implementação, que é caro memória RAM. Editando a instalação do Arduino `'Wire.h'` arquivo para alterar a constante definida `'BUFFER_LENGTH'` (e também editar o companheiro `'utility/twi.h'` arquivo para mudar TWI buffer length), ambos em vez de 16 ou 8, um usuário *poderia* reduzir significativamente a sobrecarga de memória RAM de o 'Uno' em um alvo **implementação de hardware** - O UnoArduSim, portanto, reflete essa possibilidade do mundo real **Configurar | Preferências** .

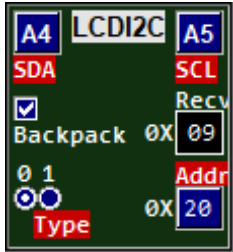


LCD de Texto I2C ('LCDI2C')

Este dispositivo 'I/O' emula um 1,2, 4-o4 linha character-LCD, em um dos três modos:

- a) mochila tipo 0 (expansor porta estilo Adafruit com hardware tendo I2C endereço de bus 0x20-0x27)
- b) tipo mochila expansor porta 1 (estilo DFRobot com hardware tendo I2C endereço de bus 0x20-0x27)
- c) não mochila (modo nativo de interface I2C integrado tendo endereço de bus I2C 0x3C-0x3F)

Apoiar o código da biblioteca para cada modo dispositivo foi fornecido dentro da pasta 'include_3rdParty' do diretório de instalação UnoArduSIm: 'Adafruit_LiquidCrystal.h', 'DFRobot_LiquidCrystal.h' e 'Native_LiquidCrystal.h', Respectivamente.



O dispositivo pode ser atribuído qualquer endereço de bus I2C usando uma entrada de dois hex-dígito em sua caixa de edição 'Addr' (ele só irá responder a I2C transações de ônibus envolvendo seu endereço atribuído). O dispositivo recebe endereço do barramento de dados e (e responde com ACK = 0 ou NAK = 1) pela respectiva drenagem aberta (pull-down-only) SDA pin. Você pode comandar LCD única gravação e dados DDRAM - você **não pode** ler dados a partir dos locais DDRAM

escritos ..



Duplo click ou **clique com o botão direito** para abrir o monitor de tela LCD janela partir do qual você também pode definir o tamanho da tela e conjunto de caracteres.

LCD de Texto SPI ('LCDSPI')

Este dispositivo 'I/O' emula um 1,2, 4-o4 linha character-LCD, em um dos dois modos:

- a) mochila (Expansor estilo porta SPI Adafruit)
- b) nenhuma mochila (modo nativo integrado SPI Interface - como mostrado abaixo)

Apoiar o código da biblioteca para cada modo dispositivo foi fornecido dentro da pasta 'include_3rdParty' do diretório de instalação UnoArduSIm: 'Adafruit_LiquidCrystal.h' ,, e 'Native_LiquidCrystal.h' , Respectivamente.



Pin 'SID' é dados seriais em, 'SS' é o baixo ativo dispositivo-selecionar, 'SCK' é o relógio pin, e 'RS' é o de dados / comandos pin. Você pode comandar LCD única gravação e dados DDRAM (todas as transações SPI são escritas) - você **não pode** ler dados a partir dos locais DDRAM escritas.

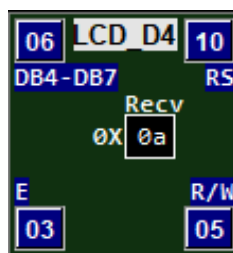
Duplo click ou **clique com o botão direito** para abrir o monitor de tela LCD janela partir do qual você também pode definir o tamanho da tela e conjunto de caracteres.



LCD de Texto D4 ('LCD')D4')

Este dispositivo 'I/O' emula um 1,2, 4-o4 linha character-LCD com uma interface de barramento de 4-bit paralelo. bytes de dados são gravados / lidos **duas metades** em seus dados 4 pins 'DB4-DB7' (onde a caixa de edição contém o *numerado mais baixo dos seus 4 números consecutivos pin*), - os dados são cronometrados no que caem sobre o bordas 'E' (permitir) pin, com direcção de dados controlado pelo 'R/W' pin, e dados LCD / modo de comando pelo 'RS' pin.

Apoiar o código da biblioteca foi fornecido dentro do 'include_3rdParty' pasta do seu diretório de instalação UnoArduSIm: 'Adafruit_LiquidCrystal.h' ,, E 'Native_LiquidCrystal.h' tanto trabalho.

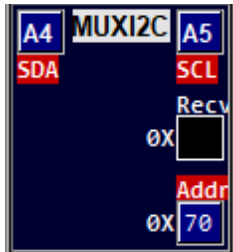


Duplo click ou **clique com o botão direito** para abrir o monitor de tela LCD janela partir do qual você também pode definir o tamanho da tela e conjunto de caracteres.



Multiplexador DEL I2C ('MUXI2C')

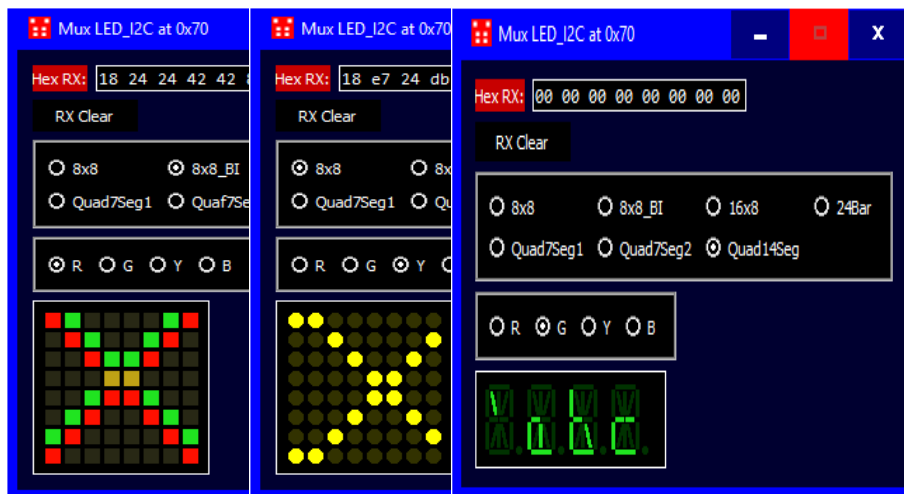
Este dispositivo 'I/O' emula um controlador HT16K33 interface I2C-(h I2C aving endereço bus 0x70-0x77) ao qual um dos vários tipos diferentes de multiplexados DEL displays podem ser anexados:



- a) 8x8 ou 16x8 DEL matriz
- b) 8x8 bicolor DEL matriz
- c) 24-bi-color-DEL barra
- d) dois estilos de 4-dígito 7 segmentos exibe
- e) um 4-dígito 14 segmentos visor alfanumérico

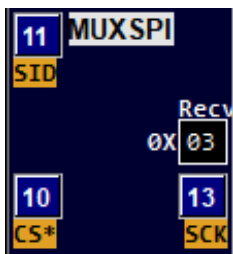
Todos são suportados pelo 'Adafruit_LEDBackpack.h' código fornecido dentro do 'include_3rdParty' pasta:

Duplo click (Ou clique com o botão direito) para abrir uma janela maior escolher e vista um dos vários DEL colorido exibida.

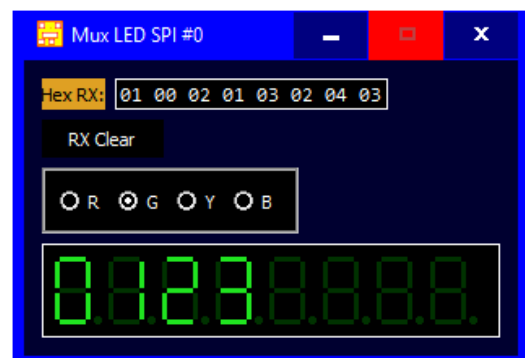


Multiplexador DEL SPI ('MUXSPI')

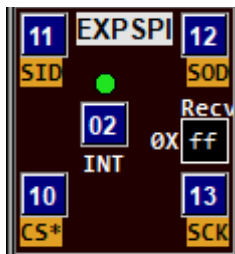
Um controlador de multiplexagem de DEL com base no MAX6219, com apoio 'MAX7219.h' código fornecido dentro da pasta para 'include_3rdParty' impelir até oito dígitos de 7 segmentos.



Duplo click (Ou clique com o botão direito) para abrir uma janela maior ver a cor 8-dígito display de 7-segment-.

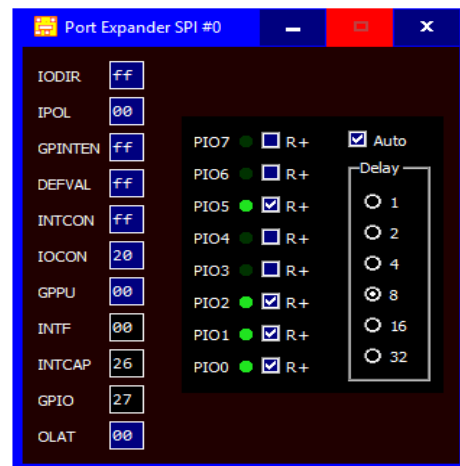


Porta de Expansão SPI ('EXPSPi')



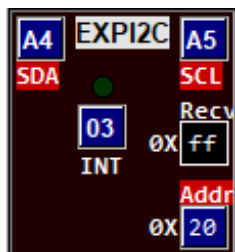
Uma 8-bit expensor porta com base no MCP23008, com apoio 'MCP23008.h' código proporcionado no interior do 'include_3rdParty' pasta. Você pode escrever para MCP23008 registros, e ler novamente o GPIO pin níveis. Interrupções pode ser habilitado em cada mudança GPIO pin - uma interrupção desencadeada vai impelir o 'INT' pin.

Duplo click (Ou clique com o botão direito) abrir **um janela maior** ver a 8 linhas de porta GPIO, e as resistências pull-up anexas. Você pode alterar pull-ups manualmente clicando, ou anexar um contador que irá periodicamente mudá-los de uma forma up-contagem. A taxa em que os incrementos de contagem é determinada pelo atraso escala para baixo fator escolhido pelo utilizador (um factor de 1 x corresponde a um incremento aproximadamente a cada 30 milissegundos; factores de atraso superiores dar uma taxa de contagem mais lentamente)

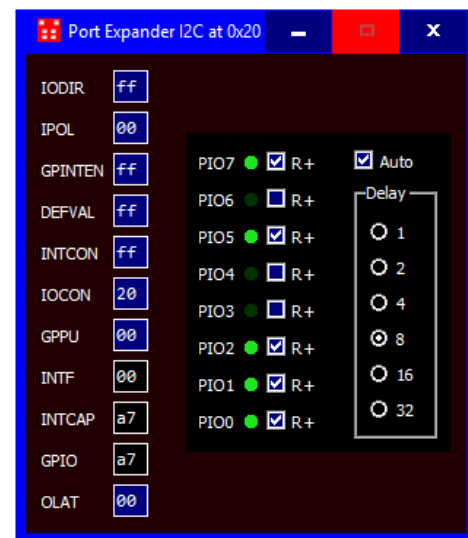


Porta de Expansão I2C ('EXPI2C')

Uma 8-bit expensor porta com base no MCP23008, com apoio 'MCP23008.h' código fornecido dentro do 'include_3rdParty' pasta. Capacidades coincidir com o 'EXPSPi' dispositivo.



Duplo click (Ou clique com o botão direito) para abrir uma janela maior como fro o 'EXPSPi' dispositivo.



'1-Wire' Escravo ('OWIISLV')

Este 'I/O' dispositivo emula um de um pequeno conjunto de barramento '1-Wire' dispositivos conectado ao pin OWIO. Você pode criar um bus '1-Wire' (com um ou mais desses escravos '1-Wire' dispositivos) no 'Uno' pin de sua escolha. Esta cabina dispositivo pode ser usada chamando 'OneWire.h' biblioteca módulos funcionais depois de colocar um '#include <OneWire.h>' linha na parte superior do seu programa. Alternativamente, você também pode usar sinais bit-banged no OWIO para este dispositivo (embora seja muito complicado fazer isso sem causar um conflito elétrico - tal conflito ainda é possível mesmo quando se usa o 'OneWire.h' módulos funcionais, mas tal conflitos são relatados no UnoArduSim).

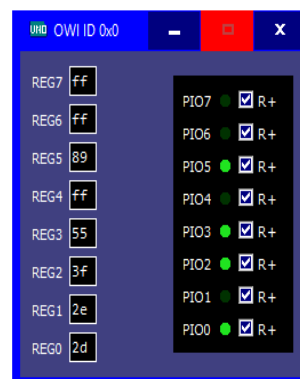
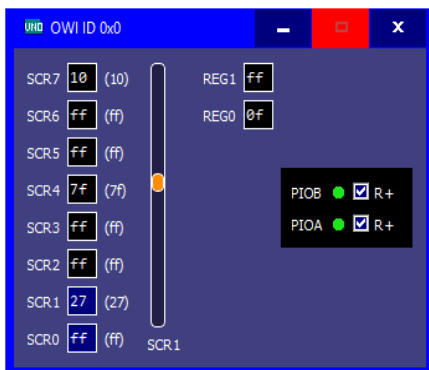


Cada OWISLV dispositivo do mundo real deve ter um i exclusivo de 8 bytes (64 bits!) número de série interno - no UnoArduSim isso é simplificado pelo usuário fornecendo um hexadecimal curto de 1 byte 'ID' valor (que é atribuído sequencialmente por default em dispositivo load / addition), mais o 'Fam' Código da família para esse dispositivo. O UnoArduSim reconhece um pequeno conjunto de códigos de Família a partir de V2.3 (0x28, 0x29, 0x3A, 0x42) cobrindo sensor de temperatura e IO paralelo (PIO) dispositivos (um código de família não reconhecido configura o dispositivo para se tornar um scratchpad genérico de 8 bytes dispositivo com sensor genérico).

Se a Família dispositivo não tiver registros PIO, registra **D0** e **D1** representar os dois primeiros bytes do bloco de notas, caso contrário eles representam o PIO "Status" (níveis reais de pin) e registrador de dados de trava PIO pin, respectivamente.

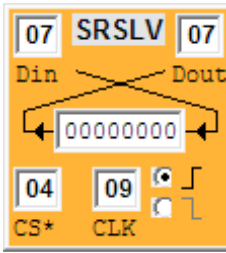
Por **clicando duas vezes** (ou **clicando com o botão direito**) no dispositivo, um maior **OWIMonitor** janela é aberto. A partir desse janela maior, você pode inspecionar todos os registradores dispositivo, alterar as localizações SCR0 e SCR1 do rascunho usando edições e um controle deslizante (novamente, SCR0 e SCR1 correspondem apenas a **D0** e **D1** se não houver um PIO) ou ajuste os pull-ups externos pin PIO. Quando o SCR0 e o SCR1 são editados, o UnoArduSim lembra esses valores editados como a "preferência" do usuário representando um valor inicial (a partir do Reinicializar) representando a saída do valor sinalizado do sensor dispositivo; o controle deslizante é redefinido para 100% (um fator de escala de 1,0) no momento da edição. Quando o Slider é subseqüentemente movido, o 'signed' O valor em SCR1 é reduzido de acordo com a posição do controle deslizante (fator de escala de 1,0 até 0,0) - seu recurso permite que você teste facilmente a resposta do seu programa para alterar os valores do sensor. .

Para um dispositivo com PIO pins, quando você define as caixas de seleção de nível pin, o UnoArduSim registra esses valores marcados como os pull-ups atuais aplicados externamente ao pins. Estes valores pull-up externos são usados junto com os dados de trava pin (registrador **D1**) para determinar os níveis reais finais de pin e acender ou apagar o DEL verde ligado ao PIO pin (o pin só vai 'HIGH' se um pull-up externo é aplicado, e o correspondente **D1** trinco é um '1').



Registro de Deslocamento Escravo ('SRSLV')

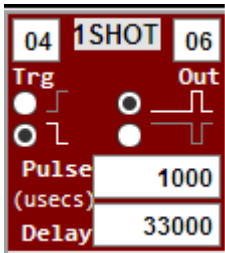
Este 'I/O' dispositivo emula um dispositivo de registro de deslocamento simples com um **SS *** ("slave-select") pin controlando o '**Dout**' saída pin (quando **SS *** é alto, '**Dout**' não é impulsionado). Seu programa poderia usar a funcionalidade do construídas-em SPI Arduino objeto e biblioteca. Alternativamente, você pode escolher criar seu próprio "bit-banged" '**Din**' e **CLK** sinais para impelir este dispositivo.



O dispositivo detecta transições de borda em sua **CLK** entrada que desencadeia mudança de seu registro - a polaridade do sentido **CLK** borda pode ser escolhido usando um controle de botão de rádio. Em cada **CLK** borda (da polaridade detectada), o registrador captura sua **Din** nivelar a posição de bit menos significativo (LSB) do registrador de deslocamento, pois os bits restantes são deslocados simultaneamente de uma posição para a posição MSB. Sempre que **SS *** é baixo, o valor atual na posição MSB do registrador de deslocamento é impulsionado para '**Dout**'.

Gerador Único-Tiro ('1SHOT')

Este 'I/O' dispositivo emula um one-shot digital que pode gerar um pulso de polaridade e largura de pulso escolhidos em seu '**Out**' pin, ocorrendo após um atraso especificado a partir de uma ponta de **Trg** (gatilho) de entrada pin. Uma vez que a borda de disparo especificada é recebida, o cronometragens começa e então um novo pulso de disparo não será reconhecido até o '**Out**' pulso foi produzido (e terminou completamente).



Uma possível utilização deste dispositivo é simular sensores de variação de ultrassom que geram um pulso de alcance em resposta a um pulso de disparo. Ele também pode ser usado onde quer que você queira gerar um sinal de entrada pin sincronizado (após o atraso escolhido) para um sinal de saída pin criado pelo seu programa.

'**Pulse**' e os valores '**Delay**' podem ser escalados a partir do janela Principal **Barra de ferramentas** Controle deslizante do fator de escala 'I/O____S' adicionando o sufixo 'S' (ou 's') para qualquer um (ou ambos).

Outro uso para este dispositivo está em testar um programa que usa interrupções, e você gostaria de ver o que acontece se um **instrução específica programa** fica interrompido. Desconecte temporariamente o 'I/O' Dispositivo que você conectou ao pin 2 (ou pin 3) e substitua por um '1SHOT' dispositivo cujo '**Out**' pin está conectado a 'Uno' pin 2 (ou pin3, respectivamente), Você pode então acionar sua entrada '**Trg**' (assumindo que a sensibilidade da borda de subida é definida lá) inserindo o par de instruções { '**digitalWrite(LOW)**' , '**digitalWrite(HIGH)**' } **somente o interior** para a instrução dentro da qual você deseja que a interrupção ocorra. Definir o '**Delay**' do 1SHOT; para cronometrar o pulso produzido em '**Out**' para acontecer dentro da instrução programa que segue este par de instrução de disparo. Note que algumas instruções mascaram interrupções (como '**SoftwareSerial.write(byte)**' , **umnd** so não pode ser interrompido.

Programável 'I/O' Dispositivo ('PROGIO')



Este 'I/O' dispositivo é na verdade um 'Uno' placa de circuito que você pode usar o programa (com um programa separado) para simular um 'I/O' dispositivo cujo comportamento você pode definir completamente. Você pode escolher até quatro pins (IO1, IO2, IO3 e IO4) que este escravo 'Uno' compartilhará em comum com o mestre (principal) Uno que aparece no meio do seu **Painel de Banco de Laboratório**. Como com outros dispositivos, qualquer conflito elétrico entre este escravo 'Uno' e o mestre 'Uno' será detectado e sinalizado. Note que todos os pins compartilhados são **diretamente** conectado, **exceto por pin 13** (onde um resistor série R-1K é assumido entre os dois pins para evitar um conflito elétrico no Reinicializar, este escravo 'Uno'

não pode ter '**I/O**' dispositivos próprio. A figura à esquerda mostra as 4 conexões pin como sendo os 4 pins do sistema SPI (**SS ***, **MISO**, **MOSI**, **SCK**) - isso permitiria a você programa este escravo como um escravo SPI genérico (ou mestre) cujo comportamento você definir.

Por **clitando duas vezes** (ou **clitando com o botão direito**) neste dispositivo, um janela maior é aberto para mostrar que este escravo 'Uno' tem seu próprio **Painel de Códigos** e associado **Painel de Variáveis**, assim como o mestre 'Uno' tem. Também tem o seu próprio **Barra de Ferramentas**, . Que você pode usar para **carga e controle execução** de um escravo programa - as ações de ícones têm os mesmos atalhos de teclado que os do janela principal. (**Carregar** é **Ctrl-L** , **Salvar** é **Ctrl-S** etc). Uma vez carregado, você pode executar de **ou** o Principal

UnoArduSim janela, ou de dentro deste Escravo Monitor janela - em ambos os casos, o Principal 'Uno' programa e Escravo 'Uno' programa permanecem trancados em sincronia com a passagem do tempo real conforme suas execuções progredem para frente. **Para escolher o Pannel de Códigos que terá o carga, pesquisa e foco execução**, **clique** em **sua barra de título janela pai - o Pannel de Códigos sem foco então tem sua barra de ferramentas ações esmaecidas**.

Algumas idéias possíveis para o escravo dispositivos que poderia ser o programado neste 'PROGIO' dispositivo estão listadas abaixo. Para emulação serial, I2C ou SPI dispositivo, você pode usar a codificação programa apropriada com matrizes para buffers de Envio e Recebimento, para para emular o comportamento complexo do dispositivo:

a) Um mestre ou escravo SPI dispositivo. UnoArduSimV2.4 foi estendido o '**SPI.h**' biblioteca para permitir o modo escravo S {PI operação através de um opcional '**mode**' parâmetro em '**SPI.begin(int mode = SPI_MASTR)**'. Explicitamente passar '**SPI_SLV**' para selecionar o modo escravo (em vez de confiar no modo mestre padrão). Você também pode agora definir uma interrupção do usuário módulo funcional (vamos chamá-lo '**onSPI**') em qualquer 'Uno' para transferir bytes chamando outra extensão adicionada '**SPI.attachInterrupt(user_onSPI)**'. **Agora** calling '**rxbyte=SPI.transfer(tx_byte)**' de dentro do seu '**user_onSPI**' módulo funcional irá limpar o sinalizador de interrupção e **Retorna imediatamente** com o byte recém-recebido em seu variável '**rxbyte**'. Alternativamente, você pode evitar anexar uma interrupção SPI e em vez disso, simplesmente ligue '**rxbyte=SPI.transfer(tx_byte)**' de dentro do seu programa principal - essa chamada **bloco execução** até que um byte SPI tenha sido transferido e irá então **Retorna** com o novo byte recebido dentro '**rxbyte**'.

b) Um genérico **serial 'I/O'** dispositivo Você precisará se comunicar entre um '**SoftwareSerial**' definido dentro do seu mestre 'Uno' programa e outro criado dentro do escravo 'Uno' programa - estes devem usar **em oposição** definiram '**txpin**' e '**rxpin**' valores de modo que o escravo 'Uno' '**SoftwareSerial**' recebe no pin em que o mestre '**SoftwareSerial**' transmite (você não pode fazer uma conexão semelhante usando a '**Serial**' porta em pins 0 e 1 em ambas as placas, como os dois '**Serial**' córregos seria impelir seus sinais TX uns contra os outros).

c) Um mestre ou escravo 'I2C' genérico dispositivo. A operação Escravo agora foi adicionada para concluir a implementação do UnoArduSim; '**Wire.h**' biblioteca (módulos funcionais '**begin(address)**', '**onReceive()**' e '**onRequest**' agora foram implementados para suportar operações no modo escravo).

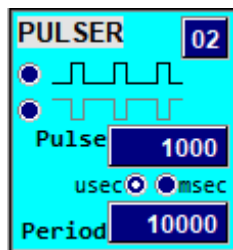
d) Um digital genérico **Pulsador**. Usando '**delayMicroseconds()**' e '**digitalWrite()**' chama dentro '**loop()**' no seu 'PROGIO' programa, você pode definir o '**HIGH**' e '**LOW**' intervalos de um trem de pulso. Adicionando um separado '**delay()**' chame dentro do seu '**setup()**' módulo funcional, você pode atrasar o início deste trem de pulso. Você pode até variar as larguras de pulso à medida que o tempo avança usando um contador variável. Você também pode usar um separado '**IOx**' pin como um gatilho para iniciar o cronometragens de um '1Shot' emulado (ou double-shot, triple-shot etc.) dispositivo, e você pode controlar a largura de pulso produzida para mudar da maneira que desejar, à medida que o tempo avança.

e) Um sinalizador aleatório. Isto é uma variação em um digital **Pulsador** que também usa chamadas para '**random()**' e '**delayMicroseconds()**' para gerar tempos aleatórios nos quais '**digitalWrite()**' um sinal em qualquer pin escolhido compartilhado com o mestre 'Uno'. Usando todos os quatro '**IOx**' O pins permitiria quatro sinais simultâneos (e únicos).

d) Um genérico 'bit-banging' dispositivo. Crie qualquer bit ou clock que você queira que o impelir sinalize para qualquer subsistema de volta no 'Uno' master. E lembre-se, **você pode usar qualquer escravo 'Uno' programa instruções ou subsistemas que você deseje**.

Pulsador Digital ('PULSER')

Este dispositivo 'I/O' emula uma simples digital pulso forma de onda gerador que produz um sinal periódico que pode ser aplicado a qualquer 'Uno' pin escolhido.



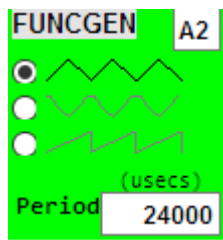
As larguras e período de impulsos (em microssegundos) pode ser definida usando edit-caixas - o período mínimo permitido é de 50 microssegundos, e o impulso de largura mínima é 10 microssegundos. Você pode escolher entre os valores cronometragens em microssegundos ('usec') e milissegundos ('msec'), e esta escolha será salvo junto com os outros valores quando você 'Save' do Configurar | I / O diálogo Dispositivos.

A polaridade também pode ser escolhido: ou impulsos positivos de ponta (0 a 5 V) ou negativos pulsos de ponta (5V a 0V).

valores 'Pulse' e 'Period' pode ser escalado a partir da principal **Tool-Bar** 'I/O ____S' controle deslizante-fator de escala por adição de um sufixo de 'S' (ou 's') para qualquer um (ou ambos). Isso permite que você altere a 'Pulse' ou valor 'Period' **dinamicamente** durante execução.

Analogóico Gerador de Funções ('FUNCGEN')

Este 'I/O' dispositivo emula um simples gerador analógico forma de onda que produz um sinal periódico que pode ser aplicado a qualquer 'Uno' pin escolhido.



O período (em microssegundos) pode ser definido usando a caixa de edição - o período mínimo permitido é de 100 microssegundos. O forma de onda que ele cria pode ser escolhido para ser sinusoidal, triangular ou dente de serra (para criar uma onda quadrada, use um 'PULSER'). Em períodos menores, menos amostras por ciclo são usadas para modelar o forma de onda produzido (apenas 4 amostras por ciclo no período = 100 microssegundos).

O 'Period' valor pode ser escalado a partir do janela principal **Barra de ferramentas** Controle deslizante de fator de escala 'I/O ____S' adicionando como um sufixo a letra 'S' (ou 's').

Motor Passo a Passo ('STEPR')

Este 'I/O' dispositivo emula um motor bipolar ou unipolar de passo de 6V com um controlador impulsor integrado impulsionado da **ou dois** (em **P1** , **P2**) **ou quatro** (em **P1** , **P2** , **P3** , **P4**) sinais de controle. O número de etapas por revolução também pode ser definido. Você pode usar o 'Stepper.h' módulos funcionais 'setSpeed()' e 'step()' para impelir o 'STEPR'. Alternativamente, o 'STEPR' *também responde* para o seu próprio 'digitalWrite()' " bit-banged "impelir.



O motor é modelado com precisão tanto mecanicamente quanto eletricamente. As quedas de tensão do motor impulsor e a relutância e a indutância variadas são modeladas juntamente com um momento de inércia realista em relação ao torque de retenção. O enrolamento do rotor do motor tem uma resistência modelada de $R = 6$ ohms e uma indutância de $L = 6$ milli-Henries que cria uma constante de tempo elétrica de 1,0 milissegundo. Por causa da modelagem realista, você notará que pulsos pin de controle muito estreito *não pegue* o motor a pisar - ambos devido ao tempo de subida da corrente finita e ao efeito da inércia do rotor. Isto concorda com o que é observado quando se está dirigindo um motor de passo real de um

'Uno' com, é claro, um apropriado (**e obrigatório**) chip do motor impulsor entre os fios do motor e o 'Uno'!

Um desafortunado erro no Arduino 'Stepper.h' código da biblioteca significa que, ao reinicializar o motor de passo, não estará na posição Passo (de quatro etapas). Para superar isso, o usuário deve usar 'digitalWrite()' em seu / sua 'setup()' rotina para inicializar os níveis de controle pin para o 'step(1)' níveis apropriados para o controle de 2-pin (0,1) ou 4-pin (1,0,1,0) e permitir que o motor 10 milissegundos se desloque para a posição inicial desejada do motor de referência de 12 Noons.

AS de V2.6, este dispositivo agora inclui um 'sync' DEL (verde para sincronizado, ou vermelho quando desligado por um ou mais passos) .Também, Além do número de passos por rotação, dois valores adicionais (ocultos) pode, opcionalmente, ser especificado no IODevs.txt arquivo para especificar o carrega- mecânica, por exemplo, valores de 20, 50, 30 especifica 20 passos por rotação, um momento de carga de inércia 50 vezes a do próprio rotor do motor, e um binário de carga de 30 por cento de torque do motor porão cheio.

Observe que **redução de marcha não é suportada diretamente** devido a falta de espaço, mas você pode emulá-lo em seu programa implementando um contador variável e apenas chamando 'step()' quando esse contador atinge 0 (para redução de marcha pelo fator N).

Pulsou Motor Passo a Passo ('PSTEPR')

Este dispositivo 'I/O' emula um 6V **micro-stepping** bipolar Motor Passo a Passo com um controlador impulsor integrado por impulsionado um 'Step' **pulsadas** pin, um activo de baixa 'EN*' (Permitir) pin, e um 'DIR' (direcção) pin . O número de passos completos por rotação também pode ser definido directamente, juntamente com o número de micro-passos por completo o passo (1,2,4,8, ou 16). Além dessas configurações, dois valores adicionais (ocultos) pode, opcionalmente, ser especificado no IODevs.txt arquivo para especificar o carrega- mecânica, por exemplo, valores de 20, 4, 50, 30 especifica 20 passos por rotação, 4 micro-passos por passo completo, um momento de carga de inércia 50 vezes que o motor de rotor em si, e um binário de carga de 30 por cento do torque do motor porão cheio.

Você deve escrever código para impelir o controle pins apropriadamente.

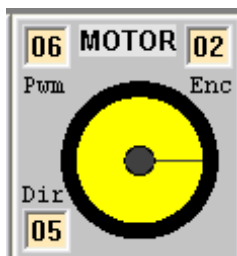


O motor é modelado com precisão ambos mecanicamente e electricamente. tensão do motor impulsor-gotas e variando relutância e indutância são modelados ao longo com um momento de inércia realistas no que diz respeito à realização de binário. O rotor do motor de enrolamento tem uma resistência modelado de $R = 6$ ohms, e uma indutância de $L = 6$ mili-Henries que cria uma constante de tempo eléctrica de 1,0 milissegundo.

este dispositivo inclui uma actividade 'STEP' amarelo DEL, e um DEL 'sync' (verde para sincronizado, ou vermelho quando desligado por um ou mais passos).

Motor DC ('MOTOR')

Este 'I/O' dispositivo emula um motor de corrente contínua de 100 volts com alimentação de 100 volts com um controlador impulsor impulsionado integrado por um sinal de modulação por largura de pulso **Pwm** entrada), e um sinal de controle de direcção **Dir** entrada). O motor também tem uma saída de codificador de roda que impele sua **Enc** saída pin. Você pode usar 'analogWrite()' para impelir o **Pwm** pin com um PWM forma de onda de 490 Hz (em pins 3,9,10,11) ou 980 Hz (em pins 5,6) do ciclo de trabalho entre 0,0 e 1,0 ('analogWrite()' valores de 0 a 255). Alternativamente, o 'MOTOR' *também responde* para o seu próprio 'digitalWrite()' " bit-banged "impelir.



O motor é modelado com precisão tanto mecanicamente quanto eletricamente. A contabilização das quedas de tensão do transistor motor impulsor e do binário real da transmissão sem carga dá uma velocidade total de aproximadamente 2 rotações por segundo e um binário de pouco mais de 5 kg-cm (ocorrendo num ciclo de trabalho PWM constante de 1,0), com um momento total de inércia do motor mais carga de 2,5 kg-cm. O enrolamento do rotor do motor tem uma resistência modelada de $R = 2$ ohms e uma indutância de $L = 300$ micro-Henries que cria uma constante de tempo eléctrica de 150 microssegundos. Devido à modelagem realista, você notará que pulsos PWM muito estreitos *não pegue* o motor a girar - ambos devido ao tempo de subida de corrente finito e ao significativo tempo de inatividade

após cada pulso estreito. Estes combinam para causar o momento insuficiente do rotor para superar o chicote de mola semelhante à mola da caixa de engrenagens sob a fricção estática. A consequência é quando se usa 'analogWrite()', um ciclo de trabalho abaixo de aproximadamente 0,125 não fará com que o motor se mova - isso está de acordo com o que é observado quando se está dirigindo um motor de engrenagens real de um 'Uno' com, é claro, um apropriado (**e obrigatório**) Motor impulsor módulo entre o motor eo 'Uno'!

O codificador do motor emulado é um sensor de interrupção ótica montado no eixo que produz um ciclo de trabalho de 50% forma de onda com 8 períodos de alto-baixo completos por revolução de roda (assim seu programa pode detectar mudanças de rotação de roda em uma resolução de 22,5 graus).

Servo Motor ('SERVO')

Este 'I/O' dispositivo emula um servo motor CC de 6 volts de alimentação de 6 volts PWM-impulsionado. Parâmetros de modelagem mecânicos e elétricos para a operação servo serão muito semelhantes aos de um servo padrão HS-422. O servo tem uma velocidade máxima de rotação de aproximadamente 60 graus em 180 milissegundos. Se o inferior esquerdo caixa de verificação está marcada, o servo torna-se um **rotação contínua** servo com a mesma velocidade máxima, mas agora a largura de pulso PWM define **Rapidez** em vez do ângulo



O seu programa deve ter um `'#include <Servo.h>'` linha antes de declarar o seu `'Servo'` instância (s) se você optar por usar a funcionalidade da biblioteca `'Servo.h'`, por exemplo `'Servo.write()'`, `'Servo.writeMicroseconds()'`. Como alternativa, o `'SERVO'` também responde `'digitalWrite()'` Sinais "bit-banged". Devido à implementação interna de UnoArduSim, você está limitado a 6 `'SERVO'` dispositivos.

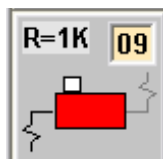
Orador Piezo ('PIEZO')



Este dispositivo permite que você "escute" sinais em qualquer 'Uno' pin escolhido, e pode ser um complemento útil para LEDs para depurar sua operação do programa. Você também pode se divertir tocando timbres apropriadamente `'tone()'` e `'delay()'` chamadas (embora não haja filtragem do forma de onda retangular, assim você não ouvirá notas "puras").

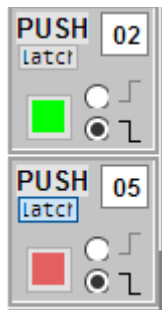
Você também pode ouvir um 'PULSER' ou 'FUNCGEN' dispositivo conectando conectando um 'PIEZO' ao pin que dispositivo impele.

Resistor de slide ('R=1K')



Este dispositivo permite ao usuário conectar a um 'Uno' pin um resistor de 1 k-Ohm para + 5V, ou um resistor de 1 k-Ohm para aterramento. Isso permite simular cargas elétricas adicionadas a um hardware real dispositivo. Ao clicar com o botão esquerdo no interruptor deslizante **corpo** você pode alternar sua seleção desejada de pull-up ou pull-down. Usar um ou vários desses dispositivos permitiria que você definisse um "código" único (ou multi) para o seu programa ler e responder.

Botão de Pressão ('PUSH')



Este 'I/O' dispositivo emula um disco normalmente aberto **momentânea OU trava** botão de pressão single-pole, single-throw (SPST) com um resistor pull-up de 10 k-ohm (ou pull-down). Se uma seleção de transição de borda ascendente é escolhida para o dispositivo, os contatos de botão de pressão serão conectados entre o dispositivo pin e + 5V, com um pull-down de 10 k-Ohm para o terra. Se uma transição de borda descendente for escolhida para o dispositivo, os contatos de botão de pressão serão conectados entre o dispositivo pin e o terra, com um pull-up de 10 k-Ohm até + 5V.

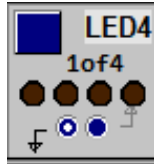
Ao clicar com o botão esquerdo do mouse ou pressionar qualquer tecla, você fecha o contato do botão de pressão. Em **momentâneo** modo, permanece fechado enquanto você segurar o botão do mouse ou a tecla, e em **robusto** modo (habilitado clicando no botão 'latch' botão) permanece fechado (e uma cor diferente) até que você pressione o botão novamente. O retorno de contato (por 1 milissegundo) será produzido toda vez que você use o **barra de espaço** para pressionar o botão.

DEL colorido ('LED')



Você pode conectar um DEL entre o 'Uno' pin escolhido (através de um resistor limitador de corrente construídas-em da série oculta de 1 k-Ohm) para o terra ou para + 5V - isso lhe dá a opção de acender o DEL quando o 'Uno' pin estiver conectado 'HIGH' ou, em vez disso quando é 'LOW'. A cor DEL pode ser escolhida para ser vermelha ('R'), amarela ('Y'), verde ('G') ou azul ('B') usando sua caixa de edição.

Linha 4-DEL ('LED4')



Você pode conectar esta linha de 4 LEDs coloridos entre o conjunto escolhido de 'Uno' pins (cada um com um resistor limitador de corrente de 1 k-Ohm da série oculta construídas-em) para o terra ou para + 5V - isso lhe dá a opção de ter os LEDs acesos quando o 'Uno' pin conectado estiver 'HIGH' ou, em vez disso quando é 'LOW'.

o '1of4' A caixa de edição pin aceita um único número pin que será considerado como significando **o primeiro de quatro consecutivos** 'Uno' pins que se conectará aos 4 LEDs.

A cor DEL ('R', 'Y', 'G' ou 'B') é uma **opção escondida** Isso pode ser **apenas ser escolhido por editando o IODevices.txt arquivo** (qual você pode criar usando **Salvar de Configurar | 'I/O' Dispositivos** caixa de diálogo).

7-Segmento DEL Dígito ('7SEG')



Você pode conectar este monitor Dígito DEL 7-Segment a um conjunto escolhido de **quatro consecutivos 'Uno' pins que dão o código hexadecimal** para o dígito exibido, ('0' a 'F'), e ligue ou desligue este dígito usando o CS * pin (ativo-LOW para ON).

Este dispositivo inclui um decodificador construídas-em que usa o **ativo-ALTO** níveis nos quatro anos consecutivos '1of4' pins para determinar o hexadecimal dígito solicitado a ser exibido. Te nível no menor número pin (aquele exibido no '1of4' caixa de edição) representa o bit menos significativo do código hexadecimal de 4 bits.

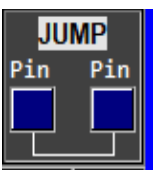
A cor dos segmentos DEL ('R', 'Y', 'G' ou 'B') é um **opção escondida** Isso pode ser **apenas ser escolhido por editando o IODevices.txt arquivo** você pode criar usando **Salvar de Configurar | 'I/O' Dispositivos** caixa de diálogo.

Deslizante Analógico

Um potenciômetro de 0-5V controlado por slider pode ser conectado a qualquer 'Uno' pin escolhido para produzir um nível de tensão analógico estático (ou que muda lentamente) que seria lido por 'analogRead()' como um valor de 0 a 1023. Use o mouse para arrastar ou clicar para pular o controle deslizante analógico.



Pin Jumper ('JUMP')



Você pode conectar dois 'Uno' pins juntos usando este dispositivo (se qualquer conflito elétrico for detectado quando você preencher o segundo número pin, a conexão escolhida não é permitida e o pin é desconectado).

Este jumper dispositivo tem uma utilidade limitada e é mais útil quando combinado com interrupções, para testes, experimentos e testes programa. fins de aprendizagem. **A partir do UnoArduSim V2.4, você pode achar que usar um 'PROGIO' dispositivo oferece mais flexibilidade que os métodos de interrupção impulsionado abaixo.**

Três usos possíveis para este dispositivo são os seguintes:

- 1) Você pode **crie uma entrada digital para testar o seu programa** que tem cronometragens mais complexo do

que pode ser produzido usando por qualquer um dos conjunto de conjunto fornecido padrão 'I/O' dispositivos, como segue:

Definir uma interrupção módulo funcional (vamos chamá-lo '**myIntr**') e Faz '**attachInterrupt(0, myIntr, RISING)**' dentro do seu '**setup()**' Conecte um **Pulsador** dispositivo para pin2 - agora '**myIntr()**' vai executar cada vez que um **Pulsador** borda ascendente ocorre. Seu '**myIntr()**' módulo funcional pode ser um algoritmo que você tem programado (usando o contador global variáveis, e talvez até '**random()**') para produzir um forma de onda de seu próprio projeto em qualquer disponível '**OUTPUT**' pin (digamos que seja pin 9). Agora **SALTAR** pin 9 para o seu desejado 'Uno' 'INPUT' pin para aplicar aquele digital forma de onda gerado àquela entrada pin (a fim testar sua resposta de programa a esse forma de onda particular). . Você pode gerar uma seqüência de pulsos, ou caracteres seriais, ou simplesmente transições de borda, toda a complexidade arbitrária e intervalos variados. Por favor note que se suas chamadas principais do programa '**micros()**' (ou liga para qualquer módulo funcional que se baseia nele), '**return**' valor **será aumentado** pelo tempo gasto dentro do seu '**myIntr()**' módulo funcional toda vez que a interrupção é acionada. Você pode produzir uma rápida explosão de bordas com tempo exato usando chamadas para '**delayMicroseconds()**' de dentro '**myIntr()**' (talvez para gerar um todo byte de um transferência de alta taxa de baud), ou simplesmente gerar uma transição por interrupção (talvez para gerar um pouco transferência de baixa taxa de baud) com o **Pulsador** dispositivo '**Period**' escolhido adequado às suas necessidades do cronometragens (lembre-se que **Pulsador** limita seu mínimo '**Period**' para 50 microssegundos).

2) Você pode **experimentar com loopbacks do subsistema**:

Por exemplo, desconecte seu 'SERIAL' 'I/O' dispositivo TX '00' pin (editar em branco), e depois **SALTAR** 'Uno' pin '01' de volta para 'Uno' pin '00' para emular um loopback de hardware do ATmega '**Serial**' subsistema. Agora no seu teste programa, dentro '**setup()**' fazer um **solteiro** '**Serial.print()**' de um palavra ou personagem, e dentro do seu '**loop()**' eco de volta todos os caracteres recebidos (quando '**Serial.available()**') fazendo um '**Serial.read()**' seguido por um '**Serial.write()**' e depois observe o que acontece. Você poderia observar que um semelhante '**SoftwareSerial**' loopback **vai falhar** (como seria na vida real - o software não pode fazer duas coisas ao mesmo tempo).

Você também pode experimentar **SPI** loop-back usando um **SALTAR** para conectar o pin 11 (MOSI) de volta ao pin 12 (MISO).





3) Você pode **contar o número e / ou medir o espaçamento de transições de nível específico em qualquer 'Uno' saída pin X** que ocorrem como resultado de um complexo Instrução Arduino ou biblioteca módulo funcional (como exemplos: '**analogWrite()**' ou '**OneWire::reset()**', ou '**Servo::write()**'), do seguinte modo:

SALTAR pin X interromper pin 2 e dentro do seu '**myIntr()**' use um '**digitalRead()**' e um '**micros()**' ligar, e compare com níveis e tempos salvos (de interrupções anteriores). Você pode alterar a sensibilidade da borda para a próxima interrupção, se necessário, usando '**detachInterrupt()**' e '**attachInterrupt()**' de **dentro** seu '**myIntr()**'. Note que você não será capaz de rastrear pin transições que ocorrem muito próximas umas das outras (mais do que o tempo total de execução seu '**myIntr()**' módulo funcional), como aqueles que acontecem com transferências I2C ou SPI, ou com alta taxa de baud '**Serial**' transferências (mesmo que sua interrupção módulo funcional não iria perturbar o cronometragens inter-edge dessas transferências produzidas por hardware). Observe também que as transferências mediadas por software (como '**OneWire::write()**' e '**SoftwareSerial::write()**') está deliberadamente protegido de interrupções (pelo código da biblioteca desativando temporariamente todas as interrupções, a fim de evitar interrupções do cronometragens), assim você não pode medir dentro daqueles que usam este método.






Embora você possa fazer essas mesmas medições de espaçamento de borda **visualmente** em um **Formas de onda Digital** janela, se você estiver interessado no espaçamento mínimo ou máximo em um grande número de transições, ou na contagem de transições, fazê-lo usando este '**myIntr()**' -mais- **SALTAR** técnica é mais conveniente. E você pode medir, por exemplo, variações no espaçamento das transições produzidas pelo main-programa (devido ao efeito do seu software ter percursos variados de execução de diferentes execução vezes), fazer um tipo de programa "perfil".

Menus









Arquivo:

<u>Carregar INO ou PDE Prog (ctrl-L)</u> 	Permite ao usuário escolher um programa arquivo com a extensão selecionada. O programa recebe imediatamente um Analisar
<u>Editar/Examinar (ctrl-E)</u>	Abre o programa carregado para visualização / edição.
<u>Salvar</u> 	Salvar o conteúdo programa editado de volta para o original programa arquivo.
<u>Salvar Como</u>	Salvar o conteúdo programa editado com um nome arquivo diferente.
<u>Próximo ('#include')</u> 	Avança o Painel de Códigos para exibir o próximo ' #include ' arquivo
<u>Anterior</u> 	Retorna o Painel de Códigos exibição para o anterior arquivo
<u>Saída</u>	Sai do UnoArduSim após lembrar o usuário de salvar qualquer arquivo (s) modificado.

Buscar:

<u>Escalar pilha de chamadas</u> 	Pule para a função de chamada anterior na pilha de chamadas - o Painel de Variáveis será ajustado para essa função
<u>Descer pilha de chamadas</u> 	Pule para a próxima função chamada na pilha de chamadas - o Painel de Variáveis será ajustará a essa função
<u>Definir texto Buscar (ctrl-f)</u> 	Ativar o Barra de ferramentas Buscar edit-box para definir o texto a ser procurado em seguida (e adiciona a primeira palavra da linha realçada no Painel de Códigos ou Painel de Variáveis se um deles tem o foco).
<u>Buscar Próximo texto</u> 	Salta para a próxima ocorrência de Texto no Painel de Códigos (se tiver o foco ativo), ou para a próxima ocorrência de Texto no Painel de Variáveis (se em vez disso, ele tiver o foco ativo).
<u>Buscar Texto anterior</u> 	Salta para a ocorrência de Texto anterior no Painel de Códigos (se tiver o foco ativo) ou a ocorrência de Texto anterior no Painel de Variáveis (se em vez disso, ele tiver o foco ativo).

Executar:

<u>Passo Dentro (F4)</u>		Passos execução para a frente por uma instrução, ou <i>em um chamado módulo funcional</i> .
<u>Passo Acima (F5)</u>		Passos execução para a frente por uma instrução, ou <i>por uma chamada módulo funcional completa</i> .
<u>Passo Fora (F6)</u>		Avança execução por <i>apenas o suficiente para deixar o módulo funcional atual</i> .
<u>Executar Para (F7)</u>		Executa o programa, <i>parando na linha programa desejada</i> - você deve primeiro clicar no realçar uma linha desejada do programa antes de usar o Executar Para.
<u>Executar Até (F8)</u>		Executa o programa até que ocorra uma gravação no variável que tinha o realçar atual no Painel de Variáveis (clique em um para estabelecer o realçar inicial).
<u>Executar (F9)</u>		Executa o programa.
<u>Parar (F10)</u>		Determina programa execução (<i>e congela o tempo</i>).
<u>Reinicializar</u>		Redefine o programa (todos os valores variáveis são redefinidos para o valor 0 e todos os variáveis do ponteiro são redefinidos para 0x0000).
<u>Animar</u>		Passa automaticamente pelas linhas programa consecutivas <i>com atraso artificial adicionado</i> e realce da linha de código atual. Operação em tempo real e sons são perdidos.
<u>Câmera Lenta</u>		Diminui o tempo por um fator de 10.

Opções:

<u>Passo Acima Structors/ Operadores</u>	Voe direto através de construtores, destruidores e sobrecarga de operador módulos funcionais durante qualquer passo (ou seja, ele não irá parar dentro destes módulos funcionais).
<u>Alocação de Registros</u>	Atribua os locais módulo funcional aos registros livre ATmega em vez de à pilha (gera um pouco de uso de RAM).
<u>Erro no Uninitialized</u>	Sinalize como um erro Analisar em qualquer lugar em que seu programa tente usar um variável sem ter inicializado seu valor pela primeira vez (ou pelo menos um valor dentro de um matriz).
<u>Adicionado 'loop()' ' Demora</u>	Adiciona 1000 microssegundos de atraso a cada vez 'loop()' é chamado (caso não haja outras chamadas programa para 'delay()' em qualquer lugar) - útil para tentar evitar ficar muito atrás do tempo real.
<u>Permitir interrupções aninhadas</u>	Permitir reativar com 'interrupts()' de dentro de uma rotina de serviço de interrupção do usuário.

Configurar:

<u>'I/O' Dispositivos</u>	Abre uma caixa de diálogo para permitir que o usuário escolha o (s) tipo (s) e números do 'I/O' dispositivos desejado. Nessa caixa de diálogo, você também pode Salvar 'I/O' dispositivos para um texto arquivo e / ou Carregar 'I/O' dispositivos de um texto salvo anteriormente (ou editado) arquivo (incluindo todas as conexões pin e configurações clicáveis e valores digitados
<u>Preferências</u>	Abre uma caixa de diálogo para permitir que o usuário defina preferências, incluindo recuo automático de linhas programa, permitindo a sintaxe Expert, escolha da fonte tipo de letra, optando por um tamanho de fonte maior, reforçando os limites do matriz, permitindo palavras-chave lógicas do operador, mostrando programa descarregando , escolha da versão 'Uno' placa de circuito e comprimento do buffer TWI (para I2C dispositivos).

VarAtualizar:

<u>Permitir Auto (-) Contrair</u>	Permita que o UnoArduSim para contrair exiba expandidos matrizes / objetos quando ficar atrás do tempo real.
<u>Mínimo</u>	Apenas atualize o Painel de Variáveis exibir 4 vezes por segundo.
<u>Realçar Alterar</u>	O Realçar alterou os valores de variável durante a execução (pode causar lentidão).

Janelas:

<u>'Serial' Monitor</u>	Conecte uma E / S serial dispositivo a pins 0 e 1 (se nenhuma) e puxe uma 'Serial' monitore o texto TX / RX janela.
<u>Restaurar tudo</u>	Restaurar todos janelas filho minimizado.
<u>Formas de Onda Digitais</u>	Restaure um janela Formas de Onda Digitais minimizado.
<u>Fomra de Onda Analógica</u>	Restaure um janela Fomra de Onda Analógica minimizado.

Socorro:

<u>Socorro rápido Arquivo</u>	Abre o PDF arquivo do UnoArduSim_QuickHelp.
<u>Socorro Completo Arquivo</u>	Abre o PDF arquivo do UnoArduSim_FullHelp.
<u>Erro Correções</u>	Veja correções significativas do erro desde o lançamento anterior.
<u>Mudança / Melhorias</u>	Veja alterações e melhorias significativas desde o lançamento anterior.
<u>Sobre</u>	Exibe versão, direitos autorais.

'Uno' Placa de circuito e 'I/O' Dispositivos

O 'Uno' e o 'I/O' dispositivos são todos modelados com precisão, e você poderá ter uma boa idéia de como o seu programas se comportará com o hardware, e todos os pin elétricos serão sinalizados.

Cronometragens

UnoArduSim executa rapidamente o suficiente em um PC ou tablet que pode *na maioria dos casos*) modelar programa ações em tempo real, **mas somente se o seu programa incorpora** pelo menos alguns pequenos '`delay()`' chamadas ou outras chamadas que naturalmente irão mantê-lo sincronizado com o tempo real (veja abaixo).

Para conseguir isso, o UnoArduSim faz uso de um temporizador de retorno de chamada Janelas módulo funcional, que permite manter um controle preciso do tempo real. O execução de um número de instruções programa é simulado durante uma fatia do temporizador, e instruções que requerem mais execução (como chamadas para '`delay()`') pode precisar usar várias fatias de timer. Cada iteração do cronômetro de retorno de chamada módulo funcional corrige a hora do sistema usando o relógio de hardware do sistema para que o programa execução seja constantemente ajustado para manter o controle em tempo real. *A única taxa de vezes execução devo ficar para trás em tempo real* é quando o usuário criou loops apertados **sem atraso adicional** ou 'I/O' dispositivos são configurados para operação com frequências 'I/O' dispositivo muito altas (e / ou taxa de baud) que gerariam um número excessivo de eventos de mudança de nível pin e sobrecarga de processamento associada. O UnoArduSim lida com essa sobrecarga saltando alguns intervalos do temporizador para compensar, e isso então diminui a progressão do programa para **abaixo em tempo real** .

Além disso, programas com matrizes grande sendo exibido, ou novamente com loops apertados **sem atraso adicional** pode causar uma alta frequência de chamada módulo funcional e gerar uma alta **Painel de Variáveis** exibir atualização de carga fazendo com que ela fique para trás em tempo real - UnoArduSim reduz automaticamente a frequência de atualização variável para tentar acompanhar, mas quando é necessária uma redução ainda maior, escolha **Mínimo** de **VarAtualizar** cardápio para especificar apenas quatro atualizações por segundo.

Modelando com precisão o tempo execução de sub-milissegundos para cada instrução ou operação programa **não está feito** - apenas estimativas muito aproximadas para a maioria foram adotadas para fins de simulação. No entanto, o cronometragens de '`delay()`' e '`delayMicroseconds()`' módulos funcionais e módulos funcionais '`millis()`' e '`micros()`' são perfeitamente precisos e **contanto que você use pelo menos um dos atrasos módulos funcionais** em um loop em algum lugar no seu programa, **ou** você usa um módulo funcional que se liga naturalmente à operação em tempo real (como '`print()`' que está ligado ao taxa de baud escolhido), então o desempenho simulado do seu programa será muito próximo do tempo real (novamente, exceto eventos de mudança de nível pin de alta frequência flagrantemente excessivos ou atualizações Variáveis permitidas pelo usuário excessivas que poderiam atrasá-lo).

Para ver o efeito das instruções individuais do programa *galinha correndo* , pode ser desejável ser capaz de retardar as coisas. Um fator de redução de tempo de 10 pode ser definido pelo usuário no menu **Executar** .

'I/O' Dispositivo Cronometragens

Estes dispositivos virtuais recebem sinalização em tempo real das mudanças que ocorrem em sua entrada pins, e produzem saídas correspondentes em sua saída pins, que podem então ser detectadas pelo 'Uno' - portanto, são inerentemente sincronizadas com programa execução. Interno 'I/O' dispositivo O cronometragens é definido pelo usuário (por exemplo, através da seleção taxa de baud ou da frequência do relógio), e os eventos do simulador são configurados para rastrear a operação interna em tempo real.

Sons

Cada 'PIEZO' dispositivo produz som correspondente às mudanças de nível elétrico que ocorrem no pin anexado, independentemente da origem de tais mudanças. Para manter os sons sincronizados com o programa execução, o UnoArduSim inicia e pára a reprodução de um buffer de som associado quando o execução é iniciado / interrompido.

A partir da V2.0, o som foi modificado para usar a API de áudio do Qt - infelizmente sua QAudioOutput '`class`' não suporta o loop de buffer de som para evitar a falta de amostras de som (como pode acontecer durante maiores

atrasos operacionais do sistema operacional do Windows). Portanto, para evitar a grande maioria dos cliques incômodos e a quebra de som durante os atrasos do sistema operacional, o som é silenciado de acordo com a seguinte regra:

O som é silenciado enquanto o UnoArduSim não é a janela "ativo" (exceto quando um novo filho janela acaba de ser criado e ativado), **e até mesmo** quando UnoArduSim é a janela principal "ativo", mas o ponteiro do mouse é **lado de fora** da sua área principal do cliente janela.

Note que isto implica que o som será temporariamente silenciado como se o ponteiro do mouse está pairando **sobre uma criança janela**, e vai ficar mudo **E se essa criança janela é clicada para ativá-lo** (até que o UnoArduSim janela principal seja clicado novamente para reativá-lo) .

O som pode sempre ser desativado clicando em qualquer lugar dentro da área do cliente do KO193 principal do UnoArduSim.

Devido ao buffer, o sound tem um atraso em tempo real de até 250 milissegundos a partir da hora do evento correspondente no pin do 'PIEZO' anexado.

Limitações e Elementos não Suportados

Incluído Arquivos

Um '< >' - entre colchetes '#include' do '<Servo.h>', '<Wire.h>', '<OneWire.h>', '<SoftwareSerial.h>', '<SPI.h>', '<EEPROM.h>' e '<SD.h>' é suportado, mas estes são apenas emulados - o arquivos real não é procurado; em vez disso, sua funcionalidade é diretamente "incorporada" ao UnoArduSim e é válida para a versão do Arduino suportada fixa.

Qualquer citado '#include' (por exemplo de "supp.ino", "myutil.cpp" ou "mylib.h") é suportado, mas todos esses arquivos devem **residir no mesmo diretório como o pai programa arquivo** naquela contém a sua '#include' (não há busca feita em outros diretórios). o '#include' recurso pode ser útil para minimizar a quantidade de código programa mostrado no **Painel de Códigos** A qualquer momento. Cabeçalho arquivos com '#include' (ou seja, aqueles que têm um ".h" extensão) fará com que o simulador tente incluir o arquivo com o mesmo nome ".cpp" extensão (se também existir no diretório do pai programa).

Alocações Dinâmicas de Memória e RAM

Operadores 'new' e 'delete' são suportados, assim como o Arduino nativo 'String' objetos, **mas não chamadas diretas para** 'malloc()', 'realloc()' e 'free()' que estes dependem.

O uso excessivo de RAM para declarações variável é sinalizado em Analisar e o estouro de memória RAM é sinalizado durante programa execução. A item no menu **Opções** permite emular a alocação normal de registro de ATmega como seria feito pelo AVR compilador, ou modelar um esquema de compilação alternativo que use apenas a pilha (como uma opção de segurança caso um erro apareça em minha modelagem de alocação de registro). Se você usasse um ponteiro para examinar o conteúdo da pilha, ele deveria refletir com precisão o que apareceria em uma implementação de hardware real.

Alocações de Memória 'Flash'

Memória 'Flash' 'byte', 'int' e 'float' variáveis / matrizes e seus correspondentes módulos funcionais de acesso de leitura são suportados. Qualquer 'F()' Chamada módulo funcional ('Flash' -macro) de qualquer string literal é suportados, mas o único KL145 de acesso direto com string 'Flash' 'strcpy_P()' e 'memcpy_P()', então para usar outros módulos funcionais você precisará primeiro copiar a string 'Flash' para uma RAM normal 'String' variável, e depois trabalhar com essa RAM 'String'. Quando você usa o 'PROGMEM' Palavra-chave modificador variável, deve aparecer **em frente de** o nome variável, e esse variável **também deve ser declarado** Como 'const'.

'String' Variáveis

O nativo 'String' A biblioteca é quase totalmente suportada com algumas poucas (e pequenas) exceções.

o 'String' os operadores suportados são +, + =, <, <=, >, > =, ==, !=, e []. Observe que: 'concat()' Leva um **solteiro** argumento que é o 'String', ou 'char' ou 'int' para ser anexado ao original 'String' objeto, **não** dois argumentos, como é erroneamente declarado nas páginas web do Arduino Reference).

Bibliotecas Arduino

Somente 'Servo.h', 'SoftwareSerial.h', 'SPI.h', 'Wire.h', 'OneWire.h', 'Stepper.h', 'SD.h', 'TFT.h' e 'EEPROM.h' para o **V1.8.8 Arduino** liberar atualmente são suportados no UnoArduSim. V2.6 introduz um mecanismo para trêsrd festa de apoio à biblioteca via arquivos fornecido no 'include_3rdParty' pasta que pode ser encontrado no interior do UnoArduSim diretório de instalação. Tentando '#include' a ".cpp" e ".h" arquivos de outro, que ainda não suportado bibliotecas **não funciona** como eles irão conter instruções de montagem de baixo nível e directivas não suportados e não reconhecida arquivos!

Ponteiros

Ponteiros para tipos simples, matrizes ou objetos são todos suportados. Um ponteiro pode ser igualado a um matriz do mesmo tipo (por 'iptr = intarray'), mas depois haveria *nenhuma verificação subsequente dos limites de matrizes* em uma expressão como 'iptr[index]'.

Módulos funcionais pode retornar ponteiros ou 'const' ponteiros, mas qualquer nível subsequente de 'const' no ponteiro retornado é ignorado.

Há sim **sem suporte** para chamadas módulo funcional sendo feitas através de **módulo funcional-ponteiros declarados pelo usuário**.

'class' e 'struct' Objetos

Embora o polimorfismo e a herança (em qualquer profundidade) sejam suportados, 'class' ou 'struct' só pode ser definido para ter no máximo **1** base 'class' (ie **múltiplo**- herança não é suportada). Chamadas de inicialização do construtor Base-'class' (via notação de dois pontos) nas linhas de declaração do construtor são suportadas, mas **não** inicializações de membros usando a mesma notação de dois pontos. Isso significa que o objetos que contém 'const' não-static variáveis, ou tipo de referência variáveis, não são suportados (são possíveis somente com inicializações de membros de tempo de construção especificadas)

As sobrecargas do operador de atribuição de cópia são suportadas juntamente com os construtores de movimento e atribuições de movimentação, mas a conversão objeto definida pelo usuário ("tipo-conversão") módulos funcionais não é suportada.

Escopo

Não há suporte para o 'using' palavra-chave ou para 'namespace', ou para 'file' escopo Todas as declarações não locais são, por implementação, consideradas globais.

Qualquer 'typedef', 'struct' ou 'class' definição (isto é, que pode ser usado para futuras declarações), deve ser feito **global** escopo (**local** definições de tais itens dentro de um módulo funcional não são suportadas).

Qualificadores 'unsigned', 'const', 'volatile', 'static'

o 'unsigned' prefixo funciona em todos os contextos legais normais. o 'const' palavra-chave, quando usada, deve **preceder** o nome variável ou o nome módulo funcional ou 'typedef' nome que está sendo declarado - colocando-o após o nome causará um erro Analisar. Para Declarações módulo funcional, somente módulos funcionais com retorno de ponteiro pode ter 'const' aparecem na sua declaração.

Todos UnoArduSim variáveis são 'volatile' por implementação, então o 'volatile' A palavra-chave é simplesmente ignorada em todas as declarações variável. Módulos funcionais não podem ser declarados

'**volatile**' , nem são argumentos de chamada módulo funcional.

o '**static**' palavra-chave é permitida para variáveis normal e para membros objeto e membro-módulos funcionais, mas é explicitamente desaprovada para as próprias instâncias objeto ('**class**' / '**struct**'), para módulos funcionais não membro, e para todos os argumentos módulo funcional.

Diretrizes Compilador

'**#include**' e regular '**#define**' ambos são suportados, mas **não macro** '**#define**'. O '**#pragma**' directivas directivas e de inclusão condicional ('**#ifdef**' , '**#ifndef**' , '**#if**' , '**#endif**' , '**#else**' e '**#elif**') são também **não suportado**. O '**#line**' , '**#error**' e macros predefinidas (como '**_LINE_**' , '**_FILE_**' , '**_DATE_**' e '**_TIME_**') são também **não suportado**.

Elementos de Linguagem Arduino

Todos os elementos da linguagem nativa do Arduino são suportados com exceção do dúbio '**goto**' instrução (o único uso razoável para o qual eu posso pensar seria como um salto (para um loop infinito de salvamento e desligamento seguro) no caso de uma condição de erro com a qual o seu programa não possa lidar de outra forma)

C / C ++ - Elementos de Linguagem

"Qualificadores de campo de bits" que economizam bits para membros em definições de estrutura são **não suportado**.

'**union**' é **não suportado**.

O excêntrico "operador de vírgula" é **não suportado** (assim você não pode executar várias expressões separadas por vírgulas quando apenas uma única expressão é normalmente esperada, por '**while()**' e '**for(; ;)**' constructos).

Modelos Módulo funcional

módulos funcionais definido pelo usuário que usa a palavra-chave "modelo" para permitir que ele aceite argumentos do tipo "genérico" são **não suportado**.

Emulação em Tempo Real

Como mencionado acima, execução vezes das muitas instruções possíveis individuais do Arduino programa são **não** modelado com precisão, de modo que, a fim de executar em uma taxa em tempo real o seu programa vai precisar de algum tipo de dominador '**delay()**' instrução (pelo menos uma vez por '**loop()**'), ou uma instrução que é naturalmente sincronizada com alterações em nível pin em tempo real (como '**pulseIn()**' , '**shiftIn()**' , '**Serial.read()**' , '**Serial.print()**' , '**Serial.flush()**' etc).

Vejo **Cronometragens** e **Sons** acima para mais detalhes sobre limitações.

Notas de Lançamento

Erro Correções

V2.7.0- Março 2020

- 1) Quando o (Janelas-default) Tema OS luz foi adotado, o **Painel de Códigos** não foi mostrando a cor de realce introduzido em V2.6 (em vez disso, apenas uma realçar cinzenta resultante a partir de um sistema de substituição).
- 2) Versão 2.6 inadvertidamente quebrou auto-indent-guia Formatação na primeira `'switch()'` construir.
- 3) O novo recurso de navegação pilha de chamadas introduzido na Versão 2.6 mostrou **valores incorretos** para locais variáveis quando não estiver dentro da módulo funcional atualmente em execução, e não com membros aninhada chamadas módulo funcional.
- 4) `'TFT :: text ()'` estava funcionando, mas as funções `'TFT :: print ()'` não estavam (elas simplesmente foram bloqueadas para sempre). Além disso, `'TFT :: loadImage ()'` falhou se `'Serial.begin ()'` tivesse sido feito anteriormente (que é o caso normal e agora é necessário).
- 5) Versão 2.6 introduziu um erro que exibia o valor incorreto para 'RX' presente e passado bytes para 'I2CSLV', 'SPISLV', 'TFT', 'LCDI2C' and 'LCDSPI' dispositivos (e seu monitor janelas).
- 6) Uma alteração feita em V2.4 causada Executar | Animar destacando a saltar sobre muitos de código de linhas executado.
- 7) Desde V2.4, de-afirmando 'SS*' ou 'CS*' em um 'I/O' dispositivo na instrução imediatamente após uma `'SPI.transfer()'` causaria que dispositivo a falhar para receber o byte de dados transferido. Além disso, byte recepção `'SPI_MODE1'` e `'SPI_MODE3'` não foi marcada até ao início do próximo byte enviado pelo mestre (e o byte foi completamente perdida se o dispositivo 'CS*' foi de-seleccionado antes de seguida).
- 8) No novo `'SPI_SLV'` modo permitido desde V2.4, `'bval = SPI.transfer()'` só voltou o valor correto para `'bval'` se a transferência byte já estava completa e esperando quando `'transfer()'` foi chamado.
- 9) A caixa de 'DATA' edit on 'SPISLV' dispositivos agora recebe o padrão 0xFF quando não há mais bytes para responder com.
- 10) A sincronização estado DEL estava incorrecta para 'PSTEPR' dispositivos ter mais do que uma micro-passo por passo completo.
- 11) conflitos eléctrico causado por 'I/O' dispositivos reagir para transições no relógio 'SPI', um sinal de 'PWM', ou um `'tone'` sinal, não foram relatados, e pode levar a inexplicável (corrompido) recepção de dados.
- 12) Quando o intervalo entre as interrupções era muito pequena (menos de 250 microsegundos), um (defeituoso) mudança na V2.4 alterou a cronometragens de construídas-em módulos funcionais que utilize um sistema de temporizadores, ou loops de instrução, para gerar atrasos (exemplos de cada um são `'delay()'` e `'delayMicroseconds()'`). Uma mudança subsequente na V2.5 causada desalinhamento de `'shiftOut()'` sinais de clock de dados e quando uma interrupção aconteceu entre bits.
- 13) Aceitar um texto auto-completar construídas-em módulo funcional via a tecla Enter não conseguiu retirar tipos de parâmetros a partir do texto da chamada módulo funcional inserido.
- 14) Carregando uma nova (user-interrupção-impulsionado) programa quando um programa executado anteriormente ainda tinha uma interrupção pendente poderia causar um acidente durante a transferência (devido a defeito tentou execução da nova rotina de interrupção).
- 15) Objeto-membros auto-completação (acessível através 'ALT' direito de seta) para objetos dentro `'#include'` arquivos estão agora acessíveis logo que a sua `'#include'` arquivo é sucesso analisado.
- 16) A declaração módulo funcional ter um espaço inadvertida dentro de um nome de parâmetro módulo funcional causou uma mensagem de erro Analisar claro.
- 17) Quando execução parou em um módulo diferente da principal programa, o KK-0 | ação anterior não conseguiu se tornar habilitado.

- 18) Citado única parênteses ('{ ' e ' } ') Ainda foram contadas (incorretamente) como suportes escopo na Analisar, e também confundiu formatação auto-tab-indent ..
- 19) 'OneWire::readBytes(byte* buf, int count)' tinha sido deixando de atualizar imediatamente o exibido 'buf' conteúdo no Painel de Variáveis.
- 20) Octal-trava 'OWISLV' dispositivos mostrou que a produção pin níveis que defasados por uma gravação trava-registo.

V2.6.0- Janeiro 2020

- 1) A erro introduzido em V2.3 levou a um acidente quando o agregado **Fechar** botão foi usado no **Buscar / Substituir** diálogo (ao invés de sua Saída botão de barra de título).
- 2) Se um programa usuário fez '#include' de outro utilizador arquivos, o **Salvar** botão dentro **Editar/Examinar** teria falhado para realmente salvar uma arquivo modificado se houve um erro Analisar ou Execução existente sinalizado dentro de um arquivo diferente.
- 3) UMA **Cancelar** após um **Salvar** também pode ser confuso - por estas razões a **Salvar** e **Cancelar** funcionalidades dos botões foram alterados (veja **Mudanças e melhorias**)..
- 4) suportes desequilibradas dentro de um 'class' definição poderia causar um travamento desde V2.5.
- 5) teste lógico direto sobre 'long' valores devolvidos 'false' Se nenhum dos 16 bits de mais baixas foram definidos.
- 6) UnoArduSim tinha sido sinalizar um erro quando um ponteiro variável foi declarado como o loop variável no interior dos suportes de um 'for()' declaração.
- 7) UnoArduSim tinha sido disallowing testes lógicos que ponteiros em comparação com 'NULL' ou '0'.
- 8) UnoArduSim tinha sido disallowing aritmética ponteiro envolvendo um número inteiro variável (apenas inteiro constantes tinha sido permitido).
- 9) Interrupções definidas usando 'attachInterrupt(pin, name_func, LOW)' só havia sido detectado em um **transição** para 'LOW'.
- 10) Ao usar mais de um dispositivo 'I2CSLV', um escravo não endereçado pode interpretar dados de barramento posteriores como correspondendo ao seu endereço de barramento (ou chamada global 0x00) e sinalizar falsamente o ACK, corrompendo o nível de ACK do barramento e interrompendo um 'requestFrom()'.
- 11) Passando valor numérico '0' (ou 'NULL') Como um argumento módulo funcional para um ponteiro em uma chamada módulo funcional agora é permitido.
- 12) O nível travessão tabulação depois de um assentamento de 'switch()' construções era muito raso quando a escolha 'auto-indent formatting' de **Configurar | Preferências** foi usado.
- 13) Subtração de dois ponteiros compatíveis agora resulta em um tipo de 'int'.
- 14) UnoArduSim estava esperando um construtor padrão definido pelo usuário para um membro objeto mesmo que não tinha sido declarado como 'const'.
- 15) Em uma pausa execução, a posição desenhada de um 'STEPR', 'SERVO', ou 'MOTOR' motor pode ficar por até 30 milissegundos de movimento por trás da sua posição última calculado real.
- 16) 'Stepper::setSpeed(0)' estava causando um acidente por causa de uma divisão por zero.
- 17) Única linha 'if()', 'for()' e 'else' não causa um muitas abas auto-recuo constrói.

V2.5.0- outubro 2019

- 1) A erro introduzido na V2.4 inicialização cartão de 'SD' quebrou (causou um acidente).
- 2) Usando o subsistema 'SPI' na nova 'SPI_SLV' Modo trabalhou de forma inadequada em 'SPI_MODE1' e 'SPI_MODE3'.
- 3) Auto-conclusão pop-ups (tal como solicitado por 'ALT-right=arrow') foram fixados para módulos funcionais ter objeto parâmetros; a lista de pop-ups agora também incluem herdou os membros da classe (base-), e auto-

completação agora também aparecem para `'Serial'`.

4) Desde V2.4 o valor de retorno para `'SPI.transfer()'` e `'SPI.transfer16()'` foi incorreto se uma rotina de interrupção do usuário disparado durante essa transferência.

5) Em V2.4, formas de onda periódicas rápidas foram mostrados como tendo uma duração maior do que a sua duração efectiva evidente quando visto em zoom superior.

6) o construtor `'File::File(SdFile &sdf, char *fname)'` não estava funcionando, por isso, `'File::openNextFile()'` (Que se baseia em que construtor) também não estava funcionando.

7) UnoArduSim foi incorretamente declarando um erro de Analisar em objeto-variáveis, e objeto-retornando módulos funcionais, declarado como `'static'`.

8) instruções de atribuição com `'Servo'`, `'Stepper'` ou `'OneWire'` objeto variável na LHS, e um objeto-retornando módulo funcional ou construtor no RHS, causou um miscount interna do número de objetos associada, conduzindo a uma eventual batida.

9) testes booleanas na objetos de tipo `'File'` foram sempre retornando `'true'` mesmo se o arquivo não estava aberto.

V2.4 - maio de 2019

1) Executar Até watch-points podem ser falsamente acionados por uma gravação em um variável adjacente (1 byte menor no endereço).

2) Seleções de taxa de transmissão podem ser detectadas quando o mouse estava dentro de um `'SERIAL'` ou `'SFTSER'` Caixa de listagem suspensa dispositivo baud (mesmo quando nenhuma taxa de transmissão foi realmente clicada).

3) Desde a V2.2, erros de recepção serial ocorreram em uma taxa de transmissão de 38400.

4) Uma alteração feita na V2.3 causou **'SoftwareSerial'** às vezes reportar erroneamente uma interrupção desativada (e assim falhar na primeira recepção do RX).

5) Uma alteração feita na V2.3 levou o SPISLV dispositivos a interpretar incorretamente os valores hexadecimais diretamente inseridos em sua caixa de edição `'DATA'`.

6) Uma alteração feita em V2.3 causou SRSLV dispositivos às vezes falsamente, e silenciosamente, detectam um conflito elétrico em seu `'Dout'` pin, e assim desautorizam uma atribuição pin lá. Repetidas tentativas de anexar um pin ao `'Dout'` poderiam levar a uma eventual colisão assim que o dispositivo fosse removido.

7) Tentar anexar um `'LED4'` dispositivo passado pin 16 causaria uma falha.

8) Um erro desencadeado por chamadas repetidas rápidas para `'analogWrite(255)'` para **'MOTOR'** controle no usuário programa causado resultante **'MOTOR'** velocidades para ser incorreto (muito lento).

9) Como a V2.3, o SRSLV dispositivos não pode receber um `'Dout'` pin devido a uma detecção elétrica conflito defeituosa (e assim desautorizar uma atribuição pin).

10) Os escravos SPI agora redefinem sua lógica de transmissor e receptor quando **'SS'** pin vai **'HIGH'**.

11) Chamando `'Wire.h'` módulos funcionais `'endTransmission()'` ou `'requestFrom()'` quando as interrupções estão atualmente desativadas agora gera um erro execução (`'Wire.h'` precisa de interrupções habilitadas para funcionar).

12) O `'Ctrl-Home'` e o `'Ctrl-End'` agora funcionam como esperado no Editar/Examinar.

13) o `'OneWire'` comando de barramento `0x33 ('ROM_READ')` não estava funcionando, e desligou o ônibus.

V2.3 - dez. 2018

1) Um erro introduzido na V2.2 tornou impossível editar o valor de um elemento matriz dentro **Editar/Monitorar**

2) Desde a versão 2.0, o texto no `'RAM free'` O controle da barra de ferramentas só era visível se estivesse usando um tema escuro do Janelas-OS.

3) Em **Arquivo | Carregar** em I / O dispositivo arquivo **Carregar**, os filtros arquivo (como `'*.ino'` e `'*.txt'`) não estavam funcionando - arquivos de todos os tipos foram mostrados.

- 4) O estado “modificado” de um arquivo foi perdido após fazer **Aceitar** ou **Compilar** em um subsequente **Arquivo | Editar/Examinar se nenhuma outra edição tiver sido feita** (**Salvar** tornou-se desativado, e não houve prompt automático para **Salvar** em programa **Saída**).
- 5) Operadores '`/=`' e '`%=`' só deu resultados corretos para '`unsigned`' lado esquerdo variáveis
- 6) O condicional ternário '`(bval) ? S1:S2`' deu um resultado incorreto quando '`S1`' ou '`S2`' era uma expressão local em vez de um variável.
- 7) Módulo funcional '`noTone()`' foi corrigido para se tornar '`noTone(uint8_t pin)`'.
- 8) Um erro de longa data causou um acidente depois de prosseguir **Reinicializar** quando aquele Reinicializar foi feito no meio de um módulo funcional que foi chamado com um de seus parâmetros faltando (e então recebendo um valor inicializador padrão).
- 9) Expressões de membros (por exemplo '`myobj.var`' ou '`myobj.func()`') não estavam herdando '`unsigned`' propriedade do seu lado direito ('`var`' ou '`func()`') e, portanto, não puderam ser diretamente comparados ou combinados com outros '`unsigned`' tipos - uma atribuição intermediária a um '`unsigned`' variável foi primeiro requerido.
- 10) UnoArduSim estava insistindo que se uma definição do módulo funcional tivesse algum parâmetro com um inicializador padrão, que o módulo funcional tem um protótipo anterior declarado.
- 11) Chamadas para '`print(byte bvar, base)`' erroneamente promovido '`bvar`' para um '`unsigned long`', e assim imprimiu muitos dígitos.
- 12) '`String(unsigned var)`' e '`concat(unsigned var)`' e operadores '`+=(unsigned)`' e '`+(unsigned)`' criado incorretamente '`signed`' seqüências de caracteres em vez disso.
- 13) Um 'R=1K' dispositivo carregado de um **IODevices.txt** arquivo com posição '`U`' foi erroneamente desenhado com o seu cursor (sempre) no **posição oposta** da sua posição elétrica verdadeira.
- 14) Tentando confiar no padrão '`inverted=false`' argumento ao declarar um '`SoftwareSerial()`' objeto causou um acidente e passando '`inverted=true`' só funcionou se o usuário programa fez um subsequente '`digitalWrite(txpin, LOW)`' a fim de estabelecer primeiro o tempo ocioso necessário '`LOW`' nível no **TX** pin
- 15) O 'I2CSLV' dispositivos não respondeu a alterações em suas caixas de edição pin (os padrões A4 e A5 permaneceram em vigor).
- 16) 'I2CSLV' e 'SPISLV' dispositivos não detectaram e corrigiram edições parciais quando o mouse saiu de suas bordas
- 17) Os valores de Pin para dispositivos que seguiram um SPISLV ou SRSLV foram gravados incorretamente no **IODevs.txt** arquivo como hexadecimais.
- 18) Tentar ligar mais do que um MIS SPISLV dispositivo ao pin 12 gerou sempre um erro Elétrico Conflito.
- 19) Mudar um pin de '`OUTPUT`' de volta a **Modo** '`INPUT`' Falha ao redefinir o nível de trava de dados pin para '`LOW`'.
- 20) Usando '`sendStop=false`' em chamadas para '`Wire.endTransmission()`' ou '`Wire.requestFrom()`' causou uma falha.
- 21) UnoArduSim permitiu indevidamente um '`SoftwareSerial`' recepção ocorrer simultaneamente com um '`SoftwareSerial`' transmissão.
- 22) Variáveis declarado com '`enum`' Não foi possível atribuir ao tipo um novo valor após sua linha de declaração, e o UnoArduSim não estava reconhecendo os membros do 'enum' quando referenciado com um (legal) '`enumname::`' prefixo.

V2.2 - jun. 2018

- 1) Chamar um módulo funcional com menos argumentos do que o necessário por sua definição (quando aquele módulo funcional era “definido para frente”, isto é, quando não tinha uma linha de declaração protótipo anterior) causava uma violação de memória e travamento.
- 2) Desde V2.1, **Formas de onda** não tinha sido atualizado durante **Executar** (apenas em **Parar** depois de um **Passo**) - além do que, além do mais, **Painel de Variáveis** valores não estavam sendo atualizados durante longos

Passo operações.

- 3) Alguns menores **Forma de onda** problemas de rolagem e zoom que existiam desde a V2.0 foram corrigidos.
- 4) Mesmo nas primeiras versões, o Reinicializar em $t = 0$ com um PULSER ou FUNCGEN, cujo período faria seu último ciclo antes de $t = 0$ seja apenas um **parcial** ciclo, resultou em sua **Forma de onda** depois de $t = 0$ sendo deslocado de sua posição verdadeira por essa (ou a quantidade restante) do ciclo fracionário, para a direita ou para a esquerda (respectivamente).
- 5) Corrigidos alguns problemas com realce de cor de sintaxe em **Editar/Examinar**.
- 5) Desde V2.0, clicar para expandir um objeto em um matriz de objetos não funcionou corretamente.
- 6) `'delay(long)'` foi corrigido para ser `'delay(unsigned long)'` e `'delayMicroseconds(long)'` foi corrigido para ser `'delayMicroseconds(unsigned int)'`.
- 7) A partir da V2.0, módulos funcionais anexado usando `'attachInterrupt()'` não estavam sendo verificados como sendo válidos módulos funcionais para esse fim (ou seja, `'void'` retornar e não ter parâmetros de chamada).
- 8) O impacto de `'noInterrupts()'` em módulos funcionais `'micros()'`, `'mills()'`, `'delay()'`, `'pulseInLong()'`; seu impacto sobre `'Stepper::step()'` e depois `'read()'` e `'peek()'` timeouts; em todas as recepções seriais RX e `'Serial'` transmissão, agora é reproduzido com precisão.
- 9) O tempo gasto dentro das rotinas de interrupção do usuário agora é contabilizado no valor retornado por `'pulseIn()'`, o atraso produzido por `'delayMicroseconds()'`, e na posição das bordas no visor **Formas de Onda Digitais**.
- 10) Chamadas para **objeto-membro** módulos funcionais que faziam parte de expressões complexas maiores, ou estavam dentro de chamadas módulo funcional tendo múltiplos argumentos complexos, e, g, `'myobj.memberfunc1() + count/2'` ou `'myfunc(myobj.func1(), count/3)'`, teria valores incorretos computados / passados no tempo de execução devido a alocações de espaço de pilha defeituosas.
- 11) Matrizes do ponteiro variáveis funcionou corretamente, mas teve valores mostrados defeituosos mostrados no **Painel de Variáveis**.
- 12) Quando matrizes dinâmico de tipo simples foi criado com `'new'`, apenas o primeiro elemento estava recebendo uma inicialização (padrão) para o valor 0 - agora todos os elementos fazem isso.
- 13) `'noTone()'`, ou o fim de um tom finito, não mais redefine o pin (ele permanece `'OUTPUT'` e vai `'LOW'`).
- 14) A rotação contínua 'SERVO' dispositivos está agora perfeitamente estacionária a uma largura de pulso de 1500 microssegundos.
- 15) A chamada em `SdFile::ls()` (listagem do diretório do cartão SD) funcionou corretamente, mas mostrou indevidamente algumas transferências de bloco SPI duplicadas nas formas de onda janela.

V2.1.1 - Mar. 2018

- 1) Corrigidas inconsistências em idiomas não ingleses com o idioma salvo para **'myArduPrefs.txt'**, com os botões de idioma de rádio exibidos no **Preferências** caixa de diálogo e com correspondência às linhas traduzidas **'myArduPrefs.txt'**.
- 2) Alocações com `'new'` agora aceite uma dimensão matriz inteira que não seja uma constante.
- 3) Clicando no **Painel de Variáveis** para expandir um matriz multidimensional mostraria um vazio supérfluo `'[]'` par de parênteses.
- 4) Referências de elementos Matriz com caracteres supérfluos à direita (por exemplo, `'y[2]12'`) não foram pegos como erros no tempo de Analisar (os caracteres extras foram simplesmente ignorados).

V2.1 - mar. 2018

- 1) Um erro nas novas versões V2.0.x fez com que o heap Janelas crescesse a cada atualização no **Painel de Variáveis** -- depois de milhões de atualizações (muitos minutos de execução), um acidente poderia resultar.
- 2) Ligações para o 'static' membro módulos funcionais usando duplo-cólon `'::'` notação falhou ao Analisar

quando dentro `'if()'`, `'while()'`, `'for()'` e `'switch()'` parênteses, e quando dentro de expressões usadas como argumentos de chamada módulo funcional ou índices matriz.

V2.0.2 Fev. 2018

- 1) Um erro introduzido na V2.0 causou **Arquivo | Carregar** falhar se um `'#include'` refere-se a um arquivo ausente ou vazio
- 2) Dentro de um **IODEVS.TXT** arquivo, ele **'I/O'** o nome 'One-Shot' era esperado em vez do antigo 'Oneshot'; ambos são aceitos agora.

V2.0.1 - jan. 2018

- 3) Em idiomas de idioma diferente do inglês, **'en'** foi mostrado incorretamente como selecionado em **Preferências**, tornando reverter para Inglês inábil (exigindo des-seleção e re-seleção).
- 4) Tinha sido possível para o usuário deixar um valor de caixa de edição Dispositivo pin em um estado incompleto (como 'A_') e deixar os 'DATA' bits de um 'SRS:V' incompletos.
- 5) O número máximo de Sliders Analógico foi limitado a 4 (agora corrigido para 6).
- 6) UnoArduSim não insiste mais em `'='` aparecendo em uma inicialização agregada matriz.
- 7) UnoArduSim insistiu que o argumento "inverted_logic" seja fornecido para `'SoftwareSerial()'`.
- 8) As operações de mudança de bit agora permitem turnos maiores que o tamanho do variável deslocado.

V2.0 - dez. 2017

- 1) Todos módulos funcionais que foram declarados como **'unsigned'** tinha, no entanto, retornado valores como se fossem **'signed'**. Isso não teve efeito se o **'return'** valor foi atribuído a um **'unsigned'** variável, mas teria causado uma imprópria interpretação negativa se tinha `MSB == 1`, e foi então atribuído a um **'signed'** variável, ou testado em uma desigualdade.
- 2) Analógico Sliders estavam atingindo apenas um máximo `'analogRead()'` valor de 1022, não o correto 1023.
- 3) Um erro inadvertidamente introduzido de volta em V1.7.0 na lógica usada para acelerar o manuseio do sistema SPI O pin SCK causou transferências de SPI para `'SPI_MODE1'` e `'SPI_MODE3'` falhar após o primeiro byte transferido (um extra espúrio Transição SCK seguiu cada byte). Também atualizações para uma caixa 'SPISLV' editar 'DATA' para bytes transferidos foram atrasadas,
- 4) O dispositivo DEL não estava listando o 'B' (para azul) como uma opção de cor (embora tenha sido aceito).
- 5) As configurações para 'SPISLV' e 'I2CSLV' dispositivos não estavam sendo salvas no usuário **'I/O' Dispositivos** arquivo
- 6) Cópia **'Servo'** instâncias falhou devido a um defeito `'Servo::Servo(Servo &tcopy)'` implementação de construtor de cópias.
- 7) fora de alcance `'Servo.writeMicroseconds()'` os valores foram detectados corretamente como um erro, mas os valores-limite declarados que acompanhavam o texto da mensagem de erro estavam errados.
- 8) Um taxa de baud legal de 115200 não foi aceito quando carregado de um **'I/O' Dispositivos** texto arquivo.
- 9) pin elétrica conflitos causada por um Deslizante Analógico dispositivo anexado nem sempre foram detectados.
- 10) Em casos raros, passando um ponteiro de string defeituoso (com a terminação 0 da terminação) **'String'** O módulo funcional pode causar o travamento do UnoArduSim.
- 11) O **Painel de Códigos** poderia realçar a linha de erro Analisar atual no **errado** Módulo programa (quando `'#include'` foi usado).
- 12) Carregar um 'I/O' Dispositivos arquivo que tinha um dispositivo que (indevidamente) impelir contra 'Uno' pin 13 causou um travamento do programa no pop-up da mensagem de erro.
- 13) UnoArduSim tinha permitido por engano o usuário faça o colar de caracteres não-hexadecimais no buffer expandidos TX janelas para SPISLV e I2CSLV.
- 14) Inicializações de linha de declaração falhou quando o valor do lado direito era o **'return'** valor de um

membro objeto-módulo funcional (como em `'int angle = myservo1.read();'`).

15) `'static'` membro variáveis ter explícito `'ClassName::'` os prefixos não eram reconhecidos se aparecessem no início de uma linha (por exemplo, em uma atribuição a uma base - `'class'` variável),

16) Chamando `'delete'` em um ponteiro criado por `'new'` só foi reconhecida se a notação módulo funcional parênteses foi usada, como em `'delete(pptr)'`.

17) Implementação UnoArduSim de `'noTone()'` estava incorretamente insistindo que um argumento pin fosse fornecido.

18) Alterações que aumentaram bytes 'RAM' globais em um programa que usava `'String'` variáveis (via **Editar/Examinar** ou **Arquivo | Carregar**), poderia levar à corrupção nesse espaço global 'Uno' devido à exclusão de heap do `'String'` objetos pertencente ao antigo programa ao usar (incorretamente) o heap pertencente ao novo programa. Em algumas circunstâncias, isso pode levar a uma queda do programa. Embora um segundo Carregar ou Analisar tenha resolvido o problema, este erro foi finalmente corrigido.

19) Os valores de retorno para `'Wire.endTransmission()'` e `'Wire.requestFrom()'` ambos estavam presos em 0 - agora foram consertados.

V1.7.2 - fev. 2017

1) Interrupções no pin 2 também estavam sendo (inadvertidamente) acionadas pela atividade do sinal no pin 3 (e vice-versa).

V1.7.1 - fev. 2017

1) Módulo funcional `'delayMicroseconds()'` estava produzindo um atraso na *mili*-segundos (1000 vezes muito grandes).

2) tipo-conversão explícito de um `'unsigned'` variável para um tipo inteiro mais longo resultou em um incorreto (`'signed'`) resultado.

3) Literais hexadecimais maiores que `0x7FFF` são agora `'long'` por definição, e assim irá gerar agora `'long'` expressões aritméticas resultantes em que se envolvem.

4) Um erro inadvertidamente introduzido pelo V1.7.0 impedia o estilo alternativo de C++ tipo-conversão de literais numéricos (por exemplo, `'(long)1000*3000'` não foi aceito).

5) `'Serial'` não ocupa mais seus muitos bytes em RAM 'Uno' se nunca for necessário pelo usuário programa.

6) O variáveis global declarado pelo usuário não ocupa mais espaço na RAM 'Uno' se nunca for realmente usado.

7) Single variáveis declarado como `'const'`, `'enum'` membros, e ponteiros para string literais, não ocupam mais espaço na RAM 'Uno' (para concordar com a compilação do Arduino),

8) Bytes de RAM necessários para `'#include'` As bibliotecas incorporadas agora estão de acordo com os resultados da compilação condicional do Arduino.

9) Usando `'new'` em uma linha de declaração real ponteiro tinha falhado (apenas um mais tarde `'new'` atribuição ao ponteiro funcionou).

10) Corrigido um erro onde um show "pendente" de um diretório de disco SD poderia causar um travamento do programa.

V1.7.0 - dez. 2016

0) Vários problemas com o manuseio de interrupções do usuário foram corrigidos:

a) Interrompe as arestas 0 e 1 que ocorreram durante um Arduino módulo funcional blocos enquanto espera (como `'pulseIn()'`, `'shiftIn()'`, `'SPI.transfer()'`, `'flush()'`, e `'write()'`) causou uma falha no fluxo execução no retorno de interrupção

b) Várias cópias do variáveis local de qualquer módulo funcional interrompido sido aparecendo no **Painel de Variáveis** (uma cópia por interrupção-retorno) e isso foi corrigido em V1.6.3, mas os outros problemas de interrupção permaneceram).

- c) Módulo funcional '**delayMicroseconds()**' não estava criando nenhum atraso se chamado de dentro de uma rotina de interrupção do usuário.
- d) Chamadas para bloquear módulos funcionais como '**pulseIn()**' de **dentro** uma interrupção a rotina não estava funcionando.
- 1) Um erro introduzido em V1.6.3 causou perda de atualização de valor no **Painel de Variáveis** durante a execução quando os valores estavam realmente mudando (isso aconteceu somente após dois ou mais **Parar** ou menu **VarAtualizar** ações do usuário). Além disso, quando um Executar Para foi feito após **Permitir Redução** tinha sido desencadeada, o **Painel de Variáveis** ocasionalmente não foi redesenhado (então valores antigos e locais-variáveis podem ter aparecido lá até o próximo Passo).
- 2) o **Painel de Códigos** realçando o comportamento do **Passo Acima** comando pode parecer enganoso em '**if()-else**' correntes - que agora foi corrigido (embora a funcionalidade real do passo estivesse correta).
- 3) Módulo funcional '**pulseIn()**' tinha definido indevidamente o tempo limite em milissegundos em vez de microssegundos - também estava reiniciando incorretamente o tempo limite quando as transições para níveis inativos e ativos foram vistas pela primeira vez.
- 4) Usando literais HEX entre 0x8000 e 0xFFFF em atribuições ou aritmética com '**long**' inteiro variáveis deu resultados incorretos devido a extensão de sinal não verificada.
- 5) Passando ou retornando para um '**float**' De qualquer '**unsigned**' tipo inteiro tendo um valor com MSB = 1 deu resultados incorretos devido a uma falha '**signed**' interpretação.
- 6) Todos '**bit_()**' módulos funcionais agora também aceita operações em '**long**' -tamanho variáveis, e UnoArduSim testa para posições de bit inválidas (que ficariam fora do tamanho variável).
- 7) Uma entrada inválida para a caixa de edição 'Pulse' (largura) em um 'PULSER' Dispositivo causou a corrupção do valor 'Period' (até corrigido pela próxima entrada de edição do usuário 'Period').
- 8) Excluindo um 'PULSER' ou 'FUNCGEN' dispositivo usando o O menu Configurar não estava removendo seu sinal periódico do pin que estava dirigindo (um Reinicializar não é mais necessário).
- 9) A capacidade de inicializar um 1-D '**char**' matriz com uma seqüência de caracteres citada estava ausente, (por exemplo '**char strg[] = "hello";** ').
- 10) A exibição hexadecimal no monitor expandidos 'SERIAL' ou 'SFTSER' janelas mostrou o caractere mais significativo incorreto para valores de byte maiores que 127.
- 11) O Forma de onda janelas não estava refletindo mudanças programáticas feitas pelo usuário '**analogWrite()**' quando o um novo valor foi 0% ou 100% ciclo de serviço.
- 12) A implementação de '**Serial.end()**' agora foi corrigido.
- 13) UMA '**myArduPrefs.txt**' arquivo com mais de 3 palavras em uma linha (ou espaços no '**I/O Dispositivos** Nome arquivo) pode causar uma falha devido a um ponteiro interno defeituoso.
- 14) A linha final de um '**I/O Dispositivos** arquivo não foi aceito se não **terminar com um avanço de linha**.
- 15) Adicionando mais de quatro Analógico Sliders causou um silencioso erro que substituiu os ponteiros DEL 'I/O' dispositivo
- 16) Começando com V1.6.0, analógico forma de onda amostras para o **primeira metade** De cada **triângulo** forma de onda foram todos **zero** (devido a um erro no cálculo da tabela forma de onda).
- 17) Fazendo um repetido **Executar Para** quando em uma linha ponto de parada não requer mais vários cliques por avanço.
- 18) A passagem de expressões de endereço para um parâmetro módulo funcional matriz não foi aceita pelo Analisador.
- 19) O módulos funcionais recursivo que retornou expressões contendo referências de ponteiro ou matriz forneceu resultados incorretos devido a sinalizadores "prontos" não redefinidos nessas expressões componentes.
- 20) Chamando '**class**' membro-módulos funcionais através **qualquer ponteiro objeto variável ou expressão de ponteiro** não estava funcionando.
- 21) O usuário módulos funcionais que retornou objetos por valor retornou com sucesso seu valor em sua primeira

chamada módulo funcional **E se** eles retornaram um objeto construído sem nome (como `'String("dog")'`) - nas chamadas subsequentes, o retorno foi ignorado devido a um sinalizador "pronto" emperrado.

22) Não houve salvaguarda para impedir o comando **Janelas | 'Serial' Monitor** from adicionando um novo `'SERIAL'` dispositivo quando realmente não havia espaço para isso.

23) Se adicionar um pins fixo pins (como 'SPISLV') causou uma mensagem pop-up pin conflito, o **Painel de Banco de Laboratório** redesenho poderia mostrar um "fantasma" duplicado dispositivo sobrepondo o 'I/O' dispositivo mais à direita (até o próximo redesenho).

24) Corrigido alguns problemas com sons não confiáveis do 'PIEZO' para sinais pin não periódicos.

25) `'PROGMEM'` variáveis também deve agora ser explicitamente declarado como `'const'` concordar com o Arduino.

26) "Sem espaço de heap" foi sinalizado incorretamente como um erro execução quando `'SD.open()'` não conseguiu encontrar o arquivo nomeado, ou um `'openNextFile()'` atingiu o último arquivo no diretório.

27) Um Analisador erro estava aceitando indevidamente um chavetas de fechamento fora de lugar `'}'`.

28) Um erro com **Painel de Variáveis** remoções no retorno do construtor member-objeto foram corrigidas (o erro aplicado apenas para o objetos que contém outros objetos como membros).

V1.6.3- setembro de 2016

1) O variáveis local de qualquer módulo funcional interrompido não estava sendo removido do **Painel de Variáveis** na interrupção da entrada módulo funcional, levando a várias cópias aparecendo lá no retorno módulo funcional de interrupção (e um possível erro execução eventual ou uma falha).

2) O Forma de onda janelas não estava refletindo mudanças programáticas `'analogWrite()'` para um novo ciclo de trabalho de 0% ou 100%.

3) A exibição hexadecimal no monitor expandidos 'SERIAL' ou 'SFTSER' janela mostrou o caractere MSB incorreto para valores de byte maiores que 127.

V1.6.2- setembro de 2016

1) As chamadas Módulo funcional feitas com o número errado ou tipo de argumentos não geraram uma mensagem de erro Analisar apropriada (somente o genérica "não é um identificador válido" mensagem apareceu).

2) o **Barra de ferramentas** botão de reset agora funciona de forma idêntica ao botão de reset 'Uno' placa de circuito.

3) O texto do erro Analisar não é mais cortado após 16 caracteres sem mostrar reticências.

V1.6.1 - ago. 2016

1) Em V1.6 uma versão 'Uno' placa de circuito no `'myArduPrefs.txt'` O arquivo que diferia do valor padrão da versão 2 causou uma exceção na inicialização (devido a um evento 13 não inicializado do pin).

2) Alterando o valor de um variável clicando duas vezes no **Painel de Variáveis** pode causar erros de "falta de alocação de memória" pop-ups (para programas com qualquer usuário definido `'class'`).

3) `'SoftwareSerial'` não permitiu o acesso a `'write(char* ptr)'` e `'write(byte* ptr, int size)'` módulos funcionais devido a uma falha na detecção de sobrecarga do módulo funcional.

4) Corrigido o problema com a inclusão automática do correspondente ".cpp" arquivo para uma biblioteca ".h" isolada `'#include'`.

V1.6 - Jun. 2016

1) Na V1.5 recuo automático na chave 'Enter' em **Editar/Examinar** (ao entrar em uma nova linha) foi perdido.

2) Detecção de pin entra em conflito com anexado forte condução externa 'I/O' dispositivos foi agora adicionado em `'Serial'` pin 1, em SPI pins SS, MOSI, e SCK, em I2C pins SCL e SDA (tudo quando o correspondente `'begin()'` é chamado), e em qualquer `'SoftwareSerial'` TX pin.

V1.5.1 - Jun. 2016

- 1) Na V1.5 as novas cores Syntax Realçar adaptáveis ao tema não foram devidamente repostas de cada vez **Editar/Examinar** foi aberto, e assim (com um tema de fundo branco) só estavam corretas a cada segunda vez.
- 2) Interromper '**RISING**' e '**FALLING**' sensibilidades tinham sido oposto para a polaridade real da borda de disparo.

V1.5 - maio de 2016

- 1) Um erro introduzido em V1.4.1 impediu a passagem de literais vazios para membro módulos funcionais que esperava um '**String**' objeto, como em '**mystring1.startsWith("Hey")**' .
- 2) Um erro no original **SD** implementação do UnoArduSim apenas permitido **SD** acesso usando chamadas para '**read()**' e '**write()**' (acesso via '**Stream**' módulos funcionais foi impedido).
- 3) Os interruptores deslizantes 'R=1K' não estavam sendo redesenhados corretamente quando o controle deslizante foi movido.
- 4) **Cancelar** na caixa de diálogo Confirm-Salvar arquivo deve ter impedido a saída do aplicativo.
- 5) Uma cotação de fechamento ou fechamento ausente '>' -parênteses em um usuário arquivo '**#include**' causaria um jeito.
- 6) Corrigido um erro no realce de sintaxe de '**String**' e usuário '**class**' ou '**struct**' e destaque prolongado para incluir chamadas de construtor módulo funcional.
- 7) Corrigidos problemas menores em **Editar/Examinar** com alterações de texto / realce e **Desfazer** botão.

V1.4.3 - abr 2016

- 1) Usando **Configurar | 'I/O' Dispositivos** para adicionar o novo dispositivos e, em seguida, remover um desses dispositivos recém-adicionados pode causar uma falha na reinicialização ou outro dispositivo parar de funcionar.
- 2) Modificando um '**String**' variável clicando duas vezes no **Painel de Variáveis** falhou (o novo '**String**' foi lido indevidamente).
- 3) Pin alterações no 'FUNCGEN' e 'PULSER' dispositivos não foram reconhecidos até que uma redefinição foi feita pela primeira vez.

V1.4.2 - mar. 2016

- 1) V1.4.1 tinha introduzido um desafortunado Analisar erro que impedia tarefas envolvendo qualquer '**class**' objetos (incluindo '**String**').
- 2) Uma correção erro incompleta feita em V1.4.1 causou '**unsigned**' valores do tipo '**char**' para imprimir como caracteres ASCII em vez de seus valores inteiros.
- 3) Os argumentos de chamada módulo funcional de expressão de membro complexos nem sempre foram reconhecidos como correspondências válidas de parâmetros módulo funcional.
- 4) **Todos** literais inteiros e expressões foram dimensionados muito generosamente (para '**long**') e, portanto, a execução não refletia **real** overflows (para negativo) que podem ocorrer no Arduino em operações de adição / multiplicação envolvendo '**int**' valores dimensionados.
- 5) Expressões envolvendo uma mistura de '**signed**' e '**unsigned**' tipos inteiros nem sempre foram tratados adequadamente '**signed**' valor seria indevidamente visto como '**unsigned**').
- 6) Nos casos pin-conflito, "valor =" mensagens de erro podem mostrar valores pin obsoletos, mesmo após um Reinicializar de um conflito anterior que o usuário já tenha liberado.

V1.4.1 - jan. 2016

- 1) Chamadas para '**print(char)**' agora imprima corretamente como caracteres ASCII (em vez de valores numéricos).

- 2) A resposta de interrupção agora está ativada por padrão quando '`attachInterrupt()`' é chamado, então não há mais necessidade em sua '`setup()`' para chamar o módulo funcional de habilitação '`interrupts()`'.
- 3) Múltiplo '`#include`' instâncias do usuário-arquivos de dentro de um arquivo agora são tratados corretamente.

V1.4 - dez. 2015

- 1) UMA **de longa data** erro marcou incorretamente um **divida por zero** condição ao dividir por um valor fracional menor que a unidade.
- 2) Fixo '`SoftwareSerial`' (que foi inadvertidamente quebrado por um '`class`' -verificação de validação de membro na V1.3 lançamentos).
- 3) As chamadas módulo funcional de fim de linha com um ponto e vírgula faltando não foram detectadas e fizeram com que o Analisador pulasse a próxima linha.
- 4) Um mal formatado '**I/O Dispositivos**' texto arquivo deu uma mensagem de erro imprópria.
- 5) Analisar Erro ao realçar a linha (adjacente) incorreta em expressões e instruções de várias linhas foi corrigido
- 6) Teste lógico de ponteiros usando o '`not`' (!) operador foi invertido.

V1.3 - outubro de 2015

- 1) Manuseio interno inadequado do scratchpad variáveis causou ocasionais " **Profundidade máxima de aninhamento do bloco de rascunho excedida** "Erros do Analisar.
- 2) Parênteses *dentro de aspas simples*parênteses ponto e vírgula, parentheses dentro de strings entre aspas e os caracteres com escape foram manipulados indevidamente.
- 3) Um Matriz com uma dimensão vazia e nenhuma lista de inicialização causou uma interrupção de RESET, e o matrizes com apenas um único elemento não foi desautorizado (e causou sua interpretação defeituosa como um ponteiro inicializado inválido).
- 4) Às vezes, os erros do Analisar às vezes levavam o realçar à linha errada (adjacente).
- 5) Passando um ponteiro para um não-const para um módulo funcional aceitando um ponteiro para um '`const`' tinha sido desaprovado (em vez do contrário).
- 6) Expressões de inicialização estavam herdando indevidamente '`PROGMEM`' qualificadores do variável sendo inicializados
- 7) '`PROGMEM`' variáveis declarado teve seu tamanho de byte incorretamente contado **duas vezes** contra a alocação de memória 'Flash' durante o processo Analisar.
- 8) Digitando na caixa de edição do 'Send' de um 'I2CSLV' às vezes causaria um acidente devido a '`sscanf`' erro
- 9) Carregar um novo programa com um novo '**I/O Dispositivos**' arquivo no seu anuário poderia causar irrelevante pin entra em conflito com **velho** Direções pin.
- 10) O tratamento de caracteres seriais com escape foi aplicado indevidamente a seqüências de caracteres recebidas, em vez de transmitidas, no campo (maior) '**Serial Monitor** buffers janela.
- 11) '`while()`' e '`for()`' loops com corpos completamente vazios, como '`while(true);`' ou '`for(int k=1;k<=100;k++);`' passou o Analisador (com uma mensagem de aviso) mas falhou na hora execução.

V1.2 - Jun. 2015

- 1) O mais simples do usuário módulos funcionais que fez chamadas para '`digitalRead()`' ou para '`analogRead()`' ou '`bit()`' poderia ter corrompido seu (muito primeiro) variável local declarado (se algum) devido a insuficiente espaço de rascunho módulo funcional alocado (se apenas dois bytes de bloco de notas fossem alocados no início da pilha módulo funcional). Qualquer expressão numérica em um módulo funcional é suficiente para causar uma alocação do bloco de notas de 4 bytes, evitando assim esse problema. Este desafortunado erro existe desde o lançamento original V1.0.
- 2) Módulos funcionais que são '`void`' com um explícito precoce '`return`' e não '`void`' módulos funcionais com mais de um '`return`' declaração, veria execução fall-through no **fechando chavetas** (se foi atingido).

- 3) Qualquer '**return**' declarações dentro '**if()**' os contextos que estavam faltando parênteses levaram a um destino de retorno para o chamador com defeito.
- 4) '**PULSER**' e larguras de pulso '**FUNCGEN**' ou períodos de valor 0 podem causar uma falha (0 é agora não permitido).
- 5) Onde não havia parênteses, '**else**' continuações após uma '**if()**' não funcionou se seguissem uma '**break**', '**continue**' ou '**return**'.
- 6) Quando **múltiplo 'enum' do utilizador- declarações foram feitas**, constantes definidas em todos, mas no primeiro '**enum**' gerado com falha '**enum**' incompatibilidade " Erros Analisar (este erro foi introduzido na V1.1).
- 7) Um identificador nulo para o último parâmetro de um módulo funcional protótipo causou um erro Analisar.
- 8) **Executar Para** os pontos de interrupção definidos em linhas complexas nem sempre foram manipulados adequadamente (e, portanto, podem ser perdidos).
- 9) '**HardwareSerial**' e '**SoftwareSerial**' usou um buffer de implementação privada com TX pendente que não foi limpo no Reinicializar (então os caracteres remanescentes da última vez podem aparecer).
- 10) O Analisador falhou em verificar a troca ilegal de bits '**float**' e aritmética de ponteiro tentada com operadores ilegais.

V1.1 - mar. 2015

- 1) Índices Matriz que foram '**byte**' ou '**char**' tamanho variáveis causou deslocamentos matriz incorretos (se um variável adjacente continha um byte não-0 alto).
- 2) O teste lógico de ponteiros testou o valor apontado para não-zero em vez do próprio valor do ponteiro.
- 3) Qualquer '**return**' declarações incorporadas dentro '**for()**' ou '**while()**' loops foram maltratados.
- 4) Listas de inicialização agregada para matrizes de objetos, ou objetos contendo outras objetos / matrizes, ou listas de inicialização completamente vazias, não estavam sendo tratadas corretamente.
- 5) Acesso de '**enum**' valores de membro usando um '**enumname::**' prefixo não suportado.
- 6) Inicialização de linha de declaração de um '**char[]**' matriz com um literal de string entre aspas não estava funcionando.
- 7) Um matriz sendo passado para um módulo funcional sem inicialização prévia foi indevidamente sinalizado com um erro "usado mas não inicializado".
- 8) Expressões de ponteiro envolvendo nomes matriz foram maltratadas.
- 9) Parâmetros Módulo funcional declarados como '**const**' não foram aceitos.
- 10) O Fomra de Onda Analógica janela não exibiu sinais PWM ('**servo.write()**' e '**analogWrite()**').
- 11) O membro módulos funcionais acessado por meio de um ponteiro objeto dava acesso a membros com defeito.
- 12) As formas de onda não estavam sendo atualizadas quando **Executar Para** ponto de parada foi atingido.
- 13) A modelagem de alocação de registros pode falhar quando um parâmetro módulo funcional é usado diretamente como um argumento para outra chamada módulo funcional

V1.0.2 - ago. 2014

Ordenação fixa de A0-A5 pins no perímetro do 'Uno' placa de circuito.

V1.0.1 - jun. 2014

Corrigido um erro que truncava pastas de edição que eram mais de três vezes o número de bytes no programa original.

V1.0 - primeiro lançamento maio 2014

Mudanças / Melhorias

V2.7.0- Março 2020

- 1) Além da linha de código atual (verde se pronto para ser executado, vermelho se o erro), UnoArduSim agora mantém, para cada módulo, o último código-line clicado pelo usuário ou pilha-navegação (destacado com um fundo escuro de oliva), tomada mais fácil configurar e encontrar linhas ponto de parada temporários (um por módulo é agora permitido, mas apenas a do módulo exibido atualmente está em vigor em um 'Run-To').
- 2) Adicionado nova 'I/O' dispositivos (e apoiando o código da biblioteca 3-party), incluindo 'SPI' e 'I2C' **porto Expanders** 'SPI' e 'I2C' **Multiplexador DEL** Controladores e monitores (DEL matrizes, 4-alfanuméricos, e 4-dígito ou 8-dígito 7 segmentos displays).
- 3) **'Wire'** operações não são mais excluiu do usuário dentro de interrupção rotinas (Isto suporta interrupções externas de um 'I2C' Porto Expander).
- 4) Digital formas de onda mostram agora um nível intermédio (entre **'HIGH'** e **'LOW'**) Quando o pin não está sendo impulsionado.
- 5) Para evitar confusão quando percorrendo ao longo única **'SPI.transfer()'** instruções, UnoArduSim agora garante que anexado 'I/O' dispositivos agora receber a sua vantagem relógio 'SCK' final (atrasou-logic) antes dos módulo funcional retornos.
- 6) Quando a auto-tab-formatação **Preferência** é ativado, digitando um chavetas fechar **'}'** dentro **Editar/Examinar** agora provoca um salto para a posição travessão guia da sua correspondente abertura-chavetas **'{'** parceiro.
- 7) UMA **Reformatar** botão foi adicionado ao **Editar/Examinar** (A causa imediata auto-tab-indent re-formatação) - este botão só é ativado quando a Preferência de auto-tab-indent está habilitado.
- 8) Uma mensagem de erro mais claras agora ocorre quando uma palavra-chave Prefixo (como **'const'**, **'unsigned'**, ou 'PROGMEM') segue um identificador em uma declaração (ele precisa preceder o identificador).
- 9) Inicializado variáveis global, mesmo quando não usado mais tarde, agora estão sempre atribuído um endereço de memória, e assim aparecerá visível.

V2.6.0 janeiro 2020

- 1) display de caracteres-LCD Adicionado dispositivos ter 'SPI', 'I2C', e interface de 4-bi-paralela. Apoiar código fonte biblioteca foi adicionado à pasta 'include_3rdParty' nova instalação (e pode ser acessado por meio de um normal, **'#include'** directiva) - os usuários podem, alternativamente, escolher, em vez escrever seu próprio módulos funcionais para impelir o dispositivo LCD.
- 2) **Painel de Códigos** destaque foi melhorado, com cores realçar separados para um código-line pronto, para um código de linha de erro, e para qualquer outro código-line.
- 3) o **Buscar** menu e ferramenta de bar 'func' ações sobem (anterior-up e de próxima para baixo) não salto para o anterior / seguinte módulo funcional linha de partida e, em vez agora (ou descer) a chamada-stack, com destaque para a linha de código relevante no chamador (ou chamado) módulo funcional, respectivamente, onde a **Painel de Variáveis** o conteúdo é ajustado para mostrar variáveis para o módulo funcional contendo o código de linha destacada no momento.
- 4) Para evitar confusão, um **Salvar** feito dentro **Editar/Examinar** causa um re- imediato **Compilar**, E se o Salvar foi bem sucedida, utilizando um subsequente Cancelar ou Saída agora só reverter texto para que última salvo texto.
- 5) Adicionou-se uma entrada-pulsadas Motor Passo a Passo ('PSTEPR') com 'STEP' (impulso), 'EN*' (permitir), e 'DIR' entradas (de direcção), e uma configuração de micro-passos-per-passo de (1,2,4,8, ou 16) .
- 6) Ambos 'STEPR' e 'PSTEPR' dispositivos agora têm um 'sync' DEL (VERDE para sincronizado, ou RED quando desligado por um ou mais passos.)

- 7) 'PULSER' dispositivos agora tem uma escolha entre microssegundos e milissegundos para 'Period' e 'Pulse'.
- 8) Construídas-em-módulo funcional auto-completação não reter o tipo de parâmetro na frente do nome do parâmetro.
- 9) Quando se muda de volta para a anterior **Painel de Códigos**, Sua linha anteriormente realçado é agora re-destacada.
- 10) Como uma ajuda para definir um ponto de interrupção temporária, usando Cancelar ou de Saída **Editar/Examinar** deixa o realçar na **Painel de Códigos** na linha última visita do cursor em **Editar/Examinar**.
- 11) A (ou 3rd party) definido pelo usuário '**class**' agora é permitido usar '**Print**' ou '**Stream**' como sua classe base. Em apoio a esta, uma nova pasta 'include_Sys' foi adicionado (na pasta de instalação UnoArduSim) que fornece o código fonte para cada base '**class**'. Neste caso, as chamadas para tal base- '**class**' módulos funcionais vai ser tratado de forma idêntica ao utilizador código (que) pode ser em degraus), em vez de como um módulo funcional construídas-em que não pode ser em degraus (tal como '**Serial.print()**').
- 12) Membro-módulo funcional auto-completação agora incluem o nome do parâmetro **ao invés de** seu tipo.
- 13) O UnoArduSim Analisar agora permite que um nome objeto em uma declaração variável a ser prefaciado por seu opcional (e correspondência) '**struct**' 'class' ou palavra-chave, seguido pela '**struct**' ou '**class**' nome.

V2.5.0 outubro 2019

- 1) Adicionado suporte para '**TFT.h**' biblioteca (com exceção de '**drawBitmap()**'), E adicionou-se uma 'TFT' associado 'I/O' Dispositivo (128 x 160 pixels). Note-se que, a fim de evitar atrasos em tempo real excessiva durante grande '**fillXXX()**' transferência, a porção sa das transferências 'SPI' **no meio do preenchimento** estará ausente do ônibus 'SPI'.
- 2) Durante grandes arquivo transferências através 'SD', um parte das transferências 'SPI' no meio da sequência de bytes será semelhante estar ausente do barramento 'SPI'.
- 3) Diminuição '**Stream**' sobrecarga -usage bytes de modo que '**RAM free**' valor corresponda melhor Arduino compilação.
- 4) UnoArduSim agora avisa o usuário quando um '**class**' tem vários membros declarou em uma linha declaração.
- 5) usando 'File | Save As' agora define o diretório atual que que salvou-no diretório.
- 6) Os dois desaparecidos '**remove()**' membro módulos funcionais foram adicionados à '**String**' classe.
- 7) UnoArduSim agora Bloqueia construtor da chamadas em um construtor-módulo funcional protótipo a menos que a definição completa do corpo módulo funcional segue imediatamente (de modo a concordar com o Arduino compilador).
- 8) Edtempo de transição de GE formas de onda digital foi reduzida para suportar a visualização de sinais de 'SPI' rápidos em maior zoom.
- 9) Un oArduSim agora permite que alguns construtores para ser declarado '**private**' ou '**protected**' (Classe para uso interno).

V2.4 de maio de 2019

- 1) Todos os arquivos de dispositivo 'I / O' agora são salvos no formato traduzido por idioma e, junto com o arquivo **Preferências**, agora estão salvos na codificação de texto UTF-8 para evitar erros de correspondência na

leitura subsequente.

2) Adicionado um novo '**PROGIO**' Dispositivo que é um escravo simples programável 'Uno' placa de circuito que compartilha até 4 pins em comum com o **Painel de Banco de Laboratório** mestre 'Uno', - um escravo 'Uno' pode ter não 'I/O' dispositivos próprio.

3) Agora você pode excluir qualquer 'I/O' dispositivo clicando nele enquanto pressiona a tecla 'Ctrl'.

4) Em **Editar/Examinar** , o preenchimento automático de texto foi adicionado para globais, built-ins e membro variáveis e módulos funcionais (use ALT-seta para a direita para solicitar uma conclusão ou **Entrar** se a lista Built-ins estiver atualmente destacando uma seleção correspondente).

5) Em **Preferências** , uma nova opção permite a inserção automática de um ponto-e-vírgula terminador de linha sobre o **Entrar** pressionar a tecla (se a linha atual for uma instrução executável que pareça autocontida e completa).

6) Pressionando '**Ctrl-S**' a partir de um **Forma de onda** janela permite salvar em um arquivo todos (**X, Y**) pontos ao longo da seção exibida de cada forma de onda (onde X é microssegundos a partir da esquerda forma de onda ponto, e Y é volts).

7) A 'SFTSER' 'dispositivo agora tem um oculto (opcional) '**inverted**' valor (**aplica-se a TX e RX**) que pode ser acrescentado após o valor da taxa de transmissão no final de sua linha em um **IODevs.txt** arquivo

8) Adicionado o '**SPISettings**' classe, módulos funcionais '**SPI.transfer16()**' , '**SPI.transfer(byte* buf, int count)**' , '**SPI.beginTransaction()**' e '**SPI.endTransaction()**' , assim como '**SPI.usingInterrupt()**' e '**SPI.notUsingInterrupt()**' .

9) Adicionada biblioteca SPI módulos funcionais '**SPI.detachInterrupt()**' junto com uma extensão de biblioteca SPI '**SPI.attachInterrupt(void myISRfunc)**' (em vez da biblioteca real módulo funcional '**SPI.attachInterrupt(void)**' (para evitar a necessidade de reconhecer genéricos '**ISR(int vector)**' declarações módulo funcional interrompidas por vetores de baixo nível).

10) O sistema SPI agora pode ser usado no modo escravo, seja fazendo '**SS**' pin (pin 10) um '**INPUT**' pin e dirigindo-o '**LOW**' depois de '**SPI.begin()**' , ou especificando '**SPI_SLV**' como o opcional '**mode**' parâmetro em '**SPI.begin(int mode=SPI_MSTR)**' (outra extensão UnoArduSim para '**SPI.h**'). Os bytes recebidos podem então ser coletados usando '**rxbyte = SPI.transfer(tx_byte)**' dentro de um módulo funcional sem interrupção SPI ou dentro de um serviço de interrupção de usuário módulo funcional anteriormente '**SPI.attachInterrupt(myISRfunc)**' . No modo escravo '**transfer()**' aguarda até que um byte de dados esteja pronto no SPDR (portanto, ele normalmente bloqueará a espera por uma recepção completa de byte, mas em uma rotina de interrupção **Retorna** imediatamente porque o byte SPI recebido já está lá). Em ambos os casos, '**tx_byte**' é colocado no SPDR, então será recebido pelo mestre SPI anexado em sua próxima '**transfer()**' .

11) O suporte ao modo Escravo foi adicionado à implementação UnoArduSim do 'Wire.h'. Módulo funcional '**begin(uint8_t slave_address)**' está agora disponível, assim como '**onReceive(void*)**' e '**onRequest(void*)**' .

12) '**Wire.end()**' e '**Wire.setClock(freq)**' agora pode ser chamado; o último para definir a frequência SCL com um '**freq**' valor de 100.000 (a frequência SCL padrão de modo padrão) ou 400.000 (modo rápido).

13) 'I2CSLV' dispositivos agora todos respondem ao 0x00 geral-chamada de endereço de barramento, e assim 0x00 não pode mais ser escolhido como um endereço de barramento I2C exclusivo para um desses escravos.

14) Os atrasos modelados do execução de operações básicas de números inteiros e de atribuição e matriz e operações de ponteiro foram reduzidas e 4 microssegundos são adicionados agora para cada operação de ponto flutuante.

V2.3 dez. 2018

- 1) O rastreamento agora foi ativado no **Barra de ferramentas** 'I/O ____ S' **controle deslizante** para contínuo e suave dimensionamento de valores 'I/O' dispositivo que o usuário adicionou o sufixo 'S'.
- 2) Um novo '**LED4**' 'I/O' dispositivo (linha de 4 LEDs acesos **4 números consecutivos pin**) foi adicionado.
- 3) Um novo '**7SEG**' 'I/O' dispositivo (7 segmentos DEL dígito com código hexadecimal em **4 números consecutivos pin** e com baixa atividade **CS** * select input), foi adicionado.
- 4) Um novo '**JUMP**' 'I/O' dispositivo que atua como um jumper de fio entre dois 'Uno' pins foi adicionado. Isso permite um '**OUTPUT**' pin para ser ligado a um '**INPUT**' pin (veja o dispositivo acima para usos possíveis deste novo recurso).
- 5) Um novo '**OWISLV**' 'I/O' dispositivo foi adicionado, e os terceiros '**<OneWire.h>**' biblioteca agora pode ser usado com '**#include**' para que o usuário programas possa testar a interface com um pequeno subconjunto do barramento '1-Wire' dispositivos.
- 6) o **Executar** cardápio **Reinicializar** comando agora está conectada ao **Reinicializar** botão.
- 7) Para mais clareza, quando **Atraso artificial 'loop()'** é selecionado sob o **Opções** menu, um explícito '**delay(1)**' chamada é adicionada à parte inferior do loop dentro '**main()**' - este é agora um atraso real que pode ser interrompido por interrupções do usuário anexadas 'Uno' pins 2 e 3.
- 8) O dreno aberto pin entra em conflito com elétrico, ou CS-selecionado, 'I/O' dispositivos (por exemplo, I2CLV ou SPISLV) são agora declarados **somente quando um verdadeiro conflito ocorre no tempo execução**, em vez de causar um erro imediato quando o dispositivo é conectado pela primeira vez.
- 9) Módulo funcional '**pulseInLong()**' agora é preciso para 4-8 microssegundos para concordar com o Arduino (a precisão anterior era de 250 microssegundos).
- 10) Erros sinalizados durante a inicialização de um variável global agora realçar que variável no **Painel de Códigos**.

V2.2 de junho de 2018

- 1) Em **Salvar** de qualquer um dos **Preferências** caixa de diálogo ou de **Configurar | 'I/O' Dispositivos**, a '**myArduPrefs.txt**' arquivo agora está salvo no diretório do programa atualmente carregado - a cada **Arquivo | Carregar** em seguida, carrega automaticamente o arquivo, junto com seu IODevs arquivo especificado, a partir daquele mesmo diretório programa.
- 2) Módulo funcional '**pulseInLong()**' **estava** faltando, mas agora foi adicionado (depende '**micros()**' para as suas medições).
- 3) Quando um usuário programa faz um '**#include**' de um '***.h**' arquivo, UnoArduSim agora também automaticamente tenta carregar o correspondente '***.c**' arquivo **E se** um correspondente '***.cpp**' arquivo não foi encontrado.
- 4) Inserção automática de um close-chavetas '}' (depois de cada chavetas aberto '{') foi adicionado a **Preferências**.
- 5) Um novo **Opções** escolha de menu agora permite '**interrupts()**' ser chamado de dentro de uma rotina de interrupção do usuário - isto é apenas para fins educacionais, uma vez que o aninhamento de interrupções deve ser evitado na prática.
- 6) Tipo-conversão de ponteiros para um '**int**' O valor agora é suportado (mas uma mensagem pop-up de aviso será exibida).
- 7) O UnoArduSim agora suporta linhas programa rotuladas '**LabelName: count++;**' para conveniência do usuário (mas '**goto**' está parado **não permitido**)
- 8) Os avisos do Execução agora ocorrem quando uma chamada '**tone()**' poderia interferir com o PWM ativo no pins 3 ou 11, quando '**analogWrite()**' interferiria com um Servo já ativo no mesmo pin, quando uma chegada de caractere serial é perdida porque as interrupções estão atualmente desabilitadas, e quando as interrupções viriam tão rápido que o UnoArduSim perderá algumas delas.

V2.1 Mar. 2018

- 1) Exibido **Painel de Variáveis** os valores são agora atualizados a cada 30 milissegundos (e a opção Mínimo ainda pode reduzir ainda mais essa taxa de atualização), mas a **VarAtualizar** opção de menu para não permitir a redução de atualização foi removida.
- 2) Operações que visam apenas uma parte dos bytes de um valor variável (como aqueles feitos através de ponteiros) agora causar a mudança para que o valor variável a ser refletido no **Painel de Variáveis** exibição.

V2.0.1 jan. 2018

- 1) Arduino módulos funcionais não documentado '**exp()**' e '**log()**' agora foram adicionados.
- 2) O 'SERVO' dispositivos agora pode ser feito com rotação contínua (portanto, a largura do pulso controla a velocidade em vez do ângulo).
- 3) Em **Editar/Examinar**, um fechamento chavetas '**}**' agora é adicionado automaticamente quando você digita uma abertura chavetas '**{**' se você selecionou isso **Preferência**.
- 4) Se você clicar no **Editar/Examinar** Barra de título janela '**x**' para sair, você agora tem a chance de abortar se você modificou, mas não salvou, o programa programa exibido.

V2.0 de setembro de 2017

- 1) A implementação foi portada para o QtCreator para que a GUI tenha algumas pequenas diferenças visuais, mas sem diferenças funcionais além de algumas melhorias:
 - a) A linha de status de mensagens na parte inferior do janela Principal e dentro do **Editar/Examinar** caixa de diálogo, foi melhorada e foi adicionado um código de cores realçar.
 - b) O espaço vertical alocado entre o **Painel de Códigos** e **Painel de Variáveis** agora é ajustável por meio de uma barra divisora que pode ser arrastada (mas não visível) na borda compartilhada.
 - c) Os valores 'I/O' dispositivo da caixa de edição agora só são validados quando o usuário move o ponteiro do mouse para fora do dispositivo - isso evita alterações automáticas inoportunas para impor valores legais enquanto o usuário está digitando.
- 2) O UnoArduSim agora suporta vários idiomas via **Configurar | Preferências**. O inglês pode sempre ser selecionado, além do idioma da localidade do usuário (contanto que uma tradução personalizada *.qm arquivo para esse idioma esteja presente na pasta UnoArduSim 'translations').
- 3) O som foi modificado para usar a API de áudio do Qt - isso exigiu muting em certas circunstâncias, a fim de evite desagregação de som irritante e cliques durante os maiores atrasos operacionais no sistema operacional do Windows causados por cliques normais do mouse do usuário - consulte a seção -Sounds para obter mais detalhes nisto.
- 4) Como conveniência do usuário, os espaços em branco agora são usados para representar um valor 0 nas caixas de edição de contagem dispositivo em **Configurar | 'I/O' Dispositivos** (Agora você pode usar a barra de espaço para remover o dispositivos).
- 5) O não dimensionado O qualificador (U) agora é opcional em 'PULSER', 'FUNCGEN' e '1SHOT' dispositivos (é o padrão assumido).
- 6) UnoArduSim agora permite (além dos valores numéricos literais) '**const**' variáveis com valor inteiro e '**enum**'

V1.7.2 - fev. 2017

1) A opção de cor azul (B) foi adicionada para DEL dispositivos.

V1.7.1 - fev. 2017

1) Sufixos 'L' e / ou 'U' agora são aceitos no final das constantes literais numéricas (para defini-las como 'long' e / ou 'unsigned') e ('0b' ou '0B' prefixado) As constantes binário agora também são aceitas. Qualquer constante numérica decimal **começando com um '0'** agora é considerado um **octal** valor. (para concordar com o Arduino).

2) Ao executar em um loop apertado do qual não há escape (por exemplo '**while(x) ; x++ ;**' Onde x é sempre verdade), clicando em **Parar** uma segunda vez garante que o programa execução realmente pára (e nessa linha defeituosa do programa).

V1.7.0 - dez. 2016

1) Um novo **Barra de ferramentas** foi adicionado um recurso que mostra bytes de RAM livre durante programa execução (representando o total de bytes usados pelo variáveis global, alocações de heap e pilha local variáveis).

2) Interrupção do usuário módulos funcionais pode agora também se chamam bloqueando Arduino módulos funcionais como '**pulseIn()**' (mas isso só deve ser usado com cautela, pois a interrupção módulo funcional não retornará até que o bloqueio módulo funcional esteja completo).

3) As interrupções do usuário não são mais desativadas durante operações de leitura de fluxo bloqueadas, portanto, o comportamento agora corresponde à operação real de leitura de fluxo do Arduino.

4) Agora você pode entrar e sair do bloqueio do Arduino módulos funcionais que pode ser interrompido (como '**delay()**' e '**pulseIn()**'), e as mensagens da Barra de Status foram aumentadas para mostrar quando você acertou uma interrupção ponto de parada dentro de um módulo funcional (ou quando você clica em Parar quando o execução está atualmente dentro de um módulo funcional).

5) Um novo **Executar Até** comando (e **Barra de ferramentas** item) foi adicionado - clique único em qualquer **Painel de Variáveis** variável (pode ser simples, um agregado matriz ou objeto, ou um elemento matriz ou membro objeto) para realçar, então faça **Executar Até** - execução irá congelar na próxima **acesso de escrita** dentro daquele agregado variável, ou para aquele único local.

6) Quando o execução congela após um **Passo, Executar Para, Executar Até** ou **Executar-então-Parar** ação, o **Painel de Variáveis** agora destaca o variável que corresponde ao **endereço local (s) que foi modificado** (se houver) pelo **o último instrução durante esse execução** - se esse local estiver oculto dentro de um matriz matriz ou objeto, ao clicar no expandir, ele fará com que o elemento ou membro da última modificação seja destacado.

7) O usuário pode agora manter uma observação especial sobre o valor de um **Variável Painel** variável / membro / elemento durante a execução - clique duas vezes nessa linha no **Painel de Variáveis** para abrir o **Editar/Monitorar Variável Valor** janela, então faça um dos **Executar** ou **Passo** comandos - o valor mostrado será atualizado durante o execução de acordo com as mesmas regras que regem as atualizações no **Painel de Variáveis**. Depois de parar o execução, você pode inserir um novo valor e **Aceitar** antes de retomar o execução (e pode **Reverter** para o **Aceitar** valor se você mudar de idéia antes disso).

8) Teclas do acelerador F4-F10 foram ajustadas para coincidir com o menu Executar **Barra de ferramentas** comandos (da esquerda para a direita).

9) Além de clicar duas vezes sobre eles, clique com o botão direito do mouse em 'SERIAL', 'SFTSER', 'SPISLV', 'I2CSLV' dispositivos agora também irá aparecer um tamanho maior TX / RX bytes / caracteres janela (e em 'SD_DRV', um arquivos-monitoramento janela).

10) A caixa de edição TX em 'SERIAL' ou 'SFTSER' não está mais desativada durante uma transmissão de caractere ativo (portanto, você pode agora anexar ou substituir o que está lá), mas um retorno de carro (ou o

botão 'Send' no filho associado; janela) será ignorada até que a transmissão retorne ao estado inativo mais uma vez (os caracteres agora são mostrados em itálico quando a transmissão está pronta para começar, ela está ativa). Além disso, o usuário agora é avisado em um fluxo serial '**begin()**' se eles já tivessem começado mais cedo o dispositivo anexado (agora em andamento), pois não haveria sincronização de quadros, levando a erros de recepção.

11) O padrão adicionado '**loop()**' o atraso foi aumentado de 250 microssegundos para um milissegundo para não cair muito para trás em tempo real, quando o usuário deixa de incluir alguns '**delay()**' (explícito ou natural) em algum lugar dentro '**loop()**' ou dentro de um módulo funcional que ele chama.

12) Matrizes e tipos simples foram agora adicionados ao suporte para a alocação de heap '**new**' instrução.

13) Verificações mais extensas (e mensagens de erro associadas) foram adicionadas para acessos de endereços fora do limite do usuário programa (ou seja, fora da 'Uno' RAM, ou fora do 'Flash' para '**PROGMEM**' acessos).

14) Valores de ponteiro no **Painel de Variáveis** agora mais se assemelham aos valores reais do ponteiro Arduino.

15) O usuário '**myArduPrefs.txt**' arquivo está agora carregado em cada **Arquivo | Carregar**, não apenas no lançamento do UnoArduSim.

16) Um erro Analisar agora é sinalizado ao tentar '**attachInterrupt()**' para um usuário módulo funcional que não é '**void**' retornar, ou que tem parâmetros módulo funcional, ou que não foi declarado em algum lugar antes '**attachInterrupt()**'.

17) '**static**' membro-variáveis agora são exibidos no topo da **Painel de Variáveis** como globais, em vez de aparecer dentro de cada instância de um objeto (expandidos).

18) Módulo funcional '**availableForWrite()**' foi adicionado à implementação de '**Serial**'.

19) Tudo especial '**PROGMEM**', '**typedef**' gostar '**prog_char**' e '**prog_int16**' agora foram removidos (eles foram preteridos no Arduino).

20) Mensagens de erro aprimoradas para erros do Analisar causados por tipos de declaração incorretos ou inválidos.

21) O tamanho máximo permitido de programa foi aumentado.

V1.6.3 - set. 2016

1) Adicionado um melhorada mensagem de erro analisar quando '**attachInterrupt()**' refere-se a uma interrupção módulo funcional que não foi **prototipado anteriormente**.

2) Adicionada uma mensagem de erro Analisar aprimorada para listas de inicialização matriz multidimensionais.

V1.6.2 - set. 2016

1) Adicionado um **Buscar-Text** editar o controle para o **Barra de ferramentas** para simplificar a pesquisa de texto (no **Painel de Códigos** e **Painel de Variáveis**).

2) o **Barra de ferramentas** O botão Reinicializar agora funciona de forma idêntica ao botão LK1041 placa de circuito Reinicializar.

V1.6.1 - ago. 2016

Adicionada uma verificação para evitar o carregamento e a análise duplicados de versões anteriores '**#include**' arquivos, .

V1.6 - Jun. 2016

1) Adicionado um novo '1SHOT' (one-shot) 'I/O' Dispositivo que gera um pulso após um atraso escolhido a partir

de uma borda de sinal de disparo da polaridade selecionada.

2) Adicionado um novo recurso torna os valores de caixa de edição 'I/O' dispositivo facilmente ***dimensionado*** durante o execução arrastando um controle deslizante de Escala 'I/O____S' global no **Barra de ferramentas** (apenas digite uma única letra 's' ou 'S' depois de um valor para indicar a escala).

V1.5.1 - Jun. 2016

1) Suporte foi agora adicionado para a biblioteca EEPROM módulos funcionais '**update()**', '**put()**' e '**get()**', e para acesso de bytes via notação matriz, por exemplo '**EEPROM[k]**'.

2) **Permitir Auto (-) Contrair** foi adicionado ao menu **VarAtualizar** para permitir o controle explícito sobre se o expandidos matrizes / objetos será contraído automaticamente quando o execução estiver atrasado em tempo real.

3) Os personagens de um '**String**' variável agora também pode ser acessado via notação matriz, por exemplo '**mystring[k]**'.

V1.5 - maio de 2016

1) **Editar/Examinar** agora tem o atalho ctrl-E, e tem um novo botão para **Compilar** (ctrl-R), mais uma caixa de erro construídas-em Analisar, para permitir o teste de edições sem precisar fechar a janela.

2) **Editar/Examinar** agora também suporta **Refazer** tem um novo **Salvar** (ctrl-S) (equivalente a **Aceitar** mais um mais tarde janela **Salvar**), e agora dá uma escolha de '**Tab**' tamanho (uma nova preferência que pode ser salva usando **Configurar | Preferências**).

3) Todas as caixas de edição graváveis agora seguem as cores de tema do Janelas OS escolhidas e, para co, todas as caixas de edição 'RECV' somente leitura usam texto branco em fundo preto. o **Editar/Examinar** As cores background e syntax-realçar agora também se adaptam ao tema escolhido.

4) O UnoArduSim agora permite uma escolha de fonte - essa escolha, e seu tamanho, foram movidos para o **Configurar | Preferências** (assim pode ser salvo no '**myArduPrefs.txt**' arquivo).

5) Arduino pré-definidos valores literais binário (como '**B01011011**') agora são permitidos.

6) Escapeado hex, octal e 4-dígito Unicode citados seqüências de caracteres agora podem ser usados como literais numéricos.

7) Depois de fazer um clique inicial com o mouse em um pad 'PUSH' dispositivo, o usuário pode usar um pressionamento de tecla (qualquer tecla) para pressionar os contatos do botão de pressão.

8) **Editar/Examinar** agora libera seu estado inicial somente leitura temporário (e remove o realce da linha inicial selecionada) após uma breve indicação visual em flash.

9) UnoArduSim agora verifica múltiplas '**Stepper**' e '**Servo**' pin conflitos, ou seja, o usuário defeituoso programa tenta se conectar ao pins já anexado a anteriormente '**Stepper**' ou '**Servo**' variáveis

10) Um erro Analisar causado por um lado esquerdo ou direito ausente em um operador (sem uma expressão LHS ou RHS ou variável) agora gera uma mensagem de erro clara.

11) O não utilizado '**String**' classe '**flags**' O membro variável foi removido para concordar com o Arduino V1.6.6. UMA '**String**' objeto agora ocupa 6 bytes (mais sua alocação de heap de caracteres).

V1.4.2 - mar. 2016

1) O módulos funcionais definido para a frente (ou seja, aqueles sem declaração protótipo antes de sua primeira chamada) agora gera apenas avisos (não erros analisar) quando a definição módulo funcional mais recente retorna o tipo inferido de seu primeiro uso.

2) Matrizes com uma dimensão igual a 1 não é mais rejeitado (para concordar com as regras padrão do C ++).

- 3) As caixas de edição não estão mais definidas para preto no plano de fundo branco - elas agora adotam a paleta definida pelo tema Janelas OS em uso.
- 4) 'SERIAL', 'SFTSER', 'SPISLV' e 'I2CSLV' dispositivo Monitor expandidos janelas (aberto com um clique duplo) agora adota a cor de fundo de seu pai 'I/O' Dispositivo.

V1.4 - dez. 2015

- 1) 'Stepper.h' Funcionalidade de biblioteca e associados 'I/O' dispositivos foram adicionados agora.
- 2) **Todas as configurações e valores do 'I/O' Dispositivo** (além de seu pins selecionado) agora também são salvos como parte do usuário escolhido 'I/O' Dispositivos texto arquivo para mais tarde recarregar.
- 3) **DEL** A cor do 'I/O' dispositivo agora pode ser definida como vermelha, amarela ou verde usando uma caixa de edição no dispositivo.
- 4) Os inicializadores de declaração Variável agora têm permissão para abranger várias linhas.
- 5) Os índices Matriz agora podem ser eles mesmos elementos matriz.
- 6) **Configurar | Preferências** agora inclui uma caixa de seleção para permitir 'and', 'or', 'not' Palavras-chave a utilizar em vez do padrão C '&&', '||' e '!' Operadores lógicos.
- 7) "Show Programa Descarregando" foi movido para **Configurar | Preferências**

V1.3 - outubro de 2015

- 1) o '**PUSH**' dispositivo agora tem uma caixa de seleção "push-like" rotulada 'latch' para torná-los "travados" (em vez de "momentâneos"), ou seja, eles trancam na posição fechada (e mudam de cor) quando pressionados, até serem pressionados novamente para liberar os contatos.
- 2) A capacidade total 'SPISLV' dispositivos foi adicionada com seleção de nó ('MODE0', 'MODE1', 'MODE2' ou 'MODE3'). Clicar duas vezes abre um janela de buffers TX / RX onde os próximos bytes REPLY (TX) podem ser definidos e para visualizar bytes passados recebidos (RX). O escravo de registro de deslocamento simples dispositivo da versão anterior foi renomeado para se tornar um 'SRSLV' dispositivo.
- 3) **Negrito** tipo de letra agora pode ser escolhido para o **Painel de Códigos** e **Painel de Variáveis** (do menu **Opções**) e **negrito destaque de palavras-chave e operadores** agora pode ser ligado / desligado em **Editar/Examinar**.
- 4) UnoArduSim agora permite 'bool' como sinônimo de 'boolean'.
- 5) Para maior clareza no relatório de erros, as declarações variável não podem mais abranger múltiplas linhas (exceto pelo matrizes com listas de inicialização).
- 6) Sintaxe velocidade de coloração em **Editar/Examinar** foi melhorado (isso será perceptível com programas maior).
- 7) Um overhead opcional de 200 microssegundos (no menu **Opções**) foi adicionado a cada chamada de 'loop()' - isto é para tentar evitar cair muito para trás em tempo real no caso em que o usuário programa não adicionou 'delay()' em qualquer lugar (veja discussão Cronometragens abaixo).

V1.2 de junho de 2015

- 1) A biblioteca SD está agora totalmente implementada e um (pequeno) disco SD 8Mbyte 'I/O' dispositivo ('SD_DRV') foi adicionado (e a funcionalidade testada em todos os exemplos de Arduino SD programas).
- 2) Assim como o Arduino, o UnoArduSim agora converterá automaticamente um argumento módulo funcional em seu endereço ao chamar um módulo funcional esperando que um ponteiro seja passado.

3) Mensagens de erro Analisar agora são mais apropriadas quando faltam ponto e vírgula e após declarações não reconhecidas.

4) Obsoleto **Painel de Variáveis** Destaques da linha agora são removidos na chamada / retorno módulo funcional.

V1.1 - mar. 2015

1) O janela principal pode agora ser maximizado ou redimensionado para **Painel de Códigos** e **Painel de Variáveis** mais amplo (para telas maiores).

2) Um novo menu Buscar (com **Botões da barra de ferramentas**) foram adicionados para permitir uma navegação mais rápida **Painel de Códigos** e **Painel de Variáveis** (PgUp e PgDown, ou pesquisa de texto com seta para cima, seta para baixo).

3) o **Editar/Examinar** O janela agora permite saltos de navegação ctrl-PgUp e ctrl-PgDn (para a próxima linha vazia), e aumentou **Buscar / Substituir** funcionalidade.

4) UMA novo item foi adicionado ao menu **VarAtualizar** para permitir que o usuário selecione uma abordagem de economia de computação sob **Painel de Variáveis** atualizar cargas.

5) O 'Uno' pins e o DEL anexado agora refletem as alterações feitas no 'I/O' dispositivos mesmo quando o tempo é congelado (ou seja, mesmo quando o execução é interrompido).

6) Outro usuário módulos funcionais agora pode ser chamado de dentro de uma interrupção do usuário módulo funcional (de acordo com a atualização para o Arduino 1.06).

7) UMA **fonte maior** agora pode ser escolhido no menu **Opções**.

V1.0.1 - jun. 2014

Forma de onda janelas agora rotular analógico pins como A0-A5 em vez de 14-19.

V1.0 - primeiro lançamento maio 2014