

UnoArduSimV2.1 Ayuda Completa

Tabla de Contenido

Visión de Conjunto

Panel de Código, Preferencias y Editar / Examinar

Panel de Código

Preferencias

Editar / Examinar

Panel de Variables y ventana Editar / Monitorear Variable

Panel del Banco de Laboratorio

El 'Uno'

Dispositivos 'I / O'

'Serial' Monitor ('SERIAL')

Software de serie ('SFTSER')

Unidad de disco SD ('SD_DRV')

Esclavo Registro de Desplazamiento ('SRSLV')

Esclavo SPI configurable ('SPISLV')

Esclavo I2C ('I2CSLV')

Motor Paso a Paso ('STEPR')

Motor de CC ('MOTOR')

ServoMotor ('SERVO')

Generador de Pulso Digital ('PULSER')

Un-Trago ('1SHOT')

Generador de funciones analógicas ('FUNCGEN')

Altavoz piezoeléctrico ('PIEZO')

Pulsador ('PUSH')

Resistencia del interruptor deslizante ('R = 1K')

LED de color ('LED')

Control Deslizante Analógico

Menús

Archivo:

Cargar INO o PDE Prog (ctrl-L)

Editar / Examinar (ctrl-E)

Guardar

Guardar Como

Siguiente ('#include')

Anterior

Salida

Encontrar:

Encontrar Función / Var Siguiente

Encontrar Función Anterior / Var

Establecer tTexto de Búsqueda ctrl-F)

Buscar texto Siguiente

Buscar texto Anterior

Ejecutar:

Paso En (F4)

Paso Sobre (F5)

Paso Afuera (F6)

Ejecutar Hacia (F7)

Ejecutar Hasta (F8)

Ejecutar (F9)

Detener (F10)

Reiniciar

[Animar](#)
[Camara lenta](#)

Opciones:

[Paso por encima de Structors / Operadores](#)
[Modelo de Asignación de Registro](#)
[Error en Sin inicializar](#)
[Se agregó 'loop\(\)' Delay](#)

Configurar:

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

VarActualizar:

[Permitir colapso automático\(-\)](#)
[Mínimo](#)
[Resaltar los Cambios](#)

Ventanas:

['Serial' Monitor](#)
[Restaura todo](#)
[Pin Formes de Onda Digital](#)
[Forma de Onda Analógico del Pin](#)

Ayuda:

[Archivo de Ayuda Rápida](#)
[Archivo de ayuda completa](#)
[Corrección de errores](#)
[Cambio / Mejoras](#)
[Acerca de](#)

Modelado

['Uno' Placa y 'I / O' Devices](#)
[Sincronización](#)
[Sincronización del Dispositivo 'I / O'](#)
[Sonidos](#)

Limitaciones y elementos no admitidos

[Archivos incluidos](#)
[Asignaciones dinámicas de memoria y RAM](#)
[Asignaciones de Memoria 'Flash'](#)
['String' Variables](#)
[Bibliotecas Arduino](#)
[Punteros](#)
[Objetos 'class' y 'struct'](#)
[Alcance](#)
[Calificadores 'unsigned' , 'const' , 'volatile' , 'static'](#)
[Directivas del Compilador](#)
[Elementos de Lenguaje Arduino](#)
[C / C ++ - Elementos de Lenguaje](#)
[Plantillas de funciones](#)
[Emulación en Tiempo Real](#)

Notas de lanzamiento

[Corrección de Errores](#)
[V2.1-Mar. 2018](#)
[V2.0.2-Feb. 2018](#)
[V2.0.1-Enero 2018](#)
[V2.0-Dic. 2017](#)
[V1.7.2 - Feb.2017](#)
[V1.7.1 - Feb.2017](#)
[V1.7.0 - Dic.2016](#)
[V1.6.3- Sept. 2016](#)
[V1.6.2- Sept. 2016](#)
[V1.6.1 - agosto de 2016](#)
[V1.6- junio de 2016](#)
[V1.5.1 - junio de 2016](#)
[V1.5 - mayo de 2016](#)

[V1.4.3 - abril de 2016](#)
[V1.4.2 - Mar. 2016](#)
[V1.4.1 - enero de 2016](#)
[V1.4 - Dic. 2015](#)
[V1.3 - Oct. 2015](#)
[V1.2 - Jun 2015](#)
[V1.1 - Mar 2015](#)
[V1.0.2 - agosto de 2014](#)
[V1.0.1 - junio de 2014](#)

Cambios / Mejoras

[V2.1 Mar. 2018](#)
[V2.0.1 Enero 2018](#)
[V2.0 Dic. 2017](#)
[V1.7.2 - Feb. 2017](#)
[V1.7.1 - Feb. 2017](#)
[V1.7.0 - Dic. 2016](#)
[V1.6.3- septiembre de 2016](#)
[V1.6.2 - septiembre de 2016](#)
[V1.6.1 - Ago de 2016](#)
[V1.6 - junio de 2016](#)
[V1.5.1 - junio de 2016](#)
[V1.5 - mayo de 2016](#)
[V1.4.2 - Mar 2016](#)
[V1.4 - Dic 2015](#)
[V1.3 - Oct 2015](#)
[Versión 1.2 Junio 2015](#)
[V1.1 - Mar 2015](#)
[V1.0.1 - junio de 2014](#)

Visión de Conjunto

UnoArduSim es una herramienta de simulación gratuita **en tiempo real** (ver Modelado para **restricciones de tiempo**) que he desarrollado para el estudiante y el entusiasta de Arduino. Está diseñado para permitirle experimentar y depurar fácilmente programas Arduino **sin la necesidad de ningún hardware real** . Está dirigido a la **placa Arduino 'Uno'** , y le permite elegir entre un conjunto de dispositivos virtuales 'I / O', y configurar y conectar estos dispositivos a su virtual 'Uno' en el **Panel del Banco de Laboratorio** . - No necesita preocuparse por errores de cableado, conexiones rotas / sueltas o dispositivos defectuosos que estropeen el desarrollo y las pruebas de su programa.

UnoArduSim proporciona mensajes de error simples para cualquier error de análisis o ejecución que encuentre, y permite la depuración con **Reiniciar, Ejecutar, Ejecutar-Hacia, Ejecutar-Hasta , Detener** y operaciones de **Paso** flexibles en el **Panel de Código** , con una vista simultánea de todas las variables locales, matrices y objetos globales y actualmente activos en el **Panel de Variables**. Se proporciona comprobación de límites de matriz en tiempo de ejecución y se detectará el desbordamiento de ATmega RAM (¡y se resaltará la línea de programa culpable!). Cualquier conflicto eléctrico con los dispositivos 'I / O' conectados se marcan e informan a medida que ocurren.

Cuando se abre un archivo de programa INO o PDE, se carga en el **Panel de Código** del programa . El programa se analiza y se "compila" en un ejecutable tokenizado que luego está listo para la **ejecución simulada** (a diferencia de Arduino.exe, *no se crea un ejecutable binario independiente*) Se detecta y marca cualquier error de análisis al resaltar la línea que no pudo analizar e informar el error en la **Barra de Estado** en la parte inferior de la ventana de la aplicación UnoArduSim. Se **puede abrir una ventana** Editar / Examinar **para que pueda ver y editar una versión resaltada por sintaxis de su programa de usuario**. Los errores durante la ejecución simulada (como tasas de baudios no coincidentes) se informan en la **Barra de Estado** y en un cuadro de mensaje emergente.

UnoArduSim V2.0 es una implementación sustancialmente completa de **Arduino Programming Language V1.6.6 como se documentó en el arduino.cc** . Página web de referencia del lenguaje y con las adiciones que se indican en la versión Notas de la versión de la página de descarga. Aunque UnoArduSim no admite la implementación completa de C ++ que hace el compilador GNU subyacente de Arduino.exe, es probable que solo los programadores más avanzados encuentren que falta algún elemento C / C ++ que deseen usar (y, por supuesto, siempre son simples). codificación de soluciones alternativas para tales características faltantes). En general, he admitido solo las características C / C ++ más útiles para los aficionados y estudiantes de Arduino; por ejemplo, 'enum' y '#define' son compatibles, pero los punteros a función no lo son. Aunque los objetos definidos por el usuario ('class' y 'struct') y (la mayoría) de las sobrecargas del operador son compatibles, *la herencia múltiple no lo es* .

Debido UnoArduSim es un simulador de nivel de lengua alta, **sólo el C / C ++ declaraciones son compatibles, instrucciones de lenguaje de ensamblaje no lo son. Del mismo modo, como no es una simulación de máquina de bajo nivel, los registros de ATmega328 no son accesibles para su** lectura o escritura, aunque se emulan la asignación de registros, los pases y los retornos (esto lo elige en el menú **Opciones**).

A partir de la versión 2.0, UnoArduSim tiene soporte automático incorporado para un subconjunto limitado de las bibliotecas proporcionadas por Arduino, estas son: 'Stepper.h' , 'SD.h' , 'Servo.h' , 'SoftwareSerial.h' , 'SPI.h' , 'Wire.h' y 'EEPROM.h' (versión 2). Para cualquier '#include' de bibliotecas creadas por el usuario, UnoArduSim **no** buscará en la estructura habitual del directorio de instalación de Arduino para ubicar la biblioteca; en su lugar, **debe** copiar el encabezado correspondiente (".h") y el archivo de origen (".cpp") en el mismo directorio que el archivo de programa en el que está trabajando (sujeto, por supuesto, a la limitación de que el contenido de cualquier '#include' archivo debe ser completamente comprensible para el analizador UnoArduSim).

Desarrollé UnoArduSimV2.0 en QtCreator con soporte multilingüe, y actualmente solo está disponible para Windows [™] . La migración a Linux o MacOS es un proyecto para el futuro. UnoArduSim surgió a partir de los simuladores que había desarrollado a lo largo de los años para los cursos que impartía en la Universidad de Queen, y ha sido probado razonablemente extensamente, pero es probable que haya algunos errores

escondidos allí. Si desea informar un error, por favor describa (brevemente) en un correo electrónico a unoArduSim@gmail.com y **asegúrese de adjuntar el código fuente completo de Arduino del programa inductor de errores** para que pueda replicar el error y solucionarlo. No responderé a los informes de errores individuales, y no tengo plazos garantizados para las correcciones en una versión posterior (¡recuerde que casi siempre hay soluciones!).

Aclamaciones,

Stan Simmons, Ph. D, P. Eng.
Profesor Asociado (retirado)
Departamento de Ingeniería Eléctrica e Informática
Queen's University
Kingston, Ontario, Canadá



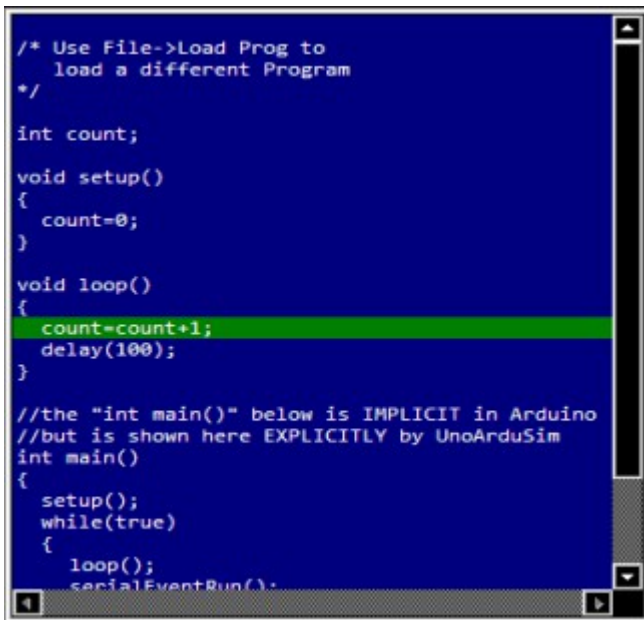
Panel de Código, Preferencias y Editar / Examinar

(Aparte: las ventanas de muestra que se muestran a continuación están todas bajo un tema de color Windows-OS elegido por el usuario que tiene un color de fondo de ventana azul oscuro).

Panel de Código

El **Panel de Código** muestra su programa de usuario y resalta su ejecución.

Después de que un programa cargado se analiza con éxito, la primera línea de '**main()**' está resaltado, y el programa está listo para su ejecución. Tenga en cuenta que '**main()**' es agregado implícitamente por Arduino (y por UnoArduSim) y no lo incluye como parte de su archivo de programa de usuario. La ejecución está bajo el control del menú **Ejecutar**, y sus botones asociados **de la Barra de Herramientas** y los accesos directos de la tecla de función.







```
/* 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();
  }
}
```






Después de ejecutar la ejecución por una (o más) instrucciones (puede usar **los botones de la Barra de Herramientas**) , , , o ), la línea del programa que se ejecutará a continuación se resaltará: la línea resaltada siempre es la siguiente línea **lista para ejecutarse**.

Del mismo modo, cuando un programa que se ejecuta realiza un punto de interrupción (temporal **Ejecutar-Hacia**), la ejecución se detiene y se pone de relieve la línea de punto de interrupción (y es entonces listo para su ejecución).

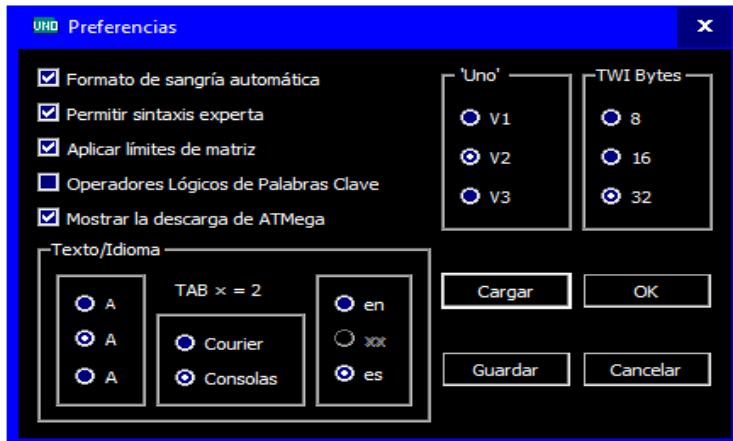
Si la ejecución del programa se detiene actualmente y hace clic en la **ventana del Panel de Código**, la línea que acaba de hacer se resalta. Esto *no* significa, sin embargo, cambiar la línea de programa actual en lo que se refiere a la ejecución del programa. Pero puede hacer que la ejecución *progrese hasta* la línea que acaba de resaltar haciendo clic en **Ejecutar-Hacia** .

botón de la Barra de Herramientas. Esta función le permite acceder rápida y fácilmente a líneas específicas en un programa, de modo que pueda avanzar línea por línea sobre una porción de programa que le interese.

Si su programa cargado tiene archivos '**#include**', puede moverse entre ellos utilizando **Archivo | Anterior** y **Archivo | Siguiente** (con Botones de la **Barra de Herramientas**  y ).

El menú **Encontrar** le permite **saltar** rápidamente entre las **funciones** en el **Panel de Códigos** (después de hacer clic en una línea dentro de él para enfocararlo) usando los comandos **Siguiente** y **Anterior** (con **los botones de la Barra de Herramientas**)  y  o atajos de teclado **AvPág** y **RePág**). **Alternativamente, puede encontrar el texto** especificado con sus comandos de texto (con **los botones de la Barra de Herramientas**)  y , o atajos de teclado **arriba-flecha** y **flecha abajo**), después de usar **Encontrar | Establecer el Texto de Búsqueda** o el botón de la **Barra de Herramientas** .

Preferencias

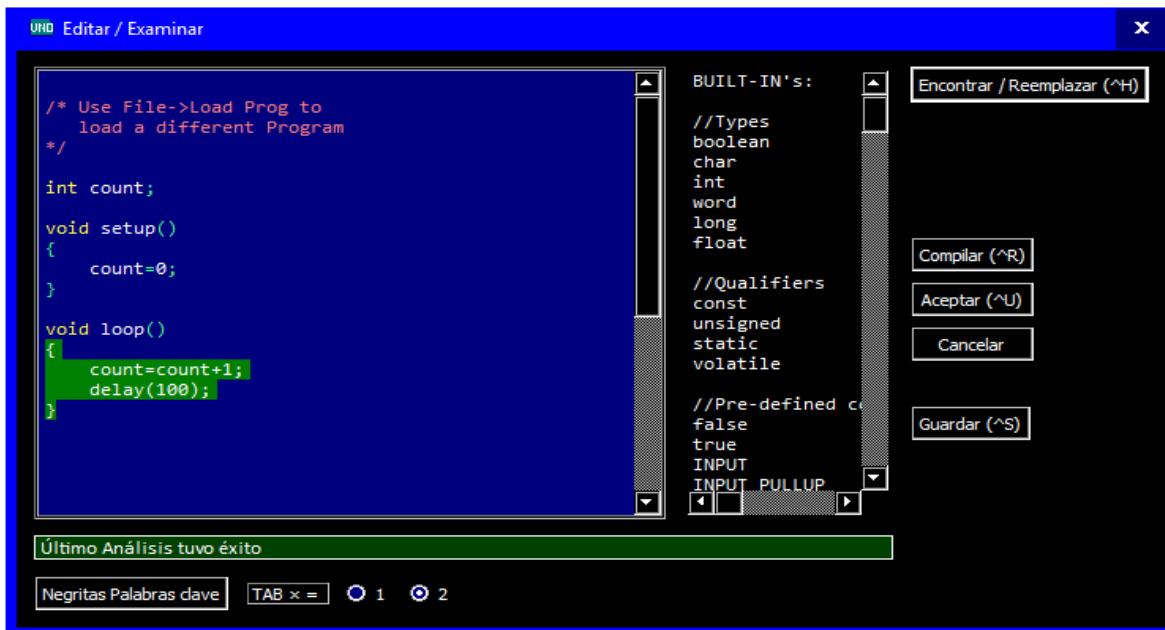


Configurar | Preferencias permiten a los usuarios para configurar el programa y las preferencias de visualización (que un usuario normalmente deseará adoptar en la próxima sesión). Por lo tanto, pueden guardarse y cargarse desde un archivo **myArduPrefs.txt** que reside en el mismo directorio que el programa 'Uno' cargado (**myArduPrefs.txt** se carga automáticamente si existe).

Este diálogo permite elegir entre dos fuentes monoespaciadas y tres tamaños de letra, y otras preferencias misceláneas. A partir de V2.0, el idioma chpice ahora está incluido. - esto siempre incluye English (**en**), más uno o dos lenguajes locales de usuario (donde

existan), y una anulación basada en el código de idioma ISO-639 de dos letras en la primera línea del archivo **myArduPrefs.txt** (si uno se proporciona allí). Las opciones solo aparecen si existe un archivo de traducción ".qm" en la carpeta de traducciones (dentro del directorio de inicio de UnoArduSim.exe).

Editar / Examinar



Al hacer doble clic en cualquier línea en el **Panel de Código** (o al utilizar el menú **Archivo**), se abre una ventana **EEditar / Examinar** para permitir cambios en su archivo de programa, con la **línea seleccionada actualmente** en el **Panel de Código** resaltada.

Esta ventana tiene capacidad de edición completa con resaltado de sintaxis dinámica (diferentes colores de resaltado se usan para palabras clave de C ++, comentarios, etc.). Hay un resaltado de sintaxis en negrita opcional y un formato de nivel de sangría automático (suponiendo que haya seleccionado eso usando **Configurar | Preferencias**). También puede seleccionar cómodamente llamadas de función incorporadas (o integradas '#define' constants) para agregarlas a su programa desde el cuadro de lista proporcionado; simplemente haga doble clic en el elemento deseado del cuadro de lista para agregarlo a su programa en la

posición actual de intercalación (los **tipos de** variable de llamada a función son solo para información y se eliminan para colocar marcadores de posición ficticios cuando se agregan a su programa).

La ventana tiene **Encontrar** (use **ctrl-F**) y capacidad **Encontrar / Reemplazar** (use **ctrl-H**) . La **ventana** Editar / Examinar **tiene** botones Deshacer (**ctrl-Z**) y Rehacer (**ctrl-Y**) (**que aparecen automáticamente**).

Para descartar **todos los cambios** que realizó desde que abrió el programa para editar, haga clic en el botón **Cancelar** . Para aceptar el estado actual, haga clic en el botón **Aceptar** y el programa se analizará automáticamente de nuevo (y se descargará al 'Uno' si no se encuentran errores) y el nuevo estado aparecerá en la ventana principal de UnoArduSim, **la Barra de Estado** .

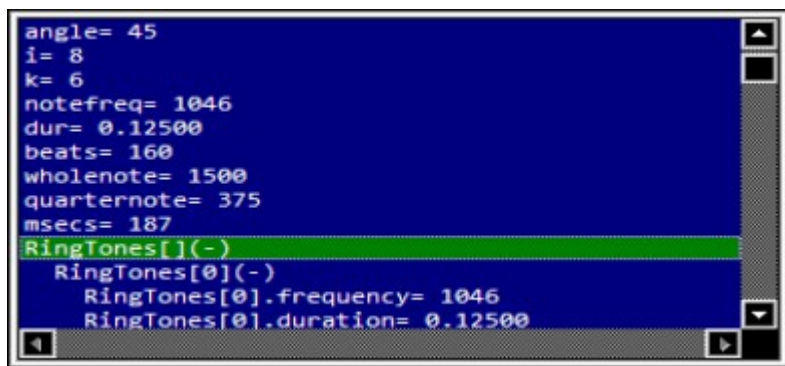
Se ha agregado un botón **Compilar** (**ctrl-R**) (más un cuadro de mensaje de **Estado del Análisis** asociado como se ve en la imagen de arriba) para permitir la prueba de ediciones sin necesidad de cerrar primero la ventana. Un botón **Guardar** (**Ctrl-S**) también se ha añadido como un acceso directo (equivalente a un **Aceptar** además de un más tarde **Guardar** separado de la ventana principal).

En **Cancelar** o **Aceptar** sin realizar modificaciones, la línea actual del Panel de Código cambia para convertirse en la **última posición Ver / Editar intercalación** , y puede usar esa función para pasar el Panel de Códigos a una línea específica (posiblemente para prepararse para **ejecutar una Ejecución**). **Para** , también puede usar **ctrl-PgDn** y **ctrl-PgUp** para saltar al salto de línea vacío siguiente (o anterior) en su programa; esto es útil para navegar rápidamente hacia arriba o hacia abajo a ubicaciones significativas (como líneas vacías entre funciones) . También puede usar **ctrl-Home** y **ctrl-End** para saltar al inicio y al final del programa, respectivamente.

El formato de sangría automática a nivel de tabulador se realiza cuando se abre la ventana, si esa opción se configuró en **Configurar | Preferencias**. También puede agregar o eliminar pestañas a un grupo de líneas consecutivas preseleccionadas utilizando las teclas *de flecha derecha* o *flecha izquierda* del teclado, pero la **sangría automática debe estar desactivada** para evitar perder sus propios niveles de pestañas.




Y para ayudar a mantener un mejor seguimiento de sus contextos y los corsés, al hacer clic en un ' { ' o ' } ' destacados corsé todo el texto entre ese aparato ortopédico y su socio de juego.



Panel de Variables y ventana Editar / Monitorear Variable



Las **Panel de Variables** se encuentra justo debajo del **Panel de Código**. Muestra los valores actuales para cada variable global / activa (en el ámbito) local / matrices / objetos en el programa cargado. A medida que la ejecución de su programa se mueve entre funciones, Los contenidos cambian para reflejar solo aquellas variables locales accesibles para la función / alcance actual, además de las variables globales declaradas por el usuario . Las variables declaradas como 'const' o como

' **PROGMEM** ' (asignadas a 'Flash' memory) tienen valores que no pueden modificarse y, por *lo* tanto, para ahorrar espacio, *no se muestran* . Las instancias 'Servo' y 'SoftwareSerial' objeto no contienen valores útiles, por lo que tampoco se muestran.

Los comandos en el menú **Encontrar** (con botones de la **Barra de Herramientas**)  y  , O atajos de teclado **AvPág** y **RePág**) le permiten **saltar** rápidamente entre **las variables** en las **variables panel** (después de la primera clic sobre una línea en su interior para darle el foco). Alternativamente, puede **encontrar el texto** especificado con sus comandos de búsqueda de texto (con **los** botones de la **Barra de Herramientas**)  y

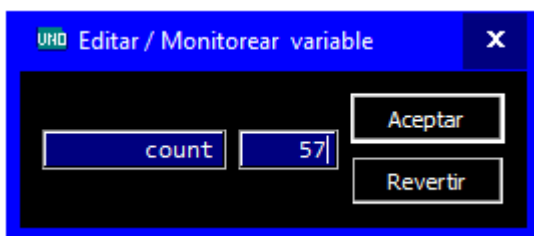
 , o atajos de teclado **arriba-flecha** y **flecha abajo**), después de usar **Encontrar | Establecer Texto de Búsqueda**  .

Las matrices y los **objetos** se muestran en formato **no expandido** o **expandido** , con un signo más (+) o menos (-) posterior , respectivamente. El símbolo de una matriz **x** muestra como **x[]** . Para expandirlo y mostrar todos los elementos de la matriz, simplemente haga clic en **x[] (+)** en el Panel de Variables. Para contraer de nuevo a una vista no expandida, haga clic en **x[] (-)** . El valor predeterminado sin expandir para un objeto **p1** muestra como **p1 (+)** . Para expandirlo y mostrar todos los miembros de la **instancia 'class'** o **'struct'** , haga clic en **p1 (+)** . en el **Panel de Variables**. Para contraer de nuevo a una vista no expandida, haga un solo clic en **p1 (-)** .

Si hace **clik solo en cualquier línea para resaltarla** (puede ser una variable simple, o la línea agregada (+) o (-) de una matriz u objeto, o un único elemento de matriz o miembro de objeto), luego **ejecuta** una **Ejecución** - Hará que la ejecución se reanude y congele en el siguiente **acceso de escritura en** cualquier lugar dentro de ese agregado seleccionado, o en esa ubicación de variable única seleccionada.

Al usar **Paso** o **Ejecutar** , las actualizaciones de los valores de variables visualizados se realizan según la configuración del usuario realizada en el menú **VarActualizar** : esto permite un rango completo de comportamiento, desde actualizaciones periódicas mínimas hasta actualizaciones inmediatas completas. Las actualizaciones reducidas o mínimas son útiles para reducir la carga de la CPU y pueden ser necesarias para evitar que la ejecución se retrase en tiempo real en lo que de otra manera sería excesiva la carga de actualización de la ventana del Panel de Variables. Cuando **Animate** está **activo** , o si se selecciona la opción de menú **Resaltar los Cambios** , los cambios en el valor de una variable durante la **ejecución** darán como resultado que su valor visualizado se actualice **inmediatamente** y se resaltará; esto hará que el **Panel de Variables** se desplace (si necesario) a la línea que contiene esa variable, y la ejecución ya no será en tiempo real !.

Cuando la ejecución se congela después del **Paso**, **Ejecutar-Hacia**, **Ejecutar-Hasta**, o **Ejecutar** -luego-**Detener**, el **Panel de Variables** destaca la variable correspondiente a la ubicación (s) dirección que quedó modificado (si lo hay) por la última instrucción durante esa ejecución (incluidas las inicializaciones de declaración de variables). Si esa instrucción llena por **completo** un **objeto o matriz** , la línea **principal (+)** o **(-)** para ese agregado se resalta. Si, en cambio, la instrucción modificó una ubicación que actualmente está visible, se resalta. Pero si la (s) ubicación (es) modificada (s) está (n) escondida (s) dentro de una matriz u objeto no expandido, esa **línea matriz** agregada obtiene una **fuentes de letra cursiva** como una señal visual de que algo dentro **sujo** fue escrito: hacer clic para expandirlo causa que su **último** elemento o miembro modificado quede resaltado.



La ventana **Editar / Monitorear** le permite seguir cualquier valor variable durante la ejecución o cambiar su valor en el medio de la ejecución del programa (detenida) (para que pueda probar cuál sería el efecto de continuar adelante con ese nuevo valor) . **Detenga la** ejecución primero, luego **haga doble clic** en la variable cuyo valor desea rastrear o cambiar. Para simplemente monitorear el valor durante la ejecución del programa, deje el diálogo abierto y luego uno de los comandos

Ejecutar o **Paso** - su valor se actualizará en Editar / Monitorear de acuerdo con las mismas reglas que gobiernan las actualizaciones en el Panel de Variables . **Para cambiar el valor de la variable** , complete el valor del cuadro de edición y **Aceptar** . Continúa la ejecución (usando cualquiera de los comandos de **Paso** o **Ejecutar**) para usar ese nuevo valor desde ese punto hacia adelante (o puedes **Revertir** al valor anterior).

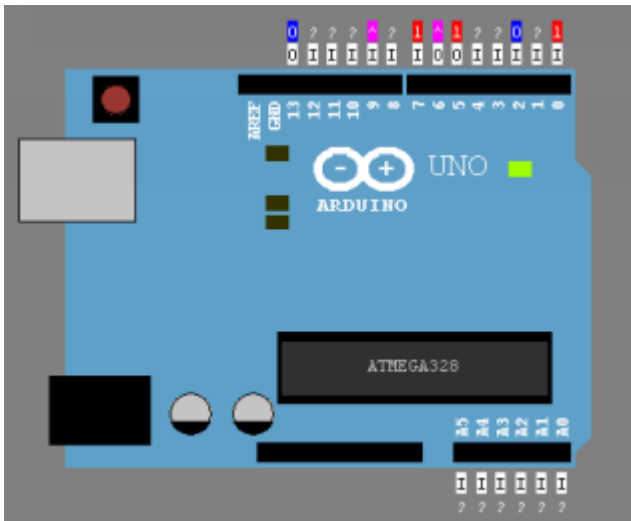
En el programa Cargar o Reiniciar note que todas las variables de valor no inicializadas se restablecen al valor 0, y todas las variables de puntero no inicializadas se restablecen a 0x0000.

Panel del Banco de Laboratorio

El Panel del Banco de Laboratorio muestra una placa de 5 voltios 'Uno' que está rodeada por un conjunto de dispositivos 'I / O' que puede seleccionar / personalizar, y conectarse a sus 'Uno' pins deseados.

El 'Uno'

Esta es una representación de la placa 'Uno' y sus LED integrados. Cuando carga un nuevo programa en UnoArduSim, si lo analiza con éxito se somete a una "descarga simulada" al 'Uno' que imita la forma en que se comporta una placa 'Uno' real: verá parpadear el LED serie RX y TX (junto con actividad en los pines 1 y 0 que están *cableados para la comunicación serial con una computadora host*). Esto es seguido inmediatamente por un pin 13 LED flash que significa que la placa se reinicia y (y UnoArduSim se detiene automáticamente en) al comienzo de la ejecución cargada de su programa. Puede evitar esta visualización y el retraso de carga asociado anulando la opción **Mostrar Descarga** desde **Configurar | Preferencias**.



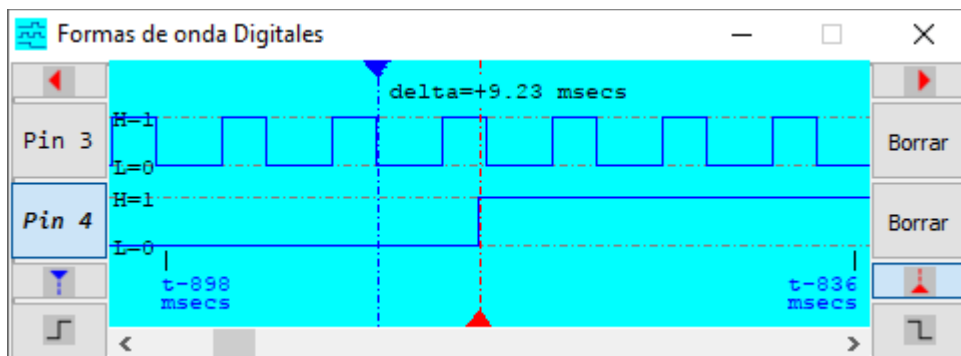
La ventana le permite visualizar los niveles lógicos digitales en todos los 20 'Uno' pins ('1' en rojo para 'HIGH' , '0' en azul para 'LOW' y '?' en gris para un voltaje indeterminado indefinido), y las indicaciones programadas ('I' para 'INPUT' , o 'O' para 'OUTPUT'). Para los pines que se pulsan usando PWM a través de 'analogWrite()' , o por 'tone()' , o por 'Servo.write()' , el color cambia a púrpura y el símbolo que se muestra se convierte en '^' .

Tenga en cuenta que **los pines digitales 0 y 1 están cableados a través de resistencias de 1 kOhm al chip USB para la comunicación en serie con una computadora host**.









A un lado: los pines digitales 0-13 aparecen como los pines del simulador 0-13, y los pines analógicos 0-5



aparecen como A0-A5. Para acceder a un pin analógico en su programa, puede hacer referencia al número pin por uno de los dos conjuntos de números equivalentes: 14-19; o A0-A5 (A0-A5 son variables *integradas 'const'* con valores 14-19). Y solo cuando se usa 'analogRead()' , se pone a disposición una tercera opción: puede, para esta instrucción, descartar el prefijo 'A' del número de pin y simplemente usar 0-5. Para acceder a los pines 14-19 en su programa utilizando 'digitalRead()' o 'digitalWrite()' , simplemente puede consultar ese número de pin, o en su lugar puede usar los alias A0-A5.

Al hacer clic con el botón izquierdo en cualquier pin 'Uno', se abrirá una ventana de **Pin Formes de Onda Digital** que muestra el **valor de un segundo de la actividad de nivel digital** en ese pin. Puede hacer clic en otros pines para agregarlos a la pantalla Pin Formes de Onda Digital (hasta un máximo de 4 formas de onda al mismo tiempo).



Haga clic para ver la página a la izquierda o a la derecha, o use las teclas Inicio, RePág, PgDn, Fin

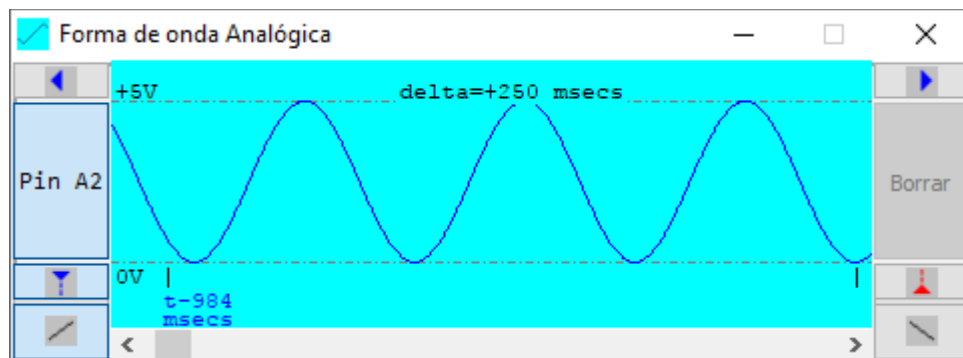
Una de las formas de onda visualizadas será la forma **de onda activa del pin** , indicada por su botón "Pin" que se muestra como presionado (como en la captura de pantalla Pin Digital Waveforms anterior). Puede seleccionar una forma de onda haciendo clic en su botón Número de pin, y luego seleccionar la polaridad de borde de interés haciendo clic en el botón de selección de polaridad de borde ascendente / descendente apropiado, , o  , O utilizando la combinación de teclas **de flecha hacia arriba y hacia abajo flecha. A continuación, puede saltar** el cursor activo (las líneas del cursor azul o rojo con su tiempo delta mostrado) hacia atrás o hacia adelante al borde digital de polaridad elegida **de este pin activo** forma de onda usando los botones del cursor (,  o ,  , dependiendo de qué cursor se activó anteriormente con  o ), o simplemente use las teclas del teclado \leftarrow y \rightarrow .

Para activar un cursor, haga clic en el botón de activación de color ( o  mostrado arriba) - *esto también desplaza la vista hacia la ubicación actual de ese cursor* . Alternativamente, puede alternar rápidamente la activación entre cursores (con sus respectivas vistas centradas) usando la tecla de acceso directo **TAB** .







Puede **saltar** el cursor actualmente activado **haciendo clic con el botón izquierdo en cualquier parte** de la región de visualización de la forma de onda en pantalla. Alternativamente, puede seleccionar la línea de cursor roja o azul haciendo clic derecho en la parte superior (para activarla), luego **arrástrela a una nueva ubicación** y lanzamiento. Cuando un cursor deseado está actualmente fuera de la pantalla, puede **hacer clic con el botón derecho en cualquier lugar** de la vista para ir a esa nueva ubicación en pantalla. Si ambos cursores ya están en la pantalla, al hacer clic con el botón derecho simplemente se alterna entre el cursor activado.

Para INCREMENTAR EL ZOOM y DISMINUIR EL ZOOM (el zoom siempre se centra en el cursor ACTIVO), use la rueda del mouse o los atajos de teclado CTRL-flecha arriba y CTRL-flecha abajo .

Haciendo en cambio un **clic derecho en cualquier pin 'Uno'** se abre una ventana de **forma de onda analógica de alfiler** que muestra el valor **pasado de un segundo de la actividad de nivel analógico** en ese pin. A diferencia de la ventana Pin Formes de Onda Digital, puede mostrar actividad analógica solo en un pin a la vez.



Haga clic para ver la página a la izquierda o a la derecha, o use las teclas Inicio, RePág, PgDn, Fin

Puede **saltar** las líneas de cursor azules o rojas al siguiente "punto de pendiente" ascendente o descendente usando los botones de flecha hacia adelante o hacia atrás (,  o ,  , de nuevo dependiendo del cursor activado, o use las teclas \leftarrow y \rightarrow) en concierto con los botones de selección de pendiente ascendente / descendente ,  (el "punto de pendiente" ocurre donde la tensión analógica pasa a través del umbral de nivel lógico-digital alto del pin ATmega). Alternativamente, puede volver a hacer clic para saltar o arrastrar estas líneas de cursor de forma similar a su comportamiento en la ventana Pin Formes de Onda Digital

Dispositivos 'I / O'

Un número de dispositivos diferentes rodea al 'Uno' en el perímetro del **Panel del Banco de Laboratorio** . Los dispositivos "pequeños" 'I / O' (de los cuales puede tener hasta 16 en total) residen a lo largo de los lados izquierdo y derecho del Panel. Los dispositivos "grandes" 'I / O' (de los cuales tiene permitido un total de 8) tienen elementos "activos" y residen en la parte superior e inferior del Panel. El número deseado de cada tipo de dispositivo 'I / O' disponible se puede establecer usando el menú **Configurar | 'I / O' Devices** .

Cada dispositivo 'I / O' tiene uno o más archivos adjuntos de pin mostrados como un **número de pin de dos dígitos** (00, 01, 02, ... 10,11,12, 13 y A0-A5, o 14-19, después de eso) en una caja de edición correspondiente. Para los números de pin 2 a 9, simplemente puede ingresar el dígito simple: el 0 inicial se proporcionará automáticamente, pero para los pines 0 y 1, primero debe ingresar el 0 inicial. Las entradas *normalmente* están en el lado izquierdo de un dispositivo 'I / O', y las salidas *normalmente* están a la derecha (*si el espacio lo permite*). Todos los dispositivos 'I / O' responderán directamente a los niveles de pin y cambios de nivel de pin, por lo que responderán a funciones de biblioteca dirigidas a sus pines adjuntos, o a '`digitalWrite()`' **programado** (para operación "bit-banged").

Puede conectar varios dispositivos al mismo pin ATmega siempre que esto no genere un **conflicto eléctrico** . Tal conflicto puede ser creado por un 'Uno' pin como '`OUTPUT`' manejando contra un dispositivo conectado de fuerte conducción (baja impedancia) (por ejemplo, conduciendo contra una salida 'FUNCGEN', o una salida del codificador 'MOTOR') , o mediante dos dispositivos conectados que compiten entre sí (por ejemplo, un botón 'PULSER' y un 'PUSH' - conectado al mismo pin). Cualquier conflicto de este tipo sería desastroso en una implementación de hardware real y, por lo tanto, no se permite, y se marcará para el usuario a través de un cuadro de mensaje emergente.

El diálogo se puede usar para permitir al usuario elegir los tipos y números de los dispositivos 'I / O' deseados. Desde este cuadro de diálogo también puede **guardar** dispositivos 'I / O' en un archivo de texto, y / o dispositivos **Cargar** 'I / O' desde un archivo de texto previamente guardado (o editado) (**incluidas todas las conexiones de PIN, y configuraciones clicables, y valores de cuadro de edición escritos a máquina**).

Tenga en cuenta que a partir de la versión 1.6, los valores en el período, el retardo y las cajas de edición de ancho de pulso en los dispositivos IO pertinentes pueden tener el sufijo con la letra 'S' (o 's'). Esto indica que se deben **escalar** según la posición de un control deslizante global '**I / O ____ S**' que aparece en la **Barra de Herramientas de la** ventana principal . Con ese control deslizante completamente a la derecha, el factor de escala es 1.0 (unidad), y con el control deslizante completamente a la izquierda, el factor de escala es 0.0 (sujeto a valores mínimos impuestos por cada dispositivo 'I / O' particular). Puede escalar más de un valor de cuadro de edición **simultáneamente** usando este control deslizante. Esta característica le permite arrastrar el control deslizante durante la ejecución para emular fácilmente el cambio de anchos de pulso, períodos y retrasos para los dispositivos 'I / O' conectados.

El resto de esta sección proporciona descripciones para cada tipo de dispositivo.

'Serial' Monitor ('SERIAL')

Este dispositivo 'I / O' permite la entrada y salida serial mediada por hardware ATmega (a través del chip 'Uno' USB) en 'Uno' pins 0 y 1. La velocidad en baudios se establece utilizando la lista desplegable en la parte inferior: la velocidad en baudios seleccionada **debe coincidir con** el valor que su programa pasa a la función '`Serial.begin()`' para una transmisión / recepción adecuada. *La comunicación serial se fija en 8 bits de datos, 1 bit de parada y ningún bit de paridad.*

Para enviar una entrada de teclado a su programa, escriba uno o más caracteres en la ventana de edición



superior (caracteres TX) y luego presione la tecla **Retorno**. (los caracteres se ponen en cursiva para indicar que las transmisiones han comenzado) - o si ya están en progreso, los caracteres añadidos estarán en cursiva. A continuación, puede utilizar las funciones '`Serial.available()`' y '`Serial.read()`' para leer los caracteres en el orden en que se recibieron en el búfer pin 0 (el carácter escrito más a la izquierda se enviará primero). Las impresiones textuales y numéricas formateadas o los valores de bytes sin formato se pueden enviar a la ventana de salida de la consola inferior (caracteres RX) llamando a las funciones Arduino '`print()`', '`println()`' o '`write()`'.

Además, **se puede abrir una ventana más grande para configurar / ver los caracteres TX y RX haciendo doble clic (o haciendo clic con el botón derecho) en este dispositivo 'Serial'**. Esta nueva ventana tiene un cuadro de edición de caracteres TX más grande, y un botón 'Send' por separado, que puede ser clickeado para enviar los caracteres TX a 'Uno' (en el pin 0). También hay una opción de casilla de verificación para volver a interpretar las secuencias de caracteres con fuga inversa, como '`\n`' o '`\t`' para visualización no cruda

Software de serie ('SFTSER')

Este dispositivo 'I / O' permite la entrada y salida serie mediada por software de biblioteca o, alternativamente, usuario "bit-banged" en cualquier par de pines 'Uno' que elija completar (**excepto los** pines 0 y 1 que están dedicados a la comunicación hardware '`Serial`'). Su programa debe tener una línea '`#include <SoftwareSerial.h>`' cerca de la parte superior si desea usar la funcionalidad de esa biblioteca. Al igual que con el 'SERIAL' dispositivo basado en hardware, la velocidad de transmisión para 'SFTSER' se establece utilizando la lista desplegable en su parte inferior - la velocidad de transmisión seleccionada debe coincidir con el valor de su programa pasa a la 'begin) la función' (para una transmisión / recepción adecuada. *La comunicación serial se fija en 8 bits de datos, 1 bit de parada y ningún bit de paridad.*

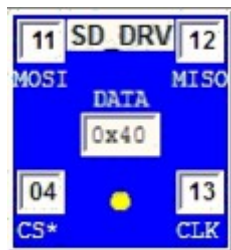


Además, al igual que con el hardware basado '`Serial`', **se puede abrir una ventana más grande para la configuración / visualización de TX y RX haciendo doble clic (o haciendo clic con el botón derecho) en el dispositivo SFTSER**.

Tenga en cuenta que a diferencia de la implementación de hardware de '`Serial`', no se proporcionan búferes TX o RX soportados por operaciones internas de interrupción ATmega, por lo que '`read()`' y '`write()`' están bloqueando (es decir, su programa no continuará hasta que se completen).

Unidad de disco SD ('SD_DRV')

Este dispositivo 'I / O' permite operaciones de entrada y salida de archivos mediadas por software de biblioteca (pero **no de "bits desorbitados"**) en los pines 'Uno' **SPI** (puede elegir qué pin **CS*** usará). Su programa puede simplemente `#include <SD.h>` línea cerca de la parte superior, y puede usar `<SD.h>` funciones O directamente puede llamar a `SdFile` functions usted mismo.

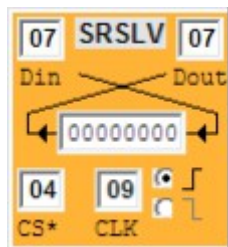


Se **puede abrir una ventana más grande que muestra directorios y archivos (y contenido) haciendo doble clic (o haciendo clic con el botón derecho) en el dispositivo 'SD_DRV'**. Todo el contenido del disco se **carga desde** un subdirectorio **SD** en el directorio del programa cargado (si existe) en `'SdVolume init()'`, **y se duplica al mismo subdirectorio SD** en el archivo `'close()'`, `'remove()'`, y en `'makeDir()'` y `'rmDir()'`.

Un LED amarillo parpadea durante las transferencias de SPI, y 'DATA' muestra el último byte de **respuesta** 'SD_DRV'. Todas las señales SPI son precisas y se pueden ver en una **ventana de forma de onda**.

Esclavo Registro de Desplazamiento ('SRSLV')

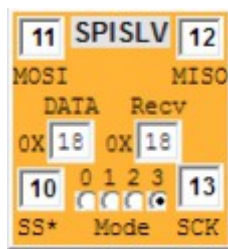
Este dispositivo 'I / O' emula un dispositivo de registro de desplazamiento simple con un pin activo **SS*** bajo ("selección de esclavo") que controla el pin de salida **Dout** (cuando **SS*** es alto, **Dout** no está **activado**). Su programa podría usar la funcionalidad del objeto y biblioteca integrados Arduino de SPI. Alternativamente, puede optar por crear sus propias señales **Din** y **CLK** "bit-banged" para conducir este dispositivo.



El dispositivo detecta las transiciones de flanco en su entrada **CLK** que activan el desplazamiento de su registro: la polaridad del flanco **CLK** detectado se puede elegir usando un control de botón de radio. En cada flanco **CLK** (de la polaridad detectada), el registro captura su nivel **Din** en la posición de bit menos significativo (LSB) del registro de desplazamiento, ya que los bits restantes se desplazan simultáneamente hacia la izquierda una posición hacia la posición MSB. Cuando **SS*** es bajo, el valor actual en la posición MSB del registro de desplazamiento se conduce a **Dout**.

Esclavo SPI configurable ('SPISLV')

Este dispositivo 'I / O' emula un esclavo SPI seleccionable en modo con un pin activo **SS*** bajo ("selección de esclavo") que controla el **pin de salida MISO** (cuando **SS*** es alto, **MISO** no está **activado**). Su programa debe tener una línea `#include <SPI.h>` si desea usar la funcionalidad del objeto y la biblioteca SPI Arduino integrados. Alternativamente, puede optar por crear sus propias señales **MOSI** y **SCK** "bit-banged" para conducir este dispositivo.

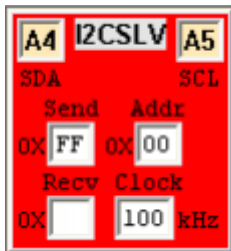


El dispositivo detecta transiciones de borde en su entrada **CLK** según el modo seleccionado (`'MODE0'`, `'MODE1'`, `'MODE2'`, o `'MODE3'`), que debe elegirse para que coincida con el modo SPI programado de su programa.

Al hacer doble clic (o hacer clic con el botón derecho) en el dispositivo, puede abrir una ventana complementaria más grande que le permite llenar el búfer de 32 bytes (para emular los dispositivos SPI que devuelven sus datos automáticamente), y ver los últimos 32 bytes recibidos (todos como pares hexadecimales). **Tenga en cuenta que** el siguiente byte del buffer de TX está automáticamente en 'DATA' solo **después de que se haya** completado un `'SPI.transfer()'` completo.

Esclavo I²C ('I2CSLV')

Este dispositivo 'I / O' solo emula un dispositivo en *modo esclavo* . Al dispositivo se le puede asignar una dirección de bus I² C usando una entrada de dos dígitos hexadecimales en su 'Addr' caja de edición (solo responderá a las transacciones de bus I² C que impliquen su dirección asignada). El dispositivo envía y recibe datos en su pin **SDA** de drenaje abierto (desplegable solamente) , y responde a la señal del reloj del bus en su pin **SCL** de drenaje abierto (solo pulldown) . Aunque 'Uno' será el maestro de bus responsable de generar la señal **SCL** , este dispositivo esclavo también reducirá **SCL** durante su fase baja para extender (si es necesario) el bus a una velocidad adecuada a su velocidad interna (que se puede configurar en su '**Clock**' caja de edición).



Su programa debe tener una línea '`#include <Wire.h>`' si desea utilizar la funcionalidad de la biblioteca '`TwoWire`' para interactuar con este dispositivo. Alternativamente, puede optar por crear sus propios datos de bits y señales de reloj para conducir este dispositivo esclavo.

Un byte único para la transmisión de vuelta al 'Uno' master se puede establecer en el 'xSend' caja de edición, y un solo byte (el más recientemente recibido) se puede ver en su (solo lectura) 'Recv' edit- caja. **Tenga en cuenta que** el valor 'Send' caja de edición siempre refleja el **siguiente** byte para la transmisión desde el búfer de datos interno de este dispositivo.

Al hacer doble clic (o hacer clic con el botón derecho) en el dispositivo, puede abrir una ventana complementaria más grande que le permite completar un búfer FIFO de 32 bytes (para emular dispositivos TWI con dicha funcionalidad), y para ver (hasta un máximo de 32) bytes de los datos recibidos más recientemente (como una pantalla de dos dígitos hexadecimales de 8 bytes por línea). El número de líneas en estas dos cajas de edición corresponde al tamaño de búfer TWI elegido (que se puede seleccionar usando **Configurar | Preferencias**). Esto se ha agregado como una opción ya que la biblioteca Arduino '`Wire.h`' utiliza **cinco** de estos búferes de RAM en su código de implementación, que es la memoria RAM cara. **Editando** el archivo Arduino installation '`Wire.h`' para cambiar la constante definida '`BUFFER_LENGTH`' (y también editando el archivo companion '`utility/twi.h`' para cambiar la longitud del búfer TWI) ambos para ser 16 o 8, un usuario *podría* reducir significativamente la sobrecarga de la memoria RAM de 'Uno' en una **implementación de hardware** dirigida - UnoArduSim por lo tanto refleja esta posibilidad del mundo real a través de **Configurar | Preferencias**.

Motor Paso a Paso ('STEPR')

Este dispositivo 'I / O' emula un motor paso a paso bipolar o unipolar de 6 V con un controlador integrado controlado por **dos** señales de control (en **P1** , **P2**) o **cuatro** (en **P1** , **P2** , **P3** , **P4**). El número de pasos por revolución también se puede establecer. Puede usar las '`Stepper.h`' funciones '`setSpeed()`' y '`step()`' para controlar 'STEPR'. Alternativamente, 'STEPR' *también responderá* a sus propias señales de unidad '`digitalWrite()`' " bit-banged".



El motor se modela con precisión tanto mecánica como eléctricamente. Las caídas de voltaje del impulsor del motor y la resistencia e inductancia variables se modelan junto con un momento realista de inercia con respecto al par de retención. El devanado del rotor del motor tiene una resistencia modelada de $R = 6$ ohmios, y una inductancia de $L = 6$ mili-Henries que crea una constante de tiempo eléctrica de 1.0 milisegundo. Debido al modelado realista, notará que los pulsos de pin de control muy angostos *no hacen que* el motor avance, ambos debido al tiempo de subida de la corriente finita y al efecto de la inercia del rotor. Esto concuerda con lo que se observa cuando se conduce un motor paso

a paso real desde un 'Uno' con, por supuesto, un chip de controlador de motor apropiado (**y requerido**) entre los cables del motor y el 'Uno'!

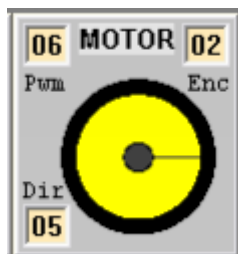
Una **falla** desafortunada en el código de la biblioteca Arduino '`Stepper.h`' significa que al reiniciar el motor paso a paso no estará en la posición 1 (de cuatro pasos). Para superar esto, el usuario debe usar

'digitalWrite()' en su rutina 'setup()' para inicializar los niveles de pin de control a los niveles 'step (1)' apropiados para 2-pin (0,1) o 4 -pin (1,0,1,0) control, y permita que el motor 10 msecs se mueva a la posición inicial deseada del motor de referencia de 12-mediodía.

Tenga en cuenta que **la reducción de engranaje no se admite directamente** debido a la falta de espacio, pero puede emularlo en su programa implementando una variable de contador módulo-N y solo llamando a 'step()' cuando ese contador llegue a 0 (para reducción de engranaje por factor N).

Motor de CC ('MOTOR')

Este dispositivo 'I/O' emula un motor de CC con engranaje 100: 1 de 6 voltios con un controlador integrado controlado por una señal de modulación de ancho de pulso (en su entrada **Pwm**) y una señal de control de dirección (en su entrada **Dir**). El motor también tiene una salida de codificador de rueda que impulsa su pin de salida **Enc**. Puede usar 'analogWrite()' para conducir el pin **Pwm** con una forma de onda PWM de 490 Hz (en los pines 3,9,10,11) o 980 Hz (en los pines 5,6) entre 0,0 y 1,0 ('analogWrite()' valores de 0 a 255). Alternativamente, 'MOTOR' también responderá a sus propias señales de unidad 'digitalWrite()' "bit-banged".



El motor se modela con precisión tanto mecánica como eléctricamente. La contabilización de las caídas de tensión del transistor del motor y el par del engranaje realista sin carga proporciona una velocidad máxima de aproximadamente 2 revoluciones por segundo y un par de parada de poco más de 5 kg-cm (con un ciclo de trabajo PWM constante de 1,0), con un momento de inercia de motor más carga de 2.5 kg-cm. El devanado del rotor del motor tiene una resistencia modelada de $R = 2$ ohmios, y una inductancia de $L = 300$ micro-Henries que crea una constante de tiempo eléctrica de 150 microsegundos. Debido a la modelación realista, notará que los impulsos PWM muy estrechos *no hacen que* el motor gire, tanto debido al tiempo de subida de la corriente finita, como a la *falta de*

tiempo significativa después de cada impulso estrecho. Estos se combinan para causar una cantidad de impulso del rotor insuficiente para superar el latigazo en forma de resorte de la caja de engranajes bajo fricción estática. La consecuencia es que al usar 'analogWrite()', un ciclo de trabajo por debajo de 0.125 no hará que el motor se mueva: esto concuerda con lo que se observa cuando se maneja un motor de engranaje real de 'Uno' con, por supuesto, un apropiado (**y se requiere**) módulo de controlador de motor entre el motor y el 'Uno'!

El codificador de motor emulado es un sensor de interrupción óptica montado en el eje que produce una forma de onda de ciclo de trabajo del 50% con 8 períodos completos alto-bajo por revolución de rueda (para que su programa pueda detectar cambios rotacionales de la rueda a una resolución de 22.5 grados).

ServoMotor ('SERVO')

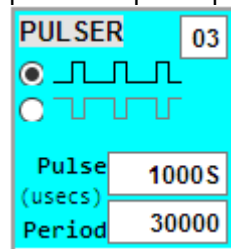
T su dispositivo 'I/O' emula un 6 voltios motor de suministro de DC servo PWM guiado en posición controlada. Los parámetros de modelado mecánico y eléctrico para la operación de servo coincidirán estrechamente con los de un servo estándar HS-422. El servo tiene una velocidad de rotación máxima de aproximadamente 60 grados en 180 mseg. Si se marca la casilla inferior izquierda, el servo se convierte en un servo de rotación continua con la misma velocidad máxima, pero ahora el ancho de pulso PWM establece la velocidad en lugar del ángulo.



Su programa debe tener una línea '#include <Servo.h>' antes de declarar su (s) instancia (s) de Servo *si elige usar la funcionalidad de la biblioteca Servo*, por ej. 'Servo.write()', 'Servo.writeMicroseconds()' Alternativamente, SERVO también responde a señales 'digitalWrite()' "bit-banged". Debido a la implementación interna de UnoArduSim, está limitado a 5 'SERVO' dispositivos.

Generador de Pulso Digital ('PULSER')

Este dispositivo 'I / O' emula un generador de forma de onda de pulso digital simple que produce una señal periódica que se puede aplicar a cualquier pin 'Uno' elegido.

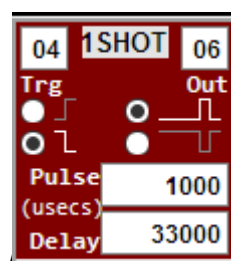


El período y los anchos de pulso (en microsegundos) se pueden configurar mediante cuadros de edición: el período mínimo permitido es de 50 microsegundos y el ancho de pulso mínimo es de 10 microsegundos. También se puede elegir la polaridad: pulsos positivos de vanguardia (0 a 5 V) o pulsos negativos de vanguardia (5 a 0 V).

Los valores de 'Pulse' y 'Period' son escalables desde el control deslizante de **Barra de Herramientas** principal 'I / O ____ S' scale-factor al sumar uno (o ambos) con la letra 'S' (o 's').

Un-Trago ('1SHOT')

Este dispositivo 'I / O' emula un Un-Trago digital que puede generar un pulso de polaridad elegida y ancho de pulso en su 'Out' pin, que se produce después de un retraso especificado desde un borde de disparo recibido en su pin de entrada **Trg** (disparador). Una vez que se recibe el filo de activación especificado, comienza el cronometraje y luego no se reconocerá un nuevo pulso de activación hasta que se haya producido el impulso 'Out' (y haya finalizado por completo).

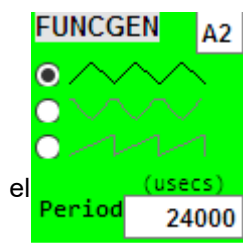


Un posible uso de este dispositivo es simular sensores de ultrasonido que generan un rango de pulso en respuesta a un pulso de activación. También puede usarse donde desee generar una señal de entrada de pin sincronizada (después de la demora elegida) a una señal de salida de pin creada por su programa.

Los valores de 'Pulse' y 'Delay' son escalables desde la ventana principal **Tool-Bar** 'I / O ____ S' scale-factor control deslizante al sumar uno (o ambos) con la letra 'S' (o 's').

Generador de funciones analógicas ('FUNCGEN')

Este dispositivo 'I / O' emula un generador de onda analógico simple que produce una señal periódica que se puede aplicar a cualquier 'Uno' pin elegido.



El período (en microsegundos) puede establecerse usando el cuadro de edición; el período mínimo permitido es de 100 microsegundos. La forma de onda que crea se puede elegir para que sea sinusoidal, triangular o diente de sierra (para crear una onda cuadrada, utilice 'PULSER' en su lugar). En períodos más pequeños, se utilizan menos muestras por ciclo para modelar la forma de onda producida (solo 4 muestras por ciclo en período = 100 usos).

El valor de 'Period' es escalable desde la ventana principal **Tool-Bar** 'I / O ____ S' scale-factor control deslizante con el sufijo con la letra 'S' (o 's').

Altavoz piezoeléctrico ('PIEZO')



Este dispositivo le permite "escuchar" las señales en cualquier 'Uno' pin elegido, y puede ser un complemento útil de los LED para depurar el funcionamiento de su programa. También puede divertirse un poco tocando los tonos de llamada con las llamadas `'tone()'` y `'delay()'` apropiadamente (aunque no hay filtrado de la forma de onda rectangular, por lo que no escuchará notas "puras").

También puede escuchar un dispositivo 'PULSER' o 'FUNCGEN' conectado al conectar un 'PIEZO' al pin que maneja el dispositivo.

Pulsador ('PUSH')



Este dispositivo 'I / O' emula un pulsador **momentáneo** normalmente abierto o **de enclavamiento de un** solo polo, de un solo paso (SPST) con una resistencia de pull-up (o pull-down) de 10 k-ohmios. Si se elige una selección de transición de borde ascendente para el dispositivo, los contactos del botón se cablearán entre el pin del dispositivo y + 5V, con un desplegable de 10 k-Ohm a tierra. Si se elige una transición de borde descendente para el dispositivo, los contactos del botón se conectarán entre el pin del dispositivo y la tierra, con un pull-up de 10 k-Ohm a + 5V.

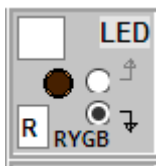
Al hacer clic con el botón izquierdo, o presione cualquier tecla, cierra el contacto del botón. En modo **momentáneo**, permanece cerrado mientras mantiene presionado el botón del mouse o cualquier tecla, y en el modo de **bloqueo** (habilitado haciendo clic en el botón 'latch' permanece cerrado (y en un color diferente) hasta que vuelva a hacer clic en el botón, o presione cualquier tecla.

Resistencia del interruptor deslizante ('R = 1K')



Este dispositivo le permite al usuario conectar a 'Uno' pin una resistencia pull-up de 1 k-Ohm a + 5V, o una resistencia pull-down de 1 k-Ohm a tierra. Esto le permite simular cargas eléctricas agregadas a un dispositivo de hardware real. Al hacer clic con el botón izquierdo en el **cuerpo del** interruptor deslizante, puede alternar la selección deseada de pull-up o pull-down. El uso de uno o varios de estos dispositivos le permitiría configurar un "código" simple (o múltiple) para que su programa lo lea y responda.

LED de color ('LED')



Puede conectar un LED entre el 'Uno' pin elegido (a través de una resistencia de limitación de corriente de 1 k-Ohm de la serie oculta incorporada) a tierra o a + 5V; esto le da la opción de tener el LED encendido cuando connected 'Uno' pin es **'HIGH'**, o en su lugar, cuando es **'LOW'**.

El color del LED se puede elegir entre rojo ('R'), amarillo ('Y'), verde ('G') o azul ('B') usando su cuadro de edición.





Control Deslizante Analógico








Se puede conectar un potenciómetro 0-5V controlado por control deslizante a cualquier pin 'Uno' elegido para producir un nivel de voltaje analógico estático (o de cambio lento) que sería leído por `'analogRead()'` como un valor de 0 a 1023. Usa el mouse para arrastrar, o haz clic para saltar, el control Deslizante Analógico.

Menús









Archivo:

 Cargar INO o PDE Prog (ctrl-L)	Permite al usuario elegir un archivo de programa que tenga la extensión seleccionada. El programa se analiza inmediatamente
Editar / Examinar (ctrl-E)	Abre el programa cargado para ver / editar.
 Guardar	Guarde los contenidos del programa editado de nuevo en el archivo de programa original.
Guardar Como	Guarde los contenidos del programa editado con un nombre de archivo diferente.
 Siguiente ('#include')	Avanza el Panel de Código para mostrar el siguiente archivo '#include'
 Anterior	Devuelve la pantalla del Panel de Código al archivo anterior
Salida	Sale de UnoArduSim después de recordar al usuario que guarde los archivos modificados.

Encontrar:

 Encontrar Función / Var Siguiente	Salta a la siguiente Función en el Panel de Código (si tiene el foco activo), o a la siguiente variable en el Panel de Variables (si en cambio tiene el foco activo).
 Encontrar Función Anterior / Var	Salta a la Función anterior en el Panel de Código (si tiene el foco activo), o a la variable anterior en el Panel de Variables (si en cambio tiene el foco activo).
 Establecer tTexto de Búsqueda ctrl-F)	Active la Barra de Herramientas Buscar cuadro de edición para definir el texto que se buscará a continuación (y agrega la primera palabra de la línea resaltada actualmente en el Panel de Código o en el Panel de Variables si uno de ellos tiene el foco).
Buscar texto Siguiente 	Vaya a la siguiente aparición de Texto en el Panel de Código (si tiene el foco activo), o a la siguiente aparición de Texto en el Panel de Variables (si en cambio tiene el foco activo).
Buscar texto Anterior 	Vaya a la aparición de texto anterior en el Panel de Código (si tiene el foco activo) o a la aparición de texto anterior en el Panel de Variables (si en cambio tiene el foco activo).

Ejecutar:

 Paso En (F4)	Ejecuta la ejecución adelante por una instrucción o <i>en una función llamada</i> .
 Paso Sobre (F5)	Ejecuta la ejecución hacia delante mediante una instrucción o <i>mediante una llamada a función completa</i> .
 Paso Afuera (F6)	Avanza en la ejecución <i>lo suficiente como para salir de la función actual</i> .
 Ejecutar Hacia (F7)	Ejecuta el programa, <i>deteniéndose en la línea de programa deseada</i> : primero debe hacer clic para resaltar la línea de programa deseada antes de ejecutar Ejecutar-Hacia.
 Ejecutar Hasta (F8)	Ejecuta el programa hasta que se produce una escritura en la variable que tenía el resaltado actual en el Panel de Variables (haga clic en uno para establecer el resaltado inicial).
 Ejecutar (F9)	Ejecuta el programa.
 Detener (F10)	Detiene la ejecución del programa (<i>y congela el tiempo</i>).
 Reiniciar	Restablece el programa (todas las variables de valor se restablecen al valor 0, y todas las variables de puntero se restablecen a 0x0000).
Animar	Automáticamente pasa líneas de programa consecutivas <i>con retardo artificial añadido</i> y resaltado de la línea de código actual. La operación en tiempo real y los sonidos se pierden.
Camara lenta	Disminuye el tiempo en un factor de 10.

Opciones:

<u>Paso por encima de Structors / Operadores</u>	Vuela directamente a través de constructores, destructores y funciones de sobrecarga del operador durante cualquier paso (es decir, no se detendrá dentro de estas funciones).
<u>Modelo de Asignación de Registro</u>	Asigne locales de función para liberar registros ATmega en lugar de a la pila (genera un uso de RAM algo reducido).
<u>Error en Sin inicializar</u>	Marcar como un error de Análisis en cualquier lugar en el que su programa intente usar una variable sin inicializar primero su valor (o al menos un valor dentro de una matriz).
<u>Se agregó 'loop()' Delay</u>	Agrega 200 microsegundos de retraso cada vez que se llama a 'loop()' (en caso de que no haya otras llamadas de programa a 'delay()' anywhere) - útil para intentar evitar quedar muy atrás en tiempo real.

Configurar:

<u>Dispositivos 'I / O'</u>	Abre un cuadro de diálogo para permitir al usuario elegir el tipo (s) y los números de los dispositivos 'I / O' deseados. Desde este diálogo, también puede Guardar dispositivos 'I / O' en un archivo de texto, y / o Cargar dispositivos 'I / O' de un archivo de texto previamente guardado (o editado) (incluyendo todas las conexiones de PIN y ajustes clicables y tipeados) valores
<u>Preferencias</u>	Abre un cuadro de diálogo para permitir al usuario establecer preferencias incluyendo la sangría automática de las líneas del programa fuente, permitiendo sintaxis experta, elección del tipo de fuente, optando por un tamaño de letra más grande, aplicando límites de matriz, permitiendo palabras clave lógicas del operador, mostrando la descarga del programa, elección de la versión 'Uno' placa y de la longitud del búfer TWI (para dispositivos I2C).

VarActualizar:

<u>Permitir colapso automático(-)</u>	Permita que UnoArduSim colapse las matrices / objetos desplegados cuando se retrasa en tiempo real.
<u>Mínimo</u>	Solo actualice Panel de Variables 4 veces por segundo.
<u>Resaltar los Cambios</u>	Resalte el valor de la variable modificada (puede causar una desaceleración).

Ventanas:

<u>'Serial' Monitor</u>	Conecte un dispositivo de E / S en serie a los pines 0 y 1 (si no tiene ninguno) y despliegue una ventana de texto TX / RX de monitor ' Serial ' mayor .
<u>Restaura todo</u>	Restaure todas las ventanas secundarias minimizadas.
<u>Pin Formes de Onda Digital</u>	Restaure una ventana de Pin Formes de Onda Digital minimizada.
<u>Forma de Onda Analógico del Pin</u>	Restaure una ventana minimizada Pin Forma de Onda Análogo

Ayuda:

<u>Archivo de Ayuda Rápida</u>	Abre el archivo UnoArduSim_QuickHelp PDF.
<u>Archivo de ayuda completa</u>	Abre el archivo PDF UnoArduSim_FullHelp.
<u>Corrección de errores</u>	Ver correcciones de errores significativas desde la versión anterior.
<u>Cambio / Mejoras</u>	Ver cambios y mejoras significativos desde la versión anterior.
<u>Acerca de</u>	Muestra la versión, copyright.

Modelado

'Uno' Placa y 'I / O' Devices

Los dispositivos 'Uno' y los adjuntos 'I / O' están todos modelados con precisión eléctrica, y usted podrá hacerse una buena idea en casa de cómo se comportarán sus programas con el hardware real, y todos los conflictos de clavijas eléctricas serán marcados.

Sincronización

UnoArduSim se ejecuta lo suficientemente rápido en una PC o tableta que puede (*en la mayoría de los casos*) acciones del programa modelo en tiempo real, **pero solo si su programa incorpora** al menos algunas llamadas 'delay()' pequeñas u otras llamadas que naturalmente mantendrán se sincronizó con el tiempo real (ver a continuación).

Para lograr esto, UnoArduSim hace uso de una función de temporizador de devolución de llamada de Windows, que le permite mantener un seguimiento preciso en tiempo real. La ejecución de varias instrucciones de programa se simula durante un segmento de temporizador, y las instrucciones que requieren una ejecución más larga (como las llamadas a 'delay()') pueden necesitar varias divisiones de temporizador. Cada iteración de la función del temporizador de devolución de llamada corrige el tiempo del sistema utilizando el reloj del hardware del sistema para que la ejecución del programa se ajuste constantemente para mantener el paso de bloqueo con el tiempo real. *El único momento en que la tasa de ejecución debe estar detrás del tiempo real es* cuando el usuario ha creado bucles ajustados **sin demora adicional** , o los dispositivos 'I / O' están configurados para funcionar con frecuencias de dispositivo 'I / O' muy altas (y / o baudios) tasa) que generaría un número excesivo de eventos de cambio de nivel de pin y sobrecarga de procesamiento asociada. UnoArduSim hace frente a esta sobrecarga omitiendo algunos intervalos de temporizador para compensar, y esto ralentiza la progresión del programa a **menos tiempo real** .

Además, los programas con matrices grandes que se muestran o vuelven a tener bucles ajustados **sin retardo adicional** pueden causar una frecuencia de llamada de función alta y generar una carga de actualización de visualización de **Panel de Variables** alta que hace que se quede atrás en tiempo real; esto se puede evitar con **Permitir Reducción** , o cuando realmente se necesita, **Mínimo**, desde el menú **VarActualizar** ,

Modelar con precisión el tiempo de ejecución de sub-milisegundos para cada instrucción u operación del programa **no se hace** - sólo cálculos muy aproximados para la mayoría han sido adoptadas con fines de simulación. Sin embargo, el tiempo de las funciones 'delay()' y 'delayMicroseconds()' y las funciones 'millis()' y 'micros()' son todas perfectamente precisas, y **siempre que utilice al menos una de las funciones de retardo** en un bucle en algún lugar de su programa, o utiliza una función que naturalmente se vincula con la operación en tiempo real (como 'print()' que está vinculada a la velocidad en baudios elegida), entonces el rendimiento simulado de su programa estará muy cerca a tiempo real (una vez más, salvo los eventos de cambio de nivel de pin de alta frecuencia descaradamente excesivos o las actualizaciones excesivas de Variables permitidas por el usuario que podrían ralentizarlo).

Para ver el efecto de las instrucciones individuales del programa *mientras se está ejecutando* , puede ser conveniente poder desacelerar las cosas. El usuario puede establecer un factor de desaceleración de tiempo de 10 en el menú **Ejecutar** .

Sincronización del Dispositivo 'I / O'

Estos dispositivos virtuales reciben señalización en tiempo real de los cambios que ocurren en sus pines de entrada, y producen salidas correspondientes en sus pines de salida que luego pueden ser detectados por el 'Uno' - por lo tanto, están sincronizados inherentemente con la ejecución del programa. La sincronización del dispositivo interno 'I / O' es establecida por el usuario (por ejemplo, mediante la selección de la velocidad en baudios o la frecuencia del reloj), y los eventos del simulador se configuran para rastrear el funcionamiento interno en tiempo real.

Sonidos

Cada dispositivo 'PIEZO' produce un sonido que corresponde a los cambios de nivel eléctrico que ocurren en el pin adjunto, independientemente de la fuente de dichos cambios. Para mantener los sonidos sincronizados con la ejecución del programa, UnoArduSim inicia y detiene la reproducción de un buffer de sonido asociado a medida que se inicia / detiene la ejecución.

A partir de V2.0, ahora se modificó el sonido para usar la API de audio Qt. Desafortunadamente, su QAudioOutput 'class' no admite el bucle del búfer de sonido para evitar que se **agoten** las muestras de sonido (como puede ocurrir durante demoras operativas de ventanas del sistema operativo más largas). Por lo tanto, para evitar la gran mayoría de los molestos clics de sonido y la interrupción del sonido durante las demoras del sistema operativo, el sonido ahora se silencia de acuerdo con la siguiente regla:

El sonido está silenciado mientras UnoArduSim no sea la ventana "activa" (excepto cuando se acaba de crear y activar una ventana secundaria), **e incluso** cuando UnoArduSim es la ventana principal "activa", pero el puntero del mouse está **fuera** de su ventana principal área de cliente de ventana.

Tenga en cuenta que esto implica que el sonido se silenciará temporalmente como rey cuando el puntero del mouse se encuentre **sobre una ventana secundaria**, y se silenciará **si se hace clic en esa ventana secundaria para activarla** (hasta que se vuelva a hacer clic en la ventana original UnoArduSim para volver a activarla).

El sonido siempre se puede silenciar haciendo clic en ianywhere dentro del área del cliente de la ventana principal de UoArduSim.

Debido al almacenamiento en búfer, sund tiene un retardo en tiempo real de hasta 250 milisegundos a partir del tiempo de evento correspondiente en el pin del 'PIEZO' adjunto.

Limitaciones y elementos no admitidos

Archivos incluidos

Un '<>' - entre corchetes '#include' de '<Servo.h>', '<Wire.h>', '<SoftwareSerial.h>', '<SPI.h>', '<EEPROM.h>' y '<SD. h>' es compatible, pero solo se emulan: no se buscan los archivos reales; en su lugar, su funcionalidad está directamente "incorporada" en UnoArduSim, y son válidas para la versión Arduino compatible.

Cualquier '#include' citado (por ejemplo, "supp.ino", "myutil.cpp" o "mylib.h") es compatible, pero todos esos archivos deben **residir en el mismo directorio como el archivo de programa principal** que contiene su '#include' (no se realizan búsquedas en otros directorios). La función '#include' puede ser útil para minimizar la cantidad de código de programa que se muestra en el Panel de Código en cualquier momento. Los archivos de encabezado con '#include' (es decir, los que tienen una extensión ".h") también provocarán que el simulador intente incluir el mismo archivo con una extensión ".cpp" (si también existe en el directorio del padre) programa).

Asignaciones dinámicas de memoria y RAM

Operadores 'new' y 'delete' son compatibles, al igual que los objetos nativos Arduino 'String', **pero no las llamadas directas a 'malloc()', 'realloc()' y 'free()' en los** que confían.

El uso excesivo de RAM para las declaraciones de variables se marca en el tiempo del Análisis, y el desbordamiento de memoria RAM se marca durante la ejecución del programa. Un ítem en Meny **Options le**

permite emular la asignación de registro normal de ATmega como lo haría el compilador de AVR, o modelar un esquema de compilación alternativo que use solo la pila (como una opción de seguridad en caso de que aparezca un error en mi asignación de registro) modelado). Si tuviera que usar un puntero para mirar el contenido de la pila, debería reflejar con precisión lo que aparecería en una implementación de hardware real.

Asignaciones de Memoria 'Flash'

'Flash' memory 'byte', 'int' y 'float' variables / matrices y sus correspondientes funciones de acceso de lectura son compatibles. Cualquier 'F()' se apoya llamada de función ("Flash"-macro) de cualquier cadena literal, pero el único soportado 'Flash' funciones de acceso directo de cuerda-Memoria son 'strcpy_P()' y 'memcpy_P()', por lo que el uso de otras funciones tendrá que copiar en primer lugar la cadena 'Flash' a una memoria de trabajo convencional 'String' variable y luego trabajar con la memoria RAM 'String'. Cuando utiliza la palabra clave 'PROGMEM' variable-modifier, debe aparecer **delante del** nombre de la variable, y esa variable **también debe declararse** como 'const'.

'String' Variables

La biblioteca native 'String' es casi completamente compatible con algunas excepciones muy (y menores).

Los operadores 'String' soportados son +, +=, <, <=, >, >=, ==, != Y[]. **Tenga en cuenta que:** 'concat()' toma un **único** argumento que es 'String', o 'char', o 'int' para ser anexado al objeto 'String' original, **no** dos argumentos como se indica erróneamente en la Referencia de Arduino páginas web).

Bibliotecas Arduino

Solo 'Stepper.h', 'SD.h', 'Servo.h', 'SoftwareSerial.h', 'SPI.h', 'Wire.h' y 'EEPROM.h' para la **versión Arduino V1.6.6** actualmente son compatibles con UnoArduSim. Tratando de '#include' los archivos ".cpp" y ".h" de otras bibliotecas aún no compatibles **no funcionará** ya que contendrán instrucciones de ensamblaje de bajo nivel y directivas no compatibles y archivos no reconocidos!

Punteros

Se admiten punteros a tipos simples, matrices u objetos. Un puntero puede equipararse a una matriz del mismo tipo (e.g. `iptr = intarray`), pero luego no habría *límites de matrices posteriores verificando* una expresión como `iptr [índice]`.

Las funciones pueden devolver punteros, o 'const' punteros, pero se ignora cualquier nivel posterior de 'const' en el puntero devuelto.

No se **admite el uso** de llamadas a funciones a través **de indicadores de función declarados** por el **usuario**.

Objetos 'class' y 'struct'

Aunque se admite el **polimorfismo** y la herencia (a cualquier profundidad), una 'class' o 'struct' solo se puede definir para tener como máximo **una** base 'class' (es decir, **la** herencia **múltiple** no es compatible). Las llamadas de inicialización base-'class' constructor (a través de la notación de dos puntos) en las líneas de declaración de constructor son compatibles, pero **no** las inicializaciones de miembros utilizando la misma notación de dos puntos. Esto significa que los objetos que contienen **variables** 'const' non-'static', o variables de tipo de referencia, no son compatibles (solo son posibles con inicializaciones de miembros de tiempo de construcción especificadas).

Las sobrecargas del operador de asignación de copias son compatibles junto con los constructores de movimientos y las asignaciones de movimiento, pero las funciones de conversión de objetos definidas por el usuario ("fundidas") no son compatibles.

Alcance

No hay soporte para la palabra clave `'using'`, ni para espacios de nombres, ni para `'file'` scope. Todas las declaraciones no locales son por implementación supuestamente globales.

Any `'typedef'`, `'struct'`, o `'class'` definición (es decir, que se puede usar para futuras declaraciones), debe tener **un** alcance **global** (las definiciones **locales** de tales elementos dentro de una función no son compatibles).

Calificadores `'unsigned'`, `'const'`, `'volatile'`, `'static'`

El prefijo `'unsigned'` funciona en todos los contextos legales normales. La palabra clave `'const'`, cuando se usa, debe **preceder** al nombre de la variable o nombre de la función o `'typedef'` name que se está declarando, colocándolo después del nombre causará un error de Análisis. Para las declaraciones de función, solo las funciones de devolución de puntero pueden tener `'const'` en su declaración.

Todas las variables de UnoArduSim son `'volatile'` por implementación, por lo que la palabra clave `'volatile'` simplemente se ignora en todas las declaraciones de variables. Las funciones no se pueden declarar `'volatile'`, ni son argumentos de llamada de función.

La palabra clave `'static'` está permitida para variables normales, y para miembros de objetos y funciones de miembros, pero explícitamente no se permite para las mismas instancias de objetos (`'class'` / `'struct'`), para funciones que no son miembros y para todos los argumentos de funciones.

Directivas del Compilador

`'#include'` y regular `'#define'` son compatibles, pero **no macro** `'#define'`. El `'#pragma'` directivas y directivas de inclusión condicional (`'#ifdef'`, `'#ifndef'`, `'#if'`, `'#endif'`, `'#else'` y `'#elif'`) tampoco son **compatibles**. El error `'#line'`, `'#'` y las macros predefinidas (como `'_LINE_'`, `'_FILE_'`, `'_DATE_'` y `'_TIME_'`) tampoco son **compatibles**.

Elementos de Lenguaje Arduino

Todos los elementos nativos del lenguaje Arduino son compatibles, con la excepción de la dudosa `'goto'` instrucción (el único uso razonable que puedo pensar sería como un salto (a un rescate y cierre seguro ciclo infinito) en el caso de una condición de error que su programa no puede resolver de otra manera).

C / C ++ - Elementos de Lenguaje

No se admiten los "calificadores de campo de bit" que ahorran bits para los miembros en las definiciones de estructura .

`'union'` **no es compatible**.

El oddball "operador de coma" **no es compatible** (por lo que no puede realizar varias expresiones separadas por comas cuando normalmente se espera una sola expresión, por ejemplo en `'while()'` y `'for (;)'` constructs).

Plantillas de funciones

Las funciones definidas por el usuario que utilizan la palabra clave "plantilla" para permitirle aceptar argumentos de tipo "genérico" **no** son *compatibles* .

Emulación en Tiempo Real

Como se indicó anteriormente, los tiempos de ejecución de las diferentes instrucciones individuales posibles del programa Arduino **no se** modelan con precisión, de modo que para ejecutar a una tasa en tiempo real su programa necesitará algún tipo de instrucción '`delay()`' dominante (al menos una vez) por '`loop()`' , o una instrucción que está naturalmente sincronizada con los cambios de nivel de pin en tiempo real (como, '`pulseIn()`' , '`shiftIn()`' , '`Serial.read()`' , '`Serial. print()`' , '`Serial.flush()`' etc.).

Vea **Modelado** y **sonidos** arriba para más detalles sobre las limitaciones.

Notas de lanzamiento

Corrección de Errores

V2.1-Mar. 2018

- 1) Un error en las nuevas versiones V2.0.x hizo que el montón de Windows creciera con cada actualización en el **Panel de Variables**: después de millones de actualizaciones (muchos minutos de ejecución) podría producirse un bloqueo..
- 2) Llamadas de función a funciones miembro '**static**' usando notación de dos puntos no se han podido analizar cuando están dentro de los corchetes '**if()**', '**while()**', '**for()**' y '**switch()**', y cuando las expresiones internas se usan argumentos de llamada de función o índices de matriz.

V2.0.2-Feb. 2018

- 1) Un error introducido en V2.0 provocó una caída de Archivo | Cargar si un '**#include**' hacía referencia a un archivo faltante o vacío.

V2.0.1-Enero 2018

- 1) En las configuraciones regionales en idioma no inglés, '**en**' se mostró incorrectamente como seleccionado en **Preferencias**, lo que hace que la reversión al inglés sea incómoda (requiriendo la des-selección y luego la re-selección).
- 2) El usuario ha podido dejar un valor de cuadro de edición de PIN del dispositivo en un estado incompleto (como '**A_**') y dejar los bits 'DATA' de una 'SRSlave' incompletos.
- 3) El número máximo de controles deslizantes analógicos se había limitado a 4 (ahora se corrigió que era 6).
- 4) UnoArduSim ya no insiste en que '=' aparezca en una inicialización de agregado de matriz.
- 5) UnoArduSim había insistido en que se diera el argumento 'inverted_logic' a '**SoftwareSerial()**' .
- 6) Las operaciones de cambio de bit ahora permiten turnos más largos que el tamaño de la variable desplazada.

V2.0-Dic. 2017

- 1) Todas las funciones que se declararon como '**unsigned**' sin embargo, ha estado devolviendo valores como si fueran '**signed**' . Esto no tuvo ningún efecto si el valor '**return**' se asignó a '**unsigned**' variable, pero habría causado una interpretación negativa incorrecta si tuviera MSB == 1, y luego se le asignó a una variable '**signed**' , o se probó en una desigualdad.
- 2) Los controles **Deslizantes Analógicos** solo alcanzaban un valor máximo '**analogRead()**' de 1022, no el 1023 correcto.
- 3) Un error introducido inadvertidamente en V1.7.0 en la lógica utilizada para acelerar el manejo del pin SPI del sistema SPI provocó que las transferencias SPI para '**SPI_MODE1**' y '**SPI_MODE3**' fallaran después de que se transfiriera el primer byte (una transición SCK adicional espuria siguió a cada byte). También se retrasaron las actualizaciones de un cuadro 'SPISLV' edit 'DATA' para los bytes transferidos,
- 4) El dispositivo LED de color no estaba enlistando 'B' (para Azul) como una opción de color (aunque fue aceptado).
- 5) La configuración de los dispositivos 'SPISLV' y 'I2CSLV' no se guardaba en el **archivo de usuario 'I / O' Devices** .

- 6) Falló la copia de instancias '**Servo**' debido a una implementación '**Servo :: Servo (Servo ©)**' copy-constructor defectuosa .
- 7) Los valores fuera de rango '**Servo.writeMicroseconds()**' se detectaron correctamente como un error, pero los valores límite indicados que acompañan al texto del mensaje de error eran incorrectos.
- 8) No se aceptó una velocidad de transmisión en baudios legal de 115200 cuando se carga desde un archivo de texto '**I / O Devices**' .
- 9) Los conflictos de pines eléctricos causados por un dispositivo Analógico Deslizante conectado no siempre se detectaron.
- 10) En raras ocasiones, pasar un puntero de cadena defectuoso (con la cadena 0-terminator faltante) a una función '**String**' podría causar que UnoArduSim se bloquee.
- 11) El **Panel de Código** podría resaltar la línea de error de Análisis actual en el módulo de programa **incorrecto** (cuando se usó '**#include**').
- 12) Al cargar un archivo de dispositivos '**I / O**' que tenía un dispositivo que conduciría (incorrectamente) contra 'Uno' pin 13, se colgó el programa en el mensaje emergente de mensaje de error.
- 13) UnoArduSim había permitido erróneamente al usuario pegar caracteres no hexadecimales en las ventanas de búfer TX expandidas para SPISLV e I2CSLV.
- 14) Las inicializaciones de línea de declaración fallaron cuando el valor del lado derecho era el valor '**return**' de una función miembro-función (como en '**int angle = myservol.read();**').
- 15) '**static**' member variables que tienen explícitos '**ClassName ::**' prefijos no fueron reconocidos si aparecían al principio de una línea (por ejemplo, en una asignación a una variable base- 'class'),
- 10) Llamar a '**delete**' en un puntero creado por '**new**' solo se reconoció si se utilizó la notación de paréntesis de funciones, como en '**delete (pptr)**' .
- 17) La implementación de UnoArduSim de '**noTone()**' insiste incorrectamente en que se proporcione un argumento pin.
- 18) Los cambios que aumentaron global 'RAM' bytes en un programa que usaba variables '**String**' (a través de **Editar / Examinar** o **Archivo | Cargar**), podrían provocar daños en ese espacio global 'Uno' debido a la eliminación del montón de los objetos '**String**' que pertenecen al programa anterior al usar (incorrectamente) el montón perteneciente al nuevo programa. En algunas circunstancias, esto podría provocar un bloqueo del programa. Aunque un segundo **Cargar** o **Análizar** resolvió el problema, este error por fin se ha solucionado.
- 19) Los valores de retorno para '**Wire.endTransmission()**' y @ '**Wire.requestFrom()**' se habían quedado atascados en 0; ahora se han corregido.

V1.7.2 - Feb.2017

- 1) Las interrupciones en el pin 2 también fueron activadas (inadvertidamente) por la actividad de la señal en el pin 3 (y viceversa).

V1.7.1 - Feb.2017

- 1) Función '**delayMicroseconds()**' estaba produciendo un retraso en -seconds **mili.** (1000 veces demasiado grande).
- 2) La conversión explícita de una variable '**unsigned**' a un tipo de entero más largo arrojó un resultado incorrecto ('**signed**').
- 3) Los literales hexadecimales mayores que 0 x7FFF son ahora '**long**' por definición, por lo que ahora generarán '**long**' expresiones aritméticas resultantes en las que se involucrarán.
- 4) Un error inadvertidamente introducido por V1.7.0 impidió la conversión alternativa de estilo C ++ de literales numéricos (por ejemplo, '**(long) 1000 * 3000**' no fue aceptado).

- 5) 'Serial' ya no ocupa muchos bytes en 'Uno' RAM si el programa de usuario nunca lo necesita.
- 6) Las variables globales declaradas por el usuario ya no ocupan espacio en 'Uno' RAM si realmente nunca se utilizan.
- 7) Las variables individuales declaradas como **miembros** 'const' ,, 'enum' y los punteros a los literales de cadenas ya no ocupan espacio en 'Uno' RAM (para estar de acuerdo con la compilación de Arduino),
- 8) Los bytes de RAM necesarios para las bibliotecas '#include' builtin ahora coinciden estrechamente con los resultados de la compilación condicional de Arduino.
- 9) Utilizando 'new' en un puntero, la línea de declaración real había fallado (solo **funcionó una** asignación 'new' posterior al puntero).
- 10) Se corrigió un error por el cual una presentación "pendiente" de un directorio de disco SD podía provocar la suspensión de un programa.

V1.7.0 - Dic.2016

- 0) Se han solucionado varios problemas con el manejo de las interrupciones del usuario:
 - a) Interrumpe los bordes 0 y 1 que ocurrieron durante una función Arduino que bloquea mientras espera (como 'pulseIn()' , 'ShiftIn()' , 'spiTransfer()' , 'flush()' y 'write()') había causado una falla en el flujo de ejecución en el retorno de interrupción
 - b) Varias copias de las variables locales de cualquier función interrumpida habían estado apareciendo en el **Panel de Variables** (una copia por interrupción-retorno) y esto fue corregido en V1.6.3, pero los otros problemas de interrupción permanecieron).
 - c) Función 'delayMicroseconds()' no estaba creando ningún retraso si se llama desde dentro de una rutina de interrupción de usuario.
 - d) Las llamadas a funciones de bloqueo como 'pulseIn()' desde **dentro de** una rutina de interrupción no han estado funcionando.
- 1) Un error introducido en V1.6.3 causó la pérdida de valor de actualización en el **Panel de Variables** mientras se ejecuta cuando en realidad estaban cambiando los valores (esto ocurrió sólo después de dos o más acciones **Detener** o el menú de usuario **VarActualizar**). Además, cuando se ejecutó Ejecutar-Hacia después de que se **haya activado** Allow Reduction , ocasionalmente el **Panel de Variables** no se volvió a dibujar (por lo que los valores antiguos y las variables locales pueden haber aparecido allí hasta el siguiente **Paso**).
- 2) El comportamiento resaltado del **Panel de Código** del comando **Paso-Sobre** podría parecer engañoso en 'if() - else' cadenas - que ahora se ha corregido (aunque la funcionalidad real de pasos era correcta).
- 3) Función 'pulseIn()' había configurado indebidamente el tiempo de espera en milisegundos en lugar de microsegundos; también reiniciaba incorrectamente el tiempo de espera cuando se veían por primera vez las transiciones a niveles inactivos y activos.
- 4) El uso de literales HEX entre 0x8000 y 0xFFFF en asignaciones o aritmética con 'long' variables enteras dio resultados incorrectos debido a la extensión de signo no verificada.
- 5) Pasar, o regresar, a 'float' de cualquier tipo 'unsigned' integer que tenga un valor con MSB = 1 dio resultados incorrectos debido a una interpretación 'signed' **incorrecta** .

- 6) Todas las **funciones** `'bit _()'` ahora también aceptan operaciones en variables de tamaño `'long'`, y UnoArduSim comprueba las posiciones de bits no válidas (que quedarían fuera del tamaño variable).
- 7) Una entrada no válida en la casilla de edición Pulse (ancho) en un dispositivo 'PULSER' causó daños en el valor de 'Period' (hasta que el siguiente usuario 'Period' edit lo corrige).
- 8) La eliminación de un dispositivo 'PULSER' o 'FUNCGEN' utilizando el menú Configurar no eliminaba la señal periódica del pin que había estado manejando (ya no se requiere un reinicio).
- 9) La capacidad de inicializar una matriz 1-D `'char'` con una cadena entrecomillada faltaba (por ejemplo, `'char strg[] = "hello";'`).
- 10) La visualización hexadecimal en las ventanas expandidas 'SERIAL' o 'SFTSER' Monitor mostró el carácter incorrecto más significativo para los valores de bytes mayores que 127.
- 11) Las ventanas de Forma de Onda no reflejaban los cambios programáticos del usuario realizados por `'analogWrite()'` cuando el nuevo valor era 0% o 100% de ciclo de trabajo.
- 12) La implementación de `'Serial.end()'` ahora ha sido arreglado
- 13) Un archivo **myUnoPrefs.txt** con más de 3 palabras en una línea (o espacios en el nombre del archivo **'I / O' Devices**) podría causar un bloqueo debido a un puntero interno defectuoso.
- 14) La línea final de un archivo de **dispositivos 'I / O'** no se aceptó si no ***finalizaba con un avance de línea***.
- 15) Agregar más de cuatro controles **Deslizantes Analógicos** causó un error silencioso que sobrescribió los indicadores LED 'I / O' Dispositivo
- 10) Comenzando con V1.6.0, las muestras de forma de onda analógica para la ***primera mitad*** de cada forma de onda ***triangular*** eran todas ***cero*** (debido a un error en el cálculo de la tabla de forma de onda).
- 17) Hacer una **ejecución** recurrente cuando está en una línea de punto de interrupción ya no requiere múltiples clics por avance.
- 18) El analizador no aceptó pasar expresiones de direcciones a un parámetro de matriz de funciones.
- 19) Las funciones recursivas que devolvieron expresiones que contenían referencias de punteros o matrices dieron resultados incorrectos debido a las marcas "listas para restablecer" en esas expresiones de componentes.
- 20) Llamar `'class'` member-functions a través de ***cualquier variable de puntero de objeto o expresión de puntero*** no funcionaba.
- 21) Las funciones de usuario que devolvían objetos por valor solo devolvían correctamente su valor en su primera llamada a función ***si*** devolvían un objeto construido sin nombre (como `'String ("perro")'`) - en llamadas posteriores la devolución se saltaba debido a un atasco "listo" "Bandera".
- 22) No había habido ninguna protección para evitar el comando **Ventanas | 'Serial' Monitor para** agregar un nuevo dispositivo `'Serial'` cuando en realidad no había espacio para eso.

23) Si agregar un dispositivo de pin fijo (como 'SPISLV') causaba un mensaje emergente de conflicto de pines, el redibujado del **Panel del Banco de Laboratorio** podría mostrar un dispositivo duplicado "fantasma" superpuesto al dispositivo 'I / O' más a la derecha (hasta el siguiente redibujado) .

24) Se corrigieron algunos problemas con sonidos 'PIEZO' no fiables para señales de pin no periódicas.

25) Las variables '**PROGMEM**' también deben declararse explícitamente como '**const**' para estar de acuerdo con Arduino.

26) "No Heap Space" se marcó incorrectamente como un error de ejecución cuando '**SD.open()**' no pudo encontrar el archivo con nombre, o '**openNextFile()**' llegó al último archivo en el directorio.

27) Un error del analizador había aceptado incorrectamente un **corsé de** cierre fuera de lugar '**}**' .

28) Se corrigió un error con la eliminación de **Panel de Variables** sobre el retorno miembro-objeto-constructor (el error se aplica solo a los objetos que contienen otros objetos como miembros).

V1.6.3- Sept. 2016

1) Las variables locales de cualquier función interrumpida no se eliminaron del **Panel de Variables** en la entrada de la función de interrupción, lo que hace que aparezcan múltiples copias en el retorno de la función de interrupción (y un posible error de ejecución o falla).

2) Las ventanas de Forma de Onda no reflejaban cambios programáticos en '**analogWrite()**' a un nuevo ciclo de trabajo de 0% o 100%.

3) La visualización hexadecimal en la ventana expandida 'SERIAL' o 'SFTSER' Monitor mostró el carácter MSB incorrecto para valores de bytes mayores que 127.

V1.6.2- Sept. 2016

1) Las llamadas de función realizadas con el número o tipo incorrecto de argumentos no habían generado un mensaje de error de Análisis apropiado (solo apareció el mensaje genérico "identificador no válido").

2) El botón de reinicio de la **Barra de Herramientas** ahora funciona de forma idéntica al botón de reinicio de la placa 'Uno'.

3) El texto de error de análisis ya no se corta después de 16 caracteres sin mostrar puntos suspensivos.

V1.6.1 - agosto de 2016

1) En V1.6 una versión de la placa 'Uno' en el archivo **myArduPrefs.txt** que difería del valor predeterminado de la versión 2 provocó una excepción al inicio (debido a un evento pin 13 no inicializado).

2) Cambiar el valor de una variable haciendo doble clic en el **Panel de Variables** podría causar ventanas emergentes de error "sin asignación de memoria" (para programas con cualquier '**class**' definido por el usuario).

3) '**SoftwareSerial**' no permitió el acceso a '**write (char * ptr)**' y '**write (byte * ptr, int size)**' funciones debido a una detección de sobrecarga de función defectuosa.

4) Se corrigió el problema con la inclusión automática del archivo ".cpp" correspondiente para .h "library-type" '**#include**' .

V1.6- junio de 2016

- 1) En V1.5, se ha perdido la sangría automática en la tecla "Volver" en **Editar / Examinar** (al ingresar una nueva línea).
- 2) Ahora se ha agregado la detección de conflictos de pines con los dispositivos externos de 'I / O' de conducción fuerte en 'Serial' pin 1, en los pines SPI SS, MOSI y SCK, en los pines I2C SCL y SDA (todo cuando el 'begin()' correspondiente se llama y en cualquier pin 'SoftwareSerial' TX declarado).

V1.5.1 - junio de 2016

- 1) En V1.5, los nuevos colores Highlight Highlight adaptables al tema no se restablecían correctamente cada vez que se abría **Editar / Examinar**, y así (con un tema de fondo blanco) eran correctos cada dos por tres.
- 2) Las **sensibilidades** de interrupción '**RISING**' y '**FALLING**' habían sido opuestas a la polaridad de disparo real.

V1.5 - mayo de 2016

- 1) Un error introducido en V1.4.1 impedía pasar literales de cadena desnudos a las funciones miembro que esperaban un objeto '**String**', como en '**mystring1.startsWith ("Hey")**'.
2) Un error en el SD original la implementación de UnoArduSim solo permitía el acceso SD usando llamadas a '**read()**' y '**write()**' (se impedía el acceso a través de las funciones '**Stream**').
3) 'R = 1K' Los interruptores deslizantes no se redibujaron correctamente cuando se movió el control deslizante.
4) "**Cancelar**" en el cuadro de diálogo Confirmar-Guardar archivo debería haber evitado la salida de la aplicación.
5) Una cita de cierre faltante o un cierre '>' -parenthesis en un archivo de usuario '**#include**' causaría un bloqueo.
6) Se corrigió un error en el resaltado de sintaxis de '**String**' y user '**class**' o '**struct**', y el resaltado ampliado para incluir llamadas a la función del constructor.
7) Se corrigieron problemas menores en **Editar / Examinar** con cambios de texto / resaltado y el botón Deshacer.

V1.4.3 - abril de 2016

- 1) Usando **Configurar | Dispositivos 'I / O'** para agregar dispositivos nuevos y luego quitar uno de esos dispositivos recién agregados podrían causar un bloqueo al reiniciarse o que otro dispositivo deje de funcionar.
- 2) La modificación de una variable '**String**' haciendo doble clic en el **Panel de Variables** falló (la nueva '**String**' se leyó incorrectamente).
- 3) Los cambios de pin en los dispositivos 'FUNCGEN' y 'PULSER' no se reconocieron hasta que se realizó un reinicio por primera vez.

V1.4.2 - Mar. 2016

- 1) V1.4.1 había **introducido** un desafortunado error de Análisis que impedía las asignaciones que implicaban cualquier objeto '**class**' (incluido '**String**').

- 2) Una corrección de error incompleta realizada en V1.4.1 causó que los valores '**unsigned**' de tipo '**char**' se imprimieran como caracteres ASCII en lugar de como valores enteros.
- 3) Los argumentos de llamada de función de expresión de miembros complejos no siempre se reconocieron como coincidencias de función-parámetro válidas.
- 4) **Todos los** literales y expresiones enteros se clasificaron con demasiada generosidad (a '**long**') y, por lo tanto, la ejecución no reflejó los desbordamientos **reales** (a negativos) que pueden ocurrir en Arduino en las operaciones de agregar / multiplicar con **valores de 'int' size**.
- 5) Las expresiones que incluían una combinación de '**signed**' y '**unsigned**' integer tipos no siempre se manejaban correctamente (el '**signed**' value se vería incorrectamente como '**unsigned**').
- 6) En los casos de conflicto de pines, los mensajes de error "value =" podrían mostrar valores de pin obsoletos incluso después de un restablecimiento de un conflicto anterior que el usuario ya había borrado.

V1.4.1 - enero de 2016

- 1) Las llamadas a '**print (char)**' ahora se imprimen correctamente como caracteres ASCII (en lugar de valores numéricos).
- 2) La respuesta a la interrupción ahora está habilitada por defecto cuando se llama a '**attachInterrupt()**' , por lo que ya no es necesario en su '**setup()**' para llamar a la función de habilitación '**interrupts()**' .
- 3) Múltiples '**#include**' instancias de archivos de usuario desde dentro de un archivo ahora se manejan correctamente.

V1.4 - Dic. 2015

- 1) Un error **antiguo** marcó incorrectamente una condición de división por **cero** cuando se divide por un valor fraccionario menor que la unidad.
- 2) Se arregló '**SoftwareSerial**' (que se **anuló** inadvertidamente mediante una comprobación de validación de miembro '**class**' - agregada en las versiones de V1.3).
- 3) Las llamadas a la función de fin de línea con un punto y coma faltante no se detectaron y provocaron que Análisis omitiera la siguiente línea.
- 4) Un archivo de texto de **dispositivos 'I / O'** mal formateado dio un mensaje de error incorrecto.
- 5) Se ha corregido el resaltado de error de Análisis de la línea incorrecta (adyacente) en expresiones y declaraciones de varias líneas
- 6) Pruebas lógicas de punteros usando '**not**' (**!**) el operador estaba invertido.

V1.3 - Oct. 2015

- 1) El manejo interno incorrecto de las variables del bloc de notas causó ocasionales " **excedió la profundidad máxima de anidamiento del bloc de notas** ". Errores de análisis.
- 2) Paréntesis *dentro de comillas simples* , llaves, puntos y comas, p arenteses dentro de las cadenas entrecomilladas, y los caracteres escapados fueron manejados incorrectamente.

- 3) Una matriz con una dimensión vacía y sin lista de inicialización provocó un bloqueo de RESETAR, y las matrices con un solo elemento no se rechazaron (y causaron su interpretación errónea como un puntero inicializado incorrectamente).
- 4) Los errores de Análisis a veces a veces resaltan la línea incorrecta (adyacente).
- 5) Pasar un puntero a un 'const' a una función que acepta un puntero a '**const**' no se ha permitido (en lugar de al revés).
- 6) Las expresiones del inicializador heredaban de forma incorrecta los calificadores '**PROGMEM**' de la variable inicializada
- 7) **Las** variables declaradas '**PROGMEM**' tuvieron su tamaño de byte contado incorrectamente **dos veces en** contra de su asignación de memoria 'Flash' durante el Análisis.
- 8) Tecleando 'x' caja de edición de un 'I2CSLV' a veces causaría un bloqueo debido a '**sscanf**' bug.
- 9) Cargar un nuevo programa con un nuevo archivo 'I / O' **Devices** en su directorio podría causar conflictos de pin irrelevantes con las **antiguas** direcciones de pin.
- 10) El manejo de caracteres en serie escapado se aplicó incorrectamente a las secuencias de caracteres recibidas, en lugar de transmitidas, en la ventana (más grande) 'Serial Monitor' buffers.
- 11) '**while()**' y '**for()**' bucles con cuerpos completamente vacíos, como '**while (verdadero);**' o '**for(int k = 1; k <= 100; k ++);**' pasó el Análisis (con un mensaje de advertencia) pero falló en el tiempo de ejecución.

V1.2 - Jun 2015

- 1) Las funciones de usuario más simples que realizan llamadas a '**digitalRead()**' o a '**analogRead()**' o '**bit()**' podrían haber dañado su (muy primera) variable local declarada (si corresponde) debido a la función asignada insuficiente scratchpad espacio (si solo se asignaron dos bytes de scratchpad al comienzo de la pila de funciones). Cualquier expresión numérica en absoluto dentro de una función es suficiente para causar una asignación de 4 bytes en el área reutilizable, y así evita este problema. Este desafortunado error ha existido desde el lanzamiento original V1.0.
- 2) Las funciones que son '**void**' con una temprana explícita '**return**', y non- '**void**' funciones con más de un '**return**' comunicado, verían la ejecución de caída a través de la corsé de cierre (si se ha alcanzado).
- 3) Todos los enunciados '**return**' dentro de '**if()**' contextos en los que faltaban corsés llevaron a un objetivo defectuoso de devolución al llamante.
- 4) 'PULSER' y 'FUNCGEN' pulse-widths o períodos de valor 0 podrían causar un bloqueo (0 ahora no está permitido).
- 5) Donde no había llaves '**else**' continuations después de '**if()**' no funcionó si siguieron un '**break**', '**continue**', o '**return**'.
- 6) Cuando se realizaron múltiples 'enum' user-declarations, las constantes definidas en todos menos en el primer 'enum' generó error " 'enum' mismatch" Error de análisis (este error se introdujo en V1.1).
- 7) Un identificador nulo para el último parámetro de un prototipo de función provocó un error de Análisis.
- 8) **Los** puntos de inflexión **Ejecutar-Hacia** establecidos en líneas complejas no siempre se manejaban correctamente (y por lo tanto se podían perder).

- 9) 'HardwareSerial' y 'SoftwareSerial' usó un buffer privado pendiente de TX de implementación privada que no se eliminó en Reiniciar (por lo que podrían aparecer los caracteres de la última vez).
- 10) El Análisis no pudo verificar el volteo ilegal de bits de 'float' , ni la aritmética del puntero con operadores ilegales.

V1.1 - Mar 2015

- 1) Los índices de matriz que eran variables 'byte' o 'char' size causaban compensaciones de matriz incorrectas (si una variable adyacente contenía un byte alto no 0).
- 2) Las pruebas lógicas de punteros probaron el valor apuntado para distinto de cero en lugar del valor del puntero.
- 3) Todas las declaraciones 'return' incrustadas dentro de 'for()' o 'while()' loops fueron mal manejadas.
- 4) Las listas de inicialización agregada para matrices de objetos u objetos que contienen otros objetos / matrices o listas de inicialización completamente vacías no se manejaban correctamente.
- 5) Acceso de 'enum' member values usando un " enumname. " el prefijo no era compatible.
- 6) La inicialización de la línea de declaración de una matriz 'char[]' con un literal de cadena entrecomillado no funcionaba.
- 7) Una matriz que se pasa a una función sin inicialización previa se marcó incorrectamente con un error "usado pero no inicializado".
- 8) Las expresiones de puntero que involucran nombres de matriz fueron mal manejadas.
- 9) Los parámetros de función declarados como 'const' no fueron aceptados.
- 10) La ventana de la Forma de Onda Analógico no mostró señales PWM ('servo.write()' y 'analogWrite()').
- 11) Las funciones miembro a las que se accedió a través de un puntero a objeto dieron acceso a miembros defectuosos.
- 12) Las formas de onda no se actualizaban cuando se alcanzaba un punto de interrupción **Ejecutar-Hacia** .
- 13) El modelo de asignación de registros podría fallar cuando un parámetro de función se utilizara directamente como argumento para otra llamada de función

V1.0.2 - agosto de 2014

Ordenación fija de los pines A0-A5 en el perímetro de la placa 'Uno' .

V1.0.1 - junio de 2014

Se corrigió un error que truncaba las correcciones de edición que eran más de tres veces la cantidad de bytes en el programa original.

V1.0 - primer lanzamiento de mayo de 2014

Cambios / Mejoras

V2.1 Mar. 2018

- 1) Los valores mostrados en el **Panel de Variables** ahora se actualizan a una velocidad de 16 veces por segundo o 4 veces por segundo (cuando la opción VarActualizar | Mínima está en efecto), y la opción VarActualizar | Permitir Reducción ha sido eliminada.
- 2) Las operaciones que se dirigen solo a una parte de los bytes del valor de una variable (como las realizadas mediante punteros) ahora hacen que el cambio en el valor de dicha variable se refleje en la pantalla del **Panel de Variables**.

V2.0.1 Enero 2018

- 1) Se han agregado las funciones de Arduino sin documentar '`exp()`' y '`log()`'.
- 2) Los dispositivos 'SERVO' ahora se pueden hacer de rotación continua (por lo que el ancho de pulso controla la velocidad en lugar del ángulo).
- 3) En **Editar / Examinar**, un corchete de cierre '`}`' ahora se agrega automáticamente cuando escribe un corchete de apertura '`{`'.
- 4) Si hace clic en '**X**' de la barra de título de la ventana **Editar / Examinar** para salir, ahora tiene la posibilidad de abortar si ha modificado, pero no ha guardado, el archivo de programa que se muestra.

V2.0 Dic. 2017

- 1) La implementación se ha transferido a QtCreator, por lo que la GUI tiene algunas diferencias visuales menores, pero no tiene diferencias funcionales aparte de algunas mejoras:
 - a) Se ha mejorado la mensajería de línea de estado en la parte inferior de la ventana principal, y dentro del cuadro de diálogo **Editar / Examinar**, y se ha agregado una codificación de color de resaltado.
 - b) El espacio vertical asignado entre el **Panel de Código** y el **Panel de Variables** ahora se puede ajustar a través de una barra separadora que puede arrastrarse (pero no visible) en su borde compartido.
 - c) Los valores de 'I' / 'O' dispositivo caja de edición ahora solo se validan una vez que el usuario ha movido el puntero del mouse fuera del dispositivo, esto evita cambios automáticos incómodos para aplicar valores legales mientras el usuario está escribiendo.
- 2) UnoArduSim ahora admite varios idiomas a través de **Configurar | Preferencias**. El inglés siempre se puede seleccionar, además del idioma de la configuración regional del usuario (siempre que haya un archivo de traducción personalizado *.qm para ese idioma en la carpeta UnoArduSim '**translations**').
- 3) Ahora se ha modificado el sonido para usar la API de audio Qt. Esto ha requerido silenciamiento en ciertas circunstancias para evitar interrupciones de sonido molestas y clics durante los retrasos operacionales de la ventana del sistema operativo más largos causados por clics normales del usuario. Consulte la sección Modelado de sonidos para obtener más información. detalle sobre esto.
- 4) Como conveniencia del usuario, los espacios en blanco se usan ahora para representar un valor de 0 en los cuadros de edición de conteo de dispositivos en **Configurar | 'I' / 'O' Devices** (por lo que ahora puede usar la barra espaciadora para eliminar dispositivos).
- 5) El calificador sin escala (U) ahora es opcional en los dispositivos 'PULSER', 'FUNCGEN' y '1SHOT' (se supone que es el predeterminado).
- 6) UnoArduSim ahora permite (además de los valores numéricos literales) '`const`' variables de valores enteros, y '`enum`' miembros, que se usarán como dimensiones en las declaraciones de matriz, siempre que esas fuentes se inicialicen explícitamente utilizando un literal numérico de valores enteros constantes.

- 7) Después de que el mouse lo ingrese por primera vez, un dispositivo **PUSH** producirá rebote de contacto (durante 1 milisegundo) cada vez que presiona la tecla de **barra espaciadora** (pero **no** para clics del mouse, ni para ninguna otra pulsación de tecla).
- 8) Clics adicionales de (**Encontrar | Establecer el Texto de Búsqueda**) ahora selecciona el siguiente palabra del texto de la línea resaltada en el Panel activo (el **Panel de Código** o el **Panel de Variables**).
- 9) El comando `'goto'` Arduino ahora está marcado como "no compatible" en UnoArduSim.
- 10) Funciones `'max()'` y `'pow()'` ahora se han incluido en la conveniencia lista de integradas dentro del cuadro de diálogo **Editar / Examinar**.

V1.7.2 - Feb. 2017

- 1) La opción de color azul (B) se ha agregado para dispositivos LED.

V1.7.1 - Feb. 2017

- 1) Los sufijos `'L'` y/o `'U'` ahora se aceptan al final de las constantes literales numéricas (para definirlos como `'long'` y/o `'unsigned'`), y (`0b` o `0B` prefijadas) las constantes binarias ahora también son aceptadas. Cualquier constante numérica completamente decimal que **comience con '0'** ahora se considera un valor **octal**. (para estar de acuerdo con Arduino).
- 2) Cuando se ejecuta en un bucle cerrado del que no hay escape (por ejemplo `'while (x); x ++;'` donde `x` siempre es verdadero), al hacer clic en **Detener** ahora se detiene la ejecución del programa (y en esa línea de programa defectuosa).

V1.7.0 – Dic. 2016

- 1) Se agregó una nueva función de **Barra de Herramientas** que muestra los bytes libres de RAM durante la ejecución del programa (teniendo en cuenta el total de bytes utilizados por las variables globales, las asignaciones de pila y las variables de pila locales).
- 2) Ahora las funciones de interrupción del usuario también pueden llamar funciones de bloqueo de Arduino como `'pulseIn()'` (pero esto solo debe usarse con precaución, ya que la función de interrupción no regresará hasta que se complete la función de bloqueo).
- 3) Las interrupciones de usuario ya no se desactivan durante las operaciones de lectura de secuencia bloqueadas, por lo que el comportamiento ahora coincide con la operación real de lectura de secuencia de Arduino.
- 4) Ahora puede entrar y salir de bloquear las funciones de Arduino que se pueden interrumpir (como `'delay()'` y `'pulseIn()'`), y los mensajes de la **Barra de Estado** se han aumentado para mostrar cuándo ha tocado un punto de interrupción de interrupción dentro de dicha función (o cuando hace clic en Detener cuando la ejecución se encuentra actualmente dentro de dicha función).
- 5) Se ha agregado un nuevo comando **Ejecutar-Hasta** (y un elemento **Tool-Bar**): haga clic en cualquier **variable del Panel de Variables** (puede ser simple, una matriz u objeto agregado, o un elemento de matriz o miembro de objeto) para resaltarla. luego haga **Ejecutar-Hasta**: la ejecución se congelará en el siguiente **acceso de escritura** dentro de esa variable agregada, o en esa única ubicación.
- 6) Cuando la ejecución se congela después de una acción **Paso**, **Ejecutar-Hacia**, **Ejecutar-Hasta** o **Ejecutar-luego-Detener**, el **Panel de Variables** ahora resalta la variable que corresponde a la **(s) ubicación (es) de dirección que se modificó** (si corresponde) por la **última instrucción durante esa ejecución**: si esa ubicación está actualmente oculta dentro de una matriz u objeto no expandido, al hacer clic para expandirla, el elemento o miembro de la última modificación quedará resaltado.

7) El usuario ahora puede mantener un control especial sobre el valor de una variable / elemento / elemento del Panel de Variable específico durante la ejecución: haga doble clic en esa línea en el Panel de Variables para abrir la ventana **Editar / Monitorear Valor de Variable** , luego haga una de las Comandos **Ejecutar o Paso** : el valor mostrado se actualizará durante la ejecución de acuerdo con las mismas reglas que rigen las actualizaciones en el Panel de Variables. Después de detener la ejecución, se le permite ingresar un nuevo valor y **Aceptarlo** antes de reanudar la ejecución (y puede **Revertir** al valor de **aceptación** previa si cambia de opinión antes de esa fecha).

8) Las teclas aceleradoras F4-F10 se han configurado para coincidir con los comandos de la **Barra de Herramientas** del menú Ejecutar (de izquierda a derecha).

9) Además de hacer doble clic en ellos, al hacer clic con el botón derecho en los dispositivos 'SERIAL', 'SFTSER', 'SPISLV', 'I2CSLV' también aparecerá una ventana de bytes / caracteres TX / RX de mayor tamaño (y en 'SD_DRV', una ventana de monitoreo de archivos).

10) El cuadro de edición de TX en 'SERIAL' o 'SFTSER' ya no está deshabilitado durante una transmisión activa de caracteres (por lo que ahora puede agregar o reemplazar lo que está allí), pero un botón de retorno de carro (o 'Send') hace clic en la ventana 'SerialMonitor' child asociada) se ignorará hasta que la transmisión vuelva al estado inactivo una vez más (los caracteres ahora se muestran en *cursiva* cuando la transmisión está lista para comenzar, o está activa). Además, el usuario ahora recibe una advertencia en una transmisión en serie '**begin()**' si ya habían comenzado antes las transmisiones del dispositivo conectado (ahora en curso), ya que entonces no habría sincronización de encuadre, lo que provocaría errores de recepción.

11) El retraso '**loop()**' agregado por defecto se ha aumentado de 250 microsegundos a un milisegundo para no quedarse tan atrás en tiempo real cuando el usuario **omite** incluir algo de '**delay()**' (explícito o natural) en algún lugar dentro de '**loop()**' o dentro de una función que llama.

12) Matrices y tipos simples ahora se han agregado al soporte para la instrucción '**new**' de asignación de **pila** .

13) Se han agregado comprobaciones más extensas (y mensajes de error asociados) para los accesos de direcciones fuera de límites del programa de usuario (es decir, fuera de 'Uno' RAM, o fuera de 'Flash' para '**PROGMEM**' accesses).

14) Los valores del puntero en el **Panel de Variables** ahora se parecen más a los valores reales del puntero Arduino.

15) El archivo myArduPrefs.txt del usuario ahora se carga en cada Archivo | Carga , no solo en el lanzamiento de UnoArduSim.

10) Ahora se marca un error de Análisis al intentar '**attachInterrupt()**' a una función de usuario que no es '**void**' return, o que tiene parámetros de función, o que no ha sido declarado en algún lugar antes de '**attachInterrupt()**' .

17) '**static**' member-variables se muestran ahora en la parte superior del **Panel de Variables** como globales, en lugar de aparecer dentro de cada instancia de un objeto (expandido).

18) Función '**availableForWrite()**' se ha agregado a la implementación de **Serial** .

19) Todos los '**PROGMEM**' , '**typedef**' like '**prog_char**' y '**prog_int16**' **especiales** ahora se han eliminado (han quedado obsoletos en Arduino).

20) Se han mejorado los mensajes de error para los errores de Análisis causados por tipos de declaraciones mal escritos o no válidos.

21) El tamaño máximo permitido del programa se ha aumentado.

V1.6.3- septiembre de 2016

1) Añadido un mensaje de error de análisis mejorado cuando '`attachInterrupt()`' se refiere a una interrupción-función que no fue *un prototipo anterior*.

2) Se agregó un mensaje de error de Análisis mejorado para listas de inicialización de matrices multidimensionales.

V1.6.2 - septiembre de 2016

1) Se agregó un control de edición de **Encontrar-Texto** a la **Barra de Herramientas** para agilizar la búsqueda de texto (en el Panel de Código y en el **Panel de Variables**).

2) El botón Reiniciar de la **Barra de Herramientas** ahora funciona de forma idéntica al botón 'Uno' placa Reiniciar.

V1.6.1 - Ago de 2016

Se ha añadido un cheque para evitar la carga duplicado y el análisis de '`ya anterior#`' `incluir` archivos,.

V1.6 - junio de 2016

1) Se agregó un nuevo dispositivo '1SHOT' (one-shot) 'I / O' que genera un pulso después de un retardo elegido desde el borde de la señal de disparo de la polaridad seleccionada.

2) Se agregó una nueva función para hacer que los valores del cuadro de edición de 'I / O' Dispositivo sean fácilmente *escalables* durante la ejecución arrastrando un control deslizante 'I / O ____ S' Scaler global en la **Barra de Herramientas** principal (simplemente escriba una sola letra 's' o 'S' después de un valor para indicar escalamiento).

V1.5.1 - junio de 2016

1) Ahora se ha agregado soporte para las funciones de la biblioteca EEPROM '`update()`' , '`put()`' y '`get()`' , y para el acceso de bytes a través de la notación de matriz, por ej. '`EEPROM [k]`' .

2) **Permitir colapso automático(-)** se ha agregado al menú **VarActualizar** para permitir el control explícito sobre si las matrices / objetos expandidos se colapsarán automáticamente cuando la ejecución se esté quedando atrás en tiempo real.

3) Los personajes de un '`String`' variable ahora también se puede acceder a través de la notación de matriz, p.ej '`mystring [k]`' .

V1.5 - mayo de 2016

1) **Ver / Editar** ahora tiene acceso directo ctrl-E, y tiene un nuevo botón para **Compilar** (ctrl-R), más un cuadro de error de Análisis incorporado, para permitir la prueba de ediciones sin necesidad de cerrar la ventana.

2) **Ver / Editar** ahora ahora también es compatible con **Rehacer** , y tiene un nuevo botón **Guardar** (ctrl-S) (equivalente a **Aceptar** más un **ahorro de** ventana principal posterior), y ahora ofrece una opción de tamaño de TAB (una nueva preferencia que se puede guardar usando **Configurar | Preferencias**).

- 3) Todas las cajas de edición editables ahora siguen los colores del tema del sistema operativo Windows elegido y, por el contrario, todas las cajas de edición de 'RECV' de solo lectura usan texto blanco sobre fondo negro. Los **colores** Editar / Examinar **fondo y resaltado de sintaxis ahora también se adaptan al tema elegido**.
- 4) UnoArduSim ahora permite una elección de fuente: esa elección y su tamaño se han movido a **Configurar | Preferencias** (por lo que se pueden guardar en el archivo **myArduPrefs.txt**).
- 5) Los valores literales binarios predefinidos de Arduino (como **'B01011011'**) ahora están permitidos.
- 6) Las secuencias de caracteres citadas hexadecimales, octales y de 4 dígitos Unicode ahora se pueden usar como literales numéricos.
- 7) Después de hacer un clic inicial con el mouse en un teclado 'PUSH', el usuario puede usar una tecla presionada (cualquier tecla) para presionar los contactos del botón.
- 8) **Editar / Examinar** ahora libera su estado inicial de solo lectura temporal (y elimina el resaltado de la línea inicial seleccionada) después de una breve indicación visual flash.
- 9) UnoArduSim ahora comprueba si hay múltiples **conflictos** **'Stepper'** y **'Servo'** pin, es decir, el programa de usuario defectuoso intenta conectarse a pines ya adjuntos a las **variables** **'Stepper'** o **'Servo'** **anteriores** .
- 10) Un error de Análisis causado por un lado izquierdo o derecho faltante a un operador (faltando una expresión o variable LHS o RHS) ahora genera un mensaje de error claro.
- 11) La variable de miembro **'String'** class **'flags'** **no utilizada** se ha eliminado para que coincida con Arduino V1.6.6. Un objeto **'String'** ahora ocupa 6 bytes (además de la asignación del montón de caracteres).

V1.4.2 - Mar 2016

- 1) Las funciones definidas hacia adelante (es decir, aquellas sin declaración de prototipo antes de su primera llamada) ahora solo generan advertencias (no errores de análisis) cuando la definición posterior de la función tipo de retorno no coincide con el tipo inferido desde su primer uso.
- 2) Las matrices que tienen una dimensión igual a 1 ya no son rechazadas (para estar de acuerdo con las reglas estándar de C ++).
- 3) los cuadros de edición ya no están configurados en negro sobre fondo blanco; ahora adoptan la paleta establecida por el tema del SO de Windows en uso.
- 4) **'SERIAL'**, **'SFTSER'**, **'SPISLV'**, y **'I2CSLV'** dispositivo expandido Las ventanas de Monitor (abiertas haciendo doble clic) ahora adoptan el color de fondo de su dispositivo padre **'I / O'**.

V1.4 - Dic 2015

- 1) Se ha agregado la funcionalidad **'Stepper.h'** library y los dispositivos **'I / O'** asociados.
- 2) **Todos los valores y configuraciones del dispositivo 'I / O'** (además de los pines seleccionados) ahora también se guardan como parte del archivo de texto del usuario **'x/ O'** elegido para su posterior recarga.

3) El color del LED 'I / O' Dispositivo ahora se puede establecer como rojo, amarillo o verde usando un cuadro de edición en el dispositivo.

4) Los inicializadores de declaración de variables ahora pueden abarcar múltiples líneas.

5) Los índices de matriz ahora se les permite a sí mismos ser elementos de matriz.

6) **Configurar | LPreferencias** ahora incluyen una casilla de verificación para permitir 'y' , 'or' , 'not' palabras clave que se utilizarán en lugar del C-estándar && , || y ! operadores logicos.

7) "Mostrar Descarga del programa" se ha movido a **Configurar | Preferencias**

V1.3 - Oct 2015

1) El dispositivo 'PUSH' ahora tiene una casilla de verificación botón-similar etiquetada 'latch' para hacer que se "enganche" (en lugar de "momentáneo"), es decir, se engancharán en la posición cerrada (y cambiarán de color) cuando presionada, hasta que se presionen nuevamente para liberar los contactos.

2) Los dispositivos de capacidad completa 'SPISLV' se han agregado con la selección de nodo ('MODE0' , 'MODE1' , 'MODE2' o 'MODE3'). Al hacer doble clic se abre una ventana de búferes de TX / RX donde se pueden definir los próximos bytes de respuesta (TX) y para ver los bytes recibidos (RX) anteriores. El dispositivo esclavo de registro de desplazamiento simple de la versión anterior se ha renombrado para convertirse en un dispositivo 'SRSLV'.

3) Ahora se puede elegir el tipo de letra *negrita* para el **Panel de Código** y el **Panel de Variables** (del menú **Opciones**), y resaltar en negrita las palabras clave y los operadores. ahora se puede activar / desactivar en **Editar / Examinar** .

4) UnoArduSim ahora permite 'bool' como sinónimo de 'boolean' .

5) Para mayor claridad en el informe de errores, ya no se permite que las declaraciones de variables abarquen varias líneas (excepto para las matrices que tienen listas de inicializadores).

6) La velocidad de coloreado de sintaxis en **Ver / Editar** se ha mejorado (esto se notará con programas más grandes).

7) Se ha agregado una sobrecarga opcional de 200 microsegundos (en el menú **Opciones**) a cada llamada de 'loop()' ; esto es para tratar de evitar caer demasiado atrás en tiempo real en el caso en que el programa de usuario no tenga 'delay()' agregado en cualquier lugar (ver Discusión de tiempo bajo **Modelado**).

Versión 1.2 Junio 2015

1) La biblioteca SD ahora está completamente implementada y se ha agregado un (pequeño) disco SD de 8Mbytes 'I / O' ('SD_DRV') (y se ha probado la funcionalidad contra todos los programas SD de muestra de Arduino).

2) Al igual que Arduino, UnoArduSim ahora convertirá automáticamente un argumento de función en su dirección cuando llame a una función esperando que se pase un puntero.

3) Los mensajes de error de Análisis ahora son más apropiados cuando faltan puntos y comas, y después de declaraciones no reconocidas.

4) Las variables viciadas Los resaltes de línea del **Panel de Variables** ahora se eliminan en la llamada / devolución de función.

V1.1 - Mar 2015

1) La ventana principal ahora se puede maximizar o cambiar de tamaño para hacer que el **Panel de Código** y el **Panel de Variables** sean más amplios (para pantallas más grandes).

2) Se ha agregado un nuevo menú Encontrar (con **botones de la Barra de Herramientas**) para permitir una navegación más rápida en el **Panel de Código** y en los **Paneles de variables** (RePág y Pág, o búsqueda de texto con flecha arriba, flecha abajo).

3) La ventana **Ver / Editar** ahora permite saltos de navegación ctrl-PgUp y ctrl-PgDn (a la siguiente línea vacía) y ha aumentado la funcionalidad **Encontrar / Reemplazar** .

4) Se ha agregado un nuevo elemento al menú **VarActualizar** para permitir al usuario seleccionar un enfoque de ahorro de cálculo bajo pesadas cargas de actualización del **Panel de Variables** .

5) 'Uno' pins y el LED adjunto ahora reflejan cualquier cambio realizado en los dispositivos 'I / O', incluso cuando el tiempo está congelado (es decir, incluso cuando se detiene la ejecución).

6) Ahora se pueden llamar otras funciones de usuario desde dentro de una función de interrupción de usuario (de acuerdo con la actualización de Arduino 1.06).

7) Una **fuentes más grande** ahora se puede elegir entre las **opciones** del menú.

V1.0.1 - junio de 2014

Las ventanas de **forma de onda** ahora etiquetan los pines analógicos como A0-A5 en lugar de 14-19.

V1.0 - primer lanzamiento de mayo de 2014