

UnoArduSimV2.7 Ayuda Copleta

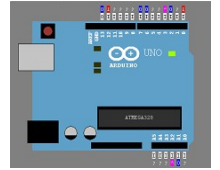


Tabla de Contenido

[Visión General](#)

[Panel de Código, Preferencias y Editar/Examinar](#)

[Panel de Código](#)

[Preferencias](#)

[Editar/Examinar](#)

[Panel de Variables y Editar/Monitorear Variable ventana](#)

[Panel del Banco de laboratorio](#)

[los 'Uno'](#)

['I/O' Dispositivos](#)

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

[Software de Serie \('SFTSER'\)](#)

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

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

[SPI Esclavo Configurable \('SPISLV'\)](#)

[I2C Esclavo TwoWire \('I2CSLV'\)](#)

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

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

[LCD del Texto D4 \('LCD_D4'\)](#)

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

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

[Puerto de Expansión SPI \('EXPSPi'\)](#)

[Puerto de Expansión I2C \('EXPI2C'\)](#)

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

[Registro de Desplazamiento Esclavo \('SRSLV'\)](#)

[Uno-Trago \('1SHOT'\)](#)

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

[Generador de Pulso Digital \('PULSER'\)](#)

[Análogo Generador de Funciones \('FUNCGEN'\)](#)

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

[Pulsado Motor Paso a Paso \('PSTEPR'\)](#)

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

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

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

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

[Pulsador \('PUSH'\)](#)

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

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

[7 segmentos DEL Dígito \('7SEG'\)](#)

[Control Deslizante Analógico](#)

[Pon Puente de Alambre \('JUMP'\)](#)

[Los Menús](#)

[Archivo:](#)

[Cargar INO o PDE Prog. \(Ctrl-L\)](#)

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

[Guardar](#)

[Guardar Como](#)

[Siguiete \('#include'\)](#)

[Anterior](#)

[Salida](#)

[Encontrar:](#)

[Subir Pila de llamadas](#)

[Desciende Pila de llamadas](#)

[Establecer texto Buscar \(ctrl-F\)](#)

[Encontrar Siguiete Texto](#)

[Encontrar Texto Anterior](#)

Ejecutar:

[Paso En \(F4\)](#)

[Paso Sobre \(F5\)](#)

[Paso Afuera \(F6\)](#)

[Ejecutar Hacia \(F7\)](#)

[Ejecutar Hasta \(F8\)](#)

[Ejecutar \(F9\)](#)

[Detener \(F10\)](#)

[Reiniciar](#)

[Animación](#)

[Camara Lenta](#)

Opciones:

[Structors Paso Sobre/ Operadores](#)

[Registro-Asignación](#)

[Error en no inicializado](#)

[Adicional 'loop\(\)' Retrasar](#)

[Permitir interrupciones anidadas](#)

Configurar:

['I/O' Dispositivos](#)

[Preferencias](#)

VarActualizar:

[Permitir Automático \(-\) Encoger](#)

[Mínimo](#)

[Resaltar Cambios](#)

Ventanas:

['Serial' Monitor](#)

[Restaura todo](#)

[Formas di Onda Digitales](#)

[Forma de Onda Analógica](#)

Ayuda:

[Ayuda rápido Archivo](#)

[Ayuda completo Archivo](#)

[Error arreglos](#)

[Cambio / Mejoras](#)

[Acerca de](#)

['Uno' Placa de Circuito y 'I/O' Dispositivos](#)

[Sincronización](#)

['I/O' Dispositivo Sincronización](#)

[Los Sonidos](#)

Limitaciones y Elementos no Soportados

[Incluido Archivos](#)

[Asignaciones de memoria dinámica y RAM](#)

[Asignaciones de memoria 'Flash'](#)

['String' Variables](#)

[Bibliotecas Arduino](#)

[Punteros](#)

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

[Alcance](#)

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

[Directivas Compilador](#)

[Elementos del Lenguaje Arduino.](#)

[C / C ++ - Elementos del Lenguaje](#)

[Plantillas Módulo funcional](#)

[Emulación en Tiempo Real](#)

Notas de Lanzamiento

[Error Arreglos](#)

[V2.7.0- Mar 2020](#)

[V2.6.0- ene 2020](#)

[V2.5.0- octubre 2019](#)

[V2.4- mayo de 2019](#)

[V2.3- Dic. 2018](#)

[V2.2- Jun. 2018](#)

[V2.1.1- Mar. 2018](#)

[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- septiembre de 2016](#)
[V1.6.2- septiembre de 2016](#)
[V1.6.1- agosto de 2016](#)
[V1.6- junio 2016](#)
[V1.5.1- junio 2016](#)
[V1.5 - Mayo 2016](#)
[V1.4.3 - Abr. 2016](#)
[V1.4.2 - Mar. 2016](#)
[V1.4.1 - Ene. 2016](#)
[V1.4 - Dic. 2015](#)
[V1.3 - Oct. 2015](#)
[V1.2 - Jun. 2015](#)
[V1.1 - Mar. 2015](#)
[V1.0.2 - Ago. 2014](#)
[V1.0.1 - Jun. 2014](#)

[V1.0 - primer lanzamiento mayo 2014](#)

[Cambios / Mejorass](#)

[V2.7.0 Mar. 2020](#)
[V2.6.0 de enero 2020](#)
[V2.5.0 Oct 2019](#)
[V2.4 de mayo de 2019](#)
[V2.3 Dic. 2018](#)
[V2.2 Jun. 2018](#)
[V2.1 Mar. 2018](#)
[V2.0.1 Ene. 2018](#)
[V2.0 Sept. 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 2016](#)
[V1.5.1 - Junio 2016](#)
[V1.5 - Mayo 2016](#)
[V1.4.2 - Mar. 2016](#)
[V1.4 - Dic. 2015](#)
[V1.3 - Oct. 2015](#)
[V1.2 Jun. 2015](#)
[V1.1 - Mar. 2015](#)
[V1.0.1 - Jun. 2014](#)

[V1.0 - primer lanzamiento mayo 2014](#)

Visión General

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

UnoArduSim proporciona mensajes de error simples para cualquier error analizar o ejecución que encuentre, y permite la depuración con **Reiniciar** , **Ejecutar** , **Ejecutar Hacia**, **Ejecutar Hasta** , **Detener** , y flexible **Paso** operaciones en el **Panel de Código** , con una vista simultánea de todos los locales variables, matrices y objetos globales y actualmente activos en el **Panel de Variables** . Ejecutar-tiempo matriz se proporciona verificación de límites, y se detectará un desbordamiento de RAM de ATmega (y la línea programa culpable resaltada!). Cualquier entra en conflicto con eléctrico adjunto 'I/O' dispositivos está marcado y reportado a medida que ocurren.

Cuando se abre un INO o PDE programa archivo, se carga en el programa **Panel de Código** . Al programa se le da una Analizar, para transformarlo en un ejecutable tokenizado que luego está listo para **ejecución simulado** (a diferencia de Arduino.exe, un ejecutable binario independiente es *no* creado) Se detecta y marca cualquier error analizar resaltando la línea que falló a analizar e informando el error en la **Barra de estado** en la parte inferior de la aplicación UnoArduSim ventana. Un **Editar/Examinar** ventana se puede abrir para permitirle ver y editar una versión resaltada en sintaxis de su usuario programa. Los errores durante ejecución simulado (como un velocidad baudios mal emparejado) se informan en la barra de estado, y a través de un cuadro de mensaje emergente.

UnoArduSim V2.7 es una implementación sustancialmente completa de la **Lenguaje de programación Arduino V1.8.8 como se documenta en el arduino.cc** . Página web de referencia del lenguaje, y con las adiciones como se indica en las Notas de la versión de la página Descarga. Aunque UnoArduSim no es compatible con la implementación completa de C ++ que Arduino.exe y la GNU compilador subyacente, es probable que solo los programadores más avanzados encuentren que falta algún elemento de C / C ++ que quieran usar (y, por supuesto, siempre hay elementos simples). codificación de soluciones para tales características que faltan). En general, solo he apoyado lo que creo que son las funciones C / C ++ más útiles para los aficionados y estudiantes de Arduino, por ejemplo, **'enum'** y **'#define'** son compatibles, pero módulo funcional-punteros no lo son. Aunque objetos definido por el usuario (**'class'** y **'struct'**) y (la mayoría) las sobrecargas de operadores son compatibles, *la herencia múltiple no es* .

Porque UnoArduSim es un simulador de lenguaje de alto nivel, **solo se admiten sentencias C / C ++** , *declaraciones en lenguaje ensamblador no son* . Del mismo modo, debido a que no es una simulación de máquina de bajo nivel, **Los registros ATmega328 no son accesibles para su programa** ya sea para lectura o escritura, aunque la asignación de registros, el paso y el retorno se emulan (usted elige eso bajo el menú **Opciones**).

A partir de V2.6, UnoArduSim tiene incorporado soporte automático para un subconjunto limitado de la Arduino proporciona bibliotecas, siendo estos: **'Stepper.h'** , **'Servo.h'** , **'SoftwareSerial.h'** , **'SPI.h'** , **'Wire.h'** , **'OneWire.h'** , **'SD.h'** , **'TFT.h'** y **'EEPROM.h'** (versión 2). V2.6 introduce un mecanismo para 3rd soporte de la biblioteca a través de las partes archivos proporciona en el **'include_3rdParty'** carpeta que se pueden encontrar dentro del directorio de instalación de UnoArduSim.

Para cualquier **'#include'** de bibliotecas creadas por el usuario, UnoArduSim **no** busque la estructura de directorio de instalación de Arduino habitual para ubicar la biblioteca; en cambio tú **necesitar** para copiar el encabezado correspondiente (".h") y la fuente (".cpp") archivo en el mismo directorio que programa archivo 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 UnoArduSim analizador).

Desarrollé UnoArduSimV2.0 en QtCreator con soporte en varios idiomas, y actualmente solo está disponible para Ventanas™. ¡Portar a Linux o MacOS, es un proyecto para el futuro! UnoArduSim surgió de los simuladores que había desarrollado a lo largo de los años para los cursos que impartí en la Universidad de Queen, y se ha probado de manera bastante extensa, pero es probable que haya algunos errores que aún se esconden allí. Si desea informar un error, por favor describalo (brevemente) en un correo electrónico a unoArduSim@gmail.com y **asegúrese de adjuntar su código fuente programa Arduino que induce error** así que puedo replicar el error y arreglarlo. No responderé a informes error individuales, y no tengo líneas de tiempo garantizadas para las correcciones en una versión posterior (recuerde que casi siempre hay soluciones alternativas).

Aclamaciones,

Stan Simmons, PhD, P.Eng.
Profesor asociado (retirado)
Departamento de Ingeniería Eléctrica e Informática
Universidad de la reina
Kingston, Ontario, Canadá







Panel de Código, Preferencias y Editar/Examinar


(Aparte: el ventanas de muestra que se muestra a continuación se encuentra bajo un Ventanas-OS elegido por el usuario tema de color que tiene un color de fondo ventana azul oscuro).

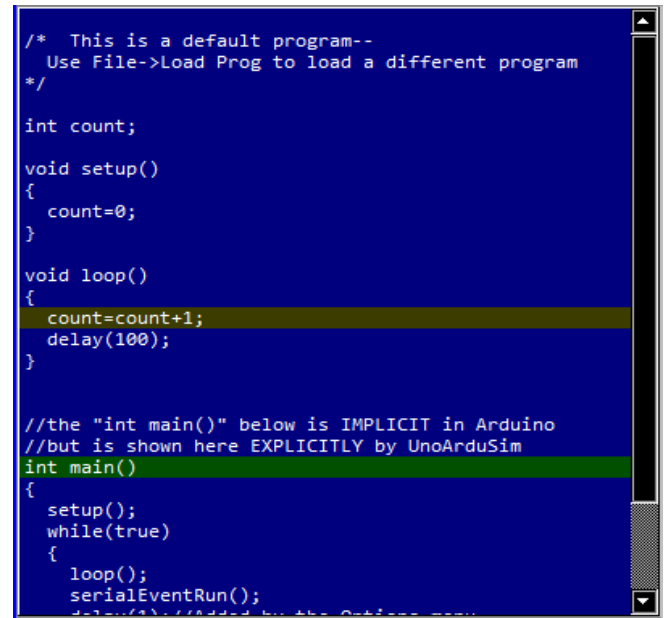
Panel de Código

los **Panel de Código** muestra su usuario programa y el resaltado rastrea su ejecución. Después de que un programa cargado tiene un Analizar exitoso, la primera línea en 'main()' está resaltado y el programa está listo para ejecución. Tenga en cuenta que 'main()' Se agrega implícitamente por Arduino (y por UnoArduSim) y lo haces **no** Inclúyelo como parte de su usuario programa archivo. Ejecución está bajo el control del menú. **Ejecutar** , y sus asociados **Barra de herramientas** botones y atajos de teclado módulo funcional.

Después de caminar ejecución por uno (o más) instrucciones (se puede utilizar **Barra de herramientas** botones  ,  ,  o ), La línea programa que será el próximo ejecutado continuación se resalta en verde - la línea verde resaltado es siempre la línea siguiente **listo para ser ejecutado** .

Del mismo modo, cuando un programa en ejecución golpea un (temporal **Ejecutar Hacia**) punto de parada, ejecución se detiene y la línea punto de parada se resalta (y luego está lista para ejecución).

Si programa ejecución se detiene en la actualidad, y que haga clic en el **Panel de Código** ventana, la línea que acaba de hacer clic se vuelve resaltado en verde oliva oscuro (como se muestra en la imagen) - los próximos-a-ser-ejecutado línea siempre permanece resaltados en verde (a partir de V2.7). Pero puedes causar ejecución *progresar hasta* la línea que acaba de resaltar haciendo clic en el **Ejecutar Hacia**  **Barra de herramientas** botón. Esta función le permite llegar rápida y fácilmente a líneas específicas en un programa para que luego pueda pasar línea por línea sobre una parte de interés de programa.





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





int count;

void setup()
{
  count=0;
}

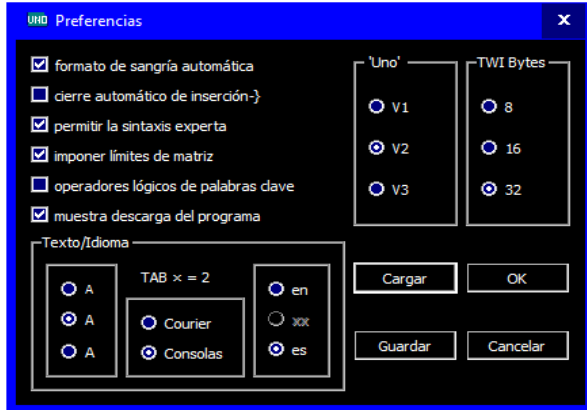
void loop()
{
  count=count+1;
  delay(100);
}

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

Si su programa cargado tiene alguna '#include' archivos, puedes moverte entre ellos usando **Archivo | Anterior** y **Archivo | Siguiente** (con **Barra de herramientas** botones  y ).

Las acciones de la **Encontrar** menú que permite **YA SEA** buscar texto en el **Panel de Código** o **Panel de Variables** (**Barra de herramientas** botones  y  o atajos de teclado **flecha hacia arriba** y **flecha hacia abajo**) **después de usar primero** **Encontrar | texto Conjunto Buscar** o **Barra de herramientas** , **O ALTERNATIVAMENTE** a **navegar por el pila de llamadas** en el **Panel de Código** (**Barra de herramientas** botones  y  o atajos de teclado **flecha hacia arriba** y **flecha hacia abajo**). Llaves **AvPág** y **Re Pág** saltar a la siguiente selección / módulo funcional anterior .

Preferencias



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

Editar/Examinar

Haciendo doble clic en cualquier línea del **Panel de Código** (o usando el menú **Archivo**), un **Editar/Examinar** ventana está abierta para permitir cambios en su programa archivo, con el **línea seleccionada actualmente** en el **Panel de Código** se resalta.

Este ventana tiene capacidad de edición completa con resaltado de sintaxis dinámico (se utilizan diferentes colores resaltar para palabras clave de C ++, comentarios, etc.) . Hay un resaltado de sintaxis negrita opcional y un formato automático de nivel de sangría (asumiendo que haya seleccionado eso usando **Configurar | Preferencias**). También puede seleccionar convenientemente

incorporado módulo funcional llamadas (o incorporado '#define' constantes) que se agregarán a su programa desde el cuadro de lista provisto - simplemente haga doble clic en el elemento del cuadro de lista deseado para agregarlo a su programa en la posición actual de caret (módulo funcional-call variable **tipos** son solo para información y se eliminan para dejar marcadores de posición ficticios cuando se agregan a su programa).

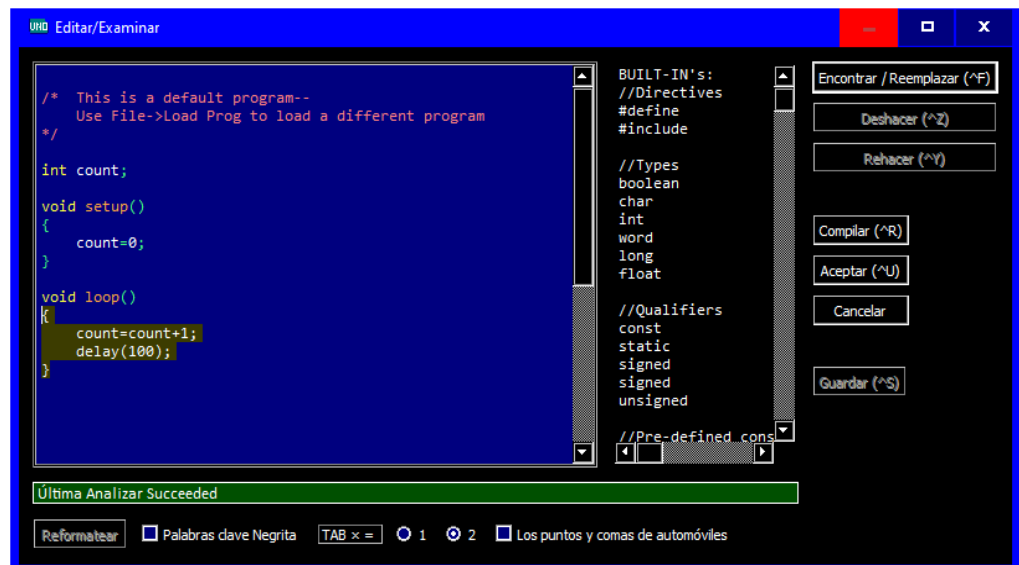
Uso ALT-flecha derecha a petición opciones de autocompletado para incorporado **variables mundial**, y para **miembro variables** y **módulos funcionales**.

El ventana tiene **Encontrar** (utilizar **Ctrl-F**) y **Encontrar / Reemplazar** capacidad (uso **Ctrl-H**) . los **Editar/Examinar** ventana tiene **Deshacer** (**Ctrl-Z**), y **Rehacer** (**ctrl-Y**) Botones (que aparecen automáticamente).

Descartar **todos los cambios** realizado desde la primera vez que abrió el programa para editarlo, haga clic en el **Cancelar** botón. Aceptar el estado actual, haga clic en **Aceptar** y el programa recibe automáticamente otro Analizar

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

Este cuadro de diálogo permite elegir entre dos fuentes mono-espaciadas y tres tamaños de letra, y otras preferencias varias. A partir de la V2.0, ahora se incluye la opción de idioma. - esto siempre incluye inglés (**en**), más uno o dos idiomas de configuración regional de usuario (donde existan), y una anulación basada en el código de



(y se descarga al 'Uno' si no se encuentran errores) y aparece el nuevo estado en el UnoArduSim ventana principal **Barra de estado** .

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

En cualquiera **Cancelar** o **Aceptar** sin modificaciones, el **Panel de Código** la línea actual cambia para convertirse en el **última posición de Editar/Examinar caret** , y puedes usar esa característica para saltar el **Panel de Código** a una línea específica (posiblemente prepararse para hacer una **Ejecutar Hacia**), También puedes usar **ctrl-PgDn** y **ctrl-PgUp** para saltar al siguiente (o anterior) salto de línea vacía en su programa, esto es útil para navegar rápidamente hacia arriba o hacia abajo a ubicaciones significativas (como líneas vacías entre módulos funcionales). También puedes usar **ctrl-Inicio** y **ctrl-End** para saltar al inicio y al final programa, respectivamente.



Formato de sangría automática 'Tab' nivel se hace cuando se abre la ventana, si esa opción se ha definido en **Configurar | Preferencias**. Usted puede hacer de nuevo que el formateo en cualquier momento haciendo clic en el **Reformatear** botón (que sólo se activa si se ha seleccionado previamente la **Preferencia sangría automática**). También puede añadir o lengüetas de borrado a sí mismo a un grupo de líneas consecutivas preseleccionados mediante el teclado **flecha correcta** o **flecha izquierda** llaves - pero **Preferencia sangría automática debe estar apagado** para evitar la pérdida de sus propios niveles de ficha personalizada.

Cuando **Auto punto y coma** se comprueba, presionando **Entrar** para finalizar una línea, se inserta automáticamente el punto y coma que termina la línea.

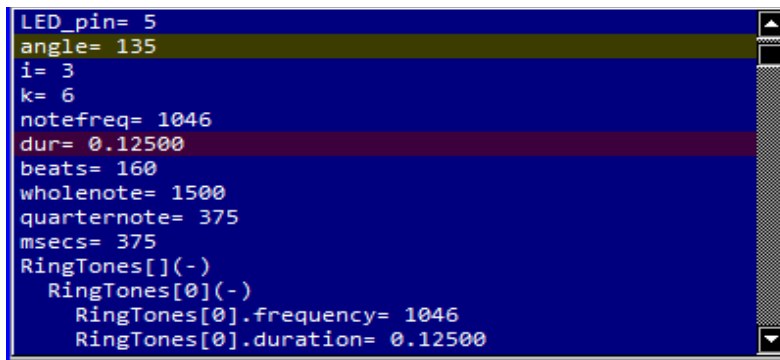
Y para ayudarlo a realizar un mejor seguimiento de sus contextos y llaves, haga clic en un ' { ' o ' } ' llave **resalta todo el texto entre ese llave y su compañero correspondiente** .

Panel de Variables y Editar/Monitorear Variable ventana

los **Panel de Variables** se encuentra justo debajo de la **Panel de Código**. Muestra los valores actuales para cada usuario global y activo (en alcance) local variable / matriz / objeto en el programa cargado. A medida que su programa ejecución se mueve entre módulos funcionales, **el contenido cambia para reflejar solo aquellos variables locales accesibles a los módulo funcional / alcance actuales, más cualquier global declarado por el usuario**. Cualquier variables declarado como 'const' o como 'PROGMEM' (asignado a 'Flash' memoria) tienen valores que no pueden cambiar, y para ahorrar espacio, estos son, por lo tanto, *no se muestra*. 'Servo' y 'SoftwareSerial' Las instancias objeto no contienen valores útiles, por lo que tampoco se muestran.

Usted puede **encontrar** especificado **texto** con sus comandos de búsqueda de texto (con **Barra de herramientas** botones  y  o atajos de teclado **flecha hacia arriba** y **flecha hacia abajo**), Después de usar primero **Encontrar | conjunto Buscar** texto o .

Matrices y objetos se muestran en cualquiera **un-expandido** o **expandido** formato, ya sea con un plus de arrastre ' (+) ' o menos ' (-) ' signo, respectivamente. El símbolo para un matriz **x** muestra como '**x[]**' . Para expandir para mostrar todos los elementos del matriz, simplemente haga clic en '**x[] (+)**' en el **Panel de Variables** . Para volver a encoger en una vista sin expandido, haga clic en '**x[] (-)**' . El valor predeterminado de un expandido para un objeto '**p1**' muestra como '**p1 (+)**' A expandir para mostrar todos los miembros de ese '**class**' o '**struct**' instancia, haga clic en '**p1 (+)**' en el **Panel de Variables** . Para volver a encoger en una vista sin expandido, haga clic en '**p1 (-)**' . Si tu **De un solo clic en cualquier línea a resaltar en oliva oscuro** (Que puede ser simple

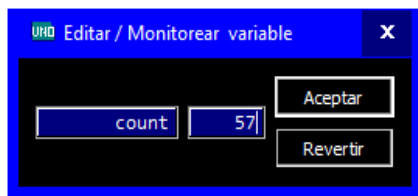


```
LED_pin= 5
angle= 135
i= 3
k= 6
notefreq= 1046
dur= 0.12500
beats= 160
wholenote= 1500
quarternote= 375
msecs= 375
RingTones[0](-)
  RingTones[0](-)
    RingTones[0].frequency= 1046
    RingTones[0].duration= 0.12500
```


variable, o el agregado ' (+) ' o ' (-) ' línea de un matriz o objeto, o un solo elemento matriz o objeto miembros), a continuación, hacer una **Ejecutar Hasta** hará que ejecución para reanudar y congelar en la próxima *acceso de escritura* en cualquier lugar dentro de ese agregado seleccionado, o para que seleccionan única ubicación variable.

Cuando usas **Paso** o **Ejecutar** , las actualizaciones de los valores variable mostrados se realizan de acuerdo con las configuraciones de usuario realizadas en el menú **VarActualizar** - Esto permite un rango completo de comportamiento desde actualizaciones periódicas mínimas hasta actualizaciones completas inmediatas. Las actualizaciones reducidas o mínimas son útiles para reducir la carga de la CPU y pueden ser necesarias para evitar que ejecución se quede atrás en tiempo real en lo que de otro modo sería excesivo **Panel de Variables** ventana carga las actualizaciones. Cuando **Animación** está en efecto , o si el **Resaltar Cambios** Se selecciona la opción de menú, cambia al valor de un variable durante **Ejecutar** resultará en la actualización de su valor mostrado *inmediatamente* , y se resalta - esto causará la **Panel de Variables** para desplazarse (si es necesario) a la línea que contiene variable y ejecución ya no será en tiempo real.

Cuando ejecución se congela después **Paso** , **Ejecutar Hacia** , **Ejecutar Hasta** o **Ejecutar** -entonces- **Detener** , la **Panel de Variables** Destaca el variable correspondiente a la *dirección (s) ubicación (es) que se modificó* (si existe) por el *ultima instrucción* durante ese ejecución (incluidas las inicializaciones de la declaración variable). Si esa instrucción *completamente* llenó un *objeto o matriz* , la *línea padre (+) o (-)* para que el agregado se destaque. Si, en cambio, la instrucción modifica un ubicación que es actualmente visible, entonces se resalta. Pero si las ubicaciones modificadas están ocultas actualmente dentro un expandido matriz o objeto un-expandido, que se agrega *línea parental* obtiene un *resaltado de letra cursiva* como una indicación visual de que se escribió algo en su interior: hacer clic en expandir provocará su *último* Elemento modificado o miembro para que se destaque.



los **Editado/Monitorear** ventana te da *la capacidad de seguir cualquier valor variable durante ejecución* , o para *cambiar su valor en medio de (detenido) programa ejecución* (para que pueda probar cuál sería el efecto de continuar adelante con ese nuevo valor). **Detener** ejecución primero, luego *doble clic izquierdo* en el variable cuyo valor desea rastrear o cambiar. Para controlar simplemente el valor durante programa ejecución, dejar el cuadro de diálogo abierto y luego uno de los **Ejecutar** o **Paso** comandos - su valor será

actualizado en **Editado/Monitorear** De acuerdo con las mismas reglas que rigen las actualizaciones en el **Panel de Variables** . *Para cambiar el valor variable* , complete el valor del cuadro de edición, y **Aceptar** . Continuar ejecución (usando cualquiera de los **Paso** o **Ejecutar** comandos) para usar ese nuevo valor desde ese punto en adelante (o puede **Revertir** al valor anterior).

En programa Cargar o Reiniciar tenga en cuenta que todos el *valor no inicializado-variables se restablece al valor 0*, y todo el *puntero no inicializado-variables se restablece a 0x0000*.

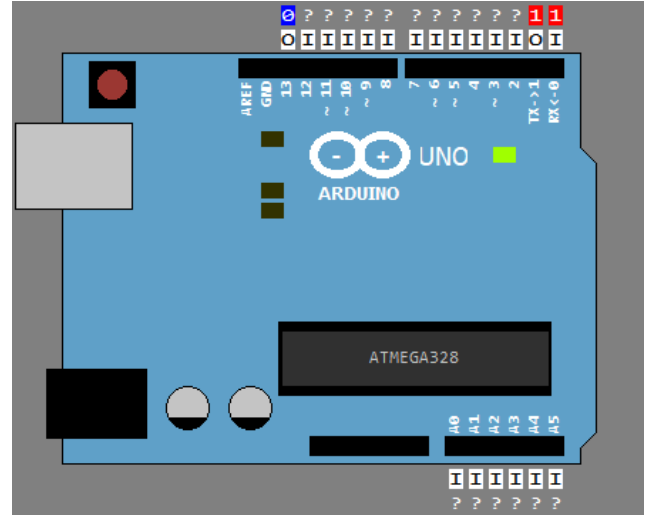
Panel del Banco de laboratorio

El Panel del Banco de laboratorio muestra un 'Uno' placa de circuito de 5 voltios rodeado por un conjunto de 'I/O' dispositivos que puede seleccionar / personalizar, y conectarse al 'Uno' pins deseado.

los 'Uno'

Esta es una representación del 'Uno' placa de circuito y sus LED a bordo. Cuando carga un nuevo programa en UnoArduSim, si lo analiza correctamente, se somete a un "descarga simulado" al 'Uno' que simula la forma en que se comporta un 'Uno' placa de circuito real. Verá el RX serial y el TX DEL parpadeando (junto con la actividad en pins 1 y 0 que son *Cableado para la comunicación en serie con una computadora host*). A esto le sigue inmediatamente un flash pin 13 DEL que significa reinicio de placa de circuito y (y UnoArduSim se detiene automáticamente) al comienzo de su programa ejecución cargado. Puede evitar esta visualización y el retraso de carga asociado deseleccionando **Mostrar Descarga** desde **Configurar | Preferencias**.

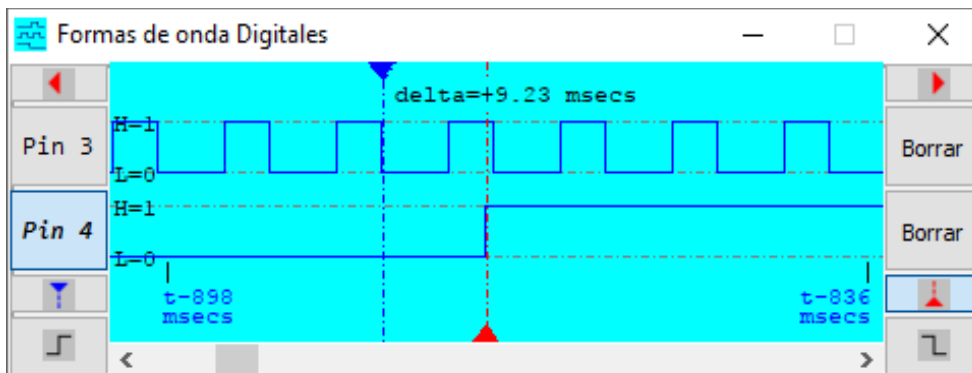
El ventana le permite visualizar los niveles lógicos digital en todos los 20 'Uno' pins ('1' en rojo para 'HIGH' , '0' en azul para 'LOW' y '?' en gris para una tensión indeterminada indefinida), y direcciones programado ('I' para 'INPUT' o 'O' para 'OUTPUT'). Para pins que están siendo pulsados usando PWM a través de 'analogWrite()' , o por 'tone()' , o por 'Servo.write()' , el color cambia a púrpura y el símbolo mostrado se convierte en '^' .



Tenga en cuenta que **Digital pins 0 y 1 están cableados a través de resistencias de 1 kOhm al chip USB para Comunicación serial con una computadora host.**

Además: Digital pins 0-13 aparece como simulador pins 0-13, y analog pins 0-5 aparece como A0-A5. Para acceder a un analog pin en su programa, puede referirse al número pin por uno de dos conjuntos de números equivalentes: 14-19; o A0-A5 (A0-A5 son incorporado 'const' variables teniendo valores 14-19). Y solo cuando se usa 'analogRead()' , se pone a disposición una tercera opción: puede, por esta única instrucción, eliminar el 'A' prefijo del número pin y simplemente use 0-5. Para acceder a pins 14-19 en su programa usando 'digitalRead()' o 'digitalWrite()' , puede simplemente referirse a ese número pin, o puede utilizar los alias A0-A5.

Clic izquierdo en cualquier 'Uno' pin se abrirá un **Formas di Onda Digitales** ventana que muestra el pasado **un segundo de valor de Actividad de nivel digital** en ese pin. Puede hacer clic en otro pins para agregarlos a la pantalla Formas di Onda Digitales (hasta un máximo de 4 formas de onda a la vez).



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

Una de las formas de onda mostradas será la **pin activo** forma de onda , indicado por su botón "Pin" que se muestra como presionado (como en la captura de pantalla Formas di Onda Digitales anterior). Puede seleccionar un forma de

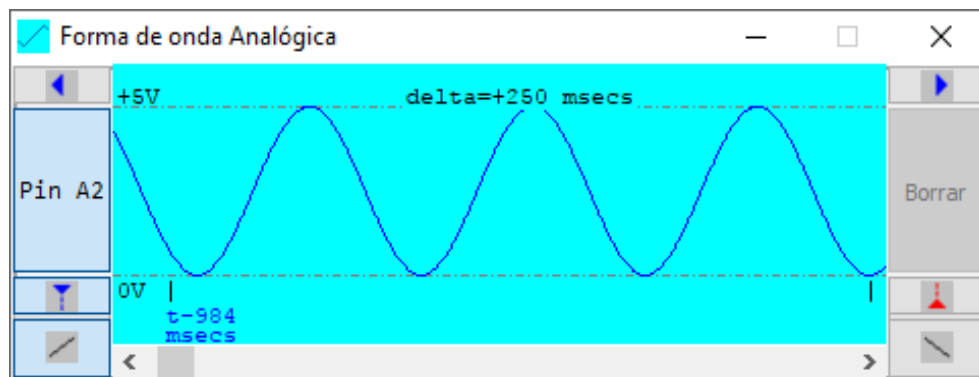
onda haciendo clic en su botón numérico Pin, y luego seleccionar la polaridad de interés de borde haciendo clic en el botón de selección de polaridad de borde ascendente / descendente apropiado, o , o usando las teclas de acceso directo **flecha arriba** y **flecha hacia abajo** . Entonces puedes **saltar** el cursor activo (ya sea las líneas del cursor azul o rojo con su tiempo delta mostrado) hacia atrás o hacia adelante hasta el borde digital de polaridad elegida **de este activo pin** forma de onda utilizando los botones del cursor (, o , (dependiendo del cursor activado anteriormente con o), o simplemente use las teclas del teclado ← y → .

Para activar un cursor, haga clic en el botón de activación de color (o mostrado anteriormente) - *esto también desplaza la vista a la ubicación actual de ese cursor* . Alternativamente, puede alternar rápidamente la activación entre cursores (con sus vistas centradas respectivamente) utilizando el acceso directo **'Tab'** llave.

Usted puede **saltar** el cursor actualmente activado por **clic izquierdo en cualquier lugar** en la región de visualización forma de onda en pantalla. Alternativamente, puede seleccionar la línea de cursor roja o azul haciendo clic derecho sobre ella (para activarla), luego **arrástrelo a una nueva ubicación** , y liberar. Cuando un cursor deseado está actualmente en algún lugar fuera de la pantalla, puede **haga clic derecho en cualquier lugar** en la vista para saltar 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 los cursores activados.

Para ZOOM IN y ZOOM OUT (el zoom siempre está centrado en el cursor ACTIVE), use la rueda del mouse o los atajos de teclado CTRL-flecha arriba y CTRL-flecha abajo.

Haciendo en cambio un **botón derecho del ratón en cualquier 'Uno' pin** abre un **Forma de Onda Analógica** ventana que muestra el **valor de un segundo pasado** de **Actividad de nivel análogo** en ese pin. A diferencia del Formas di Onda Digitales ventana, puedes muestra la actividad de análogo en un solo pin a la vez.



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

Usted puede **saltar** la las líneas de cursor azules o rojas hasta el siguiente "punto de pendiente" ascendente o descendente mediante los botones de flecha hacia adelante o hacia atrás (, o , , nuevamente dependiendo del cursor activado, o usar el ← y → teclas) en concierto con los botones de selección de pendiente ascendente / descendente , (el "punto de pendiente" se produce cuando el voltaje análogo pasa a través del umbral de nivel lógico digital de ATmega pin). Alternativamente, puede hacer clic nuevamente para saltar, o arrastrar estas líneas de cursor similares a su comportamiento en el Formas di Onda Digitales ventana

Prensado **'Ctrl-S'** dentro **ventana te permite guardar el forma de onda (X, Y) datos** a un texto archivo de su elección, donde **X** está en microsegundos desde el lado izquierdo, y **Y** Está en los pernos.

'I/O' Dispositivos

Un número de diferentes dispositivos rodean el 'Uno' en el perímetro de la **Panel del Banco de laboratorio**. "Pequeño" 'I/O' dispositivos (de los cuales se le permite hasta 16 en total) reside a lo largo de los lados izquierdo y derecho del Panel. "Grande" 'I/O' dispositivos (de los cuales se le permite hasta 8 en total) tiene elementos "activos" y se encuentra en la parte superior e inferior de la **Panel del Banco de laboratorio**. El número deseado de cada tipo de 'I/O' dispositivo disponible se puede configurar usando el menú **Configurar | 'I/O' Dispositivos**.

Cada 'I/O' dispositivo tiene uno o más accesorios pin mostrados como **dos dígito** Número pin (00, 01, 02,... 10,11,12, 13 y A0-A5, o 14-19, después de eso) en el cuadro de edición correspondiente. Para los números pin del 2 al 9, simplemente puede ingresar el dígito individual; el 0 inicial se proporcionará automáticamente, pero para pins 0 y 1 primero debe ingresar el 0 inicial. Las entradas son *normalmente* en el lado izquierdo de un 'I/O' dispositivo, y las salidas son *normalmente* a la derecha (*si el espacio lo permite*). Todos los 'I/O' dispositivos responderán directamente a los niveles de pin y los cambios a nivel de pin, por lo que responderán a cualquiera de las bibliotecas módulos funcionales orientadas a su pins adjunta, o a

programado '`digitalWrite()`' (Para operación "bit banged").

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

El cuadro de diálogo se puede usar para permitir al usuario elegir los tipos y números del 'I/O' dispositivos deseado. Desde este cuadro de diálogo también puede **Guardar** 'I/O' dispositivos a un texto archivo, y / o **Cargar** 'I/O' dispositivos de un texto archivo previamente guardado (o editado) (**Incluyendo todas las conexiones pin, y las configuraciones a las que se puede hacer clic, y cualquier valor de cuadro de edición ingresado.**).

Tenga en cuenta que a partir de la versión 1.6, los valores en los cuadros de edición de período, retardo y ancho de pulso en el IO dispositivos relevante pueden recibir el sufijo 'S' (o 's'). Eso indica que deben ser **escamoso** De acuerdo con la posición de un global '`IO ____S`' Control deslizante que aparece en el ventana principal. **Barra de herramientas.** 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 los valores mínimos impuestos por cada 'I/O' dispositivo en particular). Puede escalar más de un valor de cuadro de edición **simultáneamente** utilizando este control deslizante. Esta característica le permite arrastrar el control deslizante mientras ejecuta para emular fácilmente el ancho de pulso cambiante, los periodos y los retrasos para los 'I/O' dispositivos adjuntos.

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

Varios de estos dispositivos **Apoyar el escalado de sus valores escritos.** utilizando el control deslizante en el ventana principal **Barra de herramientas**. Si el valor de dispositivo tiene la letra 'S' como su sufijo, su valor se multiplicará por un factor de escala (entre 0.0 y 1.0) que se determina por la posición del control deslizante, sujeto a la restricción de valor mínimo de dispositivo (1.0 está totalmente a la derecha, 0.0 está completamente a la izquierda) --ver '`IO ____S`' debajo de cada



manguera dispositivos detallada a continuación.

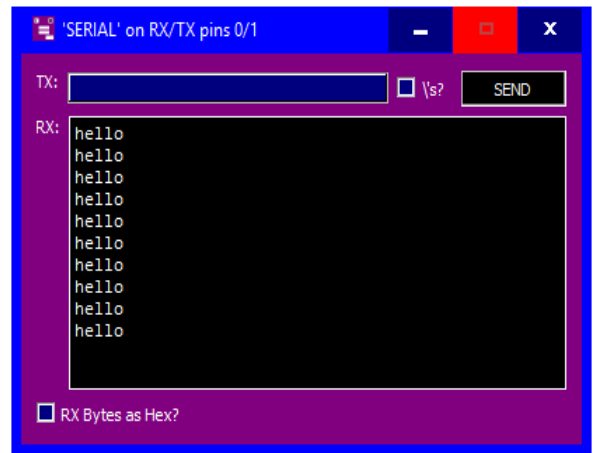
'Serial' Monitor ('SERIAL')



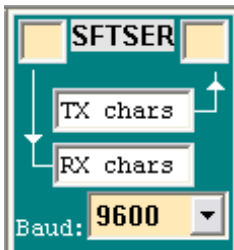
Este 'I/O' dispositivo permite una entrada y salida serie mediada por hardware ATmega (a través del chip USB 'Uno') en 'Uno' pins 0 y 1. El velocidad baudios se configura utilizando la lista desplegable en su parte inferior: el velocidad baudios seleccionado **debe coincidir con** el valor que su programa pasa al '`Serial.begin()`' módulo funcional para una correcta transmisión / recepción. *La comunicación en serie se fija en 8 bits de datos, 1 bit de parada y sin bit de paridad.* Tienes permiso para **desconectar** (blanco) **pero no reemplazar** TX pin 00, pero no RX pin 01.

Para enviar una entrada de teclado a su programa, escriba uno o más caracteres en la parte superior (caracteres TX), edite ventana y luego golpea el '**Enter**' tecla del teclado. (los caracteres se ponen en cursiva para indicar que las transmisiones han comenzado) o, si ya están en curso, los caracteres escritos que se agreguen aparecerán en cursiva. A continuación, puede utilizar el '`Serial.available()`' y '`Serial.read()`' módulos funcionales para leer los caracteres en el orden en que fueron recibidos en el búfer pin 0 (el carácter escrito más a la izquierda se enviará primero). Las impresiones en formato textual y numérico, o los valores de bytes sin formato, pueden enviarse a la salida inferior de la consola (caracteres RX) ventana llamando al Arduino '`print()`', '`println()`' o '`write()`' módulos funcionales.

Adicionalmente, **un ventana más grande para configurar / ver los caracteres TX y RX se puede abrir haciendo doble clic (o clic derecho) en este 'SERIAL' dispositivo**. Este nuevo ventana tiene un cuadro de edición de caracteres de TX más grande y un botón 'Send' separado en el que se puede hacer clic para enviar los caracteres de TX al 'Uno' (en pin 0). También hay una opción de casilla de verificación para volver a interpretar las secuencias de caracteres escapadas con barra invertida, como '`\n`' o '`\t`' para la visualización sin formato.



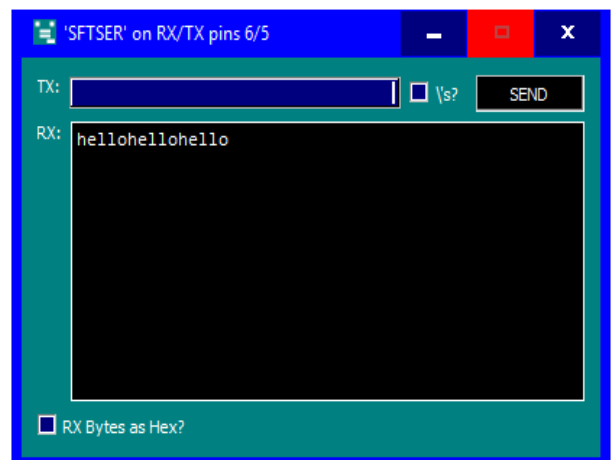
Software de Serie ('SFTSER')



Este 'I/O' dispositivo permite la entrada y salida en serie mediada por el software de la biblioteca o, alternatively, por el usuario "bit banged", en cualquier par de 'Uno' pins que elija completar (**excepto por** pins 0 y 1 dedicados al hardware. '`Serial`' comunicación). Tu programa debe tener un '`#include <SoftwareSerial.h>`' línea cerca de la parte superior si desea utilizar la funcionalidad de esa biblioteca. Al igual que con el 'SERIAL' dispositivo basado en hardware, el velocidad baudios para 'SFTSER' se configura usando la lista desplegable en su parte inferior: el velocidad baudios seleccionado debe coincidir con el valor que su programa pasa al '`begin()`' módulo funcional para una correcta transmisión / recepción. *La comunicación en serie se fija en 8 bits de datos, 1 bit de parada y sin bit de paridad.*

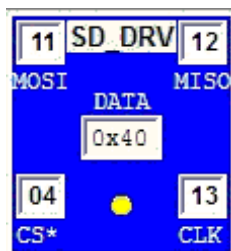
Además, al igual que con el hardware basado '`SERIAL`', **un ventana más grande para la configuración / visualización de TX y RX se puede abrir haciendo doble clic (o haciendo clic con el botón derecho) en el SFTSER dispositivo**.

Tenga en cuenta que a diferencia de la implementación de hardware de '`Serial`', no hay provisto El búfer de TX es compatible con las operaciones internas de interrupción de ATmega (solo un búfer RX), ese '`write()`' (o '`print`') las llamadas están bloqueadas (es decir, su programa no continuará hasta que se completen).



Unidad de Disco SD ('SD_DRV')

Este 'I/O' dispositivo permite el software de biblioteca mediado (pero **no** "bit banged") archivo operaciones de entrada y salida en el 'Uno' **SPI** pins (puedes elegir cual **CS *** pin utilizará). Su programa puede simplemente `'#include <SD.h>'` línea cerca de la parte superior, y puede utilizar `'<SD.h>'` módulos funcionales O llamar directamente `'SdFile'` módulos funcionales usted mismo.

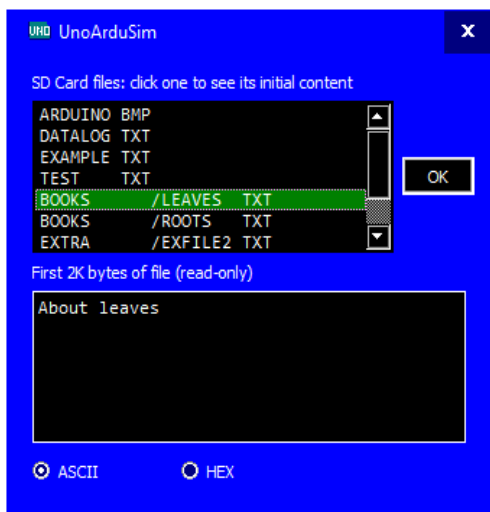


*Se puede abrir un ventana más grande mostrando directorios y archivos (y contenido) haciendo doble clic (o haciendo clic con el botón derecho) en 'SD_DRV' dispositivo . Todo el contenido del disco es **cargado desde** un **Dakota del Sur** subdirectorio en el directorio programa cargado (si existe) en*

*'SdVolume::init()' , y se refleja en lo mismo **Dakota del Sur** subdirectorio en archivo*

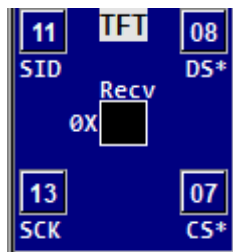
'close()' , 'remove()' , y en 'makeDir()' y 'rmDir()' .

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



Pantalla TFT ('TFT')

Este 'I/O' dispositivo emula un Adafruit™ pantalla TFT de tamaño 1280by-160 píxeles (en su rotación nativo = 0, pero cuando se utiliza la biblioteca 'TFT.h', `'TFT.begin()'` conjuntos de inicialización para la rotación = 1 que da una vista "paisaje" de 160-por-128 píxeles). Puede empujar este dispositivo llamando a la módulos funcionales del `'TFT.h'` biblioteca (que requiere en primer lugar `'#include <TFT.h>'`), o se puede utilizar el sistema SPI que le envíe propia secuencia de bytes que se empujar.

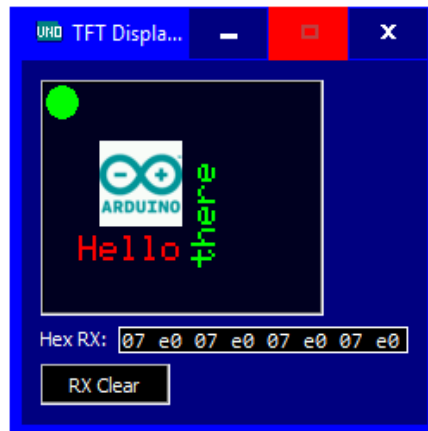


El TFT siempre está conectado a 'SPI' 'MOSI' pins (por 'SID') y 'SCK' (por 'SCK') - los que no se puede cambiar. El 'DS*' es para pin datos / comandos seleccionar (modo de datos 'LOW' selecciona), y el 'CS*' pin es la selección de chip activa baja

No hay ninguna Reiniciar pin proporciona por lo que no puede hacer un reset del hardware de este dispositivo por la conducción de un bajo pin (como el `'TFT::begin()'` módulo funcional intenta hacer

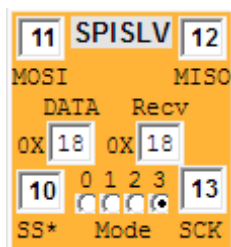
cuando se ha aprobado una serie 'reset' pin válida como el tercer parámetro a la `'TFT(int cs, int ds, int rst)'` constructor). El dispositivo Sin embargo, tiene una conexión oculta al sistema de líneas Reiniciar por lo que se reinicia cada vez que se hace clic en el principal UnoArduSim icono de la barra de herramientas Reiniciar, o el botón de reinicio 'Uno' placa de circuito ..

Por **Un doble clic** (o **botón derecho del ratón**) En este dispositivo, un ventana más grande se abre para mostrar que la pantalla LCD píxel completo 160-por-128, junto con los 8 bytes más recientemente recibido (como se muestra a continuación)



SPI Esclavo Configurable ('SPISLV')

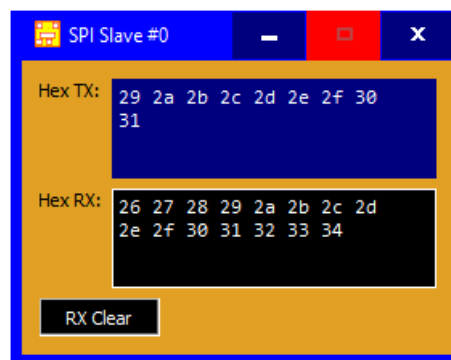
Este 'I/O' dispositivo emula un esclavo SPI en modo elegido con un activo bajo **SS *** ("slave-select") pin controlando el **MISO** salida pin (cuando **SS *** es alto, **MISO** no es empujado). Tu programa debe tener un `'#include <SPI.h>'` si desea utilizar la funcionalidad de la incorporado SPI Arduino objeto y la biblioteca. Alternativamente, puede optar por crear su propio "bit banged" **MOSI** y **SCK** Señala a empujar este dispositivo.



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

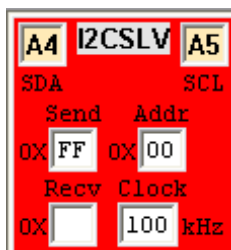
Al hacer doble clic (o hacer clic con el botón derecho) en el dispositivo, puede abrir un compañero ventana más grande que en cambio permite y Para completar el

búfer máximo de 32 bytes (para emular SPI dispositivos que automáticamente devuelve sus datos), y para ver los últimos 32 bytes recibidos (todos como pares hexadecimales). **Tenga en cuenta que** el siguiente byte buffer TX es Enviado automáticamente a 'DATA' solamente **después** una completa `'SPI.transfer()'` ¡ha completado!



I2C Esclavo TwoWire ('I2CSLV')

Este 'I/O' dispositivo solo emula un *modo esclavo* dispositivo. Al dispositivo se le puede asignar una dirección de bus I2C usando una entrada de dos hexágonos dígito en su cuadro de edición 'Addr' (solo responderá a I2C Transacciones de bus que involucren su dirección asignada). El dispositivo envía y recibe datos sobre su drenaje abierto (solo desplegable) **ASD** pin, y responde a la señal del reloj del bus en su drenaje abierto (solo bajada) **SCL** pin. Aunque el 'Uno' será el maestro de bus responsable de generar el **SCL** señal, este esclavo dispositivo también tirará **SCL** baja durante su fase baja para extender (si es necesario) el tiempo bajo del bus a uno adecuado a su velocidad interna (que se puede configurar en su cuadro de edición 'Clock').



Tu programa debe tener un `'#include <Wire.h>'` línea si desea utilizar la funcionalidad de la `'TwoWire'` Biblioteca para interactuar con este dispositivo. Alternativamente, puede elegir crear sus propios datos de bit banged y señales de reloj para empujar este esclavo dispositivo.

Se puede configurar un solo byte para la transmisión de vuelta al maestro 'Uno' en el cuadro de edición 'Send', y se puede ver un solo byte (recibido recientemente) en su cuadro de edición 'Recv' (solo lectura). **Tenga en cuenta que** El valor del cuadro de edición 'Send'

siempre refleja la **siguiente** Byte para la transmisión desde este búfer de datos interno dispositivo.

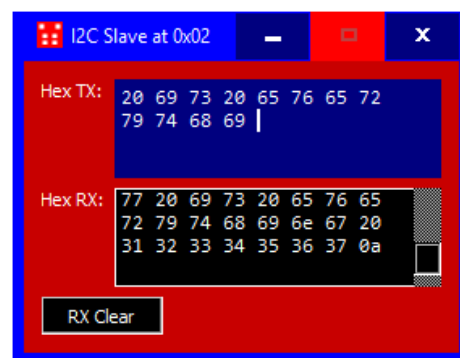
Al hacer doble clic (o hacer clic con el botón derecho) en el dispositivo, puede abrir un compañero ventana más grande que, en

cambio, le permite completar un búfer FIFO máximo de 32 bytes (para emular TWI dispositivos con dicha funcionalidad) y ver (hasta un máximo de 32) bytes de los datos recibidos más recientemente (como dos pantalla hex-dígito de 8 bytes por línea). El número de líneas en estos dos cuadros de edición corresponde al tamaño del búfer TWI elegido (que se puede seleccionar usando **Configurar | Preferencias**). Esto ha sido agregado como una opción desde el Arduino. `'Wire.h'` usos de la biblioteca **cinco** tales RAM almacena en su código de implementación, que es caro en la memoria RAM. Editando la instalación de Arduino.

`'Wire.h'` archivo para cambiar la constante definida

`'BUFFER_LENGTH'` (y también editando el compañero

`'utility/twi.h'` archivo para cambiar TWI buffer length) para que sean 16 u 8, un usuario *podría* reducir significativamente la sobrecarga de memoria RAM de El 'Uno' en un objetivo **implementación de hardware** - UnoArduSim por lo tanto refleja esta posibilidad del mundo real a través de **Configurar | Preferencias** .

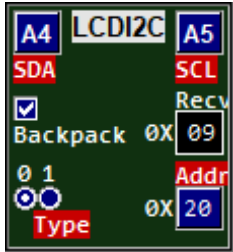


LCD del Texto I2C ('LCDI2C')

Este dispositivo 'I/O' emula a, o4 4-line 1,2 personaje-LCD, en uno de tres modos:

- a) mochila escriba 0 (expansor de puertos estilo Adafruit con hardware que tiene dirección de bus I2C 0x20-0x27)
- b) tipo mochila expansor de puertos 1 (estilo DFRobot con hardware que tiene dirección de bus I2C 0x20-0x27)
- c) no mochila (modo de interfaz I2C nativo integrado que tiene dirección de bus I2C 0x3C-0x3F)

Apoyando código de la biblioteca para cada modo dispositivo se ha previsto dentro de la carpeta 'include_3rdParty' del directorio de instalación UnoArduSIm: 'Adafruit_LiquidCrystal.h', 'DFRobot_LiquidCrystal.h' y 'Native_LiquidCrystal.h', Respectivamente.



El dispositivo se le puede asignar las direcciones de bus I2C utilizar una entrada de dos-hex-dígito en su 'Addr' cuadro de edición (que sólo responderá a I2C transacciones de bus que implican su dirección asignada). El dispositivo recibe dirección de bus y de datos (y responde con ACK = 0 o NAK = 1) en su fuga abierta (pull-down-only) SDA pin. Puede comandos LCD única escritura y datos DDRAM - te **no puedo** leer de nuevo los datos de los lugares DDRAM escrito ..



Haga doble clic o botón derecho del ratón para abrir el monitor de pantalla LCD ventana desde el que también puede establecer el tamaño de la pantalla y el juego de caracteres.

LCD del Texto SPI ('LCDSPI')

Este 'I/O' dispositivo emula una, o4 4-line 1,2 personaje-LCD, en uno de dos modos:

- a) mochila (Expansor de puertos SPI estilo Adafruit)
- b) sin la mochila (modo nativo integrado SPI interfaz - como se muestra a continuación)

Apoyando código de la biblioteca para cada modo dispositivo se ha previsto dentro de la carpeta 'include_3rdParty' del directorio de instalación UnoArduSIm: 'Adafruit_LiquidCrystal.h' ,, y 'Native_LiquidCrystal.h' , Respectivamente.



Pin 'SID' es en datos en serie, 'SS' es el activo bajo dispositivo-SELECT, 'SCK' es el reloj pin, y 'RS' son los datos / comandos pin. Puede comandos LCD única escritura y datos DDRAM (SPI todas las transacciones son escrituras) - **seno puedo** leer de nuevo los datos de los lugares DDRAM escritos.

Haga doble clic o botón derecho del ratón para abrir el monitor de pantalla LCD ventana desde el que también puede establecer el tamaño de la pantalla y el juego de caracteres.

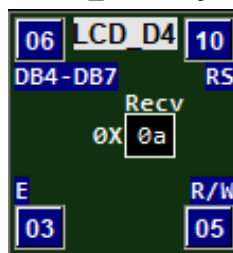


LCD del Texto D4 ('LCD_D4')

Este 'I/O' dispositivo emula una, o4 4-line 1,2 carácter-LCD que tiene una interfaz de bus de 4 bits en paralelo. bytes de datos se escriben / leído en **dos mitades** en sus 4 datos donde el cuadro de edición contiene pins 'DB4-DB7' (la el número más bajo de sus 4 números consecutivos pin), - datos tiene una velocidad en la caída de bordes en la 'E' (activar) pin, con dirección de datos controlada por el 'R/W' pin, y de datos LCD / modo de comando por el 'RS' pin.

Apoyando código de la biblioteca se ha proporcionado dentro de la 'include_3rdParty' carpeta del directorio de instalación UnoArduSIm:

'Adafruit_LiquidCrystal.h', Y
'Native_LiquidCrystal.h' ambos trabajan.

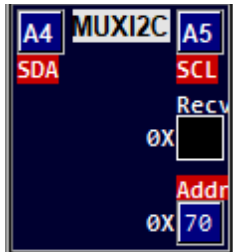


Haga doble clic o botón derecho del ratón para abrir el monitor de pantalla LCD ventana desde el que también puede establecer el tamaño de la pantalla y el juego de caracteres.



Multiplexor DEL I2C ('MUXI2C')

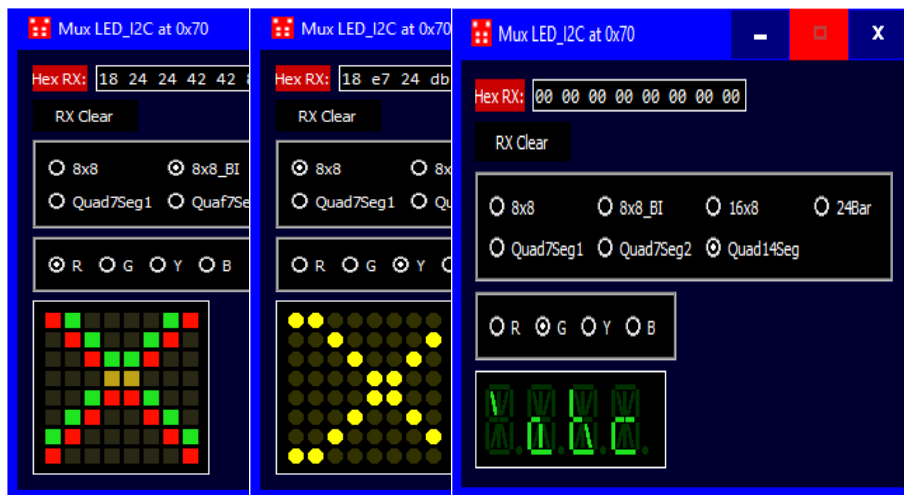
Este 'I/O' dispositivo emula un controlador HT16K33 I2C-interfaz (h I2C AVING dirección de bus 0x70-0x77) a la que uno de varios tipos diferentes de multiplexados DEL pantallas se puede unir:



- a) 8x8 o 16x8 DEL matriz
- b) 8x8 bicolor DEL matriz
- c) 24-bi color-DEL bar
- d) dos estilos de 4-dígito displays de 7 segmentos
- e) uno 14 segmentos pantalla alfanumérica 4-dígito

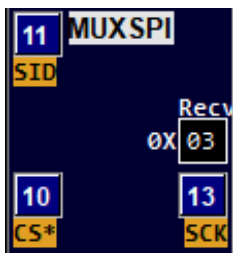
Todo son apoyados por el 'Adafruit_LEDBackpack.h' código proporciona dentro del 'include_3rdParty' carpeta:

Haga doble clic (O haga clic) para abrir un mayor ventana elegir y vista uno de los varios colores DEL pantallas.



Multiplexor DEL SPI ('MUXSPI')

Un controlador de multiplexado-DEL basado en el MAX6219, con el apoyo 'MAX7219.h' código proporciona dentro de la carpeta 'include_3rdParty' a empujar hasta ocho dígitos de 7 segmentos.

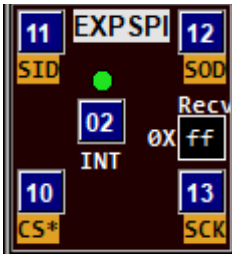


Haga doble clic (O haga clic) para abrir un mayor ventana para ver el color 8-dígito display de 7 segmentos-.

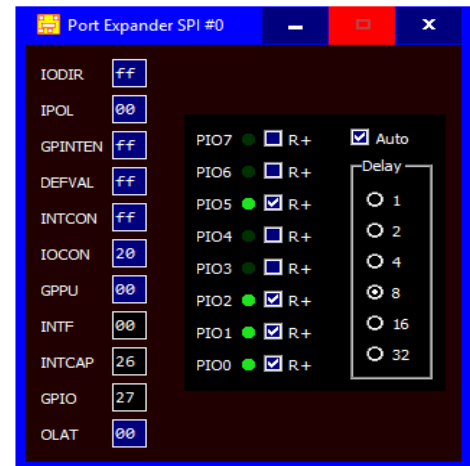


Puerto de Expansión SPI ('EXPSPi')

Un niño de 8 bits expansor de puertos basado en el MCP23008, con el apoyo 'MCP23008.h' código proporcionado dentro de la 'include_3rdParty' carpeta. Puede escribir a MCP23008 registros, y leer de nuevo el GPIO pin los niveles. Las interrupciones pueden ser activados en cada cambio GPIO pin - una interrupción disparada se empujar la 'INT' pin.



Haga doble clic (O haga clic) abrir **ventana una mayor para ver el 8 líneas de puerto GPIO**, y las resistencias pull-up adjuntos. Puede cambiar pull-ups manualmente haciendo clic, o adjuntar un contador que va a cambiar periódicamente de una manera ascendente recuento. La velocidad a la que los incrementos de recuento se determina por el retraso escala-down factor de elegido por el usuario (un factor 1x corresponde a un incremento de aproximadamente cada 30 milisegundos; factores de retardo más altas dan un ritmo más lento hasta-count)

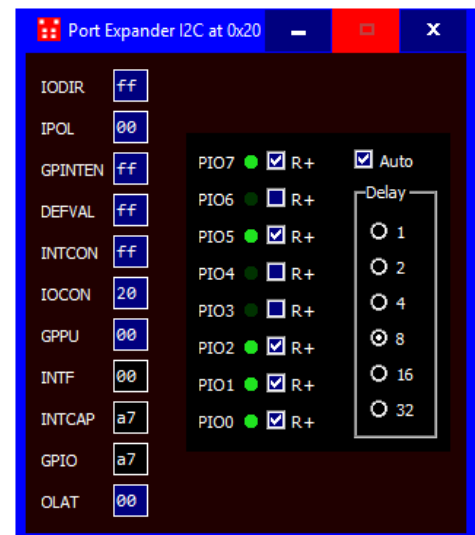


Puerto de Expansión I2C ('EXPI2C')

Un niño de 8 bits expansor de puertos basado en el MCP23008, con el apoyo 'MCP23008.h' código proporciona dentro del 'include_3rdParty' carpeta. Capacidades coinciden con el 'EXPSPi' dispositivo.



Haga doble clic (O haga clic) para abrir un mayor ventana como el vaivén 'EXPSPi' dispositivo.



'1-Wire' Esclavo ('OWIISLV')

Este 'I/O' dispositivo emula uno de un pequeño conjunto de bus '1-Wire' dispositivos conectado a pin OWIO. Puede crear un bus '1-Wire' (con uno o más de estos esclavos '1-Wire' dispositivos) en el 'Uno' pin de su elección. Esta cabina dispositivo puede utilizarse llamando al '**OneWire.h**' biblioteca módulos funcionales después de colocar un '**#include <OneWire.h>**' línea en la parte superior de su programa. Alternativamente, también puede usar señales de bit banded en OWIO para este dispositivo (aunque eso es muy difícil de hacer correctamente sin causar un conflicto eléctrico, tal conflicto todavía es posible incluso cuando se usa el '**OneWire.h**' módulos funcionales, pero dichos conflictos se reportan en UnoArduSim).

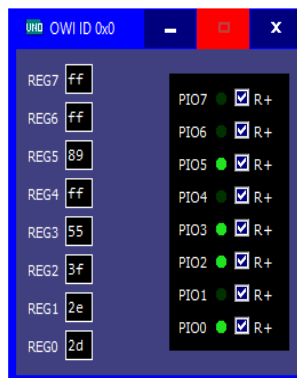
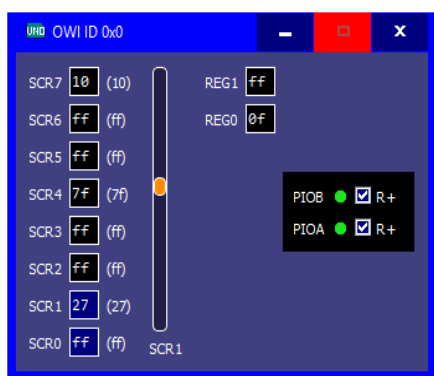


Cada OWISLV dispositivo del mundo real debe tener un i-byte i único de 64 bytes (64 bits) número de serie interno: en UnoArduSim, este es simplificado por el usuario que proporciona un hexadecimal de 1 byte corto '**ID**' valor (que se asigna secuencialmente de forma predeterminada en la carga / adición dispositivo), más el '**Fam**' Código de familia para ese dispositivo. UnoArduSim reconoce un pequeño conjunto de códigos de familia a partir de V2.3 (0x28, 0x29, 0x3A, 0x42) que cubre el sensor de temperatura, y el IO (PIO) dispositivos paralelo (un código de familia no reconocido establece el dispositivo para convertirse en un bloc de notas genérico de 8 bytes dispositivo con sensor genérico).

Si la familia dispositivo no tiene registros PIO, los registros **D0** y **D1** representar Los dos primeros bytes del scratchpad, de lo contrario, representan el PIO. Registro de "estado" (niveles pin reales) y registro de datos de cierre PIO pin, respectivamente.

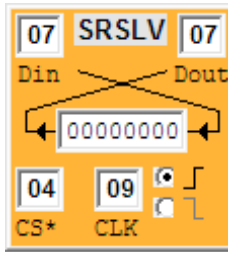
Por **doble clic** (o **clic derecho**) en el dispositivo, una más grande **OWIMonitor** ventana está abierto. Desde ese ventana más grande, puede inspeccionar todos los registros dispositivo, cambiar las ubicaciones de bloc de notas SCR0 y SCR1 usando ediciones y un control deslizante (nuevamente, SCR0 y SCR1 solo corresponden a **D0** y **D1** si no está presente el PIO), o establezca pull-ups pin PIO externos. Cuando se editan SCR0 y SCR1, UnoArduSim recuerda estos valores editados como la "preferencia" del usuario que representa un valor inicial (a partir de Reiniciar) que representa la salida del valor firmado del sensor de dispositivo; el control deslizante se restablece al 100% (un factor de escala de 1.0) En el momento de la edición. Cuando el control deslizante se mueve posteriormente, '**signed**' el valor en SCR1 se reduce de acuerdo con la posición del control deslizante (factor de escala de 1.0 a 0.0): esta función le permite probar fácilmente la respuesta de su programa para cambiar sin problemas los valores del sensor. .

Para un dispositivo con PIO pins, cuando establece las casillas de verificación de nivel pin, UnoArduSim recuerda estos valores comprobados como los pull-ups actuales aplicados externamente al pins. Estos valores de recuperación externos se utilizan junto con los datos de cierre pin (registro **D1**) para determinar los niveles reales finales de pin, y encender o apagar el DEL verde conectado al PIO pin (el pin solo va '**HIGH**' si se aplica un pull-up externo, **y** el correspondiente **D1** bit de cierre es un '**1**').



Registro de Desplazamiento Esclavo ('SRSLV')

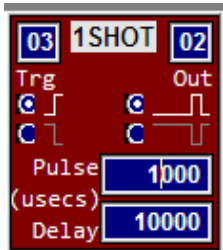
Este 'I/O' dispositivo emula un simple registro de cambios dispositivo con un activo bajo **SS *** ("slave-select") pin controlando el '**Dout**' salida pin (cuando **SS *** es alto, '**Dout**' no es empujado). Su programa podría usar la funcionalidad de la incorporado SPI Arduino objeto y la biblioteca. Alternativamente, puede optar por crear su propio "bit banged" '**Din**' y **CLK** Señala a empujar este dispositivo.



El dispositivo siente transiciones de borde en su **CLK** Entrada que activa el desplazamiento de su registro - la polaridad de lo detectado **CLK**. El borde puede ser elegido usando un control de botón de radio. En cada **CLK** borde (de la polaridad detectada), el registro captura su **Estruendo** nivel en la posición de bit menos significativo (LSB) del registro de desplazamiento, ya que los bits restantes se desplazan simultáneamente una posición hacia la posición MSB. Cuando **SS *** es bajo, el valor actual en la posición MSB del registro de desplazamiento es empujado en '**Dout**'.

Uno-Trago ('1SHOT')

Este 'I/O' dispositivo emula un digital de disparo único que puede generar un pulso de polaridad y ancho de pulso elegidos en su 'Out' pin, que se produce después de un retraso específico de un borde de activación recibido en su **Trg** (disparo) entrada pin. Una vez que se recibe el borde de activación especificado, sincronización comienza y luego no se reconocerá un nuevo impulso de activación hasta que 'Out' El pulso se ha producido (y ha terminado completamente).



Un posible uso de este dispositivo es simular Sensores de rango de ultrasonido que generan un pulso de rango en respuesta a un pulso de disparo. También se puede utilizar donde quiera que se genere una señal de entrada pin sincronizada (después de su retraso elegido) a una señal de salida pin creada por su programa.

'Pulse' y los valores 'Delay' se pueden escalar desde ventana principal **Barra de herramientas** Control deslizante del factor de escala 'I/O____S' agregando el sufijo 'S' (o 's') a uno (o ambos).

Otro uso para este dispositivo es en la prueba de un programa que usa interrupciones, y le gustaría ver qué sucede si un **instrucción específica programa** se interrumpe. Desconecte temporalmente el 'I/O' Dispositivo que ha conectado al pin 2 (o pin 3) y sustitúyalo por un '1SHOT' dispositivo cuyo 'Out' pin está conectado a 'Uno' pin 2 (o pin3, respectivamente), luego puede activar su entrada 'Trg' (asumiendo que la sensibilidad de borde ascendente se establece allí) insertando el par de instrucciones { '**digitalWrite(LOW)** ', '**digitalWrite(HIGH)** ' } **justo antes** a la instrucción dentro de la cual desea que ocurra la interrupción. Establecer el 1SHOT; s 'Delay' para sincronizar el pulso producido en 'Out' para que ocurra dentro de la instrucción programa que sigue a este par de instrucciones de activación. Tenga en cuenta que algunas instrucciones enmascaran interrupciones (como '**SoftwareSerial.write(byte)** ', un así que no puede ser interrumpido

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



Este 'I/O' dispositivo es en realidad un 'Uno' placa de circuito desnudo que puede programarse (con un programa separado) para emular un 'I/O' dispositivo cuyo comportamiento se puede definir completamente. Puede elegir hasta cuatro pines (IO1, IO2, IO3 e IO4) que este esclavo 'Uno' compartirá en común con el maestro (principal) Uno que aparece en medio de su **Panel del Banco de laboratorio**. Al igual que con otros dispositivos, cualquier conflicto eléctrico entre este esclavo 'Uno' y el maestro 'Uno' será detectado y marcado. Tenga en cuenta que todos los pines compartidos son **directamente** conectados, **excepto por pin 13** (donde se asume una resistencia de la serie R-1K entre los dos pines para evitar un conflicto eléctrico en Reiniciar, este esclavo 'Uno' no puede tener 'I/O' dispositivos de su propio. La imagen de la izquierda muestra que las 4 conexiones pin son las 4 pines del sistema SPI (SS *, MISO, MOSI, SCK): esto le permitiría a programar este esclavo como esclavo (o maestro) SPI genérico cuyo comportamiento definir.

Por **doble clic** (o **clic derecho**) en este dispositivo, se abre una ventana más grande para mostrar que este esclavo 'Uno' tiene su propio **Panel de Código** y asociado **Panel de Variables**, al igual que el maestro 'Uno' tiene. También tiene su propio **Barra de herramientas**. Que puedes usar para **carga y control ejecución** de un esclavo programa: las acciones de los iconos tienen los mismos atajos de teclado que los de la ventana principal. (**Cargar** es **Ctrl-L**, **Guardar** es **Ctrl-S** etc.). Una vez cargado, puedes ejecutar desde **ya sea** El Main UnoArduSim ventana, o desde el interior de este Esclavo Monitor ventana, en ambos casos, Main 'Uno' programa y Esclavo 'Uno' programa permanecen bloqueados en sincronización con el paso del tiempo real a medida que avanzan sus ejecuciones. **Para elegir el Panel de Código que tendrá el carga, búsqueda y enfoque ejecución, hacer clic en su barra de título ventana principal: el Panel de Código sin foco tiene su barra de herramientas acciones en gris**.

Algunas ideas posibles para el esclavo dispositivos que podría ser programado en este 'PROGIO' dispositivo se enumeran a continuación. Para la emulación serial, I2C o SPI dispositivo, puede usar la codificación programa apropiada con matrices para los buffers de envío y recepción, en orden para emular el comportamiento complejo de dispositivo:

a) Un maestro SPI o esclavo dispositivo. UnoArduSimV2.4 ha extendido el '**SPI.h**' biblioteca para permitir el modo esclavo S {operación PI a través de un opcional '**mode**' parámetro en '**SPI.begin(int mode = SPI_MASTR)**'. Pasar explícitamente '**SPI_SLV**' para seleccionar el modo esclavo (en lugar de confiar en el modo maestro predeterminado). Ahora también puede definir una interrupción de usuario módulo funcional (llamémoslo '**onSPI**') ya sea en 'Uno' para transferir bytes llamando otra extensión añadida '**SPI.attachInterrupt(user_onSPI)**'. **Ahora c** todo '**rxbyte=SPI.transfer(tx_byte)**' desde dentro de tu '**user_onSPI**' módulo funcional borrará la bandera de interrupción, y **regreso inmediatamente** con el byte recién recibido en su variable '**rxbyte**'. Alternativamente, puede evitar adjuntar una interrupción SPI, y en lugar de eso simplemente llama '**rxbyte=SPI.transfer(tx_byte)**'. Desde el interior de su programa principal - esa llamada será **bloque ejecución** hasta que se haya transferido un byte SPI, y entonces será **regreso** con el byte recién recibido en el interior '**rxbyte**'.

b) un genérico **de serie 'I/O'** dispositivo. Tendrás que comunicarte entre uno y otro. '**SoftwareSerial**' definido dentro de su maestro 'Uno' programa y otro creado dentro del esclavo 'Uno' programa - estos deben usar **opuesto** definido '**txpin**' y '**rxpin**'. Valores para que el esclavo 'Uno'. '**SoftwareSerial**' Recibe en el pin en el que el maestro '**SoftwareSerial**' transmite (No puede hacer una conexión similar usando la '**Serial**' puerto en pines 0 y 1 en ambos tableros, ya que los dos '**Serial**' arroyos haría empujar sus señales de TX entre sí).

c) Un genérico 'I2C' maestro o esclavo dispositivo. La operación Esclavo ahora se ha agregado para completar la implementación de UnoArduSim; '**Wire.h**' biblioteca (módulos funcionales '**begin(address)**', '**onReceive()**' y '**onRequest**' ahora se han implementado para soportar operaciones en modo esclavo).

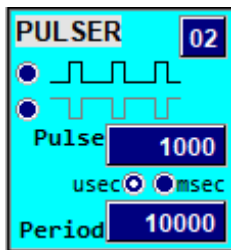
d) Un digital genérico **Pulsador**. Utilizando '**delayMicroseconds()**' y '**digitalWrite()**' llamadas dentro '**loop()**' en su 'PROGIO' programa, puede definir el '**HIGH**' y '**LOW**' Intervalos de un tren de pulsos. Añadiendo un separado '**delay()**' llama dentro de tu '**setup()**' módulo funcional, puede retrasar el inicio de este tren de pulsos. Incluso puedes variar los anchos de pulso. como el tiempo avanza utilizando un contador variable. También podría utilizar un separado '**IOX**' pin como disparador para iniciar el sincronización de un '1Shot' emulado (o doble disparo, triple disparo, etc.) dispositivo, y puede controlar el ancho del pulso producido para cambiar de la manera que desee a medida que avanza el tiempo.

e) Un señalador aleatorio. Esto es una variación en un digital **Pulsador** que también usa llamadas a '`random()`' y '`delayMicroseconds()`' para generar tiempos aleatorios en los que '`digitalWrite()`' una señal en cualquier pin elegido compartido con el maestro 'Uno'. Usando los cuatro '**IOx**' pins permitiría cuatro señales simultáneas (y únicas).

d) Un genérico 'bit-banging' dispositivo. Cree cualquier bit o patrón de reloj que desee para las señales empujar a cualquier subsistema en el maestro 'Uno'. Y recuerda, **puedes usar cualquier esclavo 'Uno' programa Instrucciones o subsistemas que desee.**

Generador de Pulso Digital ('PULSER')

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

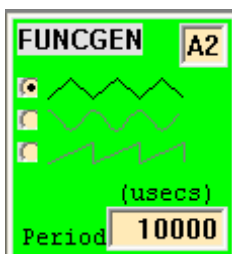


Las anchuras de época y de impulsos (en microsegundos) se pueden establecer utilizando edit-cajas-el plazo mínimo es de 50 microsegundos, y el pulso de anchura mínima es de 10 microsegundos. Se puede elegir entre los valores sincronización en microsegundos ('usec') y milisegundos ('msec'), y esta elección se guardará junto con los otros valores cuando 'Save' de la Configurar | I / O de diálogo Dispositivos.

La polaridad puede también ser elegida: o bien impulsos positivos de vanguardia (0 a 5V) o negativo pulsos de vanguardia (5V a 0V).

Los valores 'Pulse' y 'Period' se pueden escalar de la principal **Barra de herramientas 'I/O_____S'**-factor de escala control deslizante mediante la adición como un 'S' sufijo (o 's') a uno (o ambos). Esto le permite a continuación, cambiar el 'Pulse' 'Period' o el valor **dinamicamente** durante ejecución.

Analog Generador de Funciones ('FUNCGEN')



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

El período (en microsegundos) se puede configurar mediante el cuadro de edición, el período mínimo permitido es de 100 microsegundos. El forma de onda que crea puede ser elegido para ser sinusoidal, triangular o diente de sierra (para crear una onda cuadrada, use un 'PULSER' en su lugar). En períodos más pequeños, se utilizan menos muestras por ciclo para modelar el forma de onda producido (solo 4 muestras por ciclo en un período = 100 microsegundos).

El 'Period' valor puede ser escalado desde el ventana principal **Barra de herramientas** Control deslizante del factor de escala 'I/O_____S' agregando como sufijo la letra 'S' (o 's').

Motor Paso a Paso ('STEPR')

Este 'I/O' dispositivo emula un motor paso a paso bipolar o unipolar de 6 V con un controlador controlador integrado empujado de **o dos** (en **P1** , **P2**) **o cuatro** (en **P1** , **P2** , **P3** , **P4**) señales de control. El número de pasos por revolución también se puede establecer. Puedes usar el '**Stepper.h**' módulos funcionales '**setSpeed()**' y '**step()**' a empujar el 'STEPR'. Alternativamente, 'STEPR' *también responder* a tu propio '**digitalWrite()**' " Golpeó un poco las señales empujar.



El motor está modelado con precisión tanto mecánica como eléctricamente. Las caídas de tensión del motor controlador y la inductancia y la inductancia variables se modelan junto con un momento de inercia realista 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 milisegundos. Debido al modelado realista, notará que los pulsos pin de control muy estrecho *no lo obtengas* El motor debe pisar, tanto por el tiempo de aumento de la corriente finita como por el efecto de la inercia del rotor. Esto concuerda con lo que se observa cuando se maneja un motor paso a paso real desde un 'Uno'

con, por supuesto, un adecuado (**y requerido**) Chip controlador del motor entre los cables del motor y el 'Uno'!

Un error desafortunado en el Arduino. '**Stepper.h**' el código de la biblioteca significa que al reiniciar el motor paso a paso no estará en la posición Paso 1 (de cuatro pasos). Para superar esto, el usuario debe utilizar '**digitalWrite()**' en su / ella '**setup()**' rutina para inicializar los niveles de control pin a la '**step(1)**' los niveles apropiados para el control 2-pin (0,1) o 4-pin (1,0,1,0), y permiten que el motor 10 milisegundos se mueva a la posición inicial del motor deseada de referencia a las 12 del mediodía.

AS de V2.6, este dispositivo ahora incluye un 'sync' DEL (verde para sincronizada, o rojo cuando fuera a una o más etapas). También, Además del número de pasos por revolución, dos valores adicionales (ocultos) opcionalmente puede se indique en la IODevs.txt archivo para especificar el load- mecánica por ejemplo, valores de 20, 50, 30 especifica 20 pasos por revolución, un momento de carga de inercia 50 veces la del rotor del motor en sí, y un par de carga de 30 por ciento del par de torsión de sujeción del motor completo.

Tenga en cuenta que **la reducción de engranajes no es compatible directamente** debido a la falta de espacio, pero puede emularlo en su programa implementando un contador variable de módulo N y solo llamando '**step()**' cuando ese contador llega a 0 (para reducción de engranajes por factor N).

Pulsado Motor Paso a Paso ('PSTEPR')

Este 'I/O' dispositivo emula un 6V **Micro-escalonamiento** bipolar Motor Paso a Paso con un controlador integrado controlador empujado por un '**Step**' pulsadas pin, una activa baja '**EN***' (Activar) pin, y una '**DIR**' (dirección) pin . El número de pasos completos por revolución también se puede ajustar directamente, junto con el número de micro-pasos por completo el paso (1,2,4,8, o 16). Además de estos ajustes, dos valores adicionales (ocultos) pueden opcionalmente se indique en la IODevs.txt archivo para especificar el load- mecánica por ejemplo, valores de 20, 4, 50, 30 especifica 20 pasos por revolución, 4 micro-pasos por paso completo, un momento de carga de inercia 50 veces la del motor propio rotor, y un par de carga de 30 por ciento del par de sujeción del motor completo.

Debe escribir código para el control empujar pins apropiadamente.

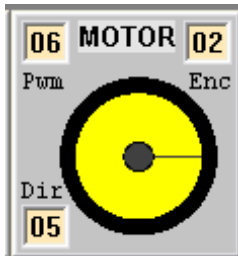


El motor se modela con precisión tanto mecánica como eléctricamente. Tensión del motor-controlador gotas y variando reticencia y la inductancia se modelan junto con un momento de inercia realista con respecto a la celebración de torque. El rotor bobinado del motor tiene una resistencia modelada de $R = 6$ ohmios y una inductancia de $L = 6$ mili-Henries que crea una constante de tiempo eléctrica de 1,0 milisegundo.

este dispositivo incluye una actividad 'STEP' amarillo DEL, y una DEL 'sync' (verde para sincronizada, o rojo cuando fuera a una o más etapas).

Motor DC ('MOTOR')

Este 'I/O' dispositivo emula un suministro de 6 voltios de 100: 1 motor de CC con un controlador controlador integrado empujado por una señal de modulación de ancho de pulso (en su **Pwm** entrada), y una señal de control de dirección (en su **Dir** entrada). El motor también tiene una salida de codificador de rueda que empuja su **Enc** salida pin. Puedes usar '**analogWrite()**' a empujar el **Pwm** pin con 490 Hz (en pins 3,9,10,11) o 980 Hz (en pins 5,6) PWM forma de onda de ciclo de trabajo entre 0.0 y 1.0 ('**analogWrite()**' valores 0 a 255). Alternativamente, 'MOTOR' *también responder* a tu propio '**digitalWrite()**' " Golpeó un poco las señales empujar.



El motor está modelado con precisión tanto mecánica como eléctricamente. Si se tienen en cuenta las caídas de tensión del transistor controlador del motor y el par de engranajes realistas sin carga, se obtiene una velocidad total de aproximadamente 2 revoluciones por segundo, y un par de detención de poco más de 5 kg-cm (que se produce a un ciclo de trabajo PWM constante de 1,0), con una Momento de inercia total del motor más la 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 al modelado realista, notará que los pulsos PWM muy estrechos *no lo obtengas* El motor debe girar, tanto por el tiempo de aumento de la corriente finita como por el tiempo de apagado significativo después de cada impulso estrecho. Estos se combinan para causar un impulso insuficiente del rotor para superar el latigazo similar a un resorte de la caja de engranajes bajo fricción estática. La consecuencia es 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 engranajes real desde un 'Uno', por supuesto, con un adecuado (**y requerido**) Motor controlador módulo entre el motor y el 'Uno'!

El codificador de motor emulado es un sensor de interrupción óptica montado en el eje que produce un ciclo de trabajo forma de onda del 50% que tiene 8 periodos completos de alta y baja por revolución de rueda (por lo que su programa puede detectar los cambios de rotación de la rueda a una resolución de 22,5 grados).

Servo Motor ('SERVO')

Este 'I/O' dispositivo emula un servomotor de CC de suministro de 6 voltios PWM-empujado controlado por posición. Los parámetros de modelado mecánico y eléctrico para el funcionamiento del 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 milisegundos . Si el abajo a la izquierda casilla de verificación está marcada, el servo se convierte en un **rotación continua** Servo con la misma velocidad máxima, pero ahora el ancho de pulso PWM establece el **velocidad** en lugar del ángulo



Tu programa debe tener un '**#include <Servo.h>**' línea antes de declarar su '**Servo**' instancias) Si elige utilizar la funcionalidad de biblioteca '**Servo.h**', p.ej '**Servo.write()**' , '**Servo.writeMicroseconds()**' Alternativamente, 'SERVO' también responde a '**digitalWrite()**' Señales "bit banged". Debido a la implementación interna. de UnoArduSim, está limitado a 6 'SERVO' dispositivos.

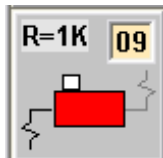
Altavoz Piezo ('PIEZO')



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

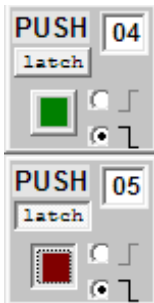
También puede escuchar un 'PULSER' conectado o 'FUNCGEN' dispositivo conectando un 'PIEZO' al pin que dispositivo empuja.

Resistor de Dslizamiento ('R=1K')



Este dispositivo le permite al usuario conectarse a un 'Uno' pin ya sea una resistencia de pull-up de 1 k-Ohm a + 5V, o una resistencia de pull-down de 1 k-Ohm a tierra. Esto le permite simular cargas eléctricas agregadas a un hardware real dispositivo. Haciendo clic izquierdo en el interruptor deslizando **cuerpo** puede alternar su selección de pull-up o de pull-down deseada. El uso de uno o varios de estos dispositivos le permitiría establecer un "código" de un solo (o de varios) bits para que su programa lea y responda.

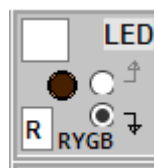
Pulsador ('PUSH')



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

Haciendo clic con el botón izquierdo en el botón o presionando cualquier tecla, cierra el contacto del botón. En **momentáneo** modo, permanece cerrado mientras mantenga presionado el botón o la tecla del mouse, y en **pestillo** modo (habilitado haciendo clic en el botón 'latch' botón) permanece cerrado (y de un color diferente) hasta que vuelva a presionar el botón. Se producirá un rebote de contacto (durante 1 milisegundo) cada vez que utilizar el **barra espaciadora** Presionar el pulsador.

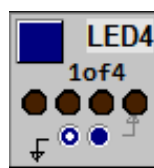
DEL coloreado ('LED')



Puede conectar un DEL entre el resistor limitador de corriente pin 'Uno' elegido (a través de un incorporado serie 1 k-Ohm oculta) a tierra o a + 5V; esto le da la opción de encender el DEL cuando el 'Uno' pin conectado está 'HIGH' o en su lugar cuando es 'LOW'.

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

4-DEL Fila ('LED4')



Puede conectar esta fila de 4 LED de colores entre el conjunto elegido de 'Uno' pins (cada uno tiene un resistor limitador de corriente de 1 kohmios incorporado oculto) a tierra o a + 5 V; esto le da la opción de tener los LED encendidos. arriba cuando el 'Uno' pin conectado es 'HIGH' o en su lugar cuando es 'LOW'.

los '1of4' El cuadro de edición pin acepta un solo número pin que se considerará como **el primero de cuatro consecutivos** 'Uno' pins que se conectará a los 4 LEDs.

El color DEL ('R', 'Y', 'G' o 'B') es un **opción oculta** eso puede ser **solo ser elegido por editando el IODevices.txt archivo** (cual puedes crear usando **Guardar** desde el **Configurar | I/O' Dispositivos** caja de diálogo).

7 segmentos DEL Dígito ('7SEG')



Puede conectar esta pantalla de 7 segmentos Dígito DEL a un conjunto elegido de **Cuatro consecutivos 'Uno' pins que dan el código hexadecimal** para el dígito visualizado deseado, ('0' a 'F'), y active o desactive este dígito usando el CS * pin (activo-BAJO para ENCENDIDO).

Este dispositivo incluye un decodificador incorporado que usa el **activo-ALTO** niveles en los cuatro consecutivos '1of4' pins para determinar el hexadecimal dígito solicitado para ser mostrado. El nivel en el número más bajo de pin (el que se muestra en el '1of4' cuadro de edición) representa el bit menos significativo del código hexadecimal de 4 bits.

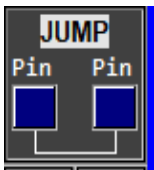
El color de los segmentos DEL ('R', 'Y', 'G' o 'B') es un **opción oculta** eso puede ser **solo ser elegido por editando el IODevices.txt archivo** puedes crear usando **Guardar** desde el **Configurar | 'I/O' Dispositivos** caja de diálogo.

Control Deslizante Analógico

Se puede conectar un potenciómetro de 0-5 V controlado por el control deslizante a cualquier 'Uno' pin elegido para producir un nivel de voltaje análogo estático (o que cambia lentamente) que será leído por 'analogRead()' como un valor de 0 a 1023. Usa el ratón para arrastrar, o haz clic para saltar, el control deslizante análogo.



Pon Puente de Alambre ('JUMP')



Puedes conectar dos 'Uno' pins junto con este dispositivo (si se detecta algún conflicto eléctrico cuando llena el segundo número pin, la conexión elegida no está permitida y esa pin está desconectada).

Este puente dispositivo tiene una utilidad limitada y es más útil cuando se combina con interrupciones, para pruebas, experimentación y programa. fines de aprendizaje. **A partir de UnoArduSim V2.4 puede encontrar que usar un 'PROGIO' dispositivo ofrece más flexibilidad que los métodos de interrupción empujado a continuación..**

Tres posibles usos para este dispositivo son los siguientes:

1) Usted puede **crear una entrada digital para probar su programa** que tiene sincronización más complejo que el que se puede producir usando cualquiera de los conjuntos de juegos suministrados Estándar 'I/O' dispositivos, como sigue:

Defina una interrupción módulo funcional (llamémosla 'myIntr') y hacer 'attachInterrupt(0, myIntr, RISING)' dentro de tu 'setup()'. Conectar un **Pulsador** dispositivo a pin2 - ahora 'myIntr()' será ejecutar cada vez que un **Pulsador** se produce un flanco ascendente. Tu 'myIntr()' módulo funcional puede ser un algoritmo que tiene programado (usando el contador global variables, y quizás incluso 'random()') para producir un forma de onda de su propio diseño en cualquier disponible 'OUTPUT' pin (digamos que es pin 9). Ahora **SALTAR** pin 9 a su 'Uno' deseado 'INPUT' pin para aplicar ese digital forma de onda generado a esa entrada pin (para probar su programa; s respuesta a ese forma de onda en particular). Puede generar una secuencia de pulsos, o caracteres en serie, o simplemente transiciones de bordes, todas de complejidad arbitraria e intervalos variables. Tenga en cuenta que si sus principales llamadas programa 'micros()' (o llama a cualquier módulo funcional que dependa de él), su 'return' valor **será incrementado** por el tiempo pasado dentro de su 'myIntr()' módulo funcional. Cada vez que la interrupción se dispara. Puede producir una ráfaga rápida de bordes cronometrados con precisión usando llamadas a 'delayMicroseconds()' de dentro 'myIntr()' (tal vez para generar un entero **byte** de un transferencia de alta velocidad baudios), o simplemente generar una transición por interrupción (tal vez para generar **un bit** de una transferencia de bajo velocidad baudios) con la **Pulsador** dispositivo 'Period' elegido de acuerdo con sus necesidades sincronización (recuerde que **Pulsador** limita su mínimo 'Period' a 50 microsegundos).

2) Usted puede **experimentar con subsistemas de bucles de retroceso:**

Por ejemplo, desconecte su 'SERIAL' 'I/O' dispositivo TX '00' pin (edítelo en un espacio en blanco), y luego **SALTAR** 'Uno' pin '01' de nuevo a 'Uno' pin '00' para emular un bucle de hardware de la ATmega 'Serial' subsistema. Ahora en tu prueba programa, por dentro 'setup()' hacer un **soltero** 'Serial.print()' de un palabra o caracter y dentro de tu 'loop()' devolver cualquier carácter recibido (cuando 'Serial.available()') haciendo un 'Serial.read()' seguido de un 'Serial.write()', y luego mira lo que pasa. Se pudo observar que una similar 'SoftwareSerial' bucle de vuelta **fallará** (como lo haría en la vida real, el software no puede hacer dos cosas a la vez).

También puedes probar **SPI** bucle de vuelta utilizando una **SALTAR** para conectar pin 11 (MOSI) de nuevo a pin 12 (MISO).



3) Usted puede contar el número y / o medir el espaciado de transiciones de nivel específicas en cualquier 'Uno' salida pin X que se producen como resultado de un complejo Instrucción de Arduino o biblioteca módulo funcional (como ejemplos: 'analogWrite()' o 'OneWire::reset()', o 'Servo::write()'), como sigue:

SALTAR pin X interrumpir pin 2 y dentro de tu 'myIntr()' utilizar una 'digitalRead()' y un 'micros()' llamada, y comparar con los niveles y tiempos guardados (de interrupciones anteriores). Puede cambiar la sensibilidad del borde para la siguiente interrupción, si es necesario, utilizando 'detachInterrupt()' y 'attachInterrupt()' desde **dentro** tu 'myIntr()'. Tenga en cuenta que no podrá realizar un seguimiento de pin transiciones que ocurren muy juntas (más cerca que el tiempo total de ejecución de tu 'myIntr()' módulo funcional), como las que ocurren con transferencias I2C o SPI, o con velocidad baudios alto 'Serial' transferencias (Aunque su interrupción módulo funcional no afectará al sincronización inter-edge de estas transferencias producidas por hardware). También tenga en cuenta que las transferencias mediadas por software (como 'OneWire::write()' y 'SoftwareSerial::write()') son protegido deliberadamente de la interrupción (por su código de biblioteca que deshabilita temporalmente todas las interrupciones, para evitar las interrupciones de sincronización), por lo que no puede medir dentro de los que utilizan este método.





Aunque en su lugar puedes hacer estas mismas medidas de espaciado de bordes **visualmente** en un **Formas de onda Digital** ventana, si está interesado en el espaciado mínimo o máximo en un gran número de transiciones, o en el conteo de transiciones, hágalo usando este 'myIntr()' -más- **SALTAR** La técnica es más conveniente. Y tu puedes mida, por ejemplo, las variaciones en el espaciado de las transiciones producidas por el programa principal (debido al efecto de su software al tomar diferentes rutas ejecución de diferentes tiempos ejecución), hacer una especie de programa "perfilado".

Los Menús







Archivo:

<u>Cargar INO o PDE Prog. (Ctrl-L)</u>	Permite al usuario elegir un programa archivo que tenga la extensión seleccionada. El programa recibe inmediatamente un Analizar.
<u>Editar/Examinar (ctrl-E)</u>	Abre el programa cargado para ver / editar.
<u>Guardar</u>	Guardar el contenido editado de programa de nuevo al programa archivo original.
<u>Guardar Como</u>	Guardar los contenidos programa editados con un nombre archivo diferente.
<u>Siguiente ('#include')</u> 	Avanza el Panel de Código para mostrar el siguiente ' #include ' archivo
<u>Anterior</u> 	Devuelve el Panel de Código mostrar a la archivo anterior
<u>Salida</u>	Sale de UnoArduSim después de recordarle al usuario que guarde los archivo modificados.

Encontrar:

<u>Subir Pila de Llamadas</u> 	Salte a la función de llamador anterior en la pila de llamadas: el Panel de Variables se ajustará a esa función
<u>Desciende Pila de Llamadas</u> 	Salte a la siguiente función llamada en la pila de llamadas: el Panel de Variables se ajustará a esa función
<u>Establecer texto Buscar (ctrl-F)</u>	Activar el Barra de herramientas Encontrar 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 Panel de Variables si uno de esos tiene el foco).
<u>Encontrar Siguiente Texto</u> 	Saltar a la siguiente aparición de texto en el Panel de Código (si tiene el foco activo), o para la siguiente aparición de Texto en el Panel de Variables (Si en cambio tiene el foco activo).
<u>Encontrar Texto Anterior</u> 	Saltar a la ocurrencia de texto anterior en el Panel de Código (si tiene el foco activo), o a la ocurrencia del texto anterior en el Panel de Variables (Si en cambio tiene el foco activo).

Ejecutar:

<u>Paso En (F4)</u>		Pasos ejecución hacia adelante por una instrucción, o <i>en un módulo funcional llamado</i> .
<u>Paso Sobre (F5)</u>		Pasos ejecución hacia adelante por una instrucción, o <i>por una llamada módulo funcional completa</i> .
<u>Paso Afuera (F6)</u>		Avances ejecución por <i>Solo lo suficiente para salir de la actual módulo funcional.</i> .
<u>Ejecutar Hacia (F7)</u>		Corre el programa, <i>detenerse en la línea programa deseada</i> - Primero debe hacer clic en resaltar una línea programa deseada antes de usar Ejecutar Hacia.
<u>Ejecutar Hasta (F8)</u>		Ejecuta el programa hasta que se produce una escritura en el variable que tenía el resaltar actual en el Panel de Variables (haga clic en uno para establecer el resaltar inicial).
<u>Ejecutar (F9)</u>		Ejecuta el programa.
<u>Detener (F10)</u>		Detiene programa ejecución (<i>y congela el tiempo</i>).
<u>Reiniciar</u>		Restablece el programa (todos los valores-variables se restablecen al valor 0, y todos los punteros-variables se restablecen a 0x0000).
<u>Animación</u>		Pasos automáticos de programa líneas consecutivas <i>con retraso artificial añadido</i> y el resaltado de la línea de código actual. La operación en tiempo real y los sonidos se pierden.
<u>Camara Lenta</u>		Reduce el tiempo por un factor de 10.

Opciones:

<u>Structors Paso Sobre/ Operadores</u>	Vuele directamente a través de constructores, destructores y sobrecarga del operador módulos funcionales durante cualquier paso (es decir, no se detendrá dentro de estos módulos funcionales).
<u>Registro-Asignación</u>	Asigne locales módulo funcional a registros libre ATmega en lugar de a la pila (genera un uso de RAM algo reducido).
<u>Error en no inicializado</u>	Marcar como un error Analizar en cualquier lugar en el que su programa intente usar un variable sin haber inicializado primero su valor (o al menos un valor dentro de un matriz).
<u>Adicional 'loop()' Retrasar</u>	Añade 1000 microsegundos de retraso cada vez. 'loop()' se llama (en caso de que no haya otras llamadas programa a 'delay()' en cualquier lugar) - útil para tratar de evitar quedarse demasiado atrás en tiempo real.
<u>Permitir interrupciones anidadas</u>	Permitir volver a habilitar con 'interrupts()' Desde dentro de una rutina de servicio de interrupción de usuario.

Configurar:

<u>'I/O' Dispositivos</u>	Abre un cuadro de diálogo para permitir al usuario elegir el tipo (s) y los números del 'I/O' dispositivos deseado. Desde este cuadro de diálogo también puede Guardar 'I/O' dispositivos a un texto archivo, y / o Cargar 'I/O' dispositivos desde un texto archivo previamente guardado (o editado) (incluidas todas las conexiones pin y las configuraciones en las que se puede hacer clic y los valores ingresados)
<u>Preferencias</u>	Abre un cuadro de diálogo para permitir al usuario establecer preferencias, incluida la sangría automática de las líneas programa de origen, lo que permite la sintaxis de Expert, la elección de la fuente tipo de letra, optar por un tamaño de fuente mayor, hacer cumplir los límites de matriz, permitir el uso de palabras clave de operador lógico, mostrar programa descarga , elección de la versión 'Uno' placa de circuito y longitud del búfer TWI (para I2C dispositivos).

VarActualizar:

<u>Permitir Automático (-) Encoger</u>	Permitir que UnoArduSim encoger muestre expandido matrices / objetos cuando se atrasa en tiempo real.
<u>Mínimo</u>	Solo actualizar el Panel de Variables mostrar 4 veces por segundo
<u>Resaltar Cambios</u>	Resaltar cambió los valores de variable durante la ejecución (puede causar una desaceleración).

Ventanas:

<u>'Serial' Monitor</u>	Conecte una E / S serial dispositivo a pins 0 y 1 (si no hay) y levante una 'Serial' monitor de texto TX / RX ventana.
<u>Restaura todo</u>	Restaurar todo el niño minimizado ventanas.
<u>Formas di Onda Digitales</u>	Restaura un Formas di Onda Digitales ventana minimizado.
<u>Forma de Onda Analógica</u>	Restaura un Forma de Onda Analógica ventana minimizado.

Ayuda:

<u>Ayuda rápido Archivo</u>	Abre el PDF archivo de UnoArduSim_QuickHelp.
<u>Ayuda completo Archivo</u>	Abre el PDF archivo de UnoArduSim_FullHelp.
<u>Error arreglos</u>	Ver importantes arreglos error desde la versión anterior.
<u>Cambio / Mejoras</u>	Ver cambios significativos y mejoras desde la versión anterior.
<u>Acerca de</u>	Muestra la versión, copyright.

'Uno' Placa de Circuito y 'I/O' Dispositivos

El 'Uno' y el 'I/O' dispositivos adjunto están modelados eléctricamente con precisión, y podrá obtener una buena idea de cómo se comportará su programas con el hardware real, y se marcará todo el pin conflictos eléctrico.

Sincronización

UnoArduSim se ejecuta lo suficientemente rápido en una PC o tableta que puede (*en la mayoría de los casos*) Modelo programa acciones en tiempo real, **pero solo si tu programa incorpora** al menos algunos pequeños ' **delay()** ' Llamadas u otras llamadas que naturalmente lo mantendrán sincronizado con el tiempo real (ver más abajo).

Para lograr esto, UnoArduSim utiliza un temporizador de devolución de llamada Ventanas módulo funcional, que le permite realizar un seguimiento preciso del tiempo real. El ejecución de una serie de instrucciones programa se simula durante un segmento de temporizador, y las instrucciones que requieren ejecución más largo (como las llamadas a ' **delay()** ') Es posible que tenga que usar múltiples segmentos de temporizador. Cada iteración del temporizador de devolución de llamada módulo funcional corrige la hora del sistema utilizando el reloj del hardware del sistema, de modo que programa ejecución se ajusta constantemente para mantener el paso de bloqueo en tiempo real. *Los únicos tiempos de ejecución debe atrasarse en tiempo real* Es cuando el usuario ha creado bucles apretados. **sin retraso adicional** , o 'I/O' dispositivos están configurados para operar con muy altas frecuencias 'I/O' dispositivo (y / o velocidad baudios) que generarían un número excesivo de eventos de cambio de nivel pin y sobrecarga de procesamiento asociado. UnoArduSim hace frente a esta sobrecarga saltándose algunos intervalos de temporizador para compensar, y esto ralentiza la progresión de programa a **abajo en tiempo real** .

Además, se muestra programas con matrices grande, o nuevamente con loops apretados **sin retraso adicional** puede causar una alta frecuencia de llamadas módulo funcional y generar una alta **Panel de Variables** la carga de actualización de la pantalla hace que se quede atrás en tiempo real: UnoArduSim reduce automáticamente la frecuencia de actualización de variable para intentar mantenerse al día, pero cuando se necesita una reducción aún mayor, elija **Mínimo** desde el **VarActualizar** menú para especificar sólo cuatro actualizaciones por segundo.

Cómo modelar con precisión el tiempo ejecución de un milisegundo para cada instrucción u operación programa **no se hace** - solo se han adoptado estimaciones muy aproximadas para la mayoría con fines de simulación. Sin embargo, el sincronización de ' **delay()** ' y ' **delayMicroseconds()** ' módulos funcionales, y módulos funcionales ' **millis()** ' y ' **micros()** ' son perfectamente exactos, y **siempre y cuando uses al menos uno de los retrasos módulos funcionales** en un bucle en algún lugar de su programa, **o** utiliza un módulo funcional que se une naturalmente a la operación en tiempo real (como ' **print()** ' que se ata al velocidad baudios elegido), entonces el rendimiento simulado de su programa será muy cercano al tiempo real (de nuevo, con la excepción de los eventos de cambio de pin excesivamente excesivos de alta frecuencia o las excesivas actualizaciones de Variables permitidas por el usuario que podrían ralentizarlo).

Para ver el efecto de las instrucciones individuales de programa w *gallina corriendo* , puede ser deseable poder ralentizar las cosas. El usuario puede configurar un factor de ralentización de 10 en el menú. **Ejecutar** .

'I/O' Dispositivo Sincronización

Estos dispositivos virtuales reciben la señalización en tiempo real de los cambios que se producen en su entrada pins, y producen salidas correspondientes en su salida pins que luego pueden ser detectadas por el 'Uno'; por lo tanto, están sincronizados de forma inherente con programa ejecución. 'I/O' dispositivo sincronización interno lo establece el usuario (por ejemplo, mediante la selección de velocidad baudios o la frecuencia de reloj), y los eventos del simulador se configuran para realizar un seguimiento del funcionamiento interno en tiempo real.

Los Sonidos

Cada 'PIEZO' dispositivo produce un sonido correspondiente 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 programa ejecución, UnoArduSim inicia y detiene la reproducción de un búfer de sonido asociado cuando se inicia / detiene ejecución.

A partir de la V2.0, el sonido ahora se ha modificado para utilizar la API de audio Qt, pero desafortunadamente su

QAudioOutput 'class' no admite el bucle del búfer de sonido para evitar quedarse sin muestras de sonido (como puede suceder durante demoras operativas de ventanas de sistema operativo más largas). Por lo tanto, para evitar la gran mayoría de los clics de sonido molestos y la ruptura del sonido durante los retrasos del sistema operativo, el sonido ahora se silencia según la siguiente regla:

El sonido está silenciado mientras UnoArduSim no sea el ventana "activo" (excepto cuando se acaba de crear y activar un nuevo ventana hijo), **e incluso** cuando UnoArduSim es el ventana principal "activo" pero el puntero del mouse es **fuera de** de Su principal área de clientes ventana.

Tenga en cuenta que esto implica que el sonido será temporalmente silenciado como rey mientras el puntero del ratón está flotando **sobre un niño ventana**, y será silenciado **Si ese niño ventana se hace clic para activarlo** (hasta que se vuelva a hacer clic en el UnoArduSim ventana principal para reactivarlo) .

El sonido siempre puede desactivarse haciendo clic en cualquier lugar dentro del área del cliente de UnoArduSim main ventana.

Debido al buffering, s La unidad tiene un retraso en tiempo real de hasta 250 milisegundos desde el tiempo de evento correspondiente en el pin del 'PIEZO' adjunto.

Limitaciones y Elementos no Soportados

Incluido Archivos

Un '< >' - entre corchetes '#include' de '<Servo.h>', '<Wire.h>', '<OneWire.h>', '<SoftwareSerial.h>', '<SPI.h>', '<EEPROM.h>' y '<SD.h>' **es** se admiten, pero solo se emulan; no se busca el archivos real; en cambio, su funcionalidad está directamente "integrada en" UnoArduSim, y son válidas para la versión Arduino soportada.

Cualquier cita '#include' (por ejemplo de "supp.ino", "myutil.cpp" o "mylib.h") es compatible, pero todos los archivos deben **residir en el mismo directorio como el padre programa archivo** ese contiene su '#include' (No hay búsqueda realizada en otros directorios). los '#include' La característica puede ser útil para minimizar la cantidad de código programa que se muestra en la **Panel de Código** en cualquier momento. Cabecera archivos con '#include' (es decir, aquellos que tienen una ".h" extensión) causará además que el simulador intente incluir el archivo del mismo nombre con un ".cpp" extensión (si también existe en el directorio del programa padre).

Asignaciones de memoria dinámica y RAM

Los operadores 'new' y 'delete' Son compatibles, al igual que Arduino nativo. 'String' objetos, **pero no llamadas directas a** 'malloc()', 'realloc()' y 'free()' que estos dependen de

El uso excesivo de RAM para las declaraciones variable se marca en el momento de Analizar, y el desbordamiento de la memoria RAM se marca durante programa ejecución. Un elemento en el menú **Opciones** le permite emular la asignación de registro ATmega normal como lo haría el AVR compilador, 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 modelo de asignación de registros). Si utilizara 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'

Memoria 'Flash' 'byte', 'int' y 'float' variables / matrices y sus correspondientes módulos funcionales de acceso de lectura son compatibles. Alguna 'F()' módulo funcional llamada ('Flash'-macro) de cualquier cadena literal **es** admitido, pero los únicos 'Flash' admitidos - módulos funcionales de acceso directo a la cadena de memoria son 'strcpy_P()' y 'memcpy_P()', así que para usar otro módulos funcionales necesitará primero copiar la cadena 'Flash' en una RAM normal 'String' variable, y luego trabajar con esa memoria RAM 'String'. Cuando usas el 'PROGMEM' Palabra clave variable-modificador, debe aparecer **en frente de** el nombre variable, y ese variable **también debe ser declarado** como 'const'.

'String' Variables

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

los 'String' los operadores soportados son +, + =, <, <=, >, > =, ==, !=, y []. Tenga en cuenta que: 'concat()' toma una **soltero** argumento que es el 'String', o 'char' o 'int' para ser anexo al original 'String' objeto, **no** dos argumentos, como se indica erróneamente en las páginas web de referencia de Arduino).

Bibliotecas Arduino

Solamente 'Servo.h', 'SoftwareSerial.h', 'SPI.h', 'Wire.h', 'OneWire.h', 'Stepper.h', 'SD.h', 'TFT.h' y 'EEPROM.h' Para el **V1.8.8 Arduino** liberar Actualmente se admiten en UnoArduSim. V2.6 introduce un mecanismo para 3rd soporte de la biblioteca a través de las partes archivos proporciona en el 'include_3rdParty' carpeta que se pueden encontrar dentro del directorio de instalación de UnoArduSim. Tratando de '#include' el ".cpp" y ".h" archivos de otra, que aún no soportado Las bibliotecas **no trabajo** ya que contendrán las instrucciones de montaje de bajo nivel y las directivas no compatibles y no reconocido archivos!

Punteros

Todos los punteros a tipos simples, matrices o objetos son compatibles. Un puntero puede equipararse a un matriz del mismo tipo (por ejemplo, 'iptr = intarray'), pero entonces habría *no hay verificación de límites matrices subsecuentes* en una expresión como 'iptr[index]'.

Módulos funcionales puede devolver punteros, o 'const' punteros, pero cualquier nivel posterior de 'const' en el puntero devuelto se ignora.

Ahi esta **sin soporte** para módulo funcional llamadas a través de **módulo funcional-punteros declarados por el usuario**.

'class' y 'struct' Objetos

Aunque se admite el poli-morfismo y la herencia (a cualquier profundidad), 'class' o 'struct' Solo se puede definir para tener como máximo **uno** base 'class' (es decir **múltiple**- la herencia no es compatible). Base: las llamadas de inicialización del constructor 'class' (a través de la notación de dos puntos) en las líneas de declaración del constructor son compatibles, pero **no** Inicializaciones de miembros usando la misma notación de dos puntos. Esto significa que objetos que contienen 'const' no son compatibles con 'static' variables o tipo de referencia variables (esto solo es posible con las inicializaciones de miembros en tiempo de construcción especificadas)

Las sobrecargas de operador de copia-asignación son compatibles junto con los constructores de movimiento y las asignaciones de movimiento, pero la conversión objeto definida por el usuario ("tipo-lanzar") módulos funcionales no es compatible.

Alcance

No hay soporte para el 'using' palabra clave, o para 'namespace', o por 'file' alcance. Todas las declaraciones no locales son, por implementación, asumidas como globales.

Alguna 'typedef', 'struct' o 'class' definición (es decir, que puede ser utilizado para futuras declaraciones), debe hacerse **global** alcance (**local** no se admiten las definiciones de dichos elementos dentro de un módulo funcional).

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

los 'unsigned' El prefijo funciona en todos los contextos legales normales. los 'const' palabra clave, cuando se utiliza, debe **preceder** el nombre variable o el nombre módulo funcional o 'typedef' Nombre que se está declarando: colocarlo después del nombre causará un error Analizar. por Declaraciones módulo funcional, solo módulos funcionales de retorno de puntero puede tener 'const' Aparecen en su declaración.

Todos UnoArduSim variables son 'volatile' por implementación, por lo que la 'volatile' palabra clave

simplemente se ignora en todas las declaraciones variable. Módulos funcionales no pueden ser declarados `'volatile'`, ni son módulo funcional-llamadas argumentos.

los `'static'` la palabra clave está permitida para variables normal, y para miembros objeto y miembro-módulos funcionales, pero está explícitamente deshabilitada para las instancias objeto en sí mismas (`'class'` / `'struct'`), para módulos funcionales no miembro, y para todos los argumentos módulo funcional.

Directivas Compilador

`'#include'` y regular `'#define'` ambos son compatibles, pero **no macro** `'#define'`. los `'#pragma'` Directivas y directivas de inclusión condicional. (`'#ifdef'`, `'#ifndef'`, `'#if'`, `'#endif'`, `'#else'` y `'#elif'`) son también **No soportado**. los `'#line'`, `'#error'` y macros predefinidas (como `'_LINE'`, `'_FILE'`, `'_DATE'` y `'_TIME'`) son también **No soportado**.

Elementos del Lenguaje Arduino.

Todos los elementos del lenguaje Arduino nativos son compatibles con la excepción del dudoso `'goto'` instrucciones (el único uso razonable que se me ocurre sería un salto (a un ciclo continuo de cierre seguro y de rescate) en el caso de una condición de error que su programa no pueda resolver)

C / C ++ - Elementos del Lenguaje

Los "calificadores de campo de bits" que guardan bits para los miembros en las definiciones de estructura son **No soportado**.

`'union'` es **No soportado**.

El "operador de coma" de bicho raro es **No soportado** (por lo que no puede realizar varias expresiones separadas por comas cuando normalmente solo se espera una expresión, por ejemplo, en `'while()'` y `'for(; ;)'` constructos).

Plantillas Módulo funcional

módulos funcionales definido por el usuario que utiliza la palabra clave "plantilla" para permitirle aceptar argumentos de tipo "genérico" son **No soportado**.

Emulación en Tiempo Real

Como se indicó anteriormente, los tiempos ejecución de las muchas instrucciones individuales posibles de Arduino programa son **no** modelado con precisión, de modo que para poder correr a una velocidad en tiempo real, su programa necesitará algún tipo de dominio `'delay()'` instrucción (al menos una vez por `'loop()'`), o una instrucción que se sincroniza naturalmente con los cambios en el nivel pin en tiempo real (como `'pulseIn()'`, `'shiftIn()'`, `'Serial.read()'`, `'Serial.print()'`, `'Serial.flush()'` etc.).

Ver **Sincronización** y **Los sonidos** arriba para más detalles sobre las limitaciones.

Notas de Lanzamiento

Error Arreglos

V2.7.0- Mar 2020

- 1) Cuando el (Ventanas-default) se adoptó el tema OS luz, la **Panel de Código** no estaba mostrando el color de resaltado introducido en V2.6 (en lugar, solamente una resaltar gris resultante de una anulación del sistema).
- 2) La versión 2.6 de forma inadvertida a la quiebra de auto-guión-ficha Formato en el primer `'switch ()'` construir.
- 3) La nueva función de navegación pila de llamadas introducido en la versión 2.6 mostró **valores incorrectos** para locales variables cuando no está dentro del módulo funcional se está ejecutando actualmente, y no con llamadas módulo funcional miembros anidada.
- 4) `'TFT :: text ()'` funcionaba, pero las funciones `'TFT :: print ()'` no funcionaban (simplemente se bloquearon para siempre). Además, `'TFT :: loadImage ()'` fallaba si `'Serial.begin ()'` se había hecho antes (que es el caso normal y ahora se requiere).
- 5) La versión 2.6 introdujo un error que muestra el valor incorrecto para 'RX' presente y pasado bytes para 'I2CSLV', 'SPISLV', 'TFT', 'LCDI2C' and 'LCDSPI' dispositivos (y su monitor ventanas).
- 6) Un cambio realizado en V2.4 causó Ejecutar | Animación destacando a pasar por alto muchos de código líneas ejecutado.
- 7) Desde V2.4, afirmando de-'SS*' 'CS*' o en un 'I/O' dispositivo en la instrucción inmediatamente después de una `'SPI.transfer ()'` haría que dispositivo a dejar de recibir el byte de datos transferidos. Además, el byte recepción en `'SPI_MODE1'` y `'SPI_MODE3'` no se encuentra en posición hasta el inicio del siguiente byte enviado por el maestro (y el byte se pierde por completo si se ha seleccionado de-la dispositivo 'CS*' antes de esa fecha).
- 8) En el nuevo `'SPI_SLV'` modo permitido desde V2.4, `'bval = SPI.transfer ()'` Sólo devuelto el valor correcto para `'bval'` Si la transferencia de bytes ya estaba completa y esperando cuando `'transfer ()'` fue llamado.
- 9) El cuadro de edición el 'DATA' 'SPISLV' dispositivos ahora recibe el valor por defecto 0xFF cuando no hay más bytes para responder con.
- 10) El estado DEL sincronización era incorrecto para 'PSTEPR' dispositivos tener más de 1 micro-paso por paso completo.
- 11) conflictos eléctrica causada por 'I/O' dispositivos reaccionar a las transiciones en el reloj 'SPI', una señal 'PWM', o una `'tone'` señal, no se informaron, y podría conducir a inexplicada (dañado) recepción de datos.
- 12) Cuando el intervalo entre interrupciones era demasiado pequeño (menos de 250 microsegundos), a (defectuoso) cambio en V2.4 alterado la sincronización de incorporado módulos funcionales que el uso ya sea del sistema temporizadores, o bucles de instrucciones, para generar retrasos (ejemplos de cada uno se `'delay ()'` y `'delayMicroseconds ()'`). Un cambio subsecuente en V2.5 causada desalineación de `'shiftOut ()'` señales de reloj de datos y cuando una interrupción pasó entre bits.
- 13) La aceptación de un texto auto-completado incorporado módulo funcional a través de la tecla Intro no se deben eliminar los tipos de parámetros a partir del texto de la llamada módulo funcional insertado.
- 14) La carga de un nuevo (fácil de interrupción-empujado) programa cuando un programa previamente funcionando todavía tenía una interrupción pendiente podría causar un accidente durante la descarga (debido a la defectuosa intentado ejecución de la nueva rutina de interrupción).
- 15) Objeto miembros auto-completos (accesible a través de 'ALT'-flecha derecha) para objetos interior `'#include'` archivos son ahora accesibles tan pronto como su `'#include'` archivo es analizado éxito.
- 16) Una declaración módulo funcional tener un espacio inadvertida dentro de un nombre de parámetro módulo funcional causó un mensaje de error claro Analizar.
- 17) Cuando ejecución detuvo en un módulo diferente de la principal programa, la Archivo | acción anterior no logró

convertirse habilitado.

18) llaves citadas solas ('{ ' y ' } ') Todavía se contaron (incorrectamente) como soportes alcanzados en el Analizar, y también confundido formato de auto-ficha-guion ..

19) 'OneWire::readBytes(byte* buf, int count)' habían sido no actualizados inmediatamente a la mostrada 'buf' contenidos en el Panel de Variables.

20) Octal-pestillo 'OWISLV' dispositivos mostraron que la producción pin niveles que se retrasó por una escritura pestillo de registro.

V2.6.0- ene 2020

- 1) Un error introdujo en V2.3 dado lugar a un accidente cuando el agregado **Cerca** botón se utiliza en el **Encontrar / Reemplazar** diálogo (en lugar de su Salida botón de la barra de título).
- 2) Si un usuario hizo programa '#include' de otro usuario archivos, la **Guardar** botón en el interior **Editar/Examinar** habría fallado en realidad un ahorro archivo modificado si hubo un error o Analizar Ejecución existente dentro de un marcado diferente archivo.
- 3) UN **Cancelar** después de **Guardar** También podría ser confuso - por estas razones el **Guardar y Cancelar** funcionalidades de botón se han cambiado (véase **Cambios y mejoras**).
- 4) soportes desequilibradas dentro de una 'class' definición podría causar un bloqueo desde V2.5.
- 5) las pruebas de lógica directa en 'long' Los valores devueltos 'false' si se establece ninguno de los 16 bits más bajos.
- 6) UnoArduSim había sido flaqueando un error cuando un puntero variable fue declarado como el bucle variable dentro de los corchetes de una 'for()' declaración.
- 7) UnoArduSim había sido no permitir pruebas lógicas que los punteros en comparación con 'NULL' o '0'.
- 8) UnoArduSim había sido no Permitiendo que la aritmética de punteros que implica un número entero variable (sólo número entero constantes habían permitido).
- 9) Las interrupciones establecer usando 'attachInterrupt(pin, name_func, LOW)' sólo se había detectado en una **transición** a 'LOW'.
- 10) Cuando se usa más de un dispositivo 'I2CSLV', un esclavo no direccionado podría interpretar los datos posteriores del bus como coincidentes con su dirección de bus (o llamada global 0x00), y así señalar falsamente ACK, corrompiendo el nivel de ACK del bus y colgando un 'requestFrom()'.
- 11) Pasando valor numérico '0' (o 'NULL') Como un argumento módulo funcional a un puntero en una llamada módulo funcional ahora se permite.
- 12) El nivel de sangría después de una tabulación de anidación 'switch()' construcciones era demasiado poco profunda cuando la elección de 'auto-indent formatting' **Configurar | Preferencias** se utilizó.
- 13) La resta de dos punteros compatibles ahora da lugar a un tipo de 'int'.
- 14) UnoArduSim había estado esperando un constructor por defecto definido por el usuario para un miembro objeto incluso si no hubiera sido declarado como 'const'.
- 15) En una ruptura ejecución, la posición dibujada de una 'STEPR', 'SERVO', o 'MOTOR' motor podría retrasarse hasta en un 30 milisegundos de movimiento detrás de su última posición calculada real.
- 16) 'Stepper::setSpeed(0)' estaba causando un accidente debido a una división por cero.
- 17) Línea sola 'if()', 'for()' y 'else' ya no causa una demasiadas pestañas auto-indentación constructos.

V2.5.0- octubre 2019

- 1) Un error introducido en V2.4 rompió la inicialización de la tarjeta 'SD' (causado un accidente).
- 2) Usando el subsistema 'SPI' en el nuevo 'SPI_SLV' Modo trabajó en forma incorrecta 'SPI_MODE1' y 'SPI_MODE3'.

- 3) Finalización automática las ventanas emergentes (según lo solicitado por 'ALT-right=arrow') se fijaron durante módulos funcionales tener objeto parámetros; la lista de las ventanas emergentes incluyen ahora también heredó miembros de la clase (base-), y auto-terminaciones ahora también aparece para '**Serial**'.
- 4) Desde V2.4 el valor de retorno de '**SPI.transfer()**' y '**SPI.transfer16()**' era incorrecta si una rutina de interrupción del usuario despedido durante la transferencia.
- 5) En V2.4, formas de onda periódicas rápidas se muestran como que tiene una duración más larga que la duración real evidente cuando se ve en mayor zoom.
- 6) el constructor '**File::File(SdFile &sdf, char *fname)**' no estaba funcionando, por lo '**File::openNextFile()**' (Que se basa en que el constructor) tampoco estaba funcionando.
- 7) UnoArduSim estaba declarando incorrectamente un error en Analizar objeto-variables y objeto-volver módulos funcionales, declarado '**static**'.
- 8) instrucciones de asignación con un '**Servo**', '**Stepper**' o '**OneWire**' objeto variable en la LHS, y un 'objeto-volver módulo funcional o constructor en el RHS, causado un miscount interna del número de objetos asociada, que conduce a una eventual choque.
- 9) pruebas booleanas sobre objetos de tipo '**File**' siempre volvían '**true**' incluso si el archivo no estaba abierto.

V2.4- mayo de 2019

- 1) Los puntos de observación Ejecutar Hasta podrían ser falsamente activados por una escritura en un variable adyacente (1 byte más bajo en la dirección).
- 2) Las selecciones de velocidad en baudios se pueden sentir cuando el ratón estaba dentro de un '**SERIAL**' o '**SFTSER**' Cuadro de lista desplegable de baudios dispositivo (Incluso cuando no se hizo clic en la velocidad en baudios).
- 3) Desde V2.2, los errores de recepción en serie ocurrieron a una velocidad de 38400.
- 4) Un cambio hecho en V2.3 causó '**SoftwareSerial**' para informar erróneamente de una interrupción deshabilitada (y por lo tanto fallar en la primera recepción de RX).
- 5) Un cambio realizado en V2.3 hizo que SPISLV dispositivos malinterpretara los valores hexadecimales introducidos directamente en su cuadro de edición '**DATA**'.
- 6) Un cambio realizado en V2.3 causó SRSLV dispositivos para detectar falsamente, y en silencio, un conflicto eléctrico en su '**Dout**' pin, y por lo tanto no permitir una asignación pin allí. Los intentos repetidos de adjuntar un pin a '**Dout**' podrían provocar un bloqueo eventual una vez que se retiró el dispositivo.
- 7) Intentar adjuntar un '**LED4**' dispositivo más allá de pin 16 causaría un bloqueo.
- 8) Un error disparado por llamadas repetidas rápidas a '**analogWrite(255)**' para '**MOTOR**' El control en el usuario programa causó resultado '**MOTOR**' Velocidades para ser incorrecto (demasiado lento).
- 9) Desde V2.3, SRSLV dispositivos no pudo asignarse un '**Dout**' pin debido a una detección eléctrica conflicto defectuosa (y por lo tanto no permitir una asignación pin).
- 10) Los esclavos SPI ahora reinician su lógica de transmisor y receptor cuando su '**SS**' pin va '**HIGH**'.
- 11) Vocación '**Wire.h**' módulos funcionales '**endTransmission()**' o '**requestFrom()**' cuando las interrupciones están actualmente deshabilitadas ahora genera un error ejecución ('**Wire.h**' necesita interrupciones habilitadas para poder trabajar).
- 12) '**Ctrl-Home**' y '**Ctrl-End**' ahora funcionan como se espera en Editar/Examinar.
- 13) los '**OneWire**' comando de bus 0x33 ('**ROM_READ**') No estaba funcionando, y colgó el autobús.

V2.3- Dic. 2018

- 1) Un error introducido en V2.2 hizo imposible editar el valor de un elemento matriz en el interior **Editar/Monitorear**.
- 2) Desde la versión 2.0, el texto en el '**RAM free**' El control de la barra de herramientas solo estaba visible si se utilizaba un tema Ventanas-OS oscuro.

- 3) En **Archivo | Cargar**, y en E / S dispositivo archivo **Cargar**, los filtros archivo (como ***.ino** y ***.txt**) no funcionaban: se mostraban archivos de todos los tipos.
- 4) El estado "modificado" de un archivo se perdió después de hacer **Aceptar** o **Compilar** en un subsiguiente **Archivo | Editar/Examinar si no se hicieron más ediciones** (**Guardar** quedó incapacitado, y no hubo un aviso automático para **Guardar** en programa **Salida**).
- 5) Los operadores **'/'** y **'%'** solo dio resultados correctos para **'unsigned'** lado izquierdo variables
- 6) El condicional ternario. **'(bval) ? s1:s2'** dio un resultado incorrecto cuando **'s1'** o **'s2'** Era una expresión local en lugar de un variable.
- 7) Módulo funcional **'noTone()'** Se ha corregido para convertirse **'noTone(uint8_t pin)'**.
- 8) Un error de larga data causó un choque después de proceder de una **Reiniciar** cuando ese Reiniciar se realizó en medio de un módulo funcional que fue llamado con uno de sus parámetros faltantes (y por lo tanto recibe un valor inicializador por defecto).
- 9) Expresiones de miembros (por ejemplo, **'myobj.var'** o **'myobj.func()'**) no estaban heredando el **'unsigned'** propiedad de su lado derecho (**'var'** o **'func()'**) y, por lo tanto, no se pueden comparar ni combinar directamente con otros **'unsigned'** tipos - una asignación intermedia a un **'unsigned'** variable fue requerido por primera vez.
- 10) UnoArduSim insistía en que si la definición de un módulo funcional tenía algún parámetro con un inicializador predeterminado, el módulo funcional tiene un prototipo anterior declarado.
- 11) Llamadas a **'print(byte bvar, base)'** promovido erróneamente **'bvar'** a una **'unsigned long'**, y así se imprimen demasiados dígitos.
- 12) **'String(unsigned var)'** y **'concat(unsigned var)'** y operadores **'+=(unsigned)'** y **'+=(unsigned)'** creado incorrectamente **'signed'** cuerdas en su lugar.
- 13) Un **'R=1K'** dispositivo cargado desde un **IODevices.txt** archivo con posición **'U'** Fue dibujado erróneamente con su deslizador (siempre) en el **posición opuesta** Desde su verdadera posición eléctrica.
- 14) Intentando confiar en el predeterminado **'inverted=false'** argumento al declarar un **'SoftwareSerial()'** objeto causó un choque, y pasando **'inverted=true'** Sólo funcionó si el usuario programa hizo un posterior **'digitalWrite(txpin, LOW)'** para establecer primero el ralentí requerido **'LOW'** nivel en el **TX** pin.
- 15) **'I2CSLV'** dispositivos no respondió a los cambios en sus cuadros de edición pin (los valores predeterminados de A4 y A5 se mantuvieron vigentes).
- 16) **'I2CSLV'** y **'SPISLV'** dispositivos no detectaron ni corrigieron ediciones parciales cuando el mouse dejó sus bordes
- 17) Los valores Pin para dispositivos que siguieron a un **SPISLV** o **SRLV** se guardaron incorrectamente en el **IODevs.txt** archivo como hexadecimales.
- 18) Intentar conectar más de un **SPISLV** dispositivo MISO a pin 12 siempre generó un error Electrical Conflicto.
- 19) Cambiando un pin de **'OUTPUT'** de regreso **Modo 'INPUT'** no se pudo restablecer el nivel de bloqueo de datos pin a **'LOW'**.
- 20) Utilizando **'sendStop=false'** en llamadas a **'Wire.endTransmission()'** o **'Wire.requestFrom()'** causó un fracaso
- 21) UnoArduSim permitió indebidamente un **'SoftwareSerial'** la recepción se produzca simultáneamente con una **'SoftwareSerial'** transmisión.
- 22) Variables declarado con **'enum'** A este tipo no se le pudo asignar un nuevo valor después de su línea de declaración, y UnoArduSim no estaba reconociendo a los miembros de **'enum'** cuando se les hacía referencia con un (legal) **'enumname::'** prefijo.

V2.2– Jun. 2018

- 1) Llamar a un módulo funcional con menos argumentos de los que necesitaba su definición (cuando ese módulo funcional estaba "definido hacia adelante", es decir, cuando no tenía una línea de declaración prototipo anterior) causó una violación de memoria y se bloqueó.
- 2) Desde la V2.1, **Formas de onda** no se había actualizado durante **Ejecutar** (solo en **Detener**, o después de un **Paso**) - Adicionalmente, **Panel de Variables** Los valores no se estaban actualizando durante mucho tiempo. **Paso** operaciones
- 3) Algunos menores **Forma de onda** Los problemas de desplazamiento y zoom que han existido desde V2.0 ahora se han solucionado.
- 4) Incluso en las versiones anteriores, el Reiniciar en $t = 0$ con un PULSER o FUNCGEN, cuyo período realizaría su último ciclo antes de $t = 0$ ser solo un *parcial* ciclo, dio lugar a su **Forma de onda** después $t = 0$ siendo desplazado de su posición verdadera por esta (o la cantidad restante) del ciclo fraccional, ya sea hacia la derecha o hacia la izquierda (respectivamente).
- 5) Se corrigieron algunos problemas con el resaltado de color de sintaxis en **Editar/Examinar** .
- 5) Desde V2.0, hacer clic en expandir un objeto en un matriz de objetos no funcionó correctamente.
- 6) '`delay(long)`' ha sido corregido para ser '`delay(unsigned long)`' y '`delayMicroseconds(long)`' ha sido corregido para ser '`delayMicroseconds(unsigned int)`' .
- 7) A partir de la V2.0, módulos funcionales adjunta usando '`attachInterrupt()`' no estaban siendo verificados como válidos módulos funcionales para ese propósito (es decir, '`void`' retorno, y al no tener parámetros de llamada).
- 8) El impacto de '`noInterrupts()`' en módulos funcionales '`micros()`', '`mills()`', '`delay()`', '`pulseInLong()`'; su impacto sobre '`Stepper::step()`' y sobre '`read()`' y '`peek()`' tiempos de espera sobre toda recepción serial RX, y sobre '`Serial`' La transmisión, ahora se reproduce con precisión.
- 9) El tiempo pasado dentro de las rutinas de interrupción del usuario ahora se contabiliza en el valor devuelto por '`pulseIn()`', el retraso producido por '`delayMicroseconds()`', y en la posición de los bordes mostrados. **Formas di Onda Digitales** .
- 10) Llamadas a **objeto-miembro** módulos funcionales que formaban parte de expresiones complejas más grandes, o que estaban dentro de las llamadas módulo funcional con múltiples argumentos complejos, e, g, '`myobj.memberfunc1() + count/2`' o '`myfunc(myobj.func1(), count/3)`', tendría valores incorrectos calculados / pasados en ejecución debido a asignaciones de espacio de pila defectuosas.
- 11) Matrices del puntero variables funcionó correctamente, pero tenía valores mostrados defectuosos mostrados en el **Panel de Variables**.
- 12) Cuando se crearon matrices dinámicos de tipo simple con '`new`', solo el primer elemento había estado recibiendo una inicialización (predeterminada) para el valor 0; ahora todos los elementos lo hacen.
- 13) '`noTone()`', o el final de un tono finito, ya no restablece el pin (permanece '`OUTPUT`' y va '`LOW`').
- 14) La rotación continua 'SERVO' dispositivos ahora está perfectamente estacionaria a 1500 microsegundos de pulso de ancho.
- 15) La llamada a `SdFile::ls()` (listado en el directorio de la tarjeta SD) funcionó correctamente, pero mostró incorrectamente algunas transferencias SPI de bloque duplicadas en las formas de onda ventana.

V2.1.1– Mar. 2018

- 1) Se corrigieron las inconsistencias en las configuraciones regionales que no estaban en inglés con el idioma guardado en '`myArduPrefs.txt`', con botones de idioma de radio mostrados en el **Preferencias** cuadro de diálogo, y con la coincidencia de las líneas traducidas en '`myArduPrefs.txt`'.
- 2) Asignaciones con '`new`' ahora acepte una dimensión matriz entera que no sea una constante.
- 3) Haciendo clic en el **Panel de Variables** a expandir un matriz multidimensional mostraría un vacío superfluo '`[]`' par de paréntesis .
- 4) Referencias de elementos Matriz con caracteres superfluos finales (por ejemplo, '`y[2]12`') no se detectaron como errores en el momento Analizar (los caracteres adicionales simplemente se ignoraban).

V2.1– Mar. 2018

- 1) Un error en las nuevas versiones V2.0.x hizo que el montón Ventanas creciera con cada actualización en el **Panel de Variables** -- después Millones de actualizaciones (muchos minutos de ejecución), podría provocar un bloqueo.
- 2) Llamadas a 'static' miembro módulos funcionales usando dos puntos dobles ' : : ' La notación falló a Analizar cuando estaba dentro ' if () ', ' while () ', ' for () ' y ' switch () ' corchetes, y cuando las expresiones internas se utilizan como módulo funcional-argumentos de llamada o índices matriz.

V2.0.2 Feb. 2018

- 1) Un error introducido en V2.0 causó un **Archivo | Cargar** chocar si un '#include' referido a un archivo faltante o vacío
- 2) Dentro de un **IOdevs.txt** archivo, el '**I/O**' se esperaba el nombre 'One-Shot' en lugar del 'Oneshot' anterior; ambos son aceptados ahora.

V2.0.1– enero 2018

- 3) En las configuraciones regionales de idiomas distintos del inglés, '**en**' se mostró incorrectamente como seleccionado en **Preferencias**, volviendo al inglés incómodo (requiriendo la deselección luego la re-selección).
- 4) Ha sido posible para el usuario dejar un valor de cuadro de edición de Dispositivo pin en un estado incompleto (como 'A_'), y dejar los bits 'DATA' de un 'SRS:V' incompleto.
- 5) El número máximo de deslizadores Análogo se había limitado a 4 (corregido ahora para ser 6).
- 6) UnoArduSim ya no insiste en '=' apareciendo en una inicialización agregada matriz.
- 7) UnoArduSim había insistido en que se proporcionara el argumento "inverted_logic" a '**SoftwareSerial()**'.
- 8) Las operaciones de cambio de bits ahora permiten cambios más largos que el tamaño del variable desplazado.

V2.0– dic. 2017

- 1) Todos los módulos funcionales que fueron declarados como '**unsigned**' Sin embargo, habían estado devolviendo valores como si fueran '**signed**'. Esto no tuvo efecto si el '**return**' valor fue asignado a un '**unsigned**' variable, pero hubiera provocado una impropia interpretación negativa si tenía MSB == 1, y luego se asignó a un '**signed**' variable, o probado en una desigualdad.
- 2) Los deslizadores Análogo solo alcanzaban un máximo. '**analogRead()**' valor de 1022, no el correcto 1023.
- 3) Un error introducido de forma inadvertida en V1.7.0 en la lógica utilizada para acelerar el manejo del sistema SPI. SCK pin provocó transferencias SPI para '**SPI_MODE1**' y '**SPI_MODE3**' fallar después del primer byte transferido (un extra falso Transición SCK siguió cada byte). También se retrasaron las actualizaciones de un cuadro de edición de 'SPISLV' 'DATA' para los bytes transferidos,
- 4) The Colored DEL dispositivo no estaba listando 'B' (para Azul) como una opción de color (aunque fue aceptada).
- 5) Las configuraciones para 'SPISLV' y 'I2CSLV' dispositivos no se guardaron en el usuario '**I/O Dispositivos** archivo.
- 6) Copiando '**Servo**' instancias fallidas debido a una falla '**Servo::Servo(Servo &toCopy)**' Implementación copia-constructor.
- 7) Fuera de rango '**Servo.writeMicroseconds()**' los valores se detectaron correctamente como un error, pero los valores límite indicados que acompañaban al texto del mensaje de error eran incorrectos.
- 8) Un velocidad baudios legal de 115200 no fue aceptado cuando se carga desde un '**I/O Dispositivos** texto archivo.
- 9) No siempre se detectó el pin conflictos eléctrico causado por un Control Deslizante Analógico dispositivo adjunto.

- 10) En raras ocasiones, pasar un puntero de cadena defectuoso (con la cadena 0-terminador faltante) a un **'String'** módulo funcional podría causar que UnoArduSim se bloquee.
- 11) El **Panel de Código** podría resaltar la línea de error Analizar actual en el **incorrecto** Módulo programa (cuando **'#include'** se utilizó).
- 12) Cargar un 'I/O' Dispositivos archivo que tenía un dispositivo que (incorrectamente) empujar contra 'Uno' pin 13 provocó un bloqueo programa en la ventana emergente del mensaje de error.
- 13) UnoArduSim había permitido erróneamente el usuario debe realizar el pegado de caracteres no hexadecimales en el búfer expandido TX ventanas para SPISLV e I2CSLV.
- 14) Inicializaciones de línea de declaración fallado cuando el valor del lado derecho era el **'return'** valor de un miembro objeto-módulo funcional (como en **'int angle = myservo1.read();'**).
- 15) **'static'** miembro variables teniendo explícito **'ClassName::'** los prefijos no se reconocían si aparecían al comienzo de una línea (por ejemplo, en una asignación a una base - 'class' variable),
- 16) Vocación **'delete'** en un puntero creado por **'new'** solo se reconoció si se usaba la notación módulo funcional paréntesis, como en **'delete(pptr)'**.
- 17) Implementación UnoArduSim de **'noTone()'** insiste incorrectamente en que se proporcione un argumento pin.
- 18) Cambios que aumentaron los bytes 'RAM' globales en un programa que usaron **'String'** variables (a través de **Editar/Examinar** o **Archivo | Cargar**), podría conducir a la corrupción en ese espacio global de 'Uno' debido a la eliminación del montón de la **'String'** objetos pertenece al antiguo programa mientras usa (incorrectamente) el montón que pertenece al nuevo programa. En algunas circunstancias, esto podría llevar a un bloqueo programa. Aunque un segundo Cargar o Analizar resolvió el problema, este error por fin se ha solucionado.
- 19) Los valores de retorno para **'Wire.endTransmission()'** y **'Wire.requestFrom()'** ambos habían estado atascados en 0 - estos ahora han sido arreglados.

V1.7.2- Feb. 2017

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

V1.7.1- Feb. 2017

- 1) Módulo funcional **'delayMicroseconds()'** estaba produciendo un retraso en **mili**-segundos (1000 veces demasiado grandes).
- 2) tipo-lanzar explícito de un **'unsigned'** variable a un tipo entero más largo produjo un error (**'signed'**) resultado.
- 3) Literales hexagonales mayor que 0x7FFF son ahora **'long'** Por definición, y así lo generará ahora. **'long'** Expresiones aritméticas resultantes en las que se involucran.
- 4) Un error introducido inadvertidamente por V1.7.0 impidió el estilo K++ tipo-lanzar de C++ de literales numéricos alternativos (por ejemplo, **'(long)1000*3000'** no fue aceptado).
- 5) **'Serial'** ya no ocupa sus muchos bytes en la RAM 'Uno' si el usuario programa nunca la necesita.
- 6) El variables global declarado por el usuario ya no ocupa espacio en la RAM 'Uno' si nunca se usan realmente.
- 7) variables solo declarado como **'const'**, **'enum'** miembros y los punteros a los literales de cadena, ya no ocupan espacio en la RAM 'Uno' (para estar de acuerdo con la compilación de Arduino),
- 8) Bytes de RAM necesarios para **'#include'** Las bibliotecas integradas ahora coinciden estrechamente con los resultados de la compilación condicional de Arduino.
- 9) Utilizando **'new'** en una línea de declaración real de puntero había fallado (sólo una posterior **'new'** asignación al puntero trabajado).
- 10) Se corrigió un error donde una presentación "pendiente" de un directorio de disco SD podría causar un bloqueo 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 un Arduino módulo funcional que bloques mientras espera (como '`pulseIn()`', '`shiftIn()`', '`SPI.transfer()`', '`flush()`', y '`write()`') había causado una falla en el flujo ejecución en el retorno de interrupción

b) Múltiples copias del variables local de cualquier módulo funcional interrumpido tenían estado apareciendo en el **Panel de Variables** (una copia por interrupción-retorno) y esto se solucionó en V1.6.3, pero los otros problemas de interrupción permanecieron).

c) Módulo funcional '`delayMicroseconds()`' no estaba creando ningún retraso si se llamaba desde dentro de una rutina de interrupción de usuario.

d) Llamadas a bloquear módulos funcionales como '`pulseIn()`' desde **dentro** una interrupcion La rutina no había estado funcionando.

1) Un error introducido en V1.6.3 causó la pérdida de la actualización de valor en el **Panel de Variables** mientras se ejecutaba cuando los valores realmente estaban cambiando (esto sucedió solo después de dos o más **Detener** o menú **VarActualizar** acciones del usuario). Además, cuando se hizo un Ejecutar Hacia después de **Permitir la reducción** había sido activado, el **Panel de Variables** ocasionalmente no se redibujó (por lo tanto, los valores antiguos y local-variables pueden haber aparecido allí hasta la próxima Paso).

2) los **Panel de Código** destacando el comportamiento de la **Paso Sobre** comando podría aparecer engañoso en '`if()-else`' cadenas - eso ahora ha sido arreglado (aunque la funcionalidad real del stepping era correcta).

3) Módulo funcional '`pulseIn()`' había establecido incorrectamente el tiempo de espera en milisegundos en lugar de microsegundos; también estaba reiniciando incorrectamente el tiempo de espera cuando se observaron por primera vez las transiciones a niveles inactivos y activos.

4) Usando literales HEX entre `0x8000` y `0xFFFF` en asignaciones o aritmética con '`long`' entero variables dio resultados incorrectos debido a sin comprobar la extensión de la señal.

5) Pasando, o regresando, a una '`float`' de cualquiera '`unsigned`' el tipo entero que tiene un valor con MSB = 1 dio resultados incorrectos debido a una falla '`signed`' interpretación.

6) Todos '`bit_()`' módulos funcionales ahora también acepta operaciones en '`long`' -el tamaño variables y UnoArduSim comprueban las posiciones de bit no válidas (que quedarían fuera del tamaño variable).

7) Una entrada no válida al cuadro de edición 'Pulse' (ancho) en un Dispositivo 'PULSER' causó daños en el valor 'Period' (hasta que se corrigió con la siguiente entrada de edición del usuario 'Period').

8) Eliminando un 'PULSER' o 'FUNCGEN' dispositivo utilizando el El menú Configurar no eliminaba su señal periódica del pin de que estaba conduciendo (ya no se requiere un Reiniciar).

9) La posibilidad de inicializar un 1-D. '`char`' matriz con una cadena entre comillas faltaba, (por ejemplo, '`char strg[] = "hello";`').

10) La visualización hexadecimal en el monitor expandido 'SERIAL' o 'SFTSER' ventanas mostró el carácter incorrecto más significativo para los valores de bytes superiores a 127.

11) El Forma de onda ventanas no reflejaba los cambios programáticos del usuario realizados por '`analogWrite()`' cuando el un nuevo valor era 0% o 100% de ciclo de trabajo.

12) La implementación de '`Serial.end()`' Ahora ha sido arreglado.

13) UNA '`myArduPrefs.txt`' archivo con más de 3 palabras en una línea (o espacios en el '**I/O** Dispositivos Nombre archivo) podría causar un bloqueo debido a un puntero interno defectuoso.

14) La línea final de un '**I/O** Dispositivos' archivo no fue aceptado si no lo hizo **terminar con un salto de línea**.

15) Agregar más de cuatro deslizadores Análogo provocó un error silencioso que sobrescribió DEL '**I/O**' dispositivo punteros

16) A partir de V1.6.0, análogo forma de onda muestras para el **primera mitad** de cada **triángulo** forma de onda fueron todos **cero** (debido a un error en el cálculo de la tabla forma de onda).

- 17) Haciendo una repetición **Ejecutar Hacia** cuando está en una línea punto de parada ya no requiere múltiples clics por avance.
- 18) Pasar expresiones de dirección a un parámetro módulo funcional matriz no fue aceptado por Analizador.
- 19) módulos funcionales recursivo que devolvió expresiones que contenían puntero o matriz desreferencias dio resultados incorrectos debido a las marcas "listas" de "restablecimiento" en esas expresiones componentes.
- 20) Vocación '**class**' miembro-módulos funcionales a través **Cualquier puntero objeto variable o expresión de puntero** no estaba funcionando.
- 21) El usuario módulos funcionales que devolvió objetos por valor solo devolvió con éxito su valor en su primera llamada módulo funcional **Si** devolvieron un objeto construido sin nombre (como '**String**("dog")' - En las llamadas subsiguientes, se omitió la devolución debido a un indicador "listo" atascado.
- 22) No había habido salvaguarda para impedir el mando. **Ventanas | 'Serial' Monitor** From añadiendo un nuevo '**SERIAL**' dispositivo cuando en realidad no había espacio para ello.
- 23) Si agregar un pin dispositivo fijo (como 'SPISLV') provocó un mensaje emergente pin conflicto, **Panel del Banco de laboratorio** redibujar podría mostrar un duplicado "fantasma" dispositivo superponiendo el 'I/O' dispositivo situado más a la derecha (hasta el siguiente redibujado).
- 24) Se corrigieron algunos problemas con los sonidos no confiables de 'PIEZO' para señales pin no periódicas.
- 25) '**PROGMEM**' variables ahora también debe ser declarado explícitamente como '**const**' De acuerdo con Arduino.
- 26) "No hay espacio de almacenamiento dinámico" se marcó incorrectamente como un error ejecución cuando un '**SD.open()**' No se pudo encontrar el nombre archivo, o un '**openNextFile()**' Llegó el último archivo en el directorio.
- 27) Un Analizador error había aceptado incorrectamente un cierre llave fuera de lugar '}'.
- 28) Un error con **Panel de Variables** se han corregido las eliminaciones en el miembro-objeto-constructor retorno (el error se aplicó solo para objetos que a su vez contienen otros objetos como miembros).

V1.6.3- septiembre de 2016

- 1) El variables local de cualquier módulo funcional interrumpido no se eliminó del **Panel de Variables** en la entrada módulo funcional de interrupción, lo que lleva a múltiples copias aparece allí en el retorno de interrupción- módulo funcional (y un posible error ejecución o un bloqueo).
- 2) El Forma de onda ventanas no reflejaba cambios programáticos en '**analogWrite()**' a un nuevo ciclo de trabajo de 0% o 100%.
- 3) La visualización hexadecimal en expandido 'SERIAL' o 'SFTSER' Monitor ventana mostró el carácter de MSB incorrecto para valores de bytes mayores que 127.

V1.6.2- septiembre de 2016

- 1) Las llamadas Módulo funcional realizadas con el número o tipo de argumentos incorrectos no generaron un mensaje de error Analizar apropiado (solo el Apareció el mensaje genérico "no es un identificador válido").
- 2) los **Barra de herramientas** El botón de reinicio ahora funciona de manera idéntica al botón de reinicio 'Uno' placa de circuito.
- 3) El texto de error Analizar ya no se corta después de 16 caracteres sin mostrar puntos suspensivos.

V1.6.1- agosto de 2016

- 1) En V1.6 una versión 'Uno' placa de circuito en el '**myArduPrefs.txt**' archivo que difería del valor predeterminado de la versión 2 causó una excepción en el inicio (debido a un evento pin 13 no inicializado).
- 2) Cambiar el valor de un variable haciendo doble clic en el **Panel de Variables** podría causar ventanas emergentes de error defectuosas "sin asignación de memoria" (para programas con cualquier definición definida por el usuario '**class**').

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

4) Se corrigió el problema con la inclusión automática de la ".cpp" archivo correspondiente para una biblioteca ".h" aislada '**#include**'.

V1.6- junio 2016

1) En v1.5 Sangría automática en la llave 'Enter' en **Editar/Examinar** (Al entrar en una nueva línea) se había perdido.

2) La detección de pin entra en conflicto con adjunta de conducción fuerte 'I/O' dispositivos ahora se ha agregado en '**Serial**' pin 1, en SPI pins SS, MOSI y SCK, en I2C pins SCL y SDA (todo cuando el correspondiente '**begin()**' se llama), y en cualquier declarada '**SoftwareSerial**' TX pin.

V1.5.1- junio 2016

1) En v1.5 los nuevos colores Syntax Resaltar adaptables a los temas no se restablecieron correctamente cada vez **Editar/Examinar** se abrió, y así (con un tema de fondo blanco) solo fueron correctos cada dos veces.

2) Interrumpir '**RISING**' y '**FALLING**' las sensibilidades habían sido opuesto a la polaridad real del borde de disparo.

V1.5 - Mayo 2016

1) Un error introducido en V1.4.1 impidió pasar literales de cuerda desnuda a miembro módulos funcionales que esperaba una '**String**' objeto, como en '**mystring1.startsWith("Hey")**' .

2) Un error en el original **Dakota del Sur** Implementación de UnoArduSim solo permitida **Dakota del Sur** acceso usando llamadas a '**read()**' y '**write()**' (acceso vía '**Stream**' módulos funcionales fue prevenido).

3) 'R=1K' Los interruptores deslizantes no se volvieron a dibujar correctamente cuando se movió el control deslizante.

4) **Cancelar** en el cuadro de diálogo Confirm-Guardar archivo debería haberse evitado la salida de la aplicación.

5) Falta una cita de cierre o cierre '>' -paréntesis en un usuario archivo '**#include**' causaría una caída.

6) Se corrigió un error en el resaltado de sintaxis de '**String**' y usuario '**class**' o '**struct**', y resaltado ampliado para incluir constructor módulo funcional llamadas.

7) Solucionados problemas menores en **Editar/Examinar** con cambios de texto / resaltado y la **Deshacer** botón.

V1.4.3 - Abr. 2016

1) Utilizando **Configurar | I/O' Dispositivos** para agregar un nuevo dispositivos, y luego quitar uno de esos nuevos dispositivos podría causar un bloqueo al reiniciar, o que otro dispositivo deje de funcionar.

2) Modificando un '**String**' variable haciendo doble clic en el **Panel de Variables** fallido (el nuevo '**String**' fue leído incorrectamente).

3) Los cambios de Pin en 'FUNCGEN' y 'PULSER' dispositivos no se reconocieron hasta que se hizo un reinicio por primera vez.

V1.4.2 - Mar. 2016

1) V1.4.1 tenía **introducido** un Analizar error desafortunado que impidió asignaciones que implican cualquier '**class**' objetos (incluyendo '**String**').

2) Una corrección error incompleta hecha en V1.4.1 causó '**unsigned**' valores de tipo '**char**' para imprimir como caracteres ASCII en lugar de como sus valores enteros.

3) Los argumentos de llamada módulo funcional de expresión de miembro complejos no siempre se reconocían

como coincidencias válidas de parámetros módulo funcional.

4) **Todos** los literales enteros y las expresiones tenían un tamaño demasiado generoso (para 'long') y, por tanto, ejecución no reflejaba la **real** desbordamientos (a negativos) que pueden ocurrir en Arduino al agregar / multiplicar operaciones que involucran 'int' valores de tamaño

5) Expresiones que involucran una mezcla de 'signed' y 'unsigned' los tipos enteros no siempre se manejaban correctamente 'signed' valor sería visto incorrectamente como 'unsigned').

6) En los casos pin-conflicto, "valor =" los mensajes de error podrían mostrar valores pin obsoletos, incluso después de un Reiniciar de un conflicto anterior que el usuario ya había borrado.

V1.4.1 - Ene. 2016

1) Llamadas a 'print(char)' ahora imprima correctamente como caracteres ASCII (en lugar de valores numéricos).

2) La respuesta de interrupción ahora está habilitada por defecto cuando 'attachInterrupt()' se llama, por lo que ya no hay ninguna necesidad en su 'setup()' para llamar a la habilitación módulo funcional 'interrupts()' .

3) Múltiple '#include' instancias de usuario-archivos desde dentro de un archivo ahora se manejan correctamente.

V1.4 - Dic. 2015

1) UNA **de larga data** error marcó incorrectamente una **dividir entre cero** Condición al dividir por un valor fraccionario menor que la unidad.

2) Fijo 'SoftwareSerial' (que fue inadvertidamente roto por un añadido 'class' -Verificación de validación de miembros en V1.3 lanzamientos).

3) Las llamadas de fin de línea módulo funcional con un punto y coma faltante no fueron capturadas, y causaron que el Analizador saltara la siguiente línea.

4) Un mal formateado 'I/O' **Dispositivos** El texto archivo dio un mensaje de error incorrecto.

5) Analizar Se ha solucionado el error al resaltar la línea incorrecta (adyacente) en las expresiones y en las declaraciones de varias líneas.

6) Prueba lógica de punteros utilizando el 'not' (!) el operador estaba invertido.

V1.3 - Oct. 2015

1) El manejo interno incorrecto del scratchpad variables causó ocasional " **Profundidad máxima de anidamiento del scratchpad excedida** "Analizar errores.

2) Paréntesis *dentro de comillas simples*, llaves, punto y coma, parentheses Dentro de las cadenas citadas, y los caracteres escapados se manejaron incorrectamente.

3) Un Matriz con una dimensión vacía y sin lista de inicialización provocó un bloqueo de RESTABLECIMIENTO, y matrices con un solo elemento no fue deshabilitado (y causó su interpretación errónea como un puntero de inicialización no válida).

4) Los errores Analizar a veces resaltar a veces la línea incorrecta (adyacente).

5) Pasando un puntero a un no-'const' a un módulo funcional aceptando un puntero a un 'const' había sido desestimado (en lugar de al revés).

6) Las expresiones de inicialización se heredaron incorrectamente. 'PROGMEM' calificando desde el variable siendo inicializado

7) 'PROGMEM' variables declarado tenía su tamaño de byte contado incorrectamente **dos veces** contra su asignación de memoria 'Flash' durante el proceso Analizar.

8) Escribiendo en el cuadro de edición 'Send' de un 'I2CSLV' A veces causaría un choque debido a 'scanf' error.

- 9) Cargando un nuevo programa con un nuevo 'I/O' **Dispositivos** archivo en su directorio podría causar pin entra en conflicto con irrelevante *antiguo* Direcciones pin.
- 10) El manejo de caracteres en serie escapado se aplicó incorrectamente a las secuencias de caracteres recibidas, en lugar de transmitidas, en el (más grande) **'Serial' Monitor** tampones ventana.
- 11) `'while()'` y `'for()'` Bucles con cuerpos completamente vacíos, tales como `'while(true);'` o `'for(int k=1;k<=100;k++);'` pasó el Analizador (con un mensaje de advertencia) pero falló en el momento ejecución.

V1.2 - Jun. 2015

- 1) El usuario módulos funcionales más simple que realizó llamadas a cualquiera `'digitalRead()'` o para `'analogRead()'` o `'bit()'` podría haber dañado su (muy primero) variable declarado local (si lo hubiera) debido a que no había suficiente espacio asignado en el bloc de notas módulo funcional (si solo se hubieran asignado dos bytes del bloc de notas al comienzo de la pila módulo funcional). Cualquier expresión numérica dentro de un módulo funcional es suficiente para causar una asignación de bloc de notas de 4 bytes, y por lo tanto evita este problema. Este desafortunado error ha existido desde la versión original V1.0.
- 2) Módulos funcionales que son `'void'` con un principio explícito `'return'`, y no `'void'` módulos funcionales con más de uno `'return'` declaración, vería ejecución caída a través de la *cierre llave* (si se alcanzó).
- 3) Alguna `'return'` declaraciones dentro `'if()'` los contextos que faltaban en llaves llevaron a un objetivo de devolución de llamada defectuoso.
- 4) **'PULSER'** y los anchos de pulso 'FUNCGEN' o los periodos de valor 0 podrían causar una caída (ahora 0 no está permitido).
- 5) Donde no había llaves, `'else'` continuaciones después de un `'if()'` No funcionó si siguieron una `'break'`, `'continue'` o `'return'`.
- 6) Cuando **múltiple 'enum' usuario- se hicieron declaraciones**, constantes definidas en todo menos la primera `'enum'` generado defectuoso " `'enum'` desajuste Errores Analizar (este error se introdujo en V1.1).
- 7) Un identificador nulo para el último parámetro de un módulo funcional prototipo provocó un error Analizar.
- 8) **Ejecutar Hacia** Los puntos de interrupción establecidos en líneas complejas no siempre se manejaron correctamente (y por lo tanto se podían pasar por alto).
- 9) `'HardwareSerial'` y `'SoftwareSerial'` usó un búfer pendiente de TX de implementación privada que no se eliminó en Reiniciar (por lo que podrían aparecer caracteres sobrantes de la última vez).
- 10) El Analizador no pudo comprobar si se realizó un cambio de bit ilegal `'float'`, y aritmética de punteros intentada con operadores ilegales.

V1.1 - Mar. 2015

- 1) Índices Matriz que se encontraban `'byte'` o `'char'` el tamaño variables causó compensaciones matriz incorrectas (si un variable adyacente contenía un byte alto que no es 0).
- 2) La prueba lógica de los punteros probó el valor apuntado a para un valor distinto de cero en lugar del propio puntero.
- 3) Alguna `'return'` declaraciones incrustadas dentro `'for()'` o `'while()'` Los bucles fueron mal manejados.
- 4) Las listas de inicialización agregada para matrices de objetos, o objetos que contienen otras objetos / matrices, o listas de inicialización completamente vacías, no se manejaron correctamente.
- 5) Acceso de `'enum'` valores de miembros usando una `'enumname::'` prefijo no fue compatible
- 6) Inicialización de línea de declaración de un `'char[]'` matriz con un literal de cadena entre comillas no estaba funcionando.
- 7) Un matriz que se pasa a un módulo funcional sin una inicialización previa se marcó incorrectamente con un error "usado pero no inicializado".

- 8) Las expresiones de puntero que implican nombres matriz fueron mal manejadas.
- 9) Módulo funcional parámetros declarados como `'const'` no fueron aceptados
- 10) El Forma de Onda Analógica ventana no mostró señales de PWM (`'servo.write()'` y `'analogWrite()'`).
- 11) El miembro módulos funcionales al que se accedió a través de un objeto-puntero dio acceso de miembro defectuoso.
- 12) Las formas de onda no se estaban actualizando cuando una **Ejecutar Hacia** punto de parada fue alcanzado.
- 13) El modelo de asignación de registros podría fallar cuando se usó un parámetro módulo funcional cuando se usó directamente como argumento para otra llamada módulo funcional

V1.0.2 - Ago. 2014

Ordenamiento fijo de A0-A5 pins en el perímetro del 'Uno' placa de circuito.

V1.0.1 - Jun. 2014

Se corrigió un error que truncaba las pastas de edición que eran más de tres veces el número de bytes en el programa original.

V1.0 - primer lanzamiento mayo 2014

Cambios / Mejoras

V2.7.0 Mar. 2020

- 1) Además del código de línea actual (verde si listo para correr, rojo si el error), UnoArduSim ahora mantiene, para cada módulo, el último código de línea hizo clic el usuario o la pila de navegación (resaltado con un fondo verde oliva oscuro), la toma de facilitar la configuración y encontrar líneas temporales punto de parada (uno por módulo ahora se permite, pero sólo el que está en el módulo que se muestra actualmente está en vigor en una 'Run-To').
- 2) Se ha añadido nueva 'I/O' dispositivos (y apoyando código de la biblioteca tercera parte), incluyendo 'SPI' y 'I2C' **Los expansores de puerto** 'SPI' y 'I2C' **Multiplexor DEL** Controladores y pantallas (DEL matrices, 4-alfanuméricos, y 4-dígito o 8 dígito-displays de 7 segmentos).
- 3) **'Wire'** operaciones ya no se desecharon de usuario dentro de las rutinas de interrupción (esta soportes interrupciones externas de un 'I2C' de expansión del puerto).
- 4) Digital formas de onda ahora muestran un nivel intermedio (entre **'HIGH'** y **'LOW'**) Cuando el pin no está siendo empujado.
- 5) Para evitar confusiones al pasar a través de más de un solo **'SPI.transfer()'** instrucciones, UnoArduSim ahora se asegura de que adjunta 'I/O' dispositivos ahora reciben su último flanco de reloj 'SCK' (lógica con retardo) antes de que los rendimientos módulo funcional.
- 6) Cuando el auto-tab-formato **Preferencia** es activada, al escribir un cierre llave **'}'** en **Editar/Examinar** ahora provoca un salto a la posición guión pestaña de su juego de apertura llave **'{'** compañero.
- 7) UN **Reformatear** botón se ha añadido a **Editar/Examinar** (A causa inmediata de auto-ficha-guión cambio de formato) - este botón se activa únicamente cuando está activada la preferencia Auto-ficha-guión.
- 8) Un mensaje de error más claro ahora se produce cuando una palabra clave de prefijo (como **'const'**, **'unsigned'**, o **'PROGMEM'**) sigue un identificador en una declaración (que tiene que preceder al identificador).
- 9) Inicializado variables mundial, incluso cuando no se usa más adelante, están ahora siempre se asigna una dirección de memoria, y así aparecerá visible.

V2.6.0 de enero 2020

- 1) visualización de caracteres LCD añadido dispositivos tener 'SPI', 'I2C', y la interfaz de 4-bi-paralelo. Apoyando código fuente de la biblioteca se ha añadido a la carpeta 'include_3rdParty' nueva instalación (y se puede acceder mediante el uso de una normal de **'#include'** Directiva) - los usuarios pueden elegir como alternativa a su lugar escribir su propio módulos funcionales a empujar la dispositivo LCD.
- 2) **Panel de Código** resaltado ha sido mejorada, con colores distintos para resaltar un código de línea, listo para un código de línea de error, y para cualquier otro código de línea.
- 3) los **Encontrar** menú y barra de herramientas 'func' acciones ascienden (anterior-y-viene abajo) ya no salto a la anterior / siguiente módulo funcional línea de salida, y en su lugar ahora (o descenso) de la pila de llamadas, destacando el código de línea relevante en la persona que llama (o llamada) módulo funcional, respectivamente, donde el **Panel de Variables** contenido se ajusta para mostrar variables para la módulo funcional que contiene el código de línea resaltada actualmente.
- 4) Para evitar confusiones, una **Guardar** en el interior hecho **Editar/Examinar** causa una inmediata re-**Compilar**, Y si el Guardar fue exitosa, utilizando un subsiguiente Cancelar o Salida ahora sólo revertir el texto a-que guardó por última vez el texto.
- 5) Añadido a-entrada pulsada Motor Paso a Paso ('PSTEPR') con 'STEP' (pulso), 'EN*' (activar), y entradas 'DIR' (dirección), y un ajuste de micro-pasos-per-etapa de (1,2,4,8, o 16) .
- 6) Tanto 'STEPR' y 'PSTEPR' dispositivos ahora tienen una 'sync' DEL (verde para sincronizada, o rojo cuando está apagada por uno o más pasos.)

- 7) 'PULSER' dispositivos ahora tienen una opción entre microsegundos y milisegundos para 'Period' y 'Pulse'.
- 8) Incorporado-módulo funcional auto-terminaciones ya no conservan el tipo de parámetro delante del nombre del parámetro.
- 9) Al volver a una anterior **Panel de Código**, Su línea destacada anteriormente ahora se re-resaltado.
- 10) Como una ayuda para el establecimiento de un punto de ruptura temporal, utilizando Cancelar o de Salida **Editar/Examinar** deja la resaltar en el **Panel de Código** en la línea visitó por última vez el cursor en **Editar/Examinar**.
- 11) A (o tercera parte) definida por el usuario **'class'** ahora está autorizado a utilizar **'Print'** o **'Stream'** como su clase base. En apoyo de esto, se ha añadido una nueva carpeta 'include_Sys' (en la carpeta de instalación UnoArduSim) que proporciona el código fuente de cada base **'class'**. En este caso, las llamadas a tales base- **'class'** módulos funcionales será tratado de forma idéntica a usuario código (que) puede ser entrado en), en lugar de como un módulo funcional incorporado que no puede ser entró en (tales como **'Serial.print()'**).
- 12) Miembro-módulo funcional auto-terminaciones ahora incluyen el nombre del parámetro **en lugar de** su tipo.
- 13) El UnoArduSim Analizar permite ahora un nombre objeto en una declaración variable ser precedido por su opcional (y juego) **'struct'** 'class' o palabra clave, seguido por el **'struct'** o **'class'** nombre.

V2.5.0 Oct 2019

- 1) Añadido soporte para **'TFT.h'** biblioteca (con la excepción de **'drawBitmap()'**), Y se añadió una 'TFT' asociado 'I/O' Dispositivo (128 por 160 píxeles). Tenga en cuenta que con el fin de evitar desfases en tiempo real excesivas durante gran **'fillXXX()'** transferencia, porción sa de las transferencias 'SPI' **en el centro del relleno** estará ausente del bus 'SPI'.
- 2) Durante archivo grandes transferencias a través de 'SD', una porción de las transferencias 'SPI' en el medio de la secuencia de bytes será parecida estar ausentes del bus 'SPI'.
- 3) Disminución **'Stream'** overhead -usage bytes de modo que **'RAM free'** valor es más compatible con Arduino compilación.
- 4) UnoArduSim ahora avisa al usuario cuando una **'class'** tiene varios miembros declararon en una línea de declaración.
- 5) El uso de 'File | Save As' ahora establece el directorio actual que se ha guardado la que en el directorio.
- 6) Los dos desaparecidos **'remove()'** miembro módulos funcionales se han añadido a la **'String'** clase.
- 7) UnoArduSim ahora Deshabilita el constructor base-llamadas en un constructor-módulo funcional prototipo menos que la definición completa del cuerpo módulo funcional sigue inmediatamente (con el fin de llegar a un acuerdo con el Arduino compilador).
- 8) Edtiempo de transición ge de digital formas de onda se redujo a apoyar la visualización de las señales 'SPI' rápidos en mayor zoom.
- 9) Naciones Unidas oArduSim ahora permite que algunos constructores para ser declaradas **'private'** o **'protected'** (Para uso clase interna).

V2.4 de mayo de 2019

- 1) Todos los archivos de dispositivos de 'I/O' ahora se guardan en un idioma traducido, y junto con el archivo de **Preferencias**, ahora se guardan en codificación de texto UTF-8 para evitar errores coincidentes en la lectura posterior.

- 2) Agregó un nuevo '**PROGIO**' Dispositivo, que es un esclavo programable 'Uno' placa de circuito que comparte hasta 4 pins en común con el **Panel del Banco de laboratorio** maestro 'Uno', - un esclavo 'Uno' puede tener no 'I/O' dispositivos propios.
- 3) Ahora puede eliminar cualquier 'I/O' dispositivo haciendo clic en él mientras presiona la tecla 'Ctrl'.
- 4) En **Editar/Examinar** , se ha agregado la terminación automática de texto para global, incorporados y miembro variables y módulos funcionales (use ALT-flecha derecha para solicitar una finalización, o **Entrar** si la lista incorporada está actualmente resaltando una selección coincidente).
- 5) En **Preferencias** , una nueva opción permite la inserción automática de un punto y coma que termina la línea en n **Entrar** pulsación de tecla (si la línea actual es una instrucción ejecutable que parece autocontenida y completa).
- 6) Prensado '**Ctrl-S**' a partir de una **Forma de onda** ventana le permite guardar todos los archivo (**X, Y**) puntos a lo largo de la sección visualizada de cada forma de onda (donde X es microsegundos desde la forma de onda de la izquierda) punto, y Y es voltios).
- 7) Un 'SFTSER' 'dispositivo ahora tiene un oculto (opcional) '**inverted**' valor (**se aplica tanto a TX como a RX**) que se puede añadir después de su valor de velocidad en baudios al final de su línea en una **IODevs.txt** archivo.
- 8) Agregó el '**SPISettings**' clase, módulos funcionales '**SPI.transfer16()**' , '**SPI.transfer(byte* buf, int count)**' , '**SPI.beginTransaction()**' y '**SPI.endTransaction()**' , tanto como '**SPI.usingInterrupt()**' y '**SPI.notUsingInterrupt()**' .
- 9) Biblioteca SPI agregada módulos funcionales '**SPI.detachInterrupt()**' junto con una extensión de la biblioteca SPI '**SPI.attachInterrupt(void myISRfunc)**' (en lugar de la biblioteca real módulo funcional '**SPI.attachInterrupt(void)**' (para evitar la necesidad de reconocer genéricos '**ISR(int vector)**' declaraciones módulo funcional de interrupción vectorial de bajo nivel).
- 10) El sistema SPI ahora se puede usar en modo esclavo, ya sea haciendo el '**SS**' pin (pin 10) un '**INPUT**' pin y conduciéndolo '**LOW**' después '**SPI.begin()**' , o especificando '**SPI_SLV**' como el opcional '**mode**' parámetro en '**SPI.begin(int mode=SPI_MSTR)**' (otra extensión de UnoArduSim a '**SPI.h**'). Los bytes recibidos pueden ser recolectados usando '**rxbyte = SPI.transfer(tx_byte)**' ya sea dentro de un módulo funcional sin interrupción SPI o dentro de un servicio módulo funcional de interrupción de usuario previamente conectado por '**SPI.attachInterrupt(myISRfunc)**' . En modo esclavo, '**transfer()**' espera hasta que un byte de datos esté listo en SPDR (por lo tanto, normalmente se bloqueará la espera de una recepción de byte completa, pero en una rutina de interrupción será **regreso** inmediatamente porque el byte SPI recibido ya está allí). En cualquier caso, '**tx_byte**' se coloca en el SPDR, por lo que será recibido por el maestro SPI adjunto en su próximo '**transfer()**' .
- 11) Se ha agregado compatibilidad con el modo Esclavo a la implementación UnoArduSim de 'Wire.h'. Módulo funcional '**begin(uint8_t slave_address)**' ya está disponible, como están '**onReceive(void*)**' y '**onRequest(void*)**' .
- 12) '**Wire.end()**' y '**Wire.setClock(freq)**' ahora se puede llamar; este último para establecer la frecuencia SCL con una '**freq**' valor de 100,000 (la frecuencia SCL de modo estándar predeterminada) o 400,000 (modo rápido).
- 13) 'I2CSLV' dispositivos ahora todos responden a la **0x00** dirección de bus de llamada general, y así **0x00** ya no se puede elegir como una dirección de bus I2C única para uno de esos esclavos.
- 14) Los retrasos ejecución modelados de enteros básicos y operaciones de asignación y matriz y las operaciones de puntero se han reducido, y ahora se agregan 4 microsegundos para cada operación de punto flotante.

V2.3 Dic. 2018

- 1) El seguimiento se ha habilitado en el **Barra de herramientas** 'I/O' **S' deslizador** para continuo y suave Escalado de los valores 'I/O' dispositivo que el usuario ha agregado el sufijo 'S'.
- 2) Un nuevo **'LED4'** 'I/O' dispositivo (fila de 4 LEDs encendidos **4 números pin consecutivos**) ha sido añadido.
- 3) Un nuevo **'7SEG'** 'I/O' dispositivo (7 segmentos DEL dígito con código hexadecimal activado **4 números pin consecutivos**, y con activo-bajo **CS** * Seleccione entrada), se ha añadido.
- 4) Un nuevo **'JUMP'** 'I/O' dispositivo que actúa como un puente de cable entre dos 'Uno' pins se ha agregado. Esto permite una **'OUTPUT'** pin para ser conectado a un **'INPUT'** pin (consulte el dispositivo más arriba para posibles usos de esta nueva función).
- 5) Un nuevo **'OWISLV'** 'I/O' dispositivo se ha agregado, y el tercero **'<OneWire.h>'** la biblioteca ahora se puede utilizar con **'#include'** de modo que el usuario programas pueda probar la interconexión con un pequeño subconjunto del bus dispositivos '1-Wire'.
- 6) los **Ejecutar** menú **Reiniciar** comando ahora está conectado a la **Reiniciar** botón.
- 7) Para mayor claridad, cuando **Demora artificial de 'loop()'** se selecciona bajo la **Opciones** menú, un explícito **'delay(1)'** La llamada se agrega a la parte inferior del bucle dentro **'main()'** - esto es ahora un retraso real que puede ser interrumpido por interrupciones de usuario adjuntas en 'Uno' pins 2 y 3.
- 8) pin entra en conflicto con eléctrico de drenaje abierto, o seleccionado por CS, 'I/O' dispositivos (por ejemplo, I2CLV, o SPISLV) ahora se declaran **solo cuando un conflicto verdadero ocurre a la hora ejecución**, en lugar de causar un error inmediato cuando el dispositivo se conecta por primera vez.
- 9) Módulo funcional **'pulseInLong()'** ahora tiene una precisión de 4-8 microsegundos para estar de acuerdo con Arduino (la precisión anterior fue de 250 microsegundos).
- 10) Los errores marcados durante la inicialización de un variable global ahora resaltar que variable en el **Panel de Código**.

V2.2 Jun. 2018

- 1) En **Guardar** de cualquiera de los **Preferencias** cuadro de diálogo, o desde **Configurar | 'I/O' Dispositivos**, la **'myArduPrefs.txt'** archivo ahora se guarda en el directorio del programa cargado actualmente - cada siguiente **Archivo | Cargar** luego carga automáticamente el archivo, junto con sus IODevs archivo especificados, de ese mismo directorio programa.
- 2) Módulo funcional **'pulseInLong()'** **había sido** falta, pero ahora se ha agregado (se basa en **'micros()'** por sus medidas).
- 3) Cuando un usuario programa hace un **'#include'** de un **'*.h'** archivo, UnoArduSim ahora también intenta automáticamente cargar el correspondiente **'*.c'** archivo **Sí** un correspondiente **'*.cpp'** archivo no fue encontrado.
- 4) Inserción automática de un cierre-llave **'}'** (después de cada llave abierto **'{'**) ha sido añadido a **Preferencias**.
- 5) Un nuevo **Opciones** opción de menú ahora permite **'interrupts()'** Para ser llamado desde dentro de una rutina de interrupción del usuario, esto es solo para fines educativos, ya que en la práctica debe evitarse el agrupamiento de interrupciones.
- 6) Tipo-lanzar de punteros a un **'int'** el valor ahora es compatible (pero aparecerá un mensaje emergente de advertencia).
- 7) UnoArduSim ahora admite líneas programa etiquetadas (por ejemplo, **'LabelName: count++;'** para comodidad del usuario (pero **'goto'** es todavía **desestimado**)
- 8) Las advertencias Ejecución ahora ocurren cuando cuando una llamada a **'tone()'** podría interferir con el PWM activo en pins 3 u 11, cuando **'analogWrite()'** interferiría con un Servo ya activo en el mismo pin, cuando se pierde la llegada de un personaje en serie porque las interrupciones están actualmente deshabilitadas, y cuando las interrupciones se producen tan rápido que UnoArduSim se perderá algunas de ellas.

V2.1 Mar. 2018

- 1) Desplegado **Panel de Variables** los valores ahora se actualizan solo cada 30 milisegundos (y la opción Mínima aún puede reducir aún más esa tasa de actualización), pero **VarActualizar** Se ha eliminado la opción de menú para no permitir la reducción de actualizaciones.
- 2) Operaciones que se dirigen solo a una parte de los bytes de un valor variable (como los realizados a través de punteros) ahora hacer que el cambio a ese valor variable se refleje en el **Panel de Variables** monitor.

V2.0.1 Ene. 2018

- 1) Undocumented Arduino módulos funcionales '**exp()**' y '**log()**' ahora se han añadido.
- 2) Ahora se puede hacer una rotación continua de 'SERVO' dispositivos (por lo que el ancho del pulso controla la velocidad en lugar del ángulo).
- 3) En **Editar/Examinar**, un cierre llave '}' Ahora se agrega automáticamente cuando escribe una apertura llave '{' si has seleccionado eso **Preferencia**.
- 4) Si hace clic en el **Editar/Examinar** Barra de título ventana '**X**' para salir, ahora tiene la oportunidad de abortar si ha modificado, pero no ha guardado, el programa archivo mostrado.

V2.0 Sept. 2017

- 1) La implementación se ha trasladado a QtCreator, por lo que la GUI tiene algunas diferencias visuales menores, pero no hay diferencias funcionales más que algunas mejoras:
 - a) El mensaje de la línea de estado en la parte inferior del ventana principal, y dentro de la **Editar/Examinar** cuadro de diálogo, se ha mejorado y se ha añadido un código de color resaltar.
 - b) El espacio vertical asignado entre el **Panel de Código** y **Panel de Variables** ahora es ajustable a través de una barra divisora que se puede arrastrar (pero no visible) en su borde compartido.
 - c) Los valores del cuadro de edición 'I/O' dispositivo ahora solo se validan una vez que el usuario ha movido el puntero del mouse fuera del dispositivo; esto evita los cambios automáticos incómodos para imponer valores legales mientras el usuario está escribiendo.
- 2) UnoArduSim ahora soporta múltiples idiomas a través de **Configurar | Preferencias**. El inglés siempre se puede seleccionar, además del idioma para la configuración regional del usuario (siempre que haya una traducción personalizada *.qm archivo para ese idioma en la carpeta UnoArduSim 'translations').
- 3) El sonido ahora se ha modificado para utilizar la API de audio Qt. Esto ha requerido silenciar en ciertas circunstancias para poder Evita molestas rupturas de sonido y clics. durante las demoras operativas de ventanas más largas del SO causadas por los clics normales del usuario (consulte la sección Sonidos para obtener más detalles) en este.
- 4) Como conveniencia para el usuario, los espacios en blanco ahora se usan para representar un valor de 0 en los cuadros de edición de dispositivo-count en **Configurar | 'I/O' Dispositivos** (por lo que ahora puede utilizar la barra espaciadora para eliminar dispositivos).
- 5) El sin escala El calificador (U) ahora es opcional en 'PULSER', 'FUNCGEN' y '1SHOT' dispositivos (es el supuesto por defecto).
- 6) UnoArduSim ahora permite (además de los valores numéricos literales) '**const**' variables de valor entero, y 'enum'

V1.7.2– Feb. 2017

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

V1.7.1– Feb. 2017

1) Sufijos 'L' y/o 'U' ahora se aceptan al final de las constantes literales numéricas (para definir las como 'long' y/o 'unsigned'), y '0b' o '0B' prefijado) ahora también se aceptan las constantes binario. Cualquier constante numérica de todos los decimales **comenzando con un '0'** ahora se considera que es una **octal** valor. (De acuerdo con Arduino).

2) Cuando se ejecuta en un bucle estrecho del que no hay escape (por ejemplo, 'while(x); x++;' donde x siempre es cierto), haciendo clic **Detener** una segunda vez ahora asegura que programa ejecución realmente se detiene (y en esa línea programa defectuosa).

V1.7.0– dic. 2016

1) Un nuevo **Barra de herramientas** se ha agregado una función que muestra libre RAM bytes durante programa ejecución (lo que representa el total de bytes utilizados por variables global, las asignaciones de pila y la pila local variables).

2) La interrupción del usuario módulos funcionales puede ahora también llamar al bloqueo Arduino módulos funcionales como 'pulseIn()' (pero esto solo debe usarse con precaución, ya que la interrupción módulo funcional no volverá hasta que se complete el bloqueo módulo funcional).

3) Las interrupciones del usuario ya no están deshabilitadas durante las operaciones de lectura de secuencias bloqueadas, por lo que el comportamiento ahora coincide con la operación de lectura de secuencias de Arduino real.

4) Ahora puede entrar y salir del bloqueo de Arduino módulos funcionales que puede ser interrumpido (como 'delay()' y 'pulseIn()'), y los mensajes de barra de estado se han aumentado para mostrar cuando ha alcanzado una interrupción punto de parada dentro de un módulo funcional (o cuando hace clic en Detener cuando ejecución se encuentra actualmente dentro de dicho módulo funcional).

5) Un nuevo **Ejecutar Hasta** comando (y **Barra de herramientas** Se ha agregado el artículo): haga clic en cualquier botón. **Panel de Variables** variable (puede ser simple, un agregado matriz o objeto, o un elemento matriz o objeto-miembro) a resaltar, entonces hazlo **Ejecutar Hasta** - ejecución se congelará en la próxima **acceso de escritura** dentro de ese agregado variable, o en esa única ubicación.

6) Cuando ejecución se congela después de una **Paso, Ejecutar Hacia, Ejecutar Hasta o Ejecutar-entonces-Detener** acción, la **Panel de Variables** Ahora destaca el variable que corresponde a la **dirección (s) ubicación (es) que se modificó** (si existe) por el **último instrucción durante esa ejecución** - Si esa ubicación está actualmente oculta dentro de un expandido matriz o objeto sin expandido, al hacer clic en expandir se hará que se resalte el elemento o miembro modificado por última vez.

7) El usuario ahora puede mantener una vigilancia especial sobre el valor de un determinado **Variable Panel** variable / miembro / elemento mientras se ejecuta: haga doble clic en esa línea en el **Panel de Variables** para abrir el **Editar/Monitorear Valor Variable** ventana, entonces haz una de las **Ejecutar** o **Paso** comandos el valor mostrado se actualizará durante ejecución de acuerdo con las mismas reglas que rigen las actualizaciones en el **Panel de Variables**. Después de detener ejecución, se le permite ingresar un nuevo valor y **Aceptar** antes de reanudar ejecución (y puede **Revertir** a la pre-**Aceptar** Valor si cambias de opinión antes de eso).

8) Las teclas del acelerador F4-F10 se han configurado para coincidir con el menú Ejecutar **Barra de herramientas** Comandos (de izquierda a derecha).

9) Además de hacer doble clic en ellos, haga clic derecho en 'SERIAL', 'SFTSER', 'SPISLV', 'I2CSLV' dispositivos ahora también mostrará un tamaño mayor de bytes / caracteres TX / RX ventana (y en 'SD_DRV', una archivos-

monitorización ventana).

10) El cuadro de edición de TX en 'SERIAL' o 'SFTSER' ya no está deshabilitado durante una transmisión de caracteres activos (por lo que ahora puede agregar o reemplazar lo que está allí), pero un clic de retorno de carro (o 'Send' haga clic en el 'Serial; Monitor secundario' asociado ventana) 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, está activa). Además, el usuario ahora está advertido en una secuencia en serie '**begin()**' si ya habían comenzado antes el dispositivo adjunto (ahora en progreso) las transmisiones, ya que entonces no habría sincronización de trama, lo que lleva a errores de recepción.

11) El valor por defecto añadido '**loop()**' el retraso se ha incrementado de 250 microsegundos a un milisegundo para no quedarse tan atrasado en tiempo real cuando el usuario se niega a incluir algunos '**delay()**' (explícito o natural) en algún lugar dentro '**loop()**' o dentro de un módulo funcional que llama.

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

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

14) Valores de puntero en el **Panel de Variables** ahora se asemejan más a los valores de puntero de Arduino reales.

15) El usuario '**myArduPrefs.txt**' archivo ahora está cargado en cada **Archivo | Cargar**, no solo en el lanzamiento de UnoArduSim.

16) Un error Analizar ahora está marcado cuando se intenta '**attachInterrupt()**' a un usuario módulo funcional que no es '**void**' regresando, o que tiene parámetros módulo funcional, o que no ha sido declarado en algún lugar antes '**attachInterrupt()**'.

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

18) Módulo funcional '**availableForWrite()**' Se ha agregado a la implementación de '**Serial**'.

19) Todo especial '**PROGMEM**', '**typedef**' me gusta '**prog_char**' y '**prog_int16**' ahora han sido eliminados (han sido desaprobados en Arduino).

20) Mensajes de error mejorados para errores Analizar causados por tipos de declaración mal escritos o inválidos.

21) Se ha aumentado el tamaño máximo permitido de programa.

V1.6.3– Sept. 2016

1) Añadido un mensaje de error mejorado analizar cuando '**attachInterrupt()**' se refiere a una interrupción-módulo funcional que no era **prototipo anterior**.

2) Se agregó un mensaje de error Analizar mejorado para las listas de inicialización de matriz multidimensionales.

V1.6.2– Sept. 2016

1) Añadido un **Encontrar-Texto** editar el control de la **Barra de herramientas** para agilizar la búsqueda de texto (en la **Panel de Código** y **Panel de Variables**).

2) los **Barra de herramientas** El botón Reiniciar ahora funciona de manera idéntica al botón 'Uno' placa de circuito Reiniciar.

V1.6.1– agosto de 2016

Se agregó una verificación para evitar la carga y el análisis duplicados de los anteriores. '**#include**' archivos, .

V1.6 - Junio 2016

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

2) Se agregó una nueva característica que hace que los valores de la caja de edición de 'I/O' dispositivo fácilmente **escamoso** durante ejecución arrastrando un control deslizante global de Escala 'I/O_____S' en la pantalla principal **Barra de herramientas** (simplemente escriba una sola letra 's' o 'S' después de un valor para indicar la escala).

V1.5.1 - Junio 2016

1) Se ha agregado soporte para la biblioteca EEPROM módulos funcionales '**update()**' , '**put()**' y '**get()**' , y para el acceso a bytes a través de la notación matriz, por ejemplo, '**EEPROM[k]**' .

2) **Permitir auto (-) Encoger** ha sido añadido al menú **VarActualizar** para permitir el control explícito sobre si expandido matrices / objetos se contraerá automáticamente cuando ejecución se quede atrás en tiempo real.

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

V1.5 - Mayo 2016

1) **Editar/Examinar** ahora tiene el atajo ctrl-E, y tiene un nuevo botón para **Compilar** (ctrl-R), más un cuadro de error incorporado Analizar, para permitir la prueba de ediciones sin necesidad de cerrar El ventana.

2) **Editar/Examinar** ahora también soporta **Rehacer**, y tiene un nuevo **Guardar** botón (ctrl-S) (equivalente a **Aceptar** más un main-ventana posterior **Guardar**), y ahora da una opción de '**Tab**' tamaño (una nueva preferencia que se puede guardar usando **Configurar | Preferencias**).

3) Todos los cuadros de edición de escritura ahora siguen los colores del tema Ventanas OS elegidos, y para el contraste, todos los cuadros de edición 'RECV' de solo lectura utilizan texto blanco sobre fondo negro. los **Editar/Examinar** Los colores de fondo y de sintaxis resaltar 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** (Así se puede guardar en el '**myArduPrefs.txt**' archivo).

5) Arduino predefinió los valores literales binario (como '**B01011011**') ahora están permitidos.

6) Ahora se pueden usar secuencias de caracteres citados de Unicode hexadecimal, octal y 4-dígito como literales numéricos.

7) Después de hacer un clic inicial del mouse en un pulsador 'PUSH' dispositivo, el usuario puede presionar una tecla (cualquier tecla) para presionar los contactos del pulsador.

8) **Editar/Examinar** ahora libera su estado temporal inicial de solo lectura (y elimina el resaltado de la línea inicial seleccionada) después de un breve destello visual.

9) UnoArduSim ahora busca múltiples '**Stepper**' y '**Servo**' pin conflictos, es decir, el usuario defectuoso programa intenta conectarse a pins que ya se adjuntó anteriormente. '**Stepper**' o '**Servo**' variables.

10) Un error Analizar causado por un lado izquierdo o derecho faltante a un operador (falta una expresión LHS o RHS o variable) ahora genera un mensaje de error claro.

11) El no utilizado '**String**' clase '**flags**' el miembro variable se ha eliminado para estar de acuerdo con Arduino V1.6.6. UNA '**String**' objeto ahora ocupa 6 bytes (más su asignación de montón de caracteres).

V1.4.2 - Mar. 2016

1) módulos funcionales definido hacia adelante (es decir, aquellos sin declaración prototipo antes de su primera llamada) ahora solo genera advertencias (no errores analizar) cuando la definición de módulo funcional posterior

no coincide con el tipo deducido de su primer uso.

2) Matrices que tiene una dimensión igual a 1 ya no se rechaza (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 Ventanas OS en uso.

4) 'SERIAL', 'SFTSER', 'SPISLV' y 'I2CSLV' dispositivo expandido El monitor ventanas (que se abre haciendo doble clic) ahora adopta el color de fondo de su padre 'I/O' Dispositivo.

V1.4 - Dic. 2015

1) 'Stepper.h' Ahora se han agregado la funcionalidad de la biblioteca y el 'I/O' dispositivos asociado.

2) **Todas las configuraciones y valores de 'I/O' Dispositivo** (además de su pins seleccionado) ahora también se guardan como parte del texto archivo 'I/O' Dispositivos elegido para recargar más tarde.

3) **DEL** El color 'I/O' dispositivo ahora se puede configurar como rojo, amarillo o verde usando un cuadro de edición en el dispositivo.

4) Los inicializadores de declaración Variable ahora pueden abarcar varias líneas.

5) Ahora se permite que los índices Matriz sean elementos matriz.

6) **Configurar | Preferencias** Ahora incluye una casilla de verificación para permitir 'and', 'or', 'not' Palabras clave que se utilizarán en lugar del estándar C '&&', '||' y '!' operadores logicos.

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

V1.3 - Oct. 2015

1) los '**PUSH**' dispositivo ahora tiene una casilla de verificación "similar a un empuje" etiquetada 'latch' para hacer que se "enganchen" (en lugar de "momentáneas"), es decir, se engancharán en la posición cerrada (y cambiarán de color) cuando se presionen, hasta que se vuelvan a presionar para liberar los contactos.

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

3) **Negrita** tipo de letra ahora puede ser elegido para el **Panel de Código** y **Panel de Variables** (desde el menú **Opciones**), y **negrita resaltado de palabras clave y operadores** ahora se puede activar / desactivar en **Editar/Examinar**.

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

5) Para mayor claridad en los informes de errores, las declaraciones variable ya no pueden abarcar varias líneas (excepto matrices que tiene listas de inicialización).

6) Sintaxis de la velocidad de color en **Editar/Examinar** ha sido mejorado (esto se notará con programas más grande).

7) Una sobrecarga opcional de 200 microsegundos (en el menú **Opciones**) ha sido añadido a cada llamada de 'loop()' - esto es para tratar de evitar quedarse demasiado atrás en tiempo real en el caso de que El usuario programa no tiene agregado 'delay()' En cualquier lugar (ver discusión Sincronización debajo).

V1.2 Jun. 2015

1) La biblioteca SD ahora está completamente implementada y se ha agregado un (pequeño) Disco SD 'I/O'

dispositivo ('SD_DRV') de 8 Mbytes (y se ha probado la funcionalidad de todas las muestras SD programas de Arduino).

2) Al igual que Arduino, UnoArduSim convertirá automáticamente un argumento módulo funcional a su dirección cuando llame a un módulo funcional esperando que se pase un puntero.

3) Los mensajes de error Analizar ahora son más apropiados cuando faltan los puntos y coma y después de las declaraciones no reconocidas.

4) Duro **Panel de Variables** los resaltados de línea ahora se eliminan en módulo funcional call / return.

V1.1 - Mar. 2015

1) El ventana principal ahora se puede maximizar o redimensionar para que el **Panel de Código** y **Panel de Variables** más ancho (para pantallas más grandes).

2) Un nuevo menú Encontrar (con **Botones de la barra de herramientas**) se han añadido para permitir una navegación más rápida en el **Panel de Código** y **Panel de Variables** (PgUp y PgDown, o búsqueda de texto con flechas hacia arriba, hacia abajo).

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

4) UNA Nuevo elemento ha sido añadido al menú. **VarActualizar** para permitir al usuario seleccionar un enfoque de ahorro de cómputo en grandes **Panel de Variables** actualizar las cargas.

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

6) Ahora se puede llamar a otro usuario módulos funcionales desde dentro de una interrupción de usuario módulo funcional (de acuerdo con la actualización a Arduino 1.06).

7) UNA **fuentes más grande** ahora se puede elegir desde el menú **Opciones**.

V1.0.1 - Jun. 2014

Forma de onda ventanas ahora etiqueta análogo pins como A0-A5 en lugar de 14-19.

V1.0 - primer lanzamiento mayo 2014