

UnoArduSimV2.1 Guida Completa

Sommario

[Panoramica](#)

[Area di Codice, Preferenze e Modificare / Esaminare](#)

[Area del Codice](#)

[Preferenze](#)

[Modificare / Esaminare](#)

[Area delle Variabili e finestra Modificare / Monitorare Variabile](#)

[Area del Banco da Laboratorio](#)

[Il 'Uno'](#)

[Dispositivi 'I / O'](#)

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

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

[Unità disco SD \('SD_DRV'\)](#)

[Schiavo Registro di Cambio \('SRSLV'\)](#)

[Schiavo SPI configurabile \('SPISLV'\)](#)

[Schiavo I2C \('I2CSLV'\)](#)

[Motore Passo-Passo \('STEPR'\)](#)

[Motore CC \('MOTOR'\)](#)

[ServoMotore \('SERVO'\)](#)

[Generatore di Impulsi \('PULSER'\)](#)

[Un-Colpo \('1SHOT'\)](#)

[Generatore di Funzioni Analogiche \('FUNCGEN'\)](#)

[Altoparlante Piezoelettrico \('PIEZO'\)](#)

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

[Resistenza interruttore a Scorrimento \('R = 1K'\)](#)

[LED Colorato \('LED'\)](#)

[Cursore Analogico](#)

[Menu](#)

[File:](#)

[Caricare INO o PDE Programma \(ctrl-L \)](#)

[Modificare / Esaminare \(ctrl-E \)](#)

[Salvare](#)

[Salva Come](#)

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

[Precedente](#)

[Esci](#)

[Trova:](#)

[Trova Funzione/Var Successiva](#)

[Trova Funzione / Var Precedente](#)

[Imposta Testo di Ricerca \(ctrl-F\)](#)

[Ricerca Prossimo Testo](#)

[Ricerca Testo Precedente](#)

[Eseguire:](#)

[Passo In \(F4\)](#)

[Passo Scavalcare \(F5\)](#)

[Passo Fuori \(F6\)](#)

[Eseguire Verso \(F7\)](#)

[Eseguire Fino A\(F8\)](#)

[Eseguire \(F9\)](#)

[Arresto \(F10\)](#)

[Resettare](#)

[Animare](#)

[Rallentatore](#)

Opzioni:

[Scavalcare Struttori / Operatori](#)
[Modellazione di Registro-Assegnazione](#)
[Errore su non inizializzato](#)
[Aggiunto 'loop\(\)' Ritardo](#)

Configurare:

[Dispositivi 'I / O'](#)
[Preferenze](#)

VarAggiorna:

[Consenti il Collasso Automatico\(-\)](#)
[Minimo](#)
[Evidenzia Modifiche](#)

Finestre:

['Serial' Monitor](#)
[Ripristinare Tutto](#)
[Pin Forme d'Onda Digitali](#)
[Pin Forma d'Onda Analogica](#)

Aiuto:

[File di Aiuto Rapido](#)
[File di Aiuto Completo](#)
[Correzioni di Bugs](#)
[Modifica / Miglioramenti](#)
[Di](#)

[Modellismo](#)

['Uno' Board e Dispositivi 'I / O'](#)
[Sincronizzazione](#)
[Sincronizzazione di Dispositivo 'I / O'](#)
[Suoni](#)

[Limitazioni e elementi non supportati](#)

[File Inclusi](#)
[Allocazioni di Memoria Dinamica e RAM](#)
[Allocazioni di memoria 'Flash'](#)
[Variabili 'String'](#)
[Librerie Arduino](#)
[Puntatori](#)
[Oggetti 'class' e 'struct'](#)
[Scopo](#)
[Qualificazioni 'unsigned' , 'const' , 'volatile' , 'static'](#)
[Direttive del Compilatore](#)
[Elementi di Lingua Arduino](#)
[C / C ++ - Elementi del Linguaggio](#)
[Modelli di funzione](#)
[Emulazione in tempo reale](#)

[Note di rilascio](#)

[Correzioni di Bug](#)

[V2.1- Mar. 2018](#)
[V2.0.2- Feb. 2018](#)
[V2.0.1- Gen. 2018](#)
[V2.0- Dic. 2017](#)
[V1.7.2- Feb.2017](#)
[V1.7.1- Feb.2017](#)
[V1.7.0- dic.2016](#)
[V1.6.3- Settembre 2016](#)
[V1.6.2- Settembre 2016](#)
[V1.6.1- agosto 2016](#)
[V1.6 - Giugno 2016](#)
[V1.5.1- giugno 2016](#)
[V1.5 - Maggio 2016](#)
[V1.4.3 - aprile 2016](#)

[V1.4.2 - marzo 2016](#)
[V1.4.1 - Gennaio 2016](#)
[V1.4 - Dicembre 2015](#)
[V1.3 - ottobre 2015](#)
[V1.2 - Giugno 2015](#)
[V1.1 - Marzo 2015](#)
[V1.0.2 - Agosto 2014](#)
[V1.0.1 - Giugno 2014](#)
Modifiche / Miglioramenti
[V2.1- Mar. 2018](#)
[V2.0.1- Gen. 2018](#)
[V2.0 settembre 2017](#)
[V1.7.2 - Febbraio 2017](#)
[V1.7.1 - Febbraio 2017](#)
[V1.7.0- dic.2016](#)
[V1.6.3- Settembre 2016](#)
[V1.6.2- Settembre 2016](#)
[V1.6.1- Ago 2016](#)
[V1.6 - Giugno 2016](#)
[V1.5.1 - Giugno 2016](#)
[V1.5 - Maggio 2016](#)
[V1.4.2 - Mar 2016](#)
[V1.4 - Dic 2015](#)
[V1.3 - Ottobre 2015](#)
[Versione 1.2 giugno 2015](#)
[V1.1 - Marzo 2015](#)
[V1.0.1 - Giugno 2014](#)

Panoramica

UnoArduSim è uno strumento di simulazione freeware ***in tempo reale*** (vedi modellazione per le **restrizioni** temporali) che ho sviluppato per gli studenti e gli appassionati di Arduino. È progettato per consentire all'utente di sperimentare e di eseguire facilmente il debug di programmi Arduino ***senza la necessità di alcun hardware effettivo*** . È indirizzato alla scheda **Arduino 'Uno'** e consente di scegliere tra una serie di dispositivi virtuali 'I / O' e di configurare e connettere questi dispositivi al proprio virtual 'Uno' nel **Area di Banco Laboratorio**. - Non è necessario preoccuparsi di errori di cablaggio, connessioni interrotte / allentate o dispositivi difettosi che compromettono lo sviluppo e il test del programma.

UnoArduSim fornisce semplici messaggi di errore per eventuali errori di analisi o esecuzione che incontra e consente il debugging con **Resettare**, **Eseguire**, **Eseguire-Verso**, **Eseguire-Fino-A**, **Arresto** e operazioni **Passo** flessibili nel riquadro di codice, con una visualizzazione simultanea di tutti i locali variabili, matrici e oggetti globali e attualmente attivi nelle variabili riquadro. Viene fornito il controllo dei limiti della matrice in fase di esecuzione e verrà rilevato l'overflow della RAM ATmega (evidenziata la riga del programma colpevole!). Eventuali conflitti elettrici con i dispositivi 'I / O' allegati sono contrassegnati e segnalati mentre si verificano.

Quando un file di programma INO o PDE viene aperto, viene caricato nel **Area del Codice** del programma . Il programma viene quindi analizzato e "compilato" in un eseguibile tokenizzato che è quindi pronto per l' **esecuzione simulata** (diversamente da Arduino.exe, *non* viene creato un eseguibile binario autonomo)
Qualsiasi errore di analisi viene rilevato e contrassegnato evidenziando la riga che non è stata analizzata e riportando l'errore sulla **Narra di Stato** nella parte inferiore della finestra dell'applicazione UnoArduSim. È possibile aprire una finestra di **modifica / visualizzazione** per consentire all'utente di visualizzare e modificare una versione evidenziata dalla sintassi del programma utente. Gli errori durante l'esecuzione simulata (come le velocità di trasmissione errate) sono riportati sulla Narra di Stato e tramite una finestra di messaggio a comparsa.

UnoArduSim V2.0 è un'implementazione sostanzialmente completa del **linguaggio di programmazione Arduino V1.6.6** *come documentato al arduino.cc* . Pagina web di riferimento della lingua e con aggiunte

come indicato nella versione Scaricare della pagina Note sulla versione. Sebbene UnoArduSim non supporti l'implementazione completa del C ++ che fa il compilatore GNU sottostante di Arduino.exe, è probabile che solo i programmatori più avanzati trovino che alcuni elementi C / C ++ che desiderano utilizzare manchi (e ovviamente ci sono sempre semplici soluzioni di codifica per tali funzionalità mancanti). In generale, ho supportato solo quelle che ritengo siano le funzionalità C / C ++ più utili per gli appassionati e gli studenti di Arduino - ad esempio, 'enum' e '#define' sono supportati, ma i puntatori di funzione non lo sono. Anche se sono supportati oggetti definiti dall'utente ('class' e 'struct') e (la maggior parte) sovraccarichi dell'operatore, *l'ereditarietà multipla no* .

Poiché UnoArduSim è un simulatore di linguaggio di alto livello, ***sono supportate solo le istruzioni C / C ++ , le istruzioni di linguaggio assembly non lo sono*** . Analogamente, poiché non si tratta di una simulazione di macchina di basso livello, i ***registri ATmega328 non sono accessibili al programma*** né per la lettura né per la scrittura, anche se l'allocazione del registro, il passaggio e il ritorno vengono emulati (lo si sceglie nel menu **Opzioni**).

A partire dalla V2.0, UnoArduSim ha il supporto automatico integrato per un sottoinsieme limitato delle librerie fornite da Arduino, che sono: 'Stepper.h' , 'SD.h' , 'Servo.h' , 'SoftwareSerial.h' , 'SPI.h' , 'Wire.h' e 'EEPROM.h' (versione 2). Per qualsiasi '#include' di librerie create dall'utente, UnoArduSim **non** cercherà la solita struttura di directory di installazione di Arduino per localizzare la libreria; invece è **necessario** copiare l'intestazione corrispondente (".h") e la sorgente ("cpp") file nella stessa directory del file di programma che il vostro stanno lavorando su (beninteso con la limitazione che il contenuto di qualsiasi '#include' file deve essere completamente comprensibile per il Analizzatore UnoArduSim).

Ho sviluppato UnoArduSimV2.0 in QtCreator con supporto multilingue, ed è attualmente disponibile solo per Windows TM . Porting su Linux o MacOS, è un progetto per il futuro. !UnoArduSim è nato dai simulatori che avevo sviluppato nel corso degli anni per i corsi che ho insegnato alla Queen's University, ed è stato testato abbastanza a lungo, ma ci sono comunque alcuni bug che si nascondono lì dentro. Se vuoi segnalare un bug, descrivilo (brevemente) in una email a unoArduSim@gmail.com e **assicurati di allegare il tuo codice sorgente Arduino completo del programma che induce** il bug in modo da poter replicare il bug e correggerlo. Non risponderò a singole segnalazioni di bug, e non ho tempistiche garantite per le correzioni in una versione successiva (ricorda che ci sono quasi sempre soluzioni alternative!).

Saluti,

Stan Simmons, Ph. D, P. Ing.
Professore associato (in pensione)
Dipartimento di ingegneria elettrica e informatica
Queen's University
Kingston, Ontario, Canada

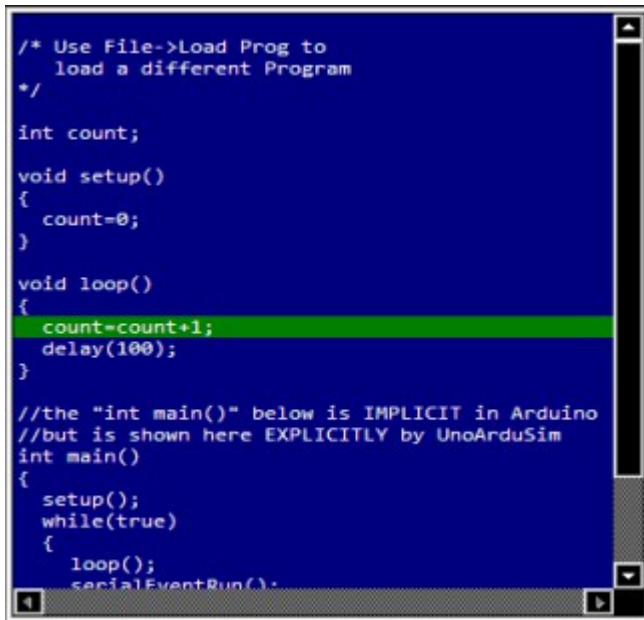


Area di Codice, Preferenze e Modificare / Esaminare

(A parte: le finestre di esempio mostrate di seguito sono tutte sotto un tema di colore Windows-OS scelto dall'utente che ha un colore di sfondo della finestra blu scuro)

Area del Codice

Il **Area di Codice** visualizza il tuo programma utente e mette in evidenza le tracce dell'esecuzione. Dopo che un programma caricato è stato analizzato correttamente, la prima riga in '**main()**' è evidenziata e il programma è pronto per l'esecuzione. Si noti che '**main()**' è implicitamente aggiunto da Arduino (e da UnoArduSim) e non lo **si** include come parte del file del programma utente. L'esecuzione è sotto il controllo del menu **Execute** e dei relativi pulsanti **Tool-Bar** e tasti funzione.







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



Dopo aver eseguito l'esecuzione con una (o più) istruzioni (è possibile utilizzare i pulsanti della **Barra degli Strumenti**. , , , o ), la linea di programma che verrà eseguita successivamente viene quindi evidenziata - la linea evidenziata è sempre la riga successiva **pronta per essere eseguita** .






Allo stesso modo, quando un programma in esecuzione raggiunge un punto di arresto (**run-to-run**) temporaneo , l'esecuzione viene interrotta e la linea di punto di arresto viene evidenziata (e quindi pronta per l'esecuzione).

Se l'esecuzione del programma è attualmente interrotta e si fa clic nella finestra del **Area del Codice** , la **riga su cui si è appena fatto clic viene evidenziata**.

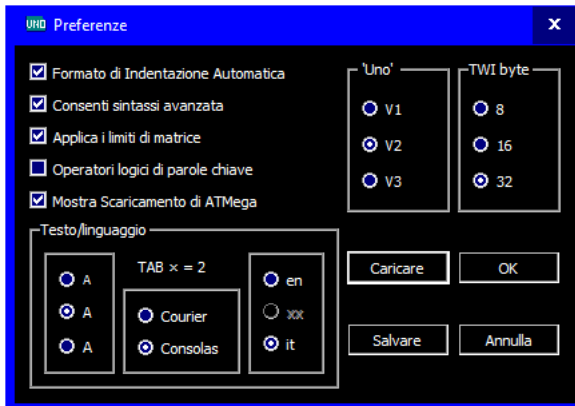
Tuttavia, ciò *non* modifica la linea di programma corrente per quanto riguarda l'esecuzione del programma. Ma puoi far *progredire* l' esecuzione *fino*

alla linea che hai appena evidenziato, quindi fare clic sul **Esegui-Verso**  Pulsante **Barra degli Strumenti**. Questa funzione consente di raggiungere in modo rapido e semplice linee specifiche in un programma in modo tale da poter successivamente effettuare il passaggio riga per riga su una porzione di programma interessata.

Se il tuo programma caricato ha '**#include**' file, puoi spostarti tra loro usando **File | Precedente** e **File | Successivo** (con Pulsanti della **Barra degli Strumenti**.  e ).

Il menu **Trova** consente di **passare** rapidamente da una **funzione** all'altra nel **Area del Codice** (dopo aver prima fatto clic su una riga al suo interno per attivarla) utilizzando i comandi **Successivo** e **Precedente** (con i pulsanti **Barra degli Strumenti**..  e  o scorciatoie da tastiera **PgDn** e **PgUp**). In alternativa, puoi **trovare il testo** specificato con i relativi comandi di testo (con i pulsanti della **Barra degli Strumenti**.  e  , o scorciatoie da tastiera **freccia su** e **freccia giù**), dopo aver prima utilizzato **Trova | Imposta Testo di Ricerca** o pulsante **Barra degli Strumenti**.  .

Preferenze

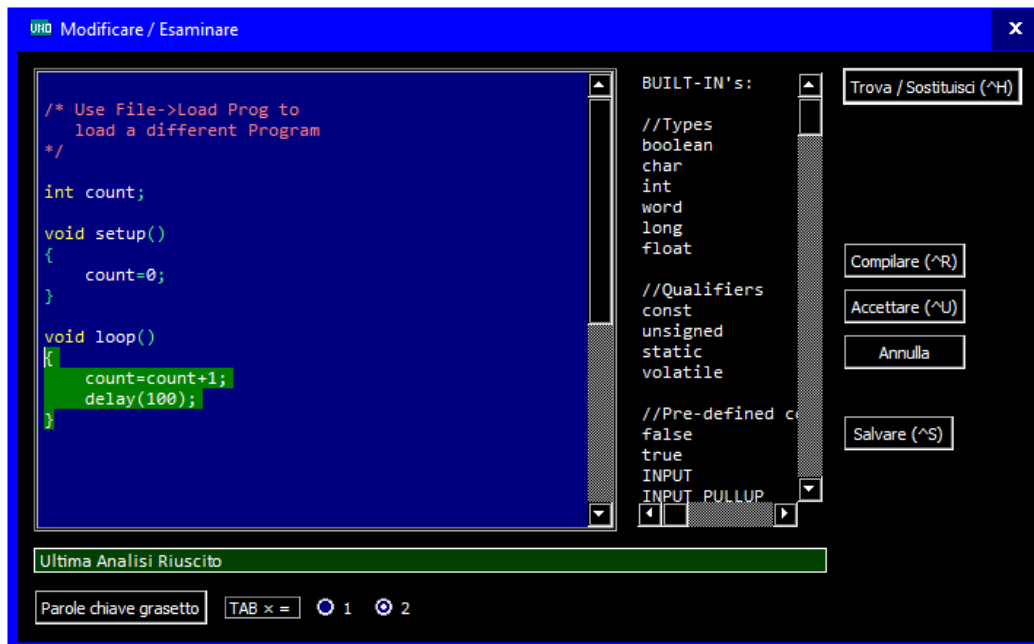


C onfigure | Le preferenze consentono agli utenti per impostare le preferenze di programma e di visualizzazione (che un utente normalmente desidera adottare alla prossima sessione). Questi possono quindi essere salvati e caricati da un file **myArduPrefs.txt** che risiede nella stessa directory del programma 'Uno' caricato (**myArduPrefs.txt** viene caricato automaticamente se esiste).

Questa finestra di dialogo consente di scegliere tra due tipi di caratteri mono-spaziati e tre tipi di dimensioni e altre preferenze varie. A partire dalla versione 2.0, ora è incluso il chipice della lingua. - questo include sempre l'inglese (**en**), più uno o due altri linguaggi locali dell'utente (laddove esistono) e un override basato sul codice della lingua ISO-639 a due lettere sulla prima riga del file **myArduPrefs.txt**

(se uno è fornito lì). Le scelte compaiono solo se esiste un file di traduzione ".qm" nella cartella delle traduzioni (all'interno della directory principale di UnoArduSim.exe).

Modificare / Esaminare



Facendo doppio clic su una qualsiasi riga nel **Area di Codice** (o utilizzando il menu **File**), viene aperta una finestra **Modificare // Esaminare** per consentire le modifiche al file di programma, con evidenziata la riga attualmente selezionata nell'Area di Codice.

Questa finestra ha funzionalità di modifica completa con evidenziazione dinamica della sintassi (i colori di evidenziazione diversi vengono utilizzati per parole chiave, commenti, ecc.). È disponibile l'evidenziazione in grassetto della sintassi opzionale e la formattazione automatica del livello di rientro (supponendo che tu abbia selezionato quello utilizzando **Configurare | Preferenze**). Puoi anche selezionare comodamente le chiamate di funzione incorporate (o incassate **#define** constants) da aggiungere al tuo programma dalla casella di elenco fornita, basta fare doppio clic sull'elemento della casella di elenco desiderato per aggiungerlo al tuo

programma nella posizione attuale del caret (i **tipi di** variabile chiamata-funzione sono solo per informazione e vengono eliminati per lasciare segnaposto fittizi quando vengono aggiunti al tuo programma).

La finestra ha le funzioni **Trova** (usa **ctrl-F**) e **Trova / Sostituisci** (usa **ctrl-H**) . La finestra **Modificare / Esaminare** ha i pulsanti **Disfare** (**ctrl-Z**) e **Rifare** (**ctrl-Y**) (**che appaiono automaticamente**).

Per annullare **tutte le modifiche** apportate dal momento in cui hai aperto il programma per la modifica, fai clic sul pulsante **Annulla** . Per accettare lo stato corrente, fare clic sul pulsante **Accetta** e il programma viene automaticamente analizzato di nuovo (e scaricato su 'Uno' se non vengono trovati errori) e il nuovo stato viene visualizzato nella Narra di Stato **della** finestra UnoArduSim principale .

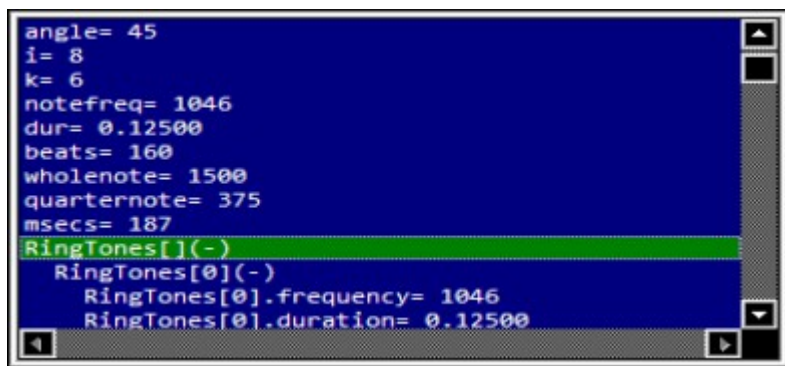
È stato aggiunto un pulsante **Compilare** (**ctrl-R**) (più un **messaggio-box Stato di Analisi** associato **nell'immagine qui sopra**) per consentire il test delle modifiche senza dover prima chiudere la finestra. Un pulsante **Salva** (**ctrl-S**) è stato aggiunto anche come scorciatoia (equivalente ad un **Accettazione** più un successivo **Salvataggio** separato dalla finestra principale).

In **Annulla** o **Accetta** senza modifiche apportate, la riga corrente del **Area di Codice** diventa l' **ultima posizione Visualizza / Modifica cursore** , ed è possibile utilizzare tale funzionalità per saltare il **Area di Codice** su una riga specifica (eventualmente per prepararsi a eseguire una **Esecuzione**). **Per** , puoi anche usare **ctrl-PgDn** e **ctrl-PgUp** per saltare alla prossima (o precedente) interruzione di riga vuota nel tuo programma - questo è utile per navigare rapidamente verso l'alto o verso il basso in posizioni significative (come le linee vuote tra le funzioni) . Puoi anche usare **ctrl-Home** e **ctrl-End** per saltare all'inizio del programma e terminare, rispettivamente.

La formattazione automatica del rientro a livello di tabulazione viene eseguita quando si apre la finestra, se l'opzione è stata impostata in **Configurare | Preferenze** . Puoi anche aggiungere o eliminare tabulazioni tu stesso a un gruppo di linee consecutive preselezionate utilizzando i tasti **freccia destra** o **sinistra della** tastiera, ma il **rientro automatico deve essere disattivato** per evitare di perdere i tuoi livelli di tabulazione.



E per aiutarti a tenere traccia dei tuoi contesti e delle tue parentesi graffe, facendo clic su ' { ' o ' } ' parentesi graffa evidenzia tutto il testo tra quella coppia e il suo partner corrispondente .



Area delle Variabili e finestra Modificare / Monitorare Variabile



Il **Area delle Variabili** trova appena sotto il **Area del Codice** Mostra i valori correnti per ogni utente globale e attivo (nell'ambito) della variabile / matrice / oggetto attivo nel programma caricato. Man mano che l'esecuzione del programma si sposta tra le funzioni, i **contenuti cambiano per riflettere solo le variabili locali accessibili alla funzione / ambito corrente, oltre a eventuali globali dichiarati dall'utente** . Qualsiasi variabile dichiarata come **'const'** o come

'PROGMEM' (assegnata alla memoria 'Flash') ha valori che non possono essere modificati e, per risparmiare spazio, questi non vengono quindi *visualizzati* . Le istanze **'Servo'** e **'SoftwareSerial'** non contengono valori utili, pertanto non vengono visualizzate.

I comandi del menu **Trova** (con i pulsanti della **Barra degli Strumenti**  e  , o scorciatoie da tastiera **PgDn** e **PgUp**) consentono di **passare** rapidamente da una **variabile** all'altra nel **Area delle Variabili** dopo aver prima fatto clic su una riga al suo interno per attivarla). In alternativa, è possibile **trovare il testo** specificato con i

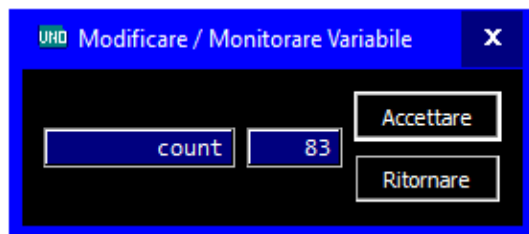
relativi comandi di ricerca testo (con i pulsanti della **Barra degli Strumenti**,  e , o scorciatoie da tastiera **freccia su** e **freccia giù**), dopo aver prima utilizzato **Trova | Imposta il Testo di Ricerca** o .

Le matrici e gli **oggetti** sono mostrati in formato **non espanso** o **espanso**, rispettivamente con un segno più (+) o meno (-). Il simbolo per una matrice **x** mostra come **x[]**. Per espanderlo per mostrare tutti gli elementi della matrice, basta fare un solo clic su **x[] (+)** nel **Area delle Variabili**. Per tornare a una vista non espansa, fai clic sulla **x[] (-)**. L'impostazione predefinita non espansa per un oggetto **p1** mostra come **p1 (+)**. Per espanderlo per mostrare tutti i membri dell'istanza 'class' o 'struct', fai clic su **p1 (+)** con un solo clic nel **Area delle Variabili**. Per tornare a una vista non espansa, fai clic su **p1 (-)**.

Se fai un **solo clic su qualsiasi linea per evidenziarlo** (può essere una variabile semplice, o la linea di aggregazione (+) o (-) di una matrice o di un oggetto, o un singolo elemento di matrice o un membro dell'oggetto), quindi eseguendo **Esegui-Fino-A** farà sì che l'esecuzione riprenda e si **blocchi** al successivo **accesso in scrittura** ovunque all'interno di **quell'aggregato** selezionato o in quella posizione di variabile singola selezionata.

Quando si usa **Passo** o **Esegui**, gli aggiornamenti dei valori delle variabili visualizzati vengono eseguiti in base alle impostazioni utente effettuate nel menu **VarAggiorna** - ciò consente una gamma completa di comportamenti, da aggiornamenti periodici minimi a aggiornamenti immediati completi. Aggiornamenti ridotti o minimi sono utili per ridurre il carico della CPU e potrebbero essere necessari per impedire che l'esecuzione si riduca in tempo reale rispetto a quelli che altrimenti sarebbero eccessivi **carichi di aggiornamento della finestra del Area delle Variabili**. Quando è attivo **Animate**, o se l'opzione di menu **Modifiche evidenziazione** è selezionata, le modifiche al valore di una variabile durante l'esecuzione comporteranno l'aggiornamento **immediato del** valore visualizzato, che verrà evidenziato. Ciò causerà lo scorrimento del Area delle Variabili (se necessario) alla riga che contiene tale variabile e l'esecuzione non sarà più in tempo reale !.

Quando l'esecuzione si blocca dopo **Passo**, **Esegui Verso**, **Esegui-Fino-A** o **Esegui -poi- Arresto**, il **Area delle Variabili** evidenzia la variabile corrispondente alle posizioni degli indirizzi che sono state modificate (se ce ne sono) dall'ultima istruzione durante quell'esecuzione (anche mediante inizializzazioni di dichiarazioni variabili). Se tale istruzione riempie **completamente** un **oggetto o una matrice**, la linea **genitore (+) o (-)** per quell'aggregato diventa evidenziata. Se, invece, l'istruzione modifica una posizione che è attualmente visibile, allora viene evidenziata. Ma se la (e) posizione (e) modificata (e) è attualmente nascosta all'interno di una matrice o oggetto non espanso, quella **linea genitoriale** aggregata ottiene un **carattere italico highlighting** come indicazione visiva che qualcosa al suo interno è stato scritto - facendo clic per espanderlo sarà poi fa in modo che l' **ultimo** elemento o membro modificato venga evidenziato.



La finestra **Modificare / Monitorare** ti dà **la possibilità di seguire qualsiasi valore di variabile durante l'esecuzione**, o di **cambiarne il valore nel mezzo dell'esecuzione (interrotta) del programma** (in modo da poter testare quale sarebbe l'effetto di continuare avanti con quel nuovo valore). **Interrompere** prima l'esecuzione, quindi **fare doppio clic** sulla variabile di cui si desidera tenere traccia o modificare il valore. Per monitorare semplicemente il valore durante l'esecuzione del programma, **lasciare la finestra di dialogo aperta** e quindi uno dei comandi **Esegui** o **Passo**: il suo valore verrà aggiornato in **Modificare /**

Monitorare in base alle stesse regole che governano gli aggiornamenti nel **Area delle Variabili**. **Per modificare il valore della variabile**, inserire il valore della casella di modifica e **accettare**. Continua l'esecuzione (usando uno qualsiasi dei comandi **Passo** o **Esegui**) per utilizzare quel nuovo valore da quel punto in avanti (o puoi ripristinare il valore precedente).

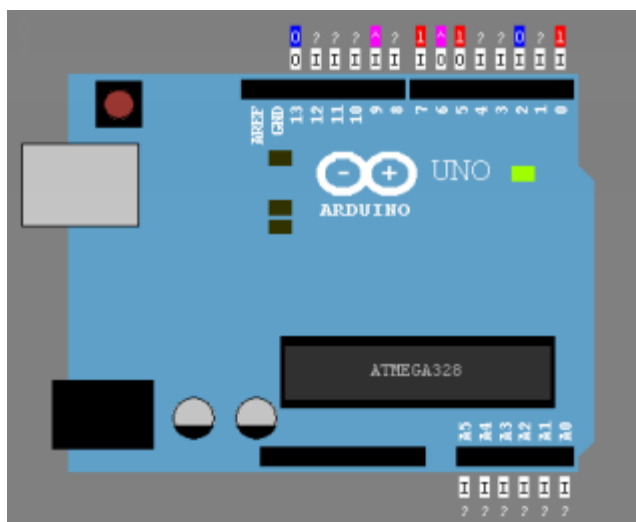
Nel programma Carica o Ripristina nota che tutte le **variabili valore non inizializzate** vengono reimpostate sul **valore 0** e tutte le **variabili puntatore non inizializzate** vengono reimpostate su **0x0000**.

Area del Banco da Laboratorio

Il Area del Banco da Laboratorio mostra una scheda a 5 volt 'Uno' che è circondata da una serie di dispositivi 'I / O' che è possibile selezionare / personalizzare e collegare ai pin 'Uno' desiderati.

Il 'Uno'

Questa è una rappresentazione della 'Uno' board e dei suoi LED integrati. Quando si carica un nuovo programma in UnoArduSim, se lo analizza correttamente subisce un "scaricamento simulato" per 'Uno' che riproduce il comportamento effettivo di 'Uno' board - vedrai lampeggiare il LED RX e TX seriale attività sui pin 1 e 0 che sono *cablati per la comunicazione seriale con un computer host*). Questo è immediatamente seguito da un flash a 13 pin che indica il reset della scheda e (e UnoArduSim si fermano automaticamente) all'inizio dell'esecuzione del programma caricato. È possibile evitare questa visualizzazione e il lag di caricamento associato deselegnando **Mostra Scaricamento** da **Configurare | Preferenze** .



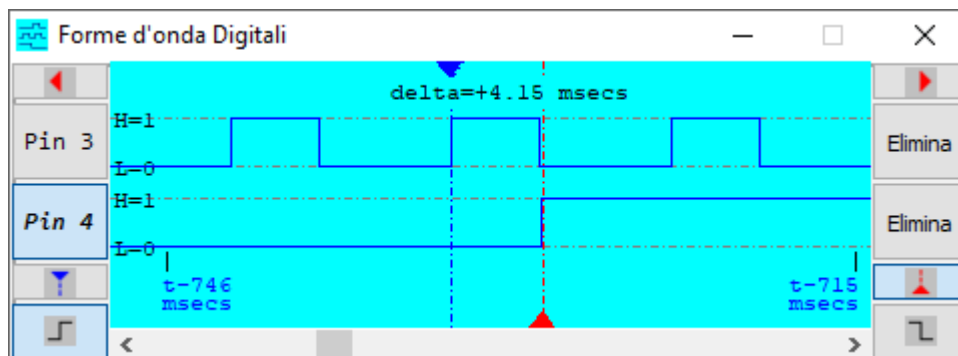
La finestra consente di visualizzare i livelli logici digitali su tutti i pin 20 'Uno' ('1' su rosso per 'HIGH' , '0' su blu per 'LOW' e '?' su grigio per una tensione indeterminata indefinita) e direzioni programmate ('I' per 'INPUT' o 'O' per 'OUTPUT'). Per i pin che vengono pulsati utilizzando PWM tramite 'analogWrite()' o 'tone()' o 'Servo.write()' , il colore diventa viola e il simbolo visualizzato diventa '^' .

Notare che i **pin digitali 0 e 1 sono cablati tramite resistori da 1-kOhm al chip USB per la comunicazione seriale con un computer host.**









A parte: i pin digitali 0-13 vengono visualizzati come pin simulatore 0-13 e i pin analogici 0-5 vengono visualizzati come A0-A5. Per accedere a un pin analogico nel



programma, è possibile fare riferimento al numero pin mediante uno dei due set di numeri equivalenti: 14-19; o A0-A5 (A0-A5 sono incassato 'const' variabili con valori 14-19). E solo quando si usa 'analogRead()' , viene resa disponibile una terza opzione: è possibile, per questa istruzione, rilasciare il prefisso 'A' dal numero pin e usare semplicemente 0-5. Per accedere ai pin 14-19 del tuo programma usando 'digitalRead()' o 'digitalWrite()' , puoi semplicemente fare riferimento a quel numero di pin, oppure puoi usare gli alias A0-A5.

Facendo clic con il tasto sinistro su qualsiasi 'Uno' pin si aprirà una finestra **Pin Forme d'Onda Digitali** che visualizza il valore passato di **un secondo di attività a livello digitale** su quel pin. È possibile fare clic su altri pin per aggiungerli al display Pin Forme d'Onda Digitali (fino a un massimo di 4 forme d'onda alla volta).



Clicca per visualizzare la pagina a sinistra o destra, oppure usa i tasti Home, PgUp, PgDn, Fine

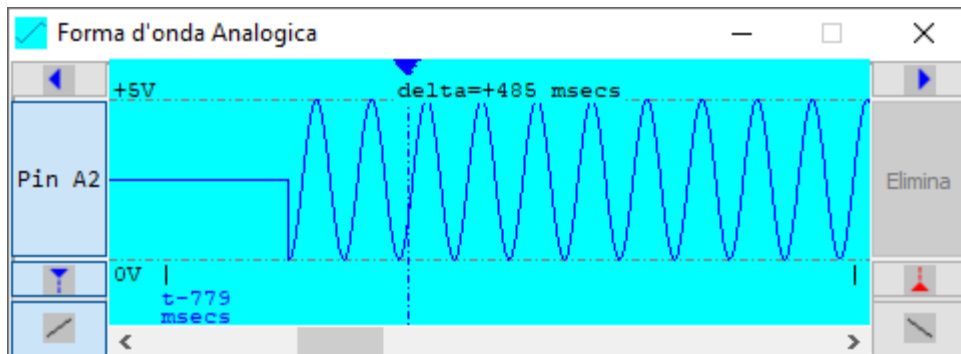
Una delle forme d'onda visualizzate sarà la **forma d'onda del pin attiva**, indicata dal suo pulsante "Pin" che viene mostrato come depresso (come nella cattura dello schermo delle Forme d'Onda Digitali Pin). È possibile selezionare una forma d'onda facendo clic sul relativo pulsante Numero pin, quindi selezionare la polarità del bordo di interesse facendo clic sul pulsante di selezione della polarità del fronte di salita / discesa appropriato, , o  oppure utilizzando i tasti di scelta rapida **freccia su** e **freccia giù**. È quindi possibile **saltare** il cursore attivo (le linee del cursore blu o rosso con il tempo di delta visualizzato) indietro o in avanti rispetto al bordo digitale della polarità selezionata *di questo pin attivo* forma d'onda usando i pulsanti del cursore (,  o , , dipende da quale cursore è stato attivato in precedenza con  o , o semplicemente utilizzare i tasti della tastiera ← e → .

Per attivare un cursore, fai clic sul suo pulsante di attivazione colorato ( o  mostrato sopra) - *anche questo salta-scorre la vista fino alla posizione corrente di quel cursore*. In alternativa, è possibile alternare rapidamente l'attivazione tra i cursori (con le rispettive viste centrate) utilizzando il tasto **TAB** della scorciatoia .







È possibile **saltare** il cursore attualmente attivato facendo **clic con il pulsante sinistro del mouse in qualsiasi punto della regione di visualizzazione della forma d'onda sullo schermo**. In alternativa, puoi selezionare la linea del cursore rossa o blu facendo clic destro sopra di essa (per attivarla), quindi *trascinala su una nuova posizione* e rilascio. Quando un cursore desiderato è attualmente fuori dallo schermo, puoi fare **clic con il pulsante destro del mouse in qualsiasi punto della vista** per passare a quella nuova posizione sullo schermo. Se entrambi i cursori sono già sullo schermo, facendo clic con il tasto destro si alterna semplicemente il cursore attivato.

Per AUMENTARE ZOOM e DIMINUIRE ZOOM (lo zoom è sempre centrato sul cursore ATTIVO), usa la rotellina del mouse o le scorciatoie da tastiera CTRL-freccia su e CTRL-freccia giù .

Facendo invece **clic con il tasto destro su qualsiasi 'Uno' pin** apre una finestra di **Forma d'Onda Analogica a spillo** che visualizza il valore **passato di un secondo di attività di livello analogico** su quel pin. A differenza della finestra Pin Forme d'Onda Digitali, è possibile visualizzare l'attività analogica su un solo pin alla volta.



Clicca per visualizzare la pagina a sinistra o destra, oppure usa i tasti Home, PgUp, PgDn, Fine

È possibile **saltare le** linee del cursore blu o rosso al successivo "punto di pendenza" in aumento o in diminuzione utilizzando i pulsanti freccia avanti o indietro (,  o , , sempre in base al cursore attivato, o utilizzare i tasti ← e →) in concerto con i pulsanti di selezione della pendenza in salita / discesa ,  (il "punto di inclinazione" si verifica quando la tensione analogica passa attraverso l'alta soglia del livello logico digitale del pin ATmega). In alternativa, è possibile fare nuovamente clic per saltare o trascinare queste linee del cursore in modo simile al loro comportamento nella finestra Forme di Forme d'Onda Digitali

Dispositivi 'I / O'

Un numero di dispositivi diversi circondano 'Uno' sul perimetro del **Area del Banco di Laboratorio** . I dispositivi "Small" 'I / O' (di cui sono consentiti fino a 16 in totale) risiedono lungo i lati sinistro e destro del Area. I dispositivi "grandi" 'I / O' (di cui sono consentiti fino a 8 in totale) dispongono di elementi "attivi" e risiedono lungo la parte superiore e inferiore del **Area di Banco del Laboratorio** . Il numero desiderato di ciascun tipo di dispositivo 'I / O' disponibile può essere impostato utilizzando il menu **Configurare | 'I / O' Dispositivi** .

Ogni dispositivo 'I / O' ha uno o più collegamenti pin visualizzati come un numero di pin a **due cifre** (00, 01, 02, ... 10,11,12, 13 e A0-A5 o 14-19, dopo quello) in una corrispondente casella di modifica .. Per i numeri di pin da 2 a 9 è sufficiente inserire la singola cifra - lo 0 iniziale verrà fornito automaticamente, ma per i pin 0 e 1 è necessario prima inserire lo 0 iniziale. Gli ingressi sono *normalmente* sul lato sinistro di un dispositivo 'I / O' e le uscite sono *normalmente* sulla destra (*spazio permettendo*). I dispositivi All 'I / O' risponderanno direttamente ai livelli dei pin e alle modifiche a livello di pin, quindi risponderanno alle funzioni della libreria indirizzate ai rispettivi pin collegati, o alla programmazione `'digitalWrite()'` (per l'operazione "bit-banged").

È possibile collegare più dispositivi allo stesso pin ATmega purché questo non crei un **conflitto elettrico** . Un tale conflitto può essere creato o da 'Uno' pin come `'OUTPUT'` che guida contro un dispositivo collegato a conduzione forte (bassa impedenza) (ad esempio, guida contro un output 'FUNCGEN' o un output 'MOTOR' encoder) oppure da due dispositivi collegati in competizione tra loro (ad esempio sia un 'PULSER' che un 'PUSH' - collegato allo stesso pin). Qualsiasi conflitto di questo tipo sarebbe disastroso in una vera implementazione hardware e quindi non consentito, e verrà segnalato all'utente tramite una finestra di messaggio a comparsa).

La finestra di dialogo può essere utilizzata per consentire all'utente di scegliere i tipi e i numeri dei dispositivi 'I / O' desiderati. Da questa finestra di dialogo è inoltre possibile **salvare i dispositivi 'I / O'** in un file di testo e / o **Caricare i dispositivi 'I / O'** da un file di testo precedentemente salvato (o modificato) (**incluse tutte le connessioni pin e le impostazioni selezionabili e qualsiasi valori digitati nella casella di modifica**).

Si noti che dalla versione 1.6, i valori nelle caselle di modifica periodo, ritardo e larghezza impulso nei relativi Dispositivi 'I / O' possono essere suffissi con la lettera 'S' (o 's'). Ciò indica che dovrebbero essere **ridimensionati in** base alla posizione di un controllo di cursore **'I / O ____ S'** che appare nella **Barra degli Strumenti.della** finestra principale . Con questo cursore completamente a destra, il fattore di scala è 1.0 (unità), e con il cursore completamente a sinistra il fattore di scala è 0.0 (soggetto ai valori minimi imposti da ciascun particolare dispositivo 'I / O'). È possibile ridimensionare più di un valore di casella di modifica **contemporaneamente** utilizzando questo cursore. Questa funzione consente di trascinare il cursore durante l'esecuzione per emulare facilmente variazioni di durata dell'impulso, periodi e ritardi per quelli collegati ai dispositivi 'I / O'.

Il resto di questa sezione fornisce descrizioni per ciascun tipo di dispositivo.

'Serial' Monitor ('SERIAL')

Questo dispositivo 'I / O' consente l'ingresso e l'uscita seriale ATmega (tramite il chip 'Uno' USB) su 'Uno' pin 0 e 1. La velocità di trasmissione viene impostata utilizzando l'elenco a discesa in basso: la velocità di trasmissione selezionata **deve corrispondere** al valore che il programma trasmette alla funzione `'Serial.begin()'` per una corretta trasmissione / ricezione. *La comunicazione seriale è fissata su 8 bit di dati, 1 bit di stop e nessun bit di parità.*



Per inviare input da tastiera al tuo programma, digita uno o più caratteri nella finestra di modifica superiore (caratteri TX), quindi premi il tasto **Invio**. (i caratteri diventano in corsivo per indicare che le trasmissioni sono iniziate) - o se già in corso, i caratteri digitati aggiunti saranno in corsivo. È quindi possibile utilizzare le **funzioni**

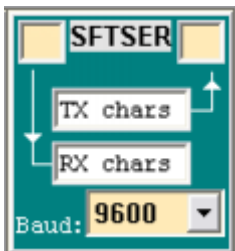
`'Serial.available()'` e `'Serial.read()'` per leggere i caratteri nell'ordine in cui sono stati ricevuti nel buffer del pin 0 (il carattere digitato più a sinistra verrà inviato per primo). Stampe testuali e numeriche formattate, o valori di byte non formattati, possono essere inviati alla finestra di output della console inferiore (caratteri RX) chiamando le

funzioni Arduino `'print()'`, `'println()'` o `'write()'`.

Inoltre, è **possibile aprire una finestra più grande per l'impostazione / visualizzazione dei caratteri TX e RX facendo doppio clic (o clic con il tasto destro) su questo dispositivo 'Serial'**. Questa nuova finestra ha una casella di modifica dei caratteri TX più grande e un pulsante 'Send' separato che può essere cliccato per inviare i caratteri TX a 'Uno' (sul pin 0). C'è anche un'opzione di spunta per reinterpretare sequenze di caratteri di \-sfuggito come '\n' o '\t' per la visualizzazione non grezza

Seriale Software ('SFTSER')

Questo dispositivo 'I / O' consente la mediazione del software della libreria o, in alternativa, l'utente "bit-banged", l'input e l'output seriale su qualsiasi coppia di pin 'Uno' che si sceglie di compilare (**eccetto per i** pin 0 e 1 che sono dedicati all'hardware `'Serial'` communication). Il tuo programma deve avere un `'#include <SoftwareSerial.h>'` line nella parte superiore se desideri utilizzare la funzionalità di tale libreria. Come con il dispositivo 'SERIAL' basato sull'hardware, la velocità di trasmissione per 'SFTSER' viene impostata utilizzando l'elenco a discesa nella parte inferiore - la velocità di trasmissione selezionata deve corrispondere al valore che il programma trasmette alla funzione `'begin()'` per corretta trasmissione / ricezione. *La comunicazione seriale è fissata su 8 bit di dati, 1 bit di stop e nessun bit di parità.*

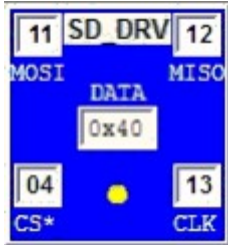


Inoltre, come per l'hardware basato su `'Serial'`, **una finestra più grande per l'impostazione / visualizzazione di TX e RX può essere aperta facendo doppio clic (o clic con il tasto destro) sul dispositivo SFTSER**.

Nota che a differenza dell'implementazione hardware di `'Serial'`, non ci sono buffer TX o RX forniti supportati da operazioni di interrupt **ATmega** interne, in modo che `'read()'` e `'write()'` stiano bloccando (ovvero, il tuo programma non procederà fino a quando non sono stati completati).

Unità disco SD ('SD_DRV')

Questo dispositivo 'I / O' consente le operazioni di input e output dei file della libreria mediata da software (ma **non** "bit-banged") sui pin 'Uno' **SPI** (è possibile scegliere quale pin **CS*** si utilizzerà). Il tuo programma può semplicemente `#include <SD.h>` line nella parte superiore, e puoi usare `<SD.h>` functions O chiamare direttamente `SdFile` functions te stesso.

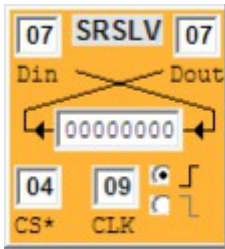


Una **finestra più grande che visualizza directory e file (e contenuto) può essere aperta facendo doppio clic (o facendo clic con il tasto destro) sul dispositivo 'SD_DRV'**. Tutto il contenuto del disco viene **caricato da** una sottodirectory **SD** nella directory del programma caricato (se esiste) su `'SdVolume init()'`, **ed è speculare alla stessa sottodirectory SD** sul file `'close()'`, `'remove()'` e su `'makeDir()'` e `'rmDir()'`.

Un LED giallo lampeggia durante i trasferimenti SPI e 'DATA' mostra l'ultimo byte di **risposta** 'SD_DRV'. Tutti i segnali SPI sono precisi e possono essere visualizzati in una **finestra di Forme d'Onda**.

Schiavo Registro di Cambio ('SRSLV')

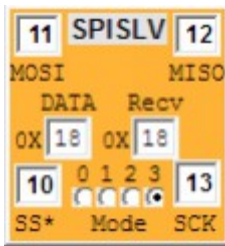
Questo dispositivo 'XXXI / O' emula un semplice dispositivo registro di cambio con **SS** attivo basso * pin controllare il pin di uscita **Dout** (ove **SS*** è alto, **Dout** non è azionato) ("slave select"). Il tuo programma potrebbe utilizzare la funzionalità dell'oggetto e della libreria Arduino SPI integrati. In alternativa, puoi scegliere di creare i tuoi segnali **Din** e **CLK** "bit-banged" per pilotare questo dispositivo.



Il dispositivo rileva transizioni di bordo sul suo ingresso **CLK** che attivano lo spostamento del suo registro - la polarità del bordo **CLK** rilevato può essere scelta usando un controllo di pulsante radio. Su ogni fronte **CLK** (della polarità rilevata), il registro acquisisce il suo livello **Din** nella posizione del bit meno significativo (LSB) del registro a scorrimento, mentre i bit rimanenti vengono spostati simultaneamente a sinistra di una posizione verso la posizione MSB. Quando **SS*** è basso, il valore corrente nella posizione MSB del registro a scorrimento viene guidato su **Dout**.

Schiavo SPI configurabile ('SPISLV')

Questo dispositivo 'XXXI / O' emula uno slave SPI modalità selezionabile con un **SS** attivo basso * ("slave select") pin controllare il pin di uscita **MISO** (ove **SS*** è alta, **MISO** non è guidato). Il tuo programma deve avere un `#include <SPI.h>` line se desideri utilizzare la funzionalità dell'oggetto e libreria SPI Arduino integrati. In alternativa, puoi scegliere di creare i tuoi segnali **MOSI** e **SCK** "bit-banged" per pilotare questo dispositivo.



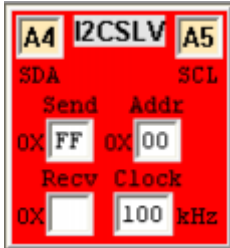
Il dispositivo rileva le transizioni di bordo sul suo ingresso **CLK in** base alla modalità selezionata (`'MODE0'`, `'MODE1'`, `'MODE2'` o `'MODE3'`), che deve essere scelto per corrispondere alla modalità SPI programmata del programma.

Facendo doppio clic (o clic con il pulsante destro del mouse) sul dispositivo, è possibile aprire una finestra compagna più ampia che invece consente di riempire il buffer massimo di 32 byte (in modo da emulare i dispositivi SPI che restituiscono automaticamente i propri dati), e vedere gli ultimi 32 byte ricevuti (tutti come coppie esadecimali).

Si noti che il successivo byte buffer TX è automaticamente associato a 'DATA' solo **dopo che è** stato completato un intero `'SPI.transfer()'` !

Schiavo I2C ('I2CSLV')

Questo dispositivo 'I / O' emula solo un dispositivo in *modalità slave*. Al dispositivo può essere assegnato un indirizzo di bus I2C utilizzando una voce a due cifre esadecimali nel suo 'Addr' casella di modifica (risponderà solo alle transazioni bus I2C che coinvolgono il suo indirizzo assegnato). Il dispositivo invia e riceve dati sul suo pin **SDA** open-drain (solo pulldown) e risponde al segnale di clock del bus sul suo piedino **SCL** a scarico aperto (solo pulldown). Sebbene 'Uno' sia il master del bus responsabile della generazione del **segnale SCL**, **questo dispositivo slave ridurrà SCL anche durante la sua fase bassa per estendere (se necessario) il tempo basso del bus a uno appropriato alla sua velocità interna (che può essere impostato nella sua 'Clock' casella di modifica).**



Il tuo programma deve avere un `#include <Wire.h>` line se desideri utilizzare la funzionalità della libreria `'TwoWire'` per interagire con questo dispositivo. In alternativa, puoi scegliere di creare i tuoi dati bit-banged e i segnali di clock per pilotare questo dispositivo slave.

Un singolo byte per la trasmissione al 'Uno' master può essere impostato nella 'Send' casella di modifica, e un singolo byte (ricevuto più di recente) può essere visualizzato nella sua (sola lettura) 'Recv' edit- scatola. ***Si noti che*** il valore 'Send' casella di modifica riflette sempre il byte **successivo** per la trasmissione dal buffer di dati interno del dispositivo.

Facendo doppio clic (o clic con il pulsante destro del mouse) sul dispositivo, è possibile aprire una finestra companion più ampia che invece consente di riempire un buffer FIFO massimo di 32 byte (in modo da emulare i dispositivi TWI con tale funzionalità) e di visualizzare (fino a un massimo di 32) byte dei dati ricevuti più di recente (come un display a due cifre esadecimali di 8 byte per riga). Il numero di righe in queste due caselle di modifica corrisponde alla dimensione del buffer TWI scelta (che può essere selezionata utilizzando **Configurare | Preferenze**). Questo è stato aggiunto come opzione poiché la libreria Arduino `'Wire.h'` utilizza **cinque** buffer RAM di questo tipo nel suo codice di implementazione, che è costoso nella memoria RAM. Modificando il file di installazione di Arduino `'Wire.h'` per modificare la costante definita `'BUFFER_LENGTH'` (e anche modificando il file companion `'utility / twi.h'` per modificare la lunghezza del buffer TWI) entrambi devono essere invece 16 o 8, un utente *potrebbe* ridurre in modo significativo il sovraccarico della memoria RAM di 'Uno' in **un'implementazione hardware** mirata - UnoArduSim rispecchia quindi questa possibilità del mondo reale tramite **Configurare | Preferenze**.

Motore Passo-Passo ('STEPR')

Questo dispositivo 'I / O' emula un motore passo-passo bipolare o unipolare da 6 V con un controller driver integrato azionato da **due** segnali di controllo (su **P1**, **P2**) o **quattro** (su **P1**, **P2**, **P3**, **P4**). È inoltre possibile impostare il numero di passaggi per giro. Puoi usare `'Stepper.h'` functions `'setSpeed ()'` e `'step ()'` per guidare 'STEPR'. In alternativa, 'STEPR' *risponderà anche* ai tuoi segnali dell'unità `'digitalWrite ()'` "bit-banged".



Il motore è accuratamente modellato sia meccanicamente che elettricamente. Le cadute di tensione del motore e la riluttanza e l'induttanza variabili sono modellate insieme a un momento di inerzia realistico rispetto alla coppia di ritenuta. L'avvolgimento del rotore del motore ha una resistenza modellata di $R = 6$ ohm e un'induttanza di $L = 6$ milli-Henries che crea una costante di tempo elettrica di 1,0 millisecondi. Grazie alla modellazione realistica si noterà che gli impulsi del pin di controllo molto stretti *non fanno muovere* il motore, sia a causa del tempo di salita della corrente finita, sia dell'effetto dell'inerzia del rotore. Questo concorda con quanto osservato quando si guida un vero motore passo-passo da un 'Uno' con, ovviamente, un appropriato (**e richiesto**) chip driver motore tra i fili del motore e il 'Uno'!

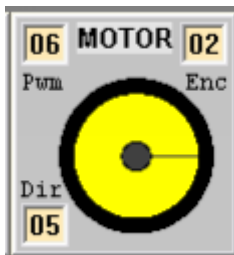
Uno sfortunato bug nel codice libreria Arduino `'Stepper.h'` indica che al reset il motore Passo-Passo non sarà in posizione Passo 1 (di quattro passi). Per ovviare a questo, l'utente deve utilizzare `'digitalWrite ()'` nella sua `'setup ()'` routine per inizializzare i livelli dei pin di controllo sui livelli `'step(1)'` appropriati per 2-

pin (0,1) o 4 -pin (1,0,1,0) controllo e consentire al motore 10 msec di spostarsi sulla posizione del motore iniziale di riferimento 12-mezzogiorno desiderata.

Notare che **la riduzione dell'ingranaggio non è direttamente supportata** a causa della mancanza di spazio, ma è possibile emularlo nel programma implementando una variabile contatore modulo-N e chiamando 'step()' quando quel contatore colpisce 0 (per riduzione dell'ingranaggio dal fattore N).

Motore CC ('MOTOR')

Questo dispositivo 'I / O' emula un motore CC a ingranaggi 100: 1 con alimentazione a 6 volt con un controller del driver integrato comandato da un segnale di modulazione della larghezza dell'impulso (sul suo ingresso **Pwm**) e un segnale di controllo della direzione (sul suo ingresso **Dir.**). Il motore ha anche un'uscita encoder ruota che aziona il suo pin di uscita **Enc**. È possibile utilizzare 'analogWrite()' per pilotare il pin **Pwm** con una forma d'onda PWM di 490 Hz (sui pin 3,9,10,11) o 980 Hz (sui pin 5,6) di duty cycle compreso tra 0,0 e 1,0 ('analogWrite()' valori da 0 a 255). In alternativa, 'MOTOR' *risponderà anche* ai tuoi segnali dell'unità 'digitalWrite()' " bit-banged".



Il motore è accuratamente modellato sia meccanicamente che elettricamente. Contabilizzazione delle cadute di tensione del transistor del motore e della coppia di ingranaggi realistica senza carico fornisce una velocità completa di circa 2 giri al secondo e una coppia di stallo di poco superiore a 5 kg-cm (che si verifica con un ciclo di lavoro PWM fisso di 1,0), con un momento di inerzia totale del motore più carico di 2,5 kg-cm. L'avvolgimento del motore ha una resistenza modellata di $R = 2$ ohm e un'induttanza di $L = 300$ micro-Henries che crea una costante di tempo elettrica di 150 microsecondi. Grazie alla modellazione realistica si noterà che gli impulsi PWM molto stretti *non fanno girare* il motore, sia a causa del tempo di salita della corrente finito, sia del tempo di spegnimento significativo dopo ogni impulso stretto. Questi si combinano per causare una quantità di moto del rotore insufficiente a superare il cuneo a molle simile al riduttore sotto l'attrito statico. La conseguenza è quando si usa 'analogWrite()', un ciclo di lavoro inferiore a circa 0,125 non farà **muovere** il motore - questo è in accordo con ciò che si osserva quando si guida un vero motoriduttore da un 'Uno' con, ovviamente, un appropriato (e **richiesto**) modulo driver motore tra il motore e 'Uno'.

Il codificatore del motore emulato è un sensore di interruzione ottica montato sull'albero che produce una forma d'onda del ciclo di lavoro del 50% con 8 periodi di interruzione della ruota completi ad alto-basso (in modo che il programma possa rilevare i cambi di rotazione della ruota con una risoluzione di 22,5 gradi).

ServoMotore ('SERVO')

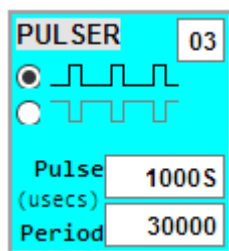
Il suo dispositivo 'I / O' emula un servomotore DC a 6 volt con alimentazione PWM comandato dalla posizione. I parametri di modellazione meccanica ed elettrica per il funzionamento del servo corrisponderanno strettamente a quelli di un servo standard HS-422. Il servo ha una velocità di rotazione massima di circa 60 gradi in 180 msec. *Se la casella di controllo in basso a sinistra è selezionata, il servo diventa un servo a rotazione continua con la stessa velocità massima, ma ora l'ampiezza dell'impulso PWM imposta la velocità piuttosto che l'angolo*



Il tuo programma deve avere '#include <Servo.h>' line prima di dichiarare le tue istanze Servo se scegli di utilizzare la funzionalità della libreria Servo, ad es. 'Servo.write()', 'Servo.writeMicroseconds()' In alternativa, SERVO risponde anche ai segnali 'digitalWrite()' " bit-banged". A causa dell'implementazione interna di UnoArduSim, sei limitato ai dispositivi 5 'SERVO'.

Generatore di Impulsi ('PULSER')

Questo dispositivo 'I / O' emula un semplice generatore di forme d'onda a impulso digitale che produce un segnale periodico che può essere applicato a qualsiasi 'Uno' pin scelto.

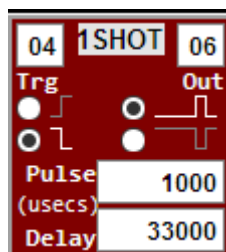


Il periodo e l'ampiezza dell'impulso (in microsecondi) possono essere impostati utilizzando le caselle di modifica: il periodo minimo consentito è 50 microsecondi e la larghezza minima dell'impulso è di 10 microsecondi. La polarità può anche essere scelta: impulsi di fronte positivo (da 0 a 5 V) o impulsi di fronte negativo (da 5 V a 0 V).

I valori di 'Pulse' e 'Period' sono scalabili dalla **Barra degli Strumenti**. principale 'I / O ____ S' controllo del cursore fattore di scala mediante il suffisso di uno (o entrambi) con la lettera 'S' (o 's').

Un-Colpo ('1SHOT')

Questo dispositivo 'I / O' emula uno scatto digitale che può generare un impulso di polarità e larghezza dell'impulso scelto sul suo 'Out' pin, che si verifica dopo un ritardo specificato da un fronte di trigger ricevuto sul suo pin di ingresso **Trg** (trigger). Una volta ricevuto il fronte di innesco specificato, inizia la temporizzazione e quindi un nuovo impulso di innesco non verrà riconosciuto fino a quando l'impulso '**Out**' è stato prodotto (e completamente finito).

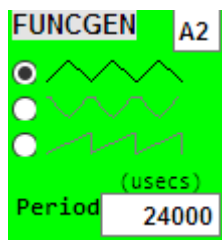


Un possibile utilizzo di questo dispositivo è la simulazione di sensori di portata ad ultrasuoni che generano un impulso di intervallo in risposta a un impulso di attivazione. Può anche essere utilizzato ovunque si desideri generare un segnale di ingresso pin sincronizzato (dopo il ritardo selezionato) su un segnale di uscita pin creato dal programma.

'Pulse' e 'Delay' i valori sono scalabili dalla **Barra degli Strumenti**. Finestra principale 'I / O ____ S' controllo cursore fattore di scala mediante il suffisso di uno (o entrambi) con la lettera 'S' (o 's').

Generatore di Funzioni Analogiche ('FUNCGEN')

Questo dispositivo 'I / O' emula un semplice generatore di forme d'onda analogico che produce un segnale periodico che può essere applicato a qualsiasi 'Uno' pin scelto.



Il periodo (in microsecondi) può essere impostato utilizzando la casella di modifica: il periodo minimo consentito è 100 microsecondi. La forma d'onda che crea può essere scelta come sinusoidale, triangolare o a dente di sega (per creare un'onda quadra, utilizzare invece 'PULSER'). In periodi più brevi, vengono utilizzati meno campioni per ciclo per modellare la forma d'onda prodotta (solo 4 campioni per ciclo al periodo = 100 usec).

Il valore 'Period' è scalabile dalla **Barra degli Strumenti**. principale **Barra degli Strumenti**. 'I / O ____ S' controllo cursore del fattore di scala mediante suffisso con la lettera 'S' (o 's').

Altoparlante Piezoelettrico ('PIEZO')



Il suo dispositivo consente di "ascoltare" i segnali su qualsiasi 'Uno' pin scelto e può essere un utile complemento ai LED per il debug delle operazioni del programma. Puoi anche divertirti un po' a suonare suonerie in modo appropriato 'tone()' e 'delay()' calls (anche se non c'è il filtraggio della forma d'onda rettangolare, quindi non sentirai le note "pure").

Puoi anche ascoltare un dispositivo collegato 'PULSER' o 'FUNCEN' collegando un 'PIEZO' al pin che il dispositivo guida.

Pulsante ('PUSH')



Questo dispositivo 'XXXI / O' emula un pulsante normalmente aperto **o interruttori** unipolari, single-laterale (SPST) con 10 k-ohm pull-up (o pull-down) resistore. Se per il dispositivo viene scelta una selezione di transizione del fronte di salita, i contatti del pulsante saranno cablati tra il pin del dispositivo e + 5V, con un pull-down da 10 k-Ohm verso terra. Se per il dispositivo viene scelta una transizione del fronte di discesa, i contatti del pulsante verranno cablati tra il pin del dispositivo e la massa, con un pull-up da 10 k-Ohm a + 5V.

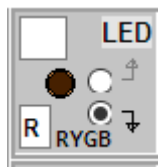
Cliccando con il tasto sinistro del mouse sul tasto o premendo un tasto qualsiasi, si chiude il contatto del pulsante. In modalità **momentanea**, rimane chiuso fino a quando si tiene premuto il tasto o il tasto del mouse e in modalità **latch** (abilitato facendo clic sul pulsante 'latch') rimane chiuso (e di un colore diverso) finché non si preme di nuovo il pulsante. Il rimbalzo del contatto (per 1 millisecondo) verrà prodotto ogni volta che si utilizza la barra spaziatrice per premere il pulsante.

Resistenza interruttore a Scorrimento ('R = 1K')



Il suo dispositivo consente all'utente di collegare a un pin 'Uno' un resistore di pull-up da 1 k-ohm a +5 V, o un resistore di pull-down da 1 k-ohm a terra. Ciò consente di simulare carichi elettrici aggiunti a un dispositivo hardware reale. Facendo clic con il pulsante sinistro del mouse sul **corpo** dell'interruttore scorrevole è possibile attivare la selezione pull-up o pull-down desiderata. L'utilizzo di uno o più di questi dispositivi consente di impostare un "codice" singolo (o multi) per il programma da leggere e rispondere.

LED Colorato ('LED')



È possibile collegare un LED tra il 'Uno' pin prescelto (tramite una resistenza nascosta da 1 k-Ohm con limitatore di corrente serie 1) a terra o a +5 V - questo consente di scegliere se il LED si illumina quando il connected 'Uno' pin è 'HIGH', o invece, quando è 'LOW'.

Il colore del LED può essere scelto per essere rosso ('R'), giallo ('Y'), verde ('G') o blu ('B') usando la sua casella di modifica.





Cursore Analogico








Un potenziometro da 0-5 V comandato da un cursore può essere collegato a qualsiasi 'Uno' pin selezionato per produrre un livello di tensione analogica statica (o lentamente variabile) che verrà letto da 'analogRead()' come valore da 0 a 1023. Usa il mouse per trascinare, o clicca per saltare, il cursore analogico.

Menu









File:

 Caricare INO o PDE Programma (ctrl-L)	Consente all'utente di scegliere un file di programma con l'estensione selezionata. Il programma viene immediatamente analizzato
Modificare / Esaminare (ctrl-E)	Apri il programma caricato per la visualizzazione / modifica.
 Salvare	Salva il contenuto del programma modificato nel file di programma originale.
Salva Come	Salva il contenuto del programma modificato con un nome di file diverso.
 Successivo ('#include')	Fa avanzare il Area del Codice per visualizzare il prossimo ' #include ' file
 Precedente	Restituisce la visualizzazione del Area del Codice al file precedente
Esci	Esce da UnoArduSim dopo aver ricordato all'utente di salvare qualsiasi file modificato.

Trova:

 Trova Funzione/Var Successiva	Passare alla funzione successiva nel Area del Codice (se ha il focus attivo) o alla variabile successiva nel Area delle Variabili (se invece ha il focus attivo).
 Trova Funzione / Var Precedente	Passare alla funzione precedente nel Area del Codice (se ha il focus attivo) o alla variabile precedente nel Area delle Variabili (se invece ha il focus attivo).
 Imposta Testo di Ricerca (ctrl-F)	Attiva la Barra degli Strumenti . Trova casella di modifica per definire il testo da cercare in seguito (e aggiunge la prima parola dalla riga attualmente evidenziata nel Area di Codice o nel Area delle Variabili se uno di questi ha lo stato attivo).
 Ricerca Prossimo Testo	Passare alla occorrenza testo successiva nel Area di Codice (se ha il focus attivo) o alla successiva ricorrenza testo nel Area delle Variabili (se invece ha il focus attivo).
 Ricerca Testo Precedente	Passare alla ricorrenza di testo precedente nel Area del Codice (se ha il focus attivo) o alla ricorrenza di testo precedente nel Area delle Variabili (se invece ha il focus attivo).

Eeguire:

 Passo In (F4)	Esegue l'esecuzione in avanti di una istruzione o <i>in una funzione chiamata</i> .
 Passo Scavalcare (F5)	Esegue l'esecuzione in avanti di una istruzione o <i>di una chiamata di funzione completa</i> .
 Passo Fuori (F6)	Esegue l'esecuzione di quanto <i>basta per lasciare la funzione corrente</i> .
 Eseguire Verso (F7)	Esegue il programma, <i>fermandosi alla riga di programma desiderata</i> - è necessario prima fare clic per evidenziare una linea di programma desiderata prima di utilizzare Eseguire Verso.
 Eseguire Fino A(F8)	Esegue il programma fino a quando non si verifica una scrittura nella variabile che ha l'evidenziazione corrente nel Area delle Variabili (fare clic su uno per stabilire l'evidenziazione iniziale).
 Eseguire (F9)	Esegue il programma.
 Arresto (F10)	Arresta l'esecuzione del programma (<i>e blocca il tempo</i>).
 Resettare	Ripristina il programma (tutte le variabili di valore vengono reimpostate sul valore 0 e tutte le variabili puntatore vengono reimpostate su 0x0000).
Animare	Passa automaticamente alle righe di programma consecutive <i>con ritardo</i> e evidenziamiento <i>artificiali</i> della riga di codice corrente. Le operazioni e i suoni in tempo reale sono persi.
Rallentatore	Rallenta il tempo di un fattore di 10.

Opzioni:

<u>Scavalcare Struttori / Operatori</u>	Volta attraverso costruttori, distruttori e funzioni di sovraccarico dell'operatore durante qualsiasi passo (cioè non si fermerà all'interno di queste funzioni).
<u>Modellazione di Registro-Assegnazione</u>	Assegna ai locals delle funzioni di liberare i registri ATmega invece che allo stack (genera un utilizzo della RAM piuttosto ridotto).
<u>Errore su non inizializzato</u>	Segnala come errore di Analisi ovunque il tuo programma tenti di utilizzare una variabile senza aver prima inizializzato il suo valore (o almeno un valore all'interno di una matrice).
<u>Aggiunto 'loop()' Ritardo</u>	Aggiunge 200 microsecondi di ritardo ogni volta che 'loop()' viene chiamato (nel caso in cui non ci siano altre chiamate di programma a 'delay()' ovunque) - utile per cercare di evitare di cadere troppo indietro rispetto al tempo reale.

Configurare:

<u>Dispositivi 'I / O'</u>	Aprire una finestra di dialogo per consentire all'utente di scegliere il tipo (i) e i numeri dei Dispositivi 'I / O' desiderati. Da questa finestra di dialogo è inoltre possibile Salvare i Dispositivi 'I / O' in un file di testo e / o Caricare i Dispositivi 'I / O' da un file di testo precedentemente salvato (o modificato) incluse tutte le pin connections e le impostazioni selezionabili e digitato valori
<u>Preferenze</u>	Aprire una finestra di dialogo che consente all'utente di impostare le preferenze tra cui il rientro automatico delle linee del programma di origine, consentendo la sintassi avanzata, la scelta del carattere tipografico, la scelta di caratteri più grandi, l'applicazione dei limiti di matrice, l'autorizzazione delle parole chiave dell'operatore logico, la visualizzazione del scaricamento del programma, scelta di 'Uno' versione della scheda e lunghezza del buffer TWI (per dispositivi I2C).

VarAggiorna:

<u>Consenti il Collasso Automatico(-)</u>	Permetti a UnoArduSim di comprimere gli array / oggetti espansi mostrati quando cadi indietro in tempo reale.
<u>Minimo</u>	Aggiorna solo la visualizzazione del Area delle Variabili 4 volte al secondo.
<u>Evidenzia Modifiche</u>	Evidenzia il valore variabile dell'ultima variabile (può causare rallentamenti).

Finestre:

<u>'Serial' Monitor</u>	Collegare un dispositivo 'I / O' seriale ai pin 0 e 1 (se nessuno) e aprire una finestra di testo più grande 'Serial' Monitor TX / RX.
<u>Ripristinare Tutto</u>	Ripristina tutte le finestre secondarie ridotte a icona.
<u>Pin Forme d'Onda Digitali</u>	Ripristinare una finestra minimizzata Pin Forme d'Onda Digitali .
<u>Pin Forma d'Onda Analogica</u>	Ripristinare una finestra Pin Forma d'Onda Analogica con perno minimizzata.

Aiuto:

<u>File di Aiuto Rapido</u>	Aprire il file PDF UnoArduSim_QuickHelp.
<u>File di Aiuto Completo</u>	Aprire il file PDF UnoArduSim_FullHelp.
<u>Correzioni di Bugs</u>	Visualizza correzioni di bug significative dalla versione precedente ..
<u>Modifica / Miglioramenti</u>	Visualizza modifiche e miglioramenti significativi rispetto alla versione precedente.
<u>Di</u>	Visualizza versione, copyright.

Modellismo

'Uno' Board e Dispositivi 'I / O'

I 'Uno' e Dispositivi 'I / O' allegati sono tutti accuratamente modellati elettricamente, e sarete in grado di ottenere una buona idea a casa di come i vostri programmi si comporteranno con l'hardware effettivo, e tutti i conflitti di pin elettrici saranno segnalati.

Sincronizzazione

UnoArduSim esegue abbastanza rapidamente su un PC o tablet che può (*nella maggior parte dei casi*) le azioni del programma modello in tempo reale, **ma solo se il tuo programma incorpora** almeno alcune piccole 'delay()' chiamate o altre chiamate che manterranno naturalmente è sincronizzato al tempo reale (vedi sotto).

Per fare ciò, UnoArduSim utilizza una funzione timer callback di Windows, che consente di tenere traccia accurata del tempo reale. L'esecuzione di un numero di istruzioni del programma viene simulata durante una porzione del timer e le istruzioni che richiedono un'esecuzione più lunga (come le chiamate a 'delay()') potrebbero richiedere l'utilizzo di più sezioni del timer. Ogni iterazione della funzione del timer di callback corregge l'ora del sistema utilizzando l'orologio hardware del sistema in modo che l'esecuzione del programma sia costantemente regolata per mantenere il passo di blocco in tempo reale. *L'unica frequenza in cui il tempo di esecuzione deve essere inferiore al tempo reale* è quando l'utente ha creato loop stretti **senza alcun ritardo aggiunto**, oppure i dispositivi 'I / O' sono configurati per funzionare con frequenze 'I / O' molto elevate (e / o baud rate) che genererebbe un numero eccessivo di eventi di cambiamento a livello di pin e un sovraccarico di elaborazione associato. UnoArduSim affronta questo sovraccarico saltando alcuni intervalli del timer per compensare, e quindi rallenta la progressione del programma al di **sotto del tempo reale**.

Inoltre, i programmi con array di grandi dimensioni visualizzati o che presentano loop stretti **senza alcun ritardo aggiunto** possono causare una frequenza di chiamata ad alta funzione e generare un elevato aggiornamento del display di **Area delle Variabili** causando un ritardo in tempo reale - questo può essere eluso utilizzando **Consenti Riduzione**, o quando veramente necessario, **Minimo**, dal menu **VarAggiorna**,

La modellazione accurata del tempo di esecuzione inferiore al millisecondo per ciascuna istruzione o operazione del programma **non viene eseguita**, ma per la simulazione sono state adottate solo stime molto approssimative. Tuttavia, i tempi delle funzioni 'delay()' e 'delayMicroseconds()' e delle funzioni 'millis()' e 'micros()' sono tutti perfettamente accurati e **purché si utilizzi almeno una delle funzioni di ritardo** in un loop da qualche parte nel tuo programma, o usi una funzione che si lega naturalmente al funzionamento in tempo reale (come 'print()' che è legato alla velocità di trasmissione scelta), quindi la performance simulata del tuo programma sarà molto vicina in tempo reale (di nuovo, salvo eventi di modifica a livello di pin ad alta frequenza eccessivamente eccessivi o eccessivi aggiornamenti variabili consentiti dall'utente che potrebbero rallentarlo).

Per vedere l'effetto delle singole istruzioni del programma *quando si esegue*, potrebbe essere desiderabile essere in grado di rallentare le cose. Un fattore di rallentamento temporale di 10 può essere impostato dall'utente nel menu **Esegui**.

Sincronizzazione di Dispositivo 'I / O'

Questi dispositivi virtuali ricevono segnalazione in tempo reale delle modifiche che si verificano sui loro pin di ingresso e producono uscite corrispondenti sui loro pin di uscita che possono essere rilevati da 'Uno' - sono quindi intrinsecamente sincronizzati con l'esecuzione del programma. La temporizzazione del dispositivo interno 'I / O' viene impostata dall'utente (ad esempio tramite la selezione della velocità di trasmissione o la frequenza di clock) e gli eventi del simulatore sono impostati per tracciare il funzionamento interno in tempo reale.

Suoni

Ciascun dispositivo 'PIEZO' produce un suono corrispondente alle variazioni del livello elettrico che si verificano sul pin collegato, indipendentemente dalla fonte di tali cambiamenti. Per mantenere i suoni sincronizzati all'esecuzione del programma, UnoArduSim avvia e interrompe la riproduzione di un buffer sonoro associato quando l'esecuzione viene avviata / interrotta.

A partire dalla V2.0, il suono è stato modificato per utilizzare l'API audio Qt, ma purtroppo QAudioOutput 'class' non supporta il looping del buffer audio per evitare l'esaurimento dei campioni audio (come può accadere durante i lunghi tempi operativi della finestra del sistema operativo). Pertanto, al fine di evitare la grande maggioranza di fastidiosi clic del suono e interruzione del suono durante i ritardi del sistema operativo, l'audio è ora disattivato in base alla seguente regola:

L'audio è disattivato fintanto che UnoArduSim non è la finestra "attiva" (tranne quando è stata appena creata e attivata una nuova finestra secondaria) **e anche** quando UnoArduSim è la finestra principale "attiva" ma il puntatore del mouse è **al** di **fuori** della sua principale area client finestra.

Si noti che questo implica che il suono verrà temporaneamente disattivato come re mentre il puntatore del mouse passa **con il** mouse **su una finestra** secondaria , e verrà disattivato **se si fa clic su questa** finestra secondaria **per attivarlo** (finché non si fa nuovamente clic sulla finestra UnoArduSim nano per riattivare ir) .

Il suono può essere sempre disattivato facendo clic su ianywhere all'interno dell'area client della finestra principale di UoArduSim.

A causa del buffering, s ha un ritardo in tempo reale fino a 250 millisecondi dal tempo dell'evento corrispondente sul pin dell'allegato 'PIEZO'.

Limitazioni e elementi non supportati

File Inclusi

A '<>' - parentesi '#include' di '<Servo.h>', '<Wire.h>', '<SoftwareSerial.h>', '<SPI.h>', '<EEPROM.h>' e '<SD.h>' è supportato ma questi sono solo emulati - i file effettivi non vengono cercati; invece la loro funzionalità è direttamente "incorporata" in UnoArduSim e sono valide per la versione Arduino supportata fissa.

Qualsiasi citazione '#include' (ad esempio "supp.ino", "myutil.cpp" o "mylib.h") è supportata, ma tutti questi file devono **risiedere nella stessa directory come il file del programma principale** che contiene i loro '#include' (non viene eseguita alcuna ricerca in altre directory). La funzione '#include' può essere utile per ridurre al minimo la quantità di codice del programma visualizzato nel **Area del Codice** in qualsiasi momento. I file di intestazione con '#include' (cioè quelli che hanno un'estensione ".h") causano anche il tentativo del simulatore di includere lo stesso file con estensione ".cpp" (se esiste anche nella directory del genitore programma).

Allocazioni di Memoria Dinamica e RAM

operatori 'new' e 'delete' sono supportati, così come gli oggetti nativi Arduino 'String', **ma non le chiamate dirette a** 'malloc()', 'realloc()' e 'free()' su cui questi si basano.

L'eccessivo utilizzo della RAM per le dichiarazioni variabili viene contrassegnato in temp di Analisi e l'overflow della memoria RAM viene contrassegnato durante l'esecuzione del programma. Un elemento su menù **Options** ti permette di emulare la normale allocazione del registro ATmega come farebbe il compilatore AVR, o di modellare uno schema di compilazione alternativo che utilizza solo lo stack (come opzione di sicurezza nel caso

in cui un bug si presenti nella mia allocazione del registro modellazione). Se si dovesse utilizzare un puntatore per esaminare il contenuto dello stack, questo dovrebbe riflettere accuratamente ciò che apparirebbe in un'implementazione hardware effettiva.

Allocazioni di memoria 'Flash'

'Flash' memory 'byte', 'int' e 'float' variables / array e le loro corrispondenti funzioni di accesso in lettura sono supportate. Qualsiasi 'F()' la chiamata di funzione ("Flash'-macro") di qualsiasi stringa letterale è supportata, ma le uniche funzioni di accesso diretto 'Flash'-memory supportate sono 'strcpy_P()' e 'memcpy_P()', quindi per utilizzare altre funzioni è necessario copiare prima la stringa di 'Flash' ad una RAM normale 'String' variabile, e quindi lavorare con quella RAM 'String'. Quando si utilizza la parola chiave modificatore di variabili 'PROGMEM', deve comparire **davanti** al nome della variabile e anche tale variabile **deve essere dichiarata** come 'const'.

Variabili 'String'

La libreria nativa 'String' è quasi completamente supportata con alcune eccezioni molto (e minori).

Gli operatori 'String' supportati sono +, +=, <, <=, >, >=, ==, !=, []. **Nota:** 'concat()' accetta un **singolo** argomento che è 'String', o 'char', o 'int' che deve essere aggiunto all'oggetto 'String' originale, **non** due argomenti come erroneamente indicato nel Riferimento Arduino pagine web).

Librerie Arduino

Solo 'Stepper.h', 'SD.h', 'Servo.h', 'SoftwareSerial.h', 'SPI.h', 'Wire.h' e 'EEPROM.h' per la versione di Arduino V1.6.6 attualmente sono supportati in UnoArduSim. Provando a '#include' i file ".cpp" e ".h" di altre librerie non ancora supportate **non funzioneranno** in quanto contengono istruzioni di assemblaggio di basso livello e direttive non supportate e file non riconosciuti!

Puntatori

Puntatori a tipi, matrici o oggetti semplici sono tutti supportati. Un puntatore può essere equiparato a una matrice dello stesso tipo (es `iptr = intarray`), ma in questo caso non ci sarebbero *limiti di array successivi che controllano* un'espressione come `iptr [index]`.

Le funzioni possono restituire puntatori o 'const' puntatori, ma qualsiasi livello successivo di 'const' sul puntatore restituito viene ignorato.

Non vi è **alcun supporto** per le chiamate di funzione effettuate tramite **i puntatori di funzione dichiarati dall'utente**.

Oggetti 'class' e 'struct'

Sebbene sia supportato il **polimorfismo** e l'ereditarietà (a qualsiasi profondità), un 'class' o 'struct' può essere definito solo per avere al massimo **una** base 'class' (cioè l'ereditarietà **multipla** non è supportata). Le chiamate di inizializzazione del costruttore 'class' (tramite notazione dei due punti) nelle righe di dichiarazione del costruttore sono supportate, ma non le **inizializzazioni dei membri che utilizzano la stessa notazione dei due punti**. Ciò significa che gli oggetti che contengono le variabili 'const' non-'static', o le variabili di tipo di riferimento, non sono supportati (quelli sono possibili solo con le inizializzazioni del membro di costruzione specificate).

I sovraccarichi dell'operatore di assegnazione delle copie sono supportati insieme ai costruttori di movimento e alle assegnazioni di movimento, ma le funzioni di conversione dell'oggetto definita dall'utente ("simile a un cast") non sono supportate.

Scopo

Non esiste alcun supporto per la parola chiave `'using'` o per gli spazi dei nomi o per `'file'` scope. Tutte le dichiarazioni non locali sono per implementazione considerate globali.

Qualsiasi `'typedef'`, `'struct'`, o `'class'` definizione (vale a dire che può essere utilizzato per dichiarazioni future), deve essere impostato a **livello globale** (le definizioni **locali** di tali elementi all'interno di una funzione non sono supportate).

Qualificazioni `'unsigned'`, `'const'`, `'volatile'`, `'static'`

Il prefisso `'unsigned'` funziona in tutti i normali contesti legali. La parola chiave `'const'`, se utilizzata, deve **precedere** il nome della variabile o il nome della funzione o il nome `'typedef'` che viene dichiarato: posizionarlo dopo il nome causerà un errore di Analisi. Per le dichiarazioni di funzioni, solo le funzioni di ritorno puntatore possono avere `'const'` nella loro dichiarazione.

Tutte le variabili UnoArduSim sono `'volatile'` per implementazione, quindi la parola chiave `'volatile'` viene semplicemente ignorata in tutte le dichiarazioni di variabile. Le funzioni non possono essere dichiarate `'volatile'`, né gli argomenti di chiamata di funzione.

La `'static'` keyword è consentita per le variabili normali e per i membri dell'oggetto e le funzioni membro, ma è esplicitamente non consentita per le istanze dell'oggetto stesse (`'class'` / `'struct'`), per le funzioni non membri e per tutti gli argomenti delle funzioni.

Direttive del Compilatore

`'#include'` e regular `'#define'` sono entrambi supportati, ma **non macro** `'#define'`. Il `'#pragma'` direttive di inclusione direttiva e condizionale (`'#ifdef'`, `'#ifndef'`, `'#se'`, `'#endif'`, `'#else'` e `'#elif'`) **non sono supportati**. La `'#linea'`, `'#errore'` e anche le macro predefinite (come `'_LINE_'`, `'_FILE_'`, `'_DATE_'` e `'_TIME_'`) **non sono supportate**.

Elementi di Lingua Arduino

Tutti gli elementi linguistici nativi di Arduino sono supportati con l'eccezione dell'istruzione **dubious** `'goto'` (l'unico uso ragionevole a cui posso pensare potrebbe essere un salto (ad un ciclo infinito di salvataggio e arresto continuo) in caso di una condizione di errore che il tuo programma non può altrimenti gestire).

C / C ++ - Elementi del Linguaggio

I "qualificatori del campo di bit" per il salvataggio di bit per i membri nelle definizioni di struttura **non sono supportati**.

`'union'` **non è supportato**.

L'oddball "operatore virgola" **non è supportato** (quindi non è possibile eseguire diverse espressioni separate da virgole quando è normalmente prevista una sola espressione, ad esempio nei costrutti `'while()'` e `'for (; ;)'`).

Modelli di funzione

Le funzioni definite dall'utente che utilizzano la parola chiave "modello" per consentirgli di accettare argomenti di tipo "generico" **non** sono **supportate** .

Emulazione in tempo reale

Come notato sopra, i tempi di esecuzione delle molte diverse singole istruzioni del programma Arduino possibili **non** sono modellate accuratamente, in modo che per poter essere eseguito in tempo reale il tuo programma abbia bisogno di qualche tipo di istruzione `'delay()'` dominante (almeno una volta per `'loop()'`) o un'istruzione che è naturalmente sincronizzata con le modifiche a livello di pin in tempo reale (ad esempio, `'pulseIn()'`, `'shiftIn()'`, `'Serial.read()'`, `'Serial.print()'`, `'Serial.flush()'` ecc.).

Vedi [Modellazione](#) e **suoni** sopra per maggiori dettagli sulle limitazioni.

Note di rilascio

Correzioni di Bug

V2.1- Mar. 2018

- 1) Un bug nelle nuove versioni V2.0.x ha causato la crescita dell'heap di Windows con ogni aggiornamento nel **Area delle Variabili**: dopo milioni di aggiornamenti (molti minuti di esecuzione) potrebbe verificarsi un arresto anomalo.
- 2) Le chiamate di funzione alle funzioni membro `'static'` utilizzando la notazione doppio punto non sono riuscite a Analisare quando si trovano all'interno delle parentesi `'if()'`, `'while()'`, `'for()'` e `'switch()'` e quando le espressioni interne sono usate come argomenti chiamata-funzione o indici di matrici.

V2.0.2- Feb. 2018

- 1) Un bug introdotto nella V2.0 causava un arresto di File | Caricare se un `'#include'` si riferiva a un file mancante o vuoto.

V2.0.1- Gen. 2018

- 1) In localizzazioni in lingua diversa dall'inglese, `'en'` è stato mostrato in modo errato come selezionato in **Preferenze**, facendo ritornare l'inglese in imbarazzo (richiedendo la deselegione e la ri-selezione).
- 2) Era possibile per l'utente lasciare un valore di casella di modifica del pin del dispositivo in uno stato incompleto (come `'A_'`), e lasciare incompleti i bit `'DATA'` di un `'SRSlave'`.
- 3) Il numero massimo di Cursori Analogici era stato limitato a 4 (corretto ora per essere 6).
- 4) UnoArduSim non insiste più su `'='` che appare in un'inizializzazione di aggregazione di matrice.
- 5) UnoArduSim aveva insistito che l'argomento `'inverted_logic'` fosse fornito a `'SoftwareSerial()'`.
- 6) Le operazioni di spostamento bit ora consentono turni più lunghi della dimensione della variabile spostata.

V2.0- Dic. 2017

- 1) Tutte le funzioni dichiarate come `'unsigned'` aveva comunque restituito valori come se fossero `'signed'`. Ciò non ha avuto alcun effetto se il valore `'return'` è stato assegnato a `'unsigned'` variabile, ma avrebbe causato un'interpretazione negativa impropria se avesse `MSB == 1`, ed è stata quindi assegnata a una variabile `'signed'`, o testata in una disuguaglianza.
- 2) Gli Cursori Analogici raggiungevano solo un valore massimo `'analogRead()'` di 1022, non il 1023 corretto.
- 3) Un errore introdotto inavvertitamente nella V1.7.0 nella logica utilizzata per accelerare la gestione del sistema SPI il pin SCK ha causato il trasferimento SPI per `'SPI_MODE1'` e `'SPI_MODE3'` dopo il primo byte trasferito (una transizione SCK extra spuria seguita da ciascun byte). Anche gli aggiornamenti di una casella `'SPISLV'` edit `'DATA'` per i byte trasferiti sono stati ritardati.
- 4) Il dispositivo LED colorato non stava elencando `'B'` (per il blu) come opzione di colore (anche se è stata accettata).
- 5) Le impostazioni per i dispositivi `'SPISLV'` e `'I2CSLV'` non venivano salvate nel file utente **Dispositivi 'I / O'**.

- 6) Copia delle istanze `'Servo'` non riuscite a causa di un'implementazione `'Servo :: Servo (Servo & Tocpy)'` copy-constructor difettosa .
- 7) Fuori intervallo `'Servo.writeMicroseconds()'` I valori sono stati rilevati correttamente come errore, ma i valori limite indicati che accompagnavano il messaggio di errore erano errati.
- 8) Non è stato accettato un baud rate legale di 115200 quando è stato caricato da un file di testo **Dispositivi 'I / O'** .
- 9) I conflitti di pin elettrici causati da un dispositivo Cursore Analogico collegato non sono sempre stati rilevati.
- 10) In rari casi, il passaggio di un puntatore stringa difettoso (con la stringa 0-terminator mancante) a una funzione `'String'` potrebbe causare l'arresto anomalo di UnoArduSim.
- 11) Il **Area del Codice** potrebbe evidenziare la riga di errore di Analisi corrente nel modulo di programma **sbagliato** (quando è stato utilizzato `'#include'`).
- 12) Il caricamento di un file **Dispositivi 'I / O'** con un dispositivo che avrebbe dovuto (erroneamente) guidare contro 'Uno' pin 13 ha causato il blocco di un programma nel pop-up del messaggio di errore.
- 13) UnoArduSim ha erroneamente permesso all'utente di incollare caratteri non esadecimali nelle finestre buffer TX espresse per SPISLV e I2CSLV.
- 14) Inizializzazioni della riga di dichiarazione non riuscite quando il valore del lato destro era il valore `'return'` da una funzione membro-oggetto (come in `'int angle = myservo1.read();'`).
- 15) Le variabili membro `'static'` con i prefissi `'ClassName ::'` espliciti non sono stati riconosciuti se sono apparsi all'inizio di una riga (ad esempio, in un assegnamento a una variabile base `'class'`),
- 16) Chiamando `'delete'` su un puntatore creato da `'new'` è stato riconosciuto solo se è stata utilizzata la notazione parentesi funzione, come in `'delete (pptr)'` .
- 17) L'implementazione di UnoArduSim di `'noTone()'` insisteva erroneamente sull'**opportunità di fornire** un argomento pin.
- 18) Le modifiche che aumentavano i 'RAM' byte globali in un programma che utilizzava `'String'` variables (tramite **Modificare / Esaminare** o **File | Carica**), potevano portare alla corruzione in quello spazio 'Uno' globale a causa dell'eliminazione dell'heap degli oggetti `'String'` che appartenevano al vecchio programma mentre si utilizza (erroneamente) l'heap appartenente al nuovo programma. In alcune circostanze ciò potrebbe causare un arresto anomalo del programma. Sebbene un secondo **Caricare** o **Analizzare** abbia risolto il problema, questo bug è stato adesso corretto.
- 19) I valori di ritorno per `'Wire.endTransmission()'` e `'Wire.requestFrom()'` erano entrambi bloccati a 0 - questi ora sono stati **corretti** .

V1.7.2- Feb.2017

- 1) Anche gli interrupt sul pin 2 venivano attivati (inavvertitamente) dall'attività del segnale sul pin 3 (e viceversa).

V1.7.1- Feb.2017

- 1) Funzione `'delayMicroseconds()'` stava producendo un ritardo in **millisecondi** . (1000 volte troppo grande).
- 2) Il casting esplicito di una variabile `'unsigned'` su un tipo intero più lungo ha prodotto un risultato errato (`'signed'`).
- 3) I valori letterali esadecimali superiori a `0 x7FFF` ora sono `'long'` per definizione e così ora generano `'long'` espressioni aritmetiche risultanti in cui vengono coinvolti.
- 4) Un bug introdotto inavvertitamente dalla V1.7.0 ha impedito il casting in stile C ++ alternativo di valori letterali numerici (ad esempio, `'(long) 1000 * 3000'` non è stato accettato).

- 5) 'Serial' non occupa più i suoi molti byte in 'Uno' RAM se non è mai necessario dal programma utente.
- 6) Le variabili globali dichiarate dall'utente non occupano più spazio in 'Uno' RAM se non vengono effettivamente utilizzate.
- 7) Singole variabili dichiarate come 'const' ,, 'enum' membri e puntatori a stringhe letterali, non occupano più spazio in 'Uno' RAM (per concordare con la compilazione Arduino),
- 8) I byte di RAM necessari per '#includono le librerie incorporate' ora corrispondono strettamente ai risultati della compilazione condizionale Arduino.
- 9) L'uso di 'new' su una riga di dichiarazione effettiva del puntatore non era riuscito (solo una successiva 'new' assegnazione al puntatore funzionava).
- 10) Risolto un bug in cui uno show "in sospeso" di una directory del disco SD poteva causare il blocco di un programma.

V1.7.0- dic.2016

- 0) Sono stati risolti numerosi problemi relativi alla gestione degli interrupt utente:
 - a) Interrompe i bordi 0 e 1 che si sono verificati durante una funzione Arduino che si blocca durante l'attesa (come 'pulseIn()' , 'shiftIn()' , 'spiTransfer()' , 'flush()' e 'write()') aveva causato un errore nel flusso di esecuzione al ritorno dell'interrupt
 - b) copie multiple delle variabili locali di qualsiasi funzione interrotta erano state visualizzate nel **pannello delle variabili** (una copia per interrupt-return) e questo era corretto in V1.6.3, ma rimanevano gli altri problemi di interrupt).
 - c) Funzione 'delayMicroseconds()' non stava creando alcun ritardo se chiamato da una routine di interruzione utente.
 - d) chiede alle funzioni di blocco come 'pulseIn()' **dall'interno** una routine di interrupt non aveva lavorato.
- 1) Un bug introdotto nella V1.6.3 causava la perdita di aggiornamento del valore nel **pannello Variabili** durante l'esecuzione quando i valori effettivamente stavano cambiando (ciò avveniva solo dopo due o più azioni **Arresto** o menu **VarAggiorna** dell'utente). Inoltre, quando è stato eseguito un Esegui Verso dopo l'**attivazione di Allow Reduction** , **occasionalmente il Area delle Variabili** non è stato ridisegnato (quindi i vecchi valori e le variabili locali potrebbero essere apparse lì fino al passaggio successivo).
- 2) Il **Area del Codice** che evidenzia il comportamento del comando Passo Scavalcare potrebbe apparire fuorviante in 'if() - else' catene - che ora è stato corretto (anche se la funzionalità di stepping effettiva era corretta).
- 3) Funzione 'pulseIn()' ha impostato erroneamente il timeout in millisecondi anziché in microsecondi, inoltre ha riavviato in modo errato il timeout quando le transizioni ai livelli inattivo e attivo sono state viste per la prima volta.
- 4) L'utilizzo di valori letterali HEX tra 0x8000 e 0xFFFF in assegnazioni o aritmetiche con le variabili 'long' integer ha dato risultati errati a causa dell'estensione di segno non spuntata.
- 5) Passare, o ritornare, a 'float' da qualsiasi 'unsigned' tipo intero con un valore con MSB = 1 ha dato risultati errati a causa di un'interpretazione 'signed' difettosa .
- 6) Tutte le **funzioni** 'bit _()' ora accettano anche operazioni su **variabili** 'long' - dimensionate, e UnoArduSim verifica le posizioni bit non valide (che non rientrano nelle dimensioni della variabile).

- 7) Un input non valido alla casella di modifica Pulse (width) su 'PULSER' Dispositivo ha causato il danneggiamento del valore 'Period' (finché non viene corretto dall'utente successivo 'Period' edit entry).
- 8) L'eliminazione di un dispositivo 'PULSER' o 'FUNCGEN' utilizzando il menu Configura non rimuoveva il suo segnale periodico dal pin che stava guidando (non è più necessario un Reset).
- 9) **Mancava** la capacità di inizializzare una matrice 1-D 'char' con una stringa quotata (ad es. 'char strg[] = "ciao";').
- 10) La visualizzazione esadecimale nella finestra espansa 'SERIAL' o 'SFTSER' Monitor ha mostrato il carattere non significativo più significativo per i valori di byte superiori a 127.
- 11) Le finestre di Forme d'Onda non riflettevano le modifiche programmatiche dell'utente effettuate da 'analogWrite()' quando il nuovo valore era 0% o 100% duty-cycle.
- 12) L'implementazione di 'Serial.end()' ora è stato corretto.
- 13) Un file **myUnoPrefs.txt** con più di 3 parole su una riga (o spazi nel nome file **Dispositivi 'I / O'**) potrebbe causare un arresto anomalo a causa di un puntatore interno difettoso.
- 14) La riga finale di un file **Dispositivi 'I / O'** non è stata accettata se non si è **conclusa con un line-feed**.
- 15) L'aggiunta di più di quattro cursori analogici causava un bug silenzioso che sovrascriveva i puntatori LED 'I / O'.
- 16) A partire dalla V1.6.0, i campioni di forme d'onda analogiche per la **prima metà** di ogni forma d'onda **triangolare** erano tutti **zero** (a causa di un bug nel calcolo della tabella forme d'onda).
- 17) **Eseguendo un Esequire Verso** ripetuto quando su una linea di punto di arresto non sono più necessari più clic per anticipo.
- 18) Il passaggio di espressioni di indirizzo a un parametro della matrice di funzione non è stato accettato dall'Analizzatore.
- 19) Le funzioni ricorsive che restituivano espressioni contenenti il puntatore o i de-riferimenti di matrice davano risultati non corretti a causa dei flag "pronti" non ripristinati su quelle espressioni componente.
- 20) Chiamare 'class' membro-funzione tramite **qualsiasi variabile puntatore oggetto o espressione puntatore** non funzionava.
- 21) Le funzioni utente che hanno restituito gli oggetti per valore hanno restituito il loro valore alla loro prima chiamata di funzione solo **se** hanno restituito un oggetto costruito senza nome (come 'String ("dog")' - nelle chiamate successive il ritorno è stato saltato a causa di un "pronto" bloccato "bandiera).
- 22) Non c'era stata alcuna salvaguardia per impedire il comando **Finestre | 'Serial' Monitor per** aggiungere un nuovo dispositivo 'Serial' quando in realtà non c'era spazio per questo.
- 23) Se l'aggiunta di un dispositivo a pin fisso (come 'SPISLV') causava un messaggio pop-up di conflitto pin, il ridisegno del **Area di Banco del Lab** poteva mostrare un duplicato dispositivo "fantasma" che sovrapponeva il dispositivo 'I / O' alla destra (fino al successivo ridisegno).
- 24) Risolti alcuni problemi con suoni inaffidabili 'PIEZO' per segnali pin non periodici.

25) '**PROGMEM**' variabili devono ora essere esplicitamente dichiarate come '**const**' per concordare con Arduino.

26) "Nessuno spazio dell'heap" è stato erroneamente contrassegnato come errore di esecuzione quando un '**SD.open()**' non ha trovato il file con nome, oppure un '**openNextFile()**' ha raggiunto l'ultimo file nella directory.

27) Un bug di **Analizzatore** ha accettato impropriamente una parentesi graffa di chiusura fuori dal luogo '**}**'.

28) È stato corretto un bug relativo alle rimozioni delle rimozioni delle **variabili** sul ritorno del costruttore membro-oggetto (il bug veniva applicato solo per gli oggetti che a loro volta contengono altri oggetti come membri).

V1.6.3- Settembre 2016

1) Le variabili locali di qualsiasi funzione interrotta non venivano rimosse dal **Area delle Variabili** sulla voce della funzione di interruzione, causando la visualizzazione di più copie sul ripristino della funzione di interruzione (e un eventuale errore di esecuzione o un arresto anomalo).

2) Le finestre di Forme d'Onda non riflettevano le modifiche programmatiche in '**analogWrite()**' a un nuovo ciclo di lavoro di 0% o 100%.

3) La visualizzazione esadecimale nella finestra espansa 'SERIAL' o 'SFTSER' Monitor ha mostrato il carattere MSB errato per valori di byte superiori a 127.

V1.6.2- Settembre 2016

1) Le chiamate di funzione eseguite con il numero o il tipo di argomenti errati non avevano generato un messaggio di errore di Analisi appropriato (appariva solo il messaggio generico di "non un identificativo valido").

2) Il pulsante di reset **Tool-Bar** ora funziona in modo identico al pulsante di reset 'Uno' board.

3) Il testo di errore di errore non viene più disattivato dopo 16 caratteri senza mostrare i puntini di sospensione.

V1.6.1- agosto 2016

1) Nella versione V1.6 una 'Uno' board nel file **myArduPrefs.txt** che differiva dal valore predefinito della versione 2 ha causato un'eccezione all'avvio (a causa di un evento pin non inizializzato 13).

2) La modifica del valore di una variabile facendo doppio clic nel **Area delle Variabili** potrebbe causare errori di errore (nessun allocazione di memoria) pop-up (per programmi con qualsiasi '**class** definito dall'utente').

3) '**SoftwareSerial**' non ha consentito l'accesso a '**write (char * ptr)**' e '**write (byte * ptr, int size)**' funzioni a causa di un rilevamento di sovraccarico della funzione difettoso.

4) Risolto il problema con l'inclusione automatica del corrispondente file ".cpp" per un isolato ".h" tipo-libreria "**#include**".

V1.6 - Giugno 2016

1) In V1.5 il rientro automatico sul tasto "Invio" in **Modificare / Esaminare** (quando si immette una nuova riga) era stato perso.

2) Il rilevamento di conflitti di pin con dispositivi esterni a conduzione diretta 'I / O' è ora stato aggiunto su 'Serial' pin 1, su SPI pin SS, MOSI e SCK, su pin I2C SCL e SDA (tutto quando il corrispondente 'begin()' viene chiamato) e su qualsiasi pin 'SoftwareSerial' TX dichiarato .

V1.5.1- giugno 2016

1) In V1.5 i nuovi colori di Sintassi adattabile tema-adattabili non sono stati reimpostati correttamente ogni volta che **Modificare / Esaminare** è stata aperta, e quindi (con un tema sfondo bianco) erano solo corretti ogni due volte.

2) Le interruzioni 'RISING' e 'FALLING' erano opposte all'effettiva polarità del fronte di innesco.

V1.5 - Maggio 2016

1) Un bug introdotto in V1.4.1 impediva il passaggio di valori letterali di stringa nuda a funzioni membro che prevedevano un oggetto 'String' , come in 'mystring1.startsWith ("Ehi")' .

2) Un bug nella SD originale l'implementazione di UnoArduSim consentiva l' accesso SD solo tramite chiamate a 'read()' e 'write()' (l'accesso tramite le funzioni 'Stream' è stato impedito).

3) 'R = 1K' Gli interruttori a scorrimento non venivano ridisegnati correttamente quando veniva spostato il cursore.

4) "Annulla" nella finestra di dialogo **Conferma-Salva** file dovrebbe aver impedito l'uscita dall'applicazione.

5) Una citazione di chiusura mancante o la chiusura di '>' -parenthesis su un file utente '#include' causerebbe un blocco.

6) Risolto un bug nell'evidenziazione della sintassi di 'String' e user 'class' o 'struct' , e l'evidenziazione estesa includeva le chiamate alle funzioni del costruttore.

7) Risolti alcuni problemi minori in **Modificare / Esaminare** con modifiche / evidenziazioni del testo e il pulsante Annulla.

V1.4.3 - aprile 2016

1) Utilizzare **Configurare | I / O | I dispositivi** per aggiungere nuovi dispositivi e quindi per rimuovere successivamente uno di questi dispositivi appena aggiunti potrebbero causare un arresto anomalo al ripristino o il blocco di un altro dispositivo.

2) Modifica di una variabile 'String' facendo doppio clic nel **Area delle Variabili** non riusciva (la nuova 'String' è stata letta in modo errato).

3) Le modifiche dei pin sui dispositivi 'FUNCGEN' e 'PULSER' non sono state riconosciute fino a quando non è stato effettuato prima un reset.

V1.4.2 - marzo 2016

1) La V1.4.1 aveva introdotto uno sfortunato bug di Analizzatore che impediva l'assegnazione di oggetti 'class' (incluso 'String').

2) Una correzione di bug incompleta eseguita in V1.4.1 ha causato 'unsigned' values di tipo 'char' per stampare come caratteri ASCII piuttosto che come valori interi.

3) Gli argomenti di chiamata di funzione di espressione membro complessa non venivano sempre riconosciuti come corrispondenze di parametri di funzione valide.

4) **Tutti i** valori letterali e le espressioni interi sono stati dimensionati in modo troppo generoso (per `'long'`) e pertanto l'esecuzione non riflette gli overflow **effettivi** (negativi) che possono verificarsi in Arduino in operazioni di aggiunta / moltiplicazione che coinvolgono i valori `'int'`.

5) Le espressioni che coinvolgono un mix di `'signed'` e `'unsigned'` integer types non sono sempre state gestite correttamente (il valore `'signed'` sarebbe stato visto impropriamente come `'unsigned'`).

6) Nei casi di conflitto di pin, i messaggi di errore "value =" potrebbero mostrare valori di pin obsoleti anche dopo un Reset da un conflitto precedente che l'utente aveva già eliminato.

V1.4.1 - Gennaio 2016

1) Le chiamate a `'print (char)'` ora vengono stampate correttamente come caratteri ASCII (anziché valori numerici).

2) La risposta agli interrupt è ora abilitata di default quando viene chiamato `'attachInterrupt()'`, quindi non c'è più bisogno di `'setup()'` per chiamare la funzione di abilitazione `'interrupts()'`.

3) Multiple `'#include'` le istanze di file utente all'interno di un file vengono ora gestite correttamente.

V1.4 - Dicembre 2015

1) Un bug di vecchia **data ha** erroneamente contrassegnato una condizione di **divisione per zero al** momento della divisione per un valore frazionario inferiore all'unità.

2) Risolto `'SoftwareSerial'` (che è stato inavvertitamente interrotto da un `'class'` aggiunto - verifica di convalida dei membri nelle versioni V1.3).

3) Le chiamate di fine linea con un punto e virgola mancante non sono state catturate e hanno causato l'errore di ignorare la riga successiva.

4) Un file di testo **'I / O'** formattato male formattato ha dato un messaggio di errore improprio.

5) Errore L'evidenziazione dell'errore della riga errata (adiacente) in espressioni e istruzioni multilinea è stata corretta

6) Test logici di puntatori utilizzando `'not' (!)` l'operatore è stato invertito.

V1.3 - ottobre 2015

1) La gestione interna impropria delle variabili del blocco scratch ha causato occasionalmente il "**superamento della massima profondità di annidamento del scratchpad**". Errori di analisi.

2) Parentesi *all'interno di virgolette singole*, parentesi graffe, punto e virgola, parentesi all'interno di stringhe tra virgolette e caratteri di \-sfuggito sono stati gestiti in modo improprio.

3) Una matrice con una dimensione vuota e nessun elenco di inizializzazione ha causato un blocco RESET e gli array con un solo elemento non sono stati disabilitati (e hanno causato la loro interpretazione errata come un puntatore inizializzato non valido).

4) A volte gli errori di analisi a volte evidenziavano la linea sbagliata (adiacente).

5) Passare un puntatore a un non `'const'` a una funzione che accetta un puntatore a `'const'` non è stata consentita (anziché viceversa).

- 6) Le espressioni di inizializzazione ereditavano **erroneamente** i qualificatori '**PROGMEM**' dalla variabile inizializzata
- 7) '**PROGMEM**' le variabili dichiarate hanno avuto il loro byte-size erroneamente contato **due volte** rispetto alla loro allocazione di memoria 'Flash' durante l'analisi.
- 8) Digitando 'Send' casella di modifica di 'I2CSLV' a volte **causava** un arresto a causa di '**sscanf**' bug.
- 9) Il caricamento di un nuovo programma con un nuovo file Dispositivi 'I / O' nella sua directory potrebbe causare conflitti di pin irrilevanti con le **vecchie** direzioni dei pin.
- 10) La gestione dei caratteri seriali di \-sfuggito è stata applicata impropriamente alle sequenze di caratteri ricevute, piuttosto che trasmesse, nella (più grande) finestra dei buffer 'Serial Monitor'.
- 11) '**while()**' e '**for()**' loop con corpi completamente vuoti, come '**while (true);**' o '**for (int k = 1; k <= 100; k ++);**' passato il Analisi (con un messaggio di avviso) ma non è riuscito al momento dell'esecuzione ..

V1.2 - Giugno 2015

- 1) La più semplice delle funzioni utente che ha effettuato chiamate a '**digitalRead()**' o '**analogRead()**' o '**bit()**' potrebbe aver corrotto la (molto prima) variabile locale dichiarata (se presente) a causa di una funzione allocata insufficiente spazio (se solo due blocchi scratchpad sono stati assegnati all'inizio dello stack di funzioni). Qualsiasi espressione numerica all'interno di una funzione è sufficiente per causare un'assegnazione dello scratchpad di 4 byte e pertanto evita questo problema. Questo sfortunato bug è stato in circolazione dalla versione originale V1.0.
- 2) Le funzioni che sono '**void**' con le prime **funzioni** esplicite '**return**' e non- '**void**' con più di una '**return**', vedrebbero l'esecuzione in cascata alla **parentesi graffa di chiusura** (se è stata raggiunta).
- 3) Qualsiasi '**return**' istruzioni all'interno '**if()**' contesti che mancavano parentesi graffe hanno portato ad un target difettosa ritorno al chiamante.
- 4) 'PULSER' e 'FUNCGEN' ampiezze di impulso o periodi di valore 0 potrebbero causare un arresto anomalo (0 non è consentito).
- 5) Dove non c'erano parentesi graffe, '**else**' continue dopo un '**if()**' non ha funzionato se hanno seguito '**break**', '**continue**' o '**return**'.
- 6) Quando sono state fatte più '**enum**' dichiarazioni utente, le costanti definite in tutto tranne il primissimo '**enum**' generato difettoso " '**enum**' mismatch" Errori di Analisi (questo errore è stato introdotto in V1.1).
- 7) Un identificatore nullo per l'ultimo parametro di un prototipo di funzione ha causato un errore di Analisi.
- 8) I punto di arresto **Esegui Verso** impostati su linee complesse non sono sempre stati gestiti correttamente (e quindi potrebbero essere persi).
- 9) '**HardwareSerial**' e '**SoftwareSerial**' ha utilizzato un buffer in attesa di implementazione privata che non è stato ripulito in Reset (così potrebbero apparire i caratteri rimanenti dell'ultima volta).
- 10) Il Analisi non è riuscito a controllare il bit-flip **errato** di '**float**' e l'aritmetica del puntatore ha tentato con operatori illegali.

V1.1 - Marzo 2015

- 1) Gli indici di array che erano **variabili** '**byte**' o '**char**' causavano offset di array errati (se una variabile adiacente conteneva un byte alto non-0).

- 2) Il test logico dei puntatori ha testato il valore puntato per il non zero anziché il valore del puntatore stesso.
- 3) Qualsiasi istruzione `'return'` incorporata nei loop `'for()'` o `'while()'` è stata gestita in modo errato.
- 4) Gli elenchi di inizializzazione aggregati per le matrici di oggetti o gli oggetti contenenti altri oggetti / matrici o gli elenchi di inizializzazione completamente vuoti non venivano gestiti correttamente.
- 5) Accesso ai valori dei membri `'enum'` utilizzando un `"enumname"`. il prefisso non era supportato.
- 6) L'inizializzazione della riga di dichiarazione di una matrice `'char[]'` con una stringa letterale quotata non funzionava.
- 7) Una matrice passata a una funzione senza un'inizializzazione precedente è stato contrassegnato in modo errato con un errore "usato ma non inizializzato".
- 8) Le espressioni di puntatore che coinvolgono nomi di array erano mal gestite.
- 9) I parametri di funzione dichiarati come `'const'` non sono stati accettati.
- 10) La finestra Forma d'Onda Analogica del perno non mostrava i segnali PWM (`'servo.write()'` e `'analogWrite()'`).
- 11) Le funzioni membro accessibili tramite un puntatore oggetto hanno fornito accessi membri difettosi.
- 12) Le forme d'onda non venivano aggiornate quando veniva raggiunto un punto di arresto **Esegui Verso**.
- 13) La modellazione dell'allocazione del registro poteva fallire quando un parametro funzione veniva usato direttamente come argomento per un'altra chiamata di funzione

V1.0.2 - Agosto 2014

Ordine fisso dei pin A0-A5 sul perimetro della scheda 'Uno' .

V1.0.1 - Giugno 2014

Risolto un bug che tagliava le paste di modifica che erano più lunghe di tre volte il numero di byte nel programma originale.

V1.0 - prima uscita maggio 2014

Modifiche / Miglioramenti

V2.1- Mar. 2018

- 1) I valori visualizzati nel **Area delle Variabili** vengono ora aggiornati a una velocità di 16 volte al secondo o 4 volte al secondo (quando è attiva l'opzione VarAggiorna | Minimo) e l'opzione VarAggiorna | Consenti la Riduzione è stata rimossa.
- 2) L'operazione che ha come target solo una parte dei byte del valore di una variabile (come quelli creati tramite i puntatori) ora fa sì che la modifica al valore di tale variabile si rifletta nella visualizzazione del **Area delle Variabili**.

V2.0.1- Gen. 2018

- 1) Le funzioni Arduino non documentate '**exp()**' e '**log()**' sono state ora aggiunte.
- 2) Ora i dispositivi 'SERVO' possono essere fatti a rotazione continua (quindi la larghezza degli impulsi controlla la velocità anziché l'angolo).
- 3) In **Modificare / Esaminare** una parentesi di chiusura '}' viene ora aggiunta automaticamente quando si digita una parentesi di apertura '{'.
- 4) Se si fa clic sulla '**X**' di barra del titolo della finestra di **Modificare / Esaminare** per uscire, si ha ora la possibilità di interrompere se è stato modificato, ma non salvato, il file di programma visualizzato.

V2.0 settembre 2017

- 1) L'implementazione è stata trasferita a QtCreator in modo che la GUI abbia alcune piccole differenze visive, ma nessuna differenza funzionale oltre a qualche miglioramento:
 - a) La messaggistica sulla riga di stato nella parte inferiore della finestra principale e all'interno della finestra di dialogo **Modificare / Esaminare** è stata migliorata e è stata aggiunta una codifica colore di evidenziazione.
 - b) Lo spazio verticale allocato tra il **Area di Codice** e il **Area delle Variabili** è ora regolabile tramite una barra di separazione trascinabile (ma non visibile) al loro bordo condiviso.
 - c) 'I / O' i valori della casella di modifica del dispositivo vengono ora convalidati solo dopo che l'utente ha spostato il puntatore del mouse all'esterno del dispositivo - questo evita modifiche automatiche per applicare i valori legali mentre l'utente sta digitando.
- 2) UnoArduSim ora supporta più lingue tramite **Configurare | Preferenze**. L'inglese può sempre essere selezionato, oltre alla lingua per le impostazioni locali dell'utente (purché nella cartella UnoArduSim 'translations' sia presente un file di traduzione personalizzato *.qm per quella lingua).
- 3) Il suono ora è stato modificato per utilizzare l'API audio Qt - questo ha richiesto il silenziamento in determinate circostanze al fine di evitare fastidiosi rotture del suono e clic durante i lunghi ritardi operativi della finestra del sistema operativo causati dai normali clic del mouse dell'utente - vedere la sezione Modellazione-Suoni per ulteriori dettagli su questo.
- 3) Per comodità dell'utente, gli spazi vuoti vengono ora utilizzati per rappresentare un valore 0 nelle caselle di modifica del conteggio del dispositivo in **Configurare | Dispositivos 'I / O'** (così ora puoi usare la barra spaziatrice per rimuovere i dispositivi).
- 5) Il qualificatore non ridimensionato (U) è ora facoltativo sui dispositivi 'PULSER', 'FUNCGEN' e '1SHOT' (è l'impostazione predefinita presunta).
- 6) UnoArduSim ora consente (oltre ai valori numerici letterali) '**const**' variabili con valore intero e '**enum**' membri, da utilizzare come dimensioni nelle dichiarazioni di matrice, a condizione che tali origini vengano inizializzate in modo esplicito utilizzando valori letterali numerici con valori interi costanti.

- 7) Dopo che il mouse è entrato per la prima volta, un dispositivo PUSH ora produrrà il rimbalzo dei contatti (per 1 millisecondo) ad ogni pressione del tasto della **barra spaziatrice** (ma **non** per i clic del mouse, né per qualsiasi altro tasto premuto).
- 8) Clic aggiuntivi di **Trova | Imposta teTesto di Ricerca** ora seleziona il **successivo** parola dal testo della riga evidenziata nel **Area** attivo (il **Area di Codice** o il **Area delle Variabili**).
- 9) Funzioni '**max()**' e '**pow()**' sono ora inclusi nella comodità elenco di incassato all'interno della finestra di dialogo **Modificare / Esaminare** .
- 10) Il comando '**goto**' Arduino è ora contrassegnato come "non supportato" in UnoArduSim.

V1.7.2 - Febbraio 2017

- 1) La scelta del colore blu (B) è stata aggiunta per i dispositivi LED.

V1.7.1 - Febbraio 2017

- 1) Suffissi '**l**' e / o '**u**' sono ora accettati alla fine delle costanti numeriche letterali (per definirli come '**long**' e / o '**unsigned**') e le costanti binarie (**0b** o **0B** prefissate) sono ora accettate. Ogni costante numerica ogni decimale che **inizia con** '**0**' è ora considerata un valore **ottale** . (per concordare con Arduino).
- 2) Quando si esegue un ciclo stretto da cui non esiste alcuna fuga (ad esempio '**while (x); x ++;**' dove **x** è sempre vero), facendo clic su Arresto una seconda volta si garantisce che l'esecuzione del programma si fermi effettivamente (e su quella riga di programma difettosa) .

V1.7.0- dic.2016

- 1) È stata aggiunta una nuova funzione **Tool-Bar** che mostra i RAM liberi durante l'esecuzione del programma (tenendo conto dei byte totali utilizzati dalle variabili globali, dalle allocazioni dell'heap e dalle variabili dello stack locale).
- 2) Le funzioni di interruzione utente possono ora anche chiamare blocchi di funzioni Arduino come '**pulseIn()**' (ma questo dovrebbe essere usato solo con cautela, poiché la funzione di interruzione non ritorna finché la funzione di blocco non è completa).
- 3) Gli interrupt utente non vengono più disabilitati durante le operazioni di lettura stream bloccate, quindi il comportamento ora corrisponde all'effettiva operazione di lettura dei flussi di Arduino.
- 4) Ora puoi entrare e uscire dal blocco delle funzioni di Arduino che possono essere interrotte (come '**delay()**' e '**pulseIn()**') e i messaggi della Narra di Stato sono stati aumentati per mostrare quando si è raggiunto un punto di interruzione dell'interrupt all'interno di tale funzione (o quando si fa clic-Alt quando l'esecuzione è attualmente all'interno di tale funzione).
- 5) È stato aggiunto un nuovo comando **Esegui-Fino-A** (e **una Barra degli Strumenti**.): un singolo clic su qualsiasi **variabile del Area delle Variabili** (può essere semplice, una matrice o un oggetto aggregato o un elemento di matrice o un membro dell'oggetto) per evidenziarlo, quindi **esegui Esegui-Fino-A** : l'esecuzione si bloccherà al successivo **accesso** in **scrittura** all'interno di quella variabile aggregata o in quella singola posizione.
- 6) Quando l'esecuzione si blocca dopo un'azione **Passo** , **Esegui-Verso** , **Esegui-Fino-A** o **Esegui** -poi- **Arresto** , il **Area delle Variabili** ora evidenzia la variabile che corrisponde alla / e posizione / i dell'indirizzo che è stata modificata (se esiste) dall'ultima istruzione durante quell'esecuzione - se quella posizione è attualmente nascosta all'interno di una matrice o di un oggetto non espanso, facendo clic per espanderlo farà in modo che l'ultimo elemento modificato o membro venga evidenziato.

7) L'utente può ora tenere un orologio speciale sul valore di una specifica variabile Variabile / membro / elemento durante l'esecuzione - fare doppio clic su quella linea nel **Area delle Variabili** per aprire la finestra **Modificare / Monitorare Variabile** , quindi eseguire una delle **Esegui** o Comandi **passo** : il valore visualizzato verrà aggiornato durante l'esecuzione in base alle stesse regole che governano gli aggiornamenti nel **Area delle Variabili** . Dopo aver interrotto l'esecuzione, è possibile immettere un nuovo valore e **accettarlo** prima di riprendere l'esecuzione (e può **tornare** al valore pre- **Accept** se si cambia idea prima).

8) I tasti di accelerazione F4-F10 sono stati impostati per corrispondere ai comandi della **Barra degli Strumenti**.del menu **Esegui** (da sinistra a destra).

9) Oltre a fare doppio clic su di essi, facendo clic con il pulsante destro del mouse su 'SERIAL', 'SFTSER', 'SPISLV', 'I2CSLV' appariranno ora anche una finestra di byte / caratteri TX / RX di grandi dimensioni (e su 'SD_DRV', una finestra di monitoraggio dei file).

10) La casella di modifica TX in 'SERIAL' o 'SFTSER' non è più disabilitata durante una trasmissione di caratteri attiva (quindi è ora possibile aggiungere o sostituire ciò che è presente), ma un ritorno a capo (o il tasto 'Send' fare clic nel associata alla finestra 'SerialMonitor' child) verrà ignorata fino a quando la trasmissione non tornerà di nuovo in stato di riposo (i caratteri ora sono in *corsivo* quando la trasmissione è pronta per iniziare, o è attiva). Inoltre, l'utente è ora avvisato su un flusso seriale '**begin()**' se avessero già avviato in precedenza le trasmissioni del dispositivo collegato (ora in corso), poiché non ci sarebbe quindi alcuna sincronizzazione di framing, portando a errori di ricezione.

11) Il ritardo predefinito '**loop()**' delay è stato aumentato da 250 microsecondi a un millisecondo, in modo da non cadere molto indietro rispetto al tempo reale quando l'utente trascura di includere qualche '**delay()**' (esplicito o naturale) da qualche parte all'interno di '**loop()**' o all'interno di una funzione che chiama.

12) Array e tipi semplici sono ora stati aggiunti al supporto per l'istruzione heap-allocating '**new**' .

13) Verifiche più estese (e messaggi di errore associati) sono state aggiunte per gli accessi agli indirizzi fuori campo del programma utente (cioè al di fuori di 'Uno' RAM, o al di fuori di 'Flash' per '**PROGMEM**' **accesses**).

14) I valori del puntatore nel **Area delle Variabili** ora assomigliano più strettamente ai valori attuali del puntatore Arduino.

15) Il file **myArduPrefs.txt** dell'utente è ora caricato in ogni **File | Carica** , non solo al lancio di UnoArduSim.

16) Un errore di Analisi è ora contrassegnato quando si tenta di '**attachInterrupt()**' a una funzione utente che non è '**void**' ritorno, o che ha parametri di funzione, o che non è stato dichiarato da qualche parte prima di '**attachInterrupt()**' .

17) '**static**' membro-variabili sono ora visualizzate nella parte superiore del **Area delle Variabili** come globals, anziché apparire all'interno di ciascuna istanza di un oggetto (espanso).

18) Funzione '**availableForWrite()**' è stato aggiunto all'implementazione di **Serial** .

19) Tutti gli speciali '**PROGMEM**' , '**typedef**' like '**prog_char**' e '**prog_int16**' ora sono stati rimossi (sono stati deprecati in Arduino).

20) Messaggi di errore migliorati per errori di analisi causati da tipi di dichiarazione errati o non validi.

21) La dimensione massima consentita del programma è stata aumentata.

V1.6.3- Settembre 2016

1) Aggiunto un messaggio di errore di analisi migliorato quando `'attachInterrupt()'` si riferisce a una funzione di interruzione che non era stata **prototipata in precedenza**.

2) Aggiunto un messaggio di errore di Analisi migliorato per gli elenchi di inizializzazione di array multidimensionali.

V1.6.2- Settembre 2016

1) Aggiunto un controllo di modifica del **Trova-Testo** alla **Barra degli Strumenti**, per semplificare la ricerca del testo (nel **Area del Codice** e nel **Area delle Variabili**).

2) Il pulsante Ripristina **Barra degli Strumenti**, ora funziona in modo identico al pulsante di reset 'Uno' board.

V1.6.1- Ago 2016

Aggiunto un controllo per evitare di caricare duplicati e l'analisi di `'già precedente#'` **includere file**,.

V1.6 - Giugno 2016

1) Aggiunto un nuovo '1SHOT' (Un-Colpo) dispositivo 'I / O' che genera un impulso dopo un ritardo scelto da un fronte del segnale di trigger della polarità selezionata.

2) Aggiunta una nuova funzione per rendere i valori 'I / O' casella di modifica del dispositivo facilmente **scalabili** durante l'esecuzione trascinando un dispositivo di scorrimento global 'I / O _____ S' Scaler sulla Barra degli Strumenti, principale (basta digitare una singola lettera 's' o 'S' dopo un valore per indicare il ridimensionamento).

V1.5.1 - Giugno 2016

1) Il supporto è stato ora aggiunto per le funzioni della libreria EEPROM `'update()'`, `'put()'` e `'get()'`, e per l'accesso ai byte tramite notazione array, ad es. `'EEPROM [k]'`.

2) **Consenti Auto(-) Collasso** è stata aggiunta al menu **VarAggiorna** per consentire il controllo esplicito sull'eventuale auto-compressione di array / oggetti espansi quando l'esecuzione è in ritardo rispetto al tempo reale.

3) I personaggi di a `'String'` ora è possibile accedere anche alla variabile tramite notazione array, per esempio `'mystring [k]'`.

V1.5 - Maggio 2016

1) **Visualizza / Modifica** ora ha una scorciatoia da tastiera ctrl-E, e ha un nuovo pulsante per **Compilare** (ctrl-R), più una casella di errore di Analisi incassato, per consentire il test delle modifiche senza dover chiudere la finestra.

2) **Visualizza / Modifica** ora ora supporta anche **Rifare**, e ha un nuovo pulsante **Salva** (ctrl-S) (equivalente a **Accetta** più una successiva finestra principale **Salva**), e ora offre una scelta di dimensioni TAB (una nuova preferenza che può essere salvata usando **Configurare | Preferenze**).

3) Tutte le caselle di modifica scrivibili ora seguono i colori del tema del sistema operativo Windows scelti e, per contrasto, tutte le finestre di sola lettura 'RECV' edit utilizzano il testo bianco su sfondo nero. I **colori di modifica dello sfondo e della sintassi** Modificare / Esaminare **ora si adattano anche al tema scelto**.

4) UnoArduSim ora consente una scelta di font: quella scelta e le dimensioni sono state spostate in **Configurare | Preferenze** (quindi può essere salvato nel file **myArduPrefs.txt**).

5) I valori letterali binari predefiniti di Arduino (come `'B01011011'`) ora sono consentiti.

6) Le sequenze di caratteri quotate in virgola mobile esadecimale, ottale e a 4 cifre possono ora essere utilizzate come valori letterali numerici.

7) Dopo aver fatto un clic iniziale del mouse su un pulsante del dispositivo 'PUSH', l'utente può quindi utilizzare un tasto premuto (qualsiasi tasto) per premere i contatti del pulsante.

8) **Modificare / Esaminare** ora rilascia il suo stato temporaneo di sola lettura iniziale (e rimuove l'evidenziazione della linea selezionata iniziale) dopo un breve segnale di flash visivo.

9) UnoArduSim ora verifica la presenza di più **conflitti** `'Stepper'` e `'Servo'` pin, ovvero il programma utente difettoso tenta di collegarsi ai pin già collegati alle variabili precedenti `'Stepper'` o `'Servo'` .

10) Un errore di errore causato da un mancino o da un lato destro di un operatore (manca un'espressione o una variabile LHS o RHS) genera ora un messaggio di errore chiaro.

11) La **variabile** inutilizzata `'String'` class `'flags'` membro è stata rimossa per concordare con Arduino V1.6.6. Un oggetto `'String'` ora occupa 6 byte (più l'allocazione dell'heap dei caratteri).

V1.4.2 - Mar 2016

1) Le funzioni definite in avanti (cioè quelle senza nessuna dichiarazione di prototipo prima della prima chiamata) ora generano solo avvertimenti (non errori di analisi) quando la definizione di funzione successiva restituisce un tipo non corrispondente al tipo desunto dal loro primo utilizzo.

2) Le matrici con una dimensione uguale a 1 non vengono più rifiutate (per concordare con le regole C ++ standard).

3) le caselle di modifica non sono più impostate su sfondo nero su sfondo bianco; ora adottano la tavolozza impostata dal tema del sistema operativo Windows in uso.

4) `'SERIAL'`, `'SFTSER'`, `'SPISLV'` e `'I2CSLV'` dispositivo espanso Le finestre di monitoraggio (aperte facendo doppio clic) ora adottano il colore di sfondo del loro genitore dispositivo `'I / O'`.

V1.4 - Dic 2015

1) Sono state aggiunte le funzionalità `'Stepper.h'` library e i dispositivi associati `'I / O'`.

2) Tutte le impostazioni e i valori di dispositivo `'I / O'` (in aggiunta ai pin selezionati) vengono ora salvati come parte del file di testo user Dispositivi `'I / O'` selezionato per il successivo ricaricamento.

3) **LED** `'I / O'` colore del dispositivo può ora essere impostato come rosso, giallo o verde utilizzando una casella di modifica sul dispositivo.

4) Gli inizializzatori di dichiarazione variabile ora possono estendersi su più righe.

5) Gli indici di array ora possono essere essi stessi elementi di array.

6) **Configurare |** Le preferenze ora includono una casella di controllo che consente di utilizzare `'and'` , `'or'` , `'not'` parole chiave da utilizzare al posto dello C standard `&&` , `||` e `!` operatori logici.

7) "Mostra Scaricamento Programmi" è stato spostato in **Configurare | Preferenze**

V1.3 - Ottobre 2015

1) Il dispositivo **'PUSH'** ora ha una casella di controllo "pulsante-simile" con l'etichetta 'latch' per renderli "latching" (anziché "momentary"), ovvero si bloccheranno nella posizione chiusa (e cambieranno colore) quando premuto, fino a quando non vengono premuti di nuovo per rilasciare i contatti.

2) Funzionalità complete 'SPISLV' sono state aggiunte con la selezione del nodo (**'MODE0'** , **'MODE1'** , **'MODE2'** o **'MODE3'**). Facendo doppio clic si apre una finestra dei buffer TX / RX in cui possono essere definiti i prossimi REPLY (TX) byte e per la visualizzazione dei byte ricevuti (RX) passati. Il semplice dispositivo slave a registro a scorrimento della versione precedente è stato rinominato per diventare un dispositivo 'SRSLV'.

3) Ora è possibile scegliere il carattere **grassetto** per il **Area del Codice** e il **Area delle Variabili** (dal menu **Opzioni**) e **evidenziare in grassetto parole chiave e operatori** ora può essere attivato / disattivato in **Modificare / Esaminare** .

4) UnoArduSim ora consente a **'bool'** come sinonimo di **'boolean'** .

5) Per chiarezza nella segnalazione degli errori, le dichiarazioni delle variabili non sono più consentite per estendersi su più righe (eccetto per gli array che hanno elenchi di inizializzatori).

6) La velocità della colorazione della sintassi in **Visualizza / Modifica** è stata migliorata (questo sarà evidente con programmi più grandi).

7) Un overhead opzionale di 200 microsecondi (nel menu **Opzioni**) è stato aggiunto a ogni chiamata di **'loop ()'** - questo è per cercare di evitare di cadere troppo indietro rispetto al tempo reale nel caso in cui il programma utente non abbia aggiunto **'delay ()'** ovunque (vedere la discussione sulla tempistica sotto **Modellazione**).

Versione 1.2 giugno 2015

1) La libreria SD è ora completamente implementata ed è stato aggiunto un (piccolo) dispositivo SD Disk 'I / O' ('SD_DRV') (e funzionalità testata su tutti i programmi SD di esempio di Arduino).

2) Come Arduino, UnoArduSim convertirà automaticamente un argomento di funzione nel suo indirizzo quando chiama una funzione in attesa del passaggio di un puntatore.

3) I messaggi di errore di analisi sono ora più appropriati quando mancano i punti e virgola e dopo le dichiarazioni non riconosciute.

4) Stale linea **Area delle Variabili** evidenzia ora vengono rimossi in funzione di chiamata / ritorno.

V1.1 - Marzo 2015

1) Ora la finestra principale può essere ingrandita o ridimensionata per ampliare il **Area del Codice** e il **Area delle Variabili** (per schermi più grandi).

2) È stato aggiunto un nuovo menu Trova (con i **pulsanti Barra degli Strumenti**.) per consentire una navigazione più rapida nel **Area del Codice** e nei **riquadri delle variabili** (PgUp e PgDown o ricerca testo con freccia su, freccia giù).

3) La finestra **Visualizza / Modifica** ora consente i salti di navigazione ctrl-PgUp e ctrl-PgDn (alla riga vuota successiva) e ha la funzionalità **Trova / Sostituisci** aumentata .

4) Un nuovo elemento è stato aggiunto al menu **VarAggiorna** per consentire all'utente di selezionare un approccio di risparmio di calcolo in caso di carichi di aggiornamento del pesante **pannello Variabili** .

5) 'Uno' pin e LED collegato riflettono ora eventuali modifiche apportate ai dispositivi 'I / O' anche quando il tempo è congelato (ovvero, anche quando l'esecuzione viene interrotta).

6) Altre funzioni utente possono ora essere richiamate da una funzione di interruzione utente (in conformità con l'aggiornamento ad Arduino 1.06).

7) Ora è possibile scegliere un **carattere più grande** dal menu **Opzioni** .

V1.0.1 - Giugno 2014

Le finestre delle **forme d'onda** ora etichettano i pin analogici come A0-A5 anziché 14-19.

V1.0 - prima uscita maggio 2014