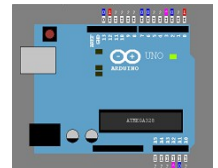


UnoArduSimV2.7 Aiuto Completo



Sommario

Panoramica

Area di Codice, Preferenze e Modificare/Esaminare

Area di Codice

Preferenze

Modificare/Esaminare

Area delle Variabili e Modificare/Monitorare Variabile finestra

Area del Banco da Laboratorio

Il 'Uno'

'I/O' Dispositivi

'Serial' Monitor ('SERIAL')

Seriale Software ('SFTSER')

Unità Disco SD ('SD_DRV')

Display TFT ('TFT')

Configurabile SPI Slchiavo ('SPISLV')

Due fili I2C Slchiavo ('I2CSLV')

LCD di Testo I2C ('LCDI2C')

LCD di Testo SPI ('LCDSPI')

LCD di Testo D4 ('LCD_D4')

Multiplexer DEL I2C ('MUXI2C')

Multiplexer DEL SPI ('MUXSPI')

Porta di Espansione SPI ('EXPSPi')

Porta di Espansione I2C ('EXPI2C')

1-Wire' Slchiavo ('OWIISLV')

Registro di Cambio Slchiavo ('SRSLV')

Impulso Singolo ('1SHOT')

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

Generatore di Impulsi ('PULSER')

Analogico Generatore di Funzioni ('FUNCGEN')

Motore Passo Passo ('STEPR')

Pulsata Motore Passo Passo ('PSTEPR')

DC Motore ('MOTOR')

ServoMotore ('SERVO')

Altoparlante Piezoelettrico ('PIEZO')

Resistenza di scorrimento ('R=1K')

Pulsante ('PUSH')

DEL colorato ('LED')

4-DEL Row ('LED4')

7 segmenti DEL Cifra ('7SEG')

Cursore Analogico

Pin Ponticello ('JUMP')

Menu

File:

Caricare INO o PDE Prog (Ctrl-L)

Modificare/Esaminare (Ctrl-E)

Salvare

Salvare Come

Il prossimo ('#include')

Precedente

Esci

Trova:

Salire Mucchio di Chiamate

Scendere Mucchio di Chiamate

Imposta il testo Cerca (Ctrl-F)

Trova Next Text

[Trova Testo precedente](#)

[Eeguire:](#)

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

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

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

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

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

[Eeguire \(F9\)](#)

[Arresto \(F10\)](#)

[Resettare](#)

[Animare](#)

[Rallentatore](#)

[Opzioni:](#)

[Passo Scavalcare Structors/ Operatori](#)

[Registrati-assegnazione](#)

[Errore su non inizializzato](#)

[Aggiunto 'loop\(\)' Ritardo](#)

[Consenti interrupt annidati](#)

[Configurare:](#)

['I/O' Dispositivi](#)

[Preferenze](#)

[VarAggiorna:](#)

[Consenti Auto \(-\) Contrarsi](#)

[Minimo](#)

[Evidenziare I cambiamenti](#)

[Finestre:](#)

['Serial' Monitor](#)

[Ripristinare tutto](#)

[Forme d'Onda Digitali](#)

[Forma d'Onde Analogica](#)

[Aiuto:](#)

[Aiuto veloce File](#)

[Aiuto completo File](#)

[Correzioni Errore](#)

[Modifica / Miglioramenti](#)

[Di](#)

['Uno' Scheda e 'I/O' Dispositivi](#)

[Sincronizzazione](#)

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

[Suoni](#)

[Limitazioni e Elementi non Supportati](#)

[Files Incluso](#)

[Allocazioni di Memoria Dinamica e RAM](#)

['Flash' Allocazioni di Memoria](#)

['String' Variabili](#)

[Librerie Arduino](#)

[Puntatori](#)

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

[Scopo](#)

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

[Direttive Compilatore](#)

[Elementi di Lingua Arduino](#)

[C / C ++ - Elementi del Linguaggio](#)

[Modelli Modulo Funzionale](#)

[Emulazione in Tempo Reale](#)

[Note di Rilascio](#)

[Correzioni Errore](#)

[V2.7.0- Mar. 2020](#)

[V2.6.0- Jan 2020](#)

[V2.5.0- ottobre 2019](#)

[V2.4- Maggio 2019](#)

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

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

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

[V2.1- Mar. 2018](#)
[V2.0.2 febbraio 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 - Aug. 2014](#)
[V1.0.1 - Giu. 2014](#)

[V1.0 - prima pubblicazione maggio 2014](#)

[Modifiche / Miglioramenti](#)

[V2.7.0- Mar. 2020](#)
[V2.6.0 gennaio 2020](#)
[V2.5.0 ottobre 2019](#)
[V2.4 maggio 2019](#)
[V2.3 dicembre 2018](#)
[V2.2 giugno 2018](#)
[V2.1 marzo 2018](#)
[V2.0.1 gennaio 2018](#)
[V2.0 settembre 2017](#)
[V1.7.2- Feb. 2017](#)
[V1.7.1 - Febbraio 2017](#)
[V1.7.0- dic. 2016](#)
[V1.6.3- Sett. 2016](#)
[V1.6.2- Sett. 2016](#)
[V1.6.1- agosto 2016](#)
[V1.6 - Giugno 2016](#)
[V1.5.1 - Giugno 2016](#)
[V1.5 - Maggio 2016](#)
[V1.4.2 - marzo 2016](#)
[V1.4 - Dicembre 2015](#)
[V1.3 - ottobre 2015](#)
[V1.2 giugno 2015](#)
[V1.1 - Marzo 2015](#)
[V1.0.1 - Giu. 2014](#)

[V1.0 - prima pubblicazione maggio 2014](#)

Panoramica

UnoArduSim è un freeware **tempo reale** (vedere per Sincronizzazione **restrizioni**) strumento di simulazione che ho sviluppato per gli studenti e gli appassionati di Arduino. È progettato per permetterti di sperimentare e facilmente eseguire il debug, Arduino programmi **senza la necessità di alcun hardware reale**. È mirato al **Arduino 'Uno'** scheda e consente di scegliere tra un set di 'I/O' Dispositivi virtuale e di configurare e connettere questi Dispositivi al 'Uno' virtuale nel **Area del Banco da Laboratorio**. - Non è necessario preoccuparsi di errori di cablaggio, connessioni interrotte / allentate o Dispositivi difettoso che ha compromesso lo sviluppo e il test di programma.

UnoArduSim fornisce semplici messaggi di errore per tutti gli errori analizzare o esecuzione che incontra e consente il debugging con **Resettare**, **Eseguire**, **Eseguire Verso**, **Eseguire Fino A**, **Arresto** e flessibile **Passo** operazioni nel **Area di Codice**, con una visione simultanea di tutte le variabili locali globali e attualmente attive, matrici e oggetti nel **Area delle Variabili**. Viene fornito il controllo dei limiti Eseguire in tempo Eseguire e verrà rilevato l'overflow della RAM ATmega (e la linea colpevole programma evidenziata!). Qualsiasi è in conflitto con elettrico collegato 'I/O' Dispositivi viene contrassegnato e segnalato quando si verificano.

Quando viene aperto un INO o PDE programma file, viene caricato nel programma **Area di Codice**. Il programma viene quindi dato a Analizzare, per trasformarlo in un eseguibile tokenizzato che è quindi pronto per **simulato esecuzione** (a differenza di Arduino.exe, è un eseguibile standalone binario *non* creato) Qualsiasi errore analizzare viene rilevato e contrassegnato evidenziando la linea che non è riuscita a analizzare e riportando l'errore sul **Barra di stato** nella parte inferiore dell'applicazione UnoArduSim finestra. Un **Modificare/Esaminare** È possibile aprire finestra per consentire all'utente di visualizzare e modificare una versione evidenziata dalla sintassi dell'utente programma. Gli errori durante il esecuzione simulato (come un velocità baud errato) sono riportati sulla barra di stato e tramite un messaggio a comparsa.

UnoArduSim V2.7 è un'implementazione sostanzialmente completa del **Arduino Programming Language V1.8.8 come documentato al arduino.cc**. Pagina web di riferimento della lingua e con aggiunte come indicato nella nota di rilascio della versione Scaricamento. Sebbene UnoArduSim non supporti l'implementazione completa di C ++ che fa il GNU compilatore di Arduino.exe, è probabile che solo i programmatori più avanzati scoprano che manca un elemento C / C ++ che desiderano usare (e ovviamente ci sono sempre dei 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 modulo funzionale non lo sono. Anche se oggetti definito dall'utente ('class' e 'struct') e (la maggior parte) sono supportati i sovraccarichi dell'operatore, *l'ereditarietà multipla non lo è*.

Perché UnoArduSim è un simulatore di linguaggio di alto livello, **sono supportate solo le istruzioni C / C ++**, le dichiarazioni di linguaggio assembly non lo sono. Allo stesso modo, perché non è una simulazione di macchina di basso livello, **I registri ATmega328 non sono accessibili per il tuo programma** sia per leggere che per scrivere, anche se l'allocazione del registro, vengono emessi il passaggio e il ritorno (lo si sceglie nel menu **Opzioni**).

Come di V2.6, UnoArduSim ha incassato supporto automatico per un sottoinsieme limitato di Arduino disponibile librerie, ovvero in: 'Stepper.h', 'Servo.h', 'SoftwareSerial.h', 'SPI.h', 'Wire.h', 'OneWire.h', 'SD.h', 'TFT.h' e 'EEPROM.h' (Versione 2). V2.6 introduce un meccanismo per 3rd supporto delle librerie interlocutore tramite files fornito nel 'include_3rdParty' cartella che possono essere trovati all'interno della directory di installazione UnoArduSim.

Per ogni '#include' di librerie create dall'utente, lo farà UnoArduSim **non** cercare la solita struttura di directory di installazione di Arduino per individuare la libreria; invece tu **bisogna** per copiare l'intestazione corrispondente (".h") e l'origine (".cpp") file nella stessa directory del file su cui stai lavorando (naturalmente a condizione che il contenuto di qualsiasi '#include' file deve essere completamente comprensibile a l'UnoArduSim analizzatore).

Ho sviluppato UnoArduSimV2.0 in QtCreator con supporto multilingue, ed è attualmente disponibile solo per FinestreTM. 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 sicuramente alcuni errori ancora nascosti lì dentro. Se desideri segnalare un errore, descrivilo (brevemente) in un'email a unoArduSim@gmail.com e **assicurati di allegare il tuo codice sorgente Arduino programma completo errore** così posso replicare il errore e sistemarlo. Non risponderò ai singoli rapporti di errore e non ho tempistiche garantite per le correzioni in una versione successiva (ricorda che ci sono quasi sempre soluzioni alternative!).

Saluti,

Stan Simmons, PhD, P.Eng.
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: l'esempio finestre mostrato sotto è tutto sotto un Finestre-OS scelto dall'utente tema del colore che ha un colore di sfondo blu scuro finestra).


Area di Codice



Il **Area di Codice** mostra il tuo utente programma e mette in evidenza le tracce del suo esecuzione.







Dopo un programma caricato ha un Analizzatore di successo, la prima linea in ' **main()** ' è evidenziato e programma è pronto per esecuzione. Nota che ' **main()** ' è implicitamente aggiunto da Arduino (e da UnoArduSim) e tu lo fai **non** includilo come parte del tuo utente programma file. Esecuzione è sotto controllo del menu **Eseguire** e il suo associato **Strumento-Bar** pulsanti e scorciatoie da tastiera modulo funzionale.

Dopo il rafforzamento esecuzione da uno (o più) istruzioni (si può usare **Strumento-Bar** pulsanti , , , o ), La linea programma che sarà eseguito successivo viene poi evidenziata in verde - la linea verde evidenziata è sempre la riga successiva **pronto per essere eseguito** .

Allo stesso modo, quando un programma in esecuzione colpisce un (temporaneo **Esegui Verso**) punto di arresto, esecuzione viene arrestato e la linea punto di arresto viene evidenziata (ed è quindi pronta per esecuzione).

Se programma esecuzione è attualmente interrotta, e si fa clic nella **Area di Codice** finestra, la linea che avete appena fatto clic diventa evidenziato in verde oliva scuro (come mostrato nella foto) - i prossimi-to-be-eseguito riga evidenziata in verde rimane sempre (come di V2.7). Ma puoi causare esecuzione *progredire fino a* la linea che hai appena evidenziato, quindi fai clic su **Esegui Verso**  **Strumento-Bar** pulsante. Questa funzione consente di raggiungere in modo rapido e semplice linee specifiche in un programma in modo da poter successivamente passare linea per linea su una porzione di interesse programma.

Se il tuo programma caricato ne ha '**#include**' files, puoi spostarti tra di loro usando **File | Precedente** e **File | Il prossimo** (con **Strumento-Bar** pulsanti  e ).

Le azioni del **Trova** menu che si permettono di **O** trovare il testo nel **Area di Codice** o **Area delle Variabili** (**Strumento-Bar** pulsanti   , o le scorciatoie da tastiera **freccia su** e **Freccia in giù**) *dopo il primo impiego* **Trova | testo Set Cerca** o **Strumento-Bar**  , **O IN ALTERNATIVA** per *navigare call-stack di* nel **Area di Codice** (**Strumento-Bar** pulsanti   e  , o le scorciatoie da tastiera **freccia su** e **Freccia in giù**). chiavi **PgDn** e **PgUp** saltare selezione alla successiva / precedente modulo funzionale ..

```

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

int count;

void setup()
{
    count=0;
}

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

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

```

Preferenze



Configurare | Preferenze consente agli utenti per impostare programma e le preferenze di visualizzazione (che un utente normalmente desidera adottare alla prossima sessione). Questi possono quindi essere salvati e caricati da a **'myArduPrefs.txt'** file che risiede nella stessa directory del 'Uno' programma 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 V 2.0, la scelta della lingua è ora inclusa. - questo include sempre l'inglese (**it**), più una o due altre lingue locali dell'utente (dove queste esistono), e un override basato sul codice della lingua ISO-639 a due

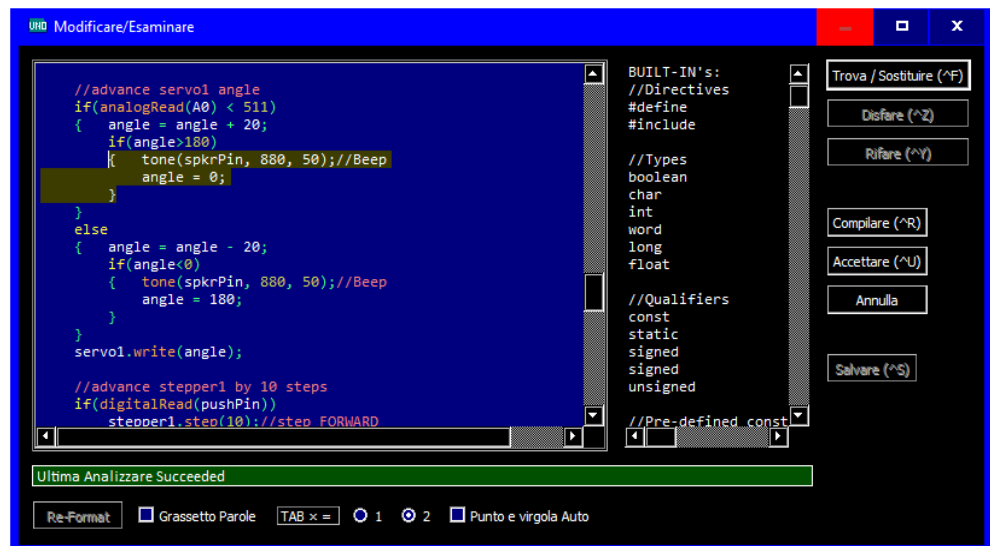
lettere sulla prima riga del **'myArduPrefs.txt'** file (se ne viene fornito uno lì). Le scelte compaiono solo se una traduzione ".qm" file esiste nella cartella delle traduzioni (all'interno la directory home di UnoArduSim.exe).

Modificare/Esaminare

Facendo doppio clic su qualsiasi riga nel file **Area di Codice** (o usando il menu **File**), un **Modificare/Esaminare** finestra è aperto per consentire modifiche al programma file, con il **linea attualmente selezionata** nel **Area di Codice** è evidenziato.

Questo finestra ha capacità di modifica completa con evidenziazione dinamica della sintassi (diversi colori evidenziare sono usati per parole chiave, commenti, ecc.) , È disponibile l'evidenziazione della sintassi grassetto opzionale e la formattazione automatica del livello di rientro (supponendo che l'utente abbia selezionato tale utilizzo **Configurare | Preferenze**). Puoi anche selezionare comodamente le chiamate incassato modulo funzionale (o

incassato **'#define'** costanti) da aggiungere nel programma dalla casella di riepilogo fornita - fai doppio clic sulla voce della casella di elenco desiderata per aggiungerla al tuo programma alla posizione attuale del cursore (modulo funzionale-chiama variabile **tipi** sono solo per informazione e sono spogliati per lasciare fantastici segnaposto quando vengono aggiunti al tuo programma).



Il finestra ha **Trova** (uso **Ctrl-F**) e **Trova / Sostituire** capacità (usare **Ctrl-H**) . Il **Modificare/Esaminare** finestra ha **Disfare** (**ctrl-Z**), e **Rifare** (**Ctrl-Y**) pulsanti (che appaiono automaticamente).

Utilizzare Alt-freccia destra per richiedere scelte auto-completamento per incassato **globale variabili**, e per **membro variabili e moduli funzionali**.

Scartare **tutti i cambiamenti** da quando hai aperto programma per la modifica, fai clic su **Annulla** pulsante. Accettare il stato attuale, fare clic su **Accettare** pulsante e il programma riceve automaticamente un altro Analizzare (e viene scaricato nel 'Uno' se non vengono trovati errori) e il nuovo stato appare nello UnoArduSim finestra principale **Barra di stato** .

UN **Compilare** (**Ctrl-R**) (più un associato **Stato Analizzare** messaggio-box come visto nell'immagine sopra) è stato aggiunto per consentire il test delle modifiche senza dover prima chiudere finestra. UN **Salvare** (**Ctrl-S**) è stato aggiunto anche come scorciatoia (equivalente a un **Accettare** più una successiva separata **Salvare** dal principale finestra).

Su entrambi **Annulla** o **Accettare** senza modifiche fatte, il **Area di Codice** la linea corrente cambia per diventare il **ultima posizione Modificare/Esaminare** e puoi usare quella funzione per saltare il **Area di Codice** su una linea specifica (possibilmente per prepararsi a fare un **Eseguire Verso**), Puoi anche usare **ctrl-PgDn** e **ctrl-PgUp** per passare alla successiva (o precedente) interruzione della linea 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 moduli funzionali). Puoi anche usare **ctrl-casa** e **ctrl-end** per saltare all'inizio del programma e terminare, rispettivamente.

Formattazione di rientro automatico 'Tab' livello si fa quando si apre la finestra, se questa opzione è stata impostata sotto **Configurare | Preferenze**. Si può rifare che la formattazione in qualsiasi momento facendo clic sul **Re-Format** Pulsante (è abilitato solo se è stato precedentemente selezionato il **Preferenza automatico indentazione**). È inoltre possibile aggiungere o schede di eliminazione a voi stessi di un gruppo di righe consecutive preselezionati utilizzando la tastiera **freccia destra** o **freccia sinistra** chiavi - ma **Preferenza automatico indentazione deve essere spento** per evitare di perdere i propri livelli di scheda personalizzata.




quando **Punti e virgola automatici** è selezionato, premendo **accedere** per terminare una riga, viene automaticamente inserito il punto e virgola della linea.

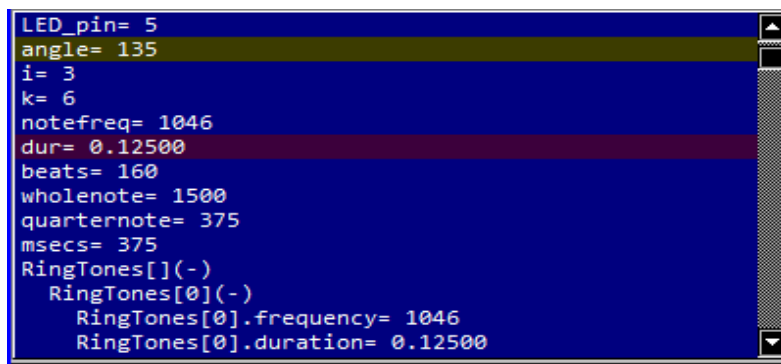
E per aiutarti a tenere traccia dei tuoi contesti e parentesi graffe, facendo clic su a ' { ' o ' } ' parentesi graffa **evidenzia tutto il testo tra quel parentesi graffa e il suo partner corrispondente** .

Area delle Variabili e Modificare/Monitorare Variabile finestra

Il **Area delle Variabili** si trova appena sotto il **Area di Codice**. Mostra i valori correnti per ogni utente globale e attivo (in-scopo) locale variabile / matrice / oggetto nel programma caricato. Come programma esecuzione si sposta tra moduli funzionali, i contenuti cambiano per riflettere solo quelli locali variabili accessibili all'attuale modulo funzionale / scopo, più eventuali globali dichiarati dall'utente. Qualsiasi variabili dichiarato come 'const' o come

'PROGMEM' (assegnato a 'Flash' memoria) hanno valori che non possono cambiare, e per risparmiare spazio questi sono quindi *non visualizzato*. 'Servo' e 'SoftwareSerial' Le istanze oggetto non contengono valori utili, pertanto non vengono visualizzate.

Puoi **trova** specificato **testo** con i suoi comandi di testo di ricerca (con **Strumento-Bar** pulsanti  e , o le scorciatoie da tastiera **freccia su** e **Freccia in giù**), Dopo il primo impiego **Trova | set Cerca** testo o  .



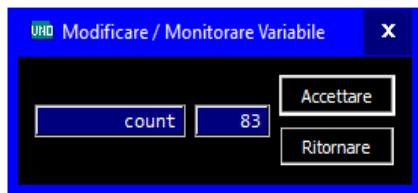
```
LED_pin= 5
angle= 135
i= 3
k= 6
notefreq= 1046
dur= 0.12500
beats= 160
wholenote= 1500
quarternote= 375
msec= 375
RingTones[0](-)
RingTones[0](-)
RingTones[0].frequency= 1046
RingTones[0].duration= 0.12500
```

Matrici e **oggetti** sono mostrati in entrambi **un-espansi** o **espansi** formato, con un plus finale ' (+) ' o meno ' (-) ' segno, rispettivamente. Il simbolo per un matrice **x** mostra come '**x** [] ' . Per espandere per mostrare tutti gli elementi del matrice, basta fare un solo clic su '**x** [] (+) ' nel **Area delle Variabili** . Per contrarsi tornare a una vista non espansi, fare clic su '**x** [] (-) ' . Il default un-espansi per un oggetto '**p1** ' mostra come '**p1** (+) ' Per espandere per mostrare tutti i membri di quello '**class** ' o '**struct** ' istanza, clic singolo su '**p1** (+) ' nel **Area delle Variabili** . Per contrarsi tornare a una vista non espansi, fare clic su '**p1** (-) ' . Se tu *singolo-clic su qualsiasi linea per evidenziare in verde oliva scuro* (Può essere semplice variabile, o l'aggregato ' (+) ' o ' (-) ' linea di un matrice o oggetto, o un singolo elemento o matrice oggetto-membro), poi facendo un **Eseguire Fino A** causerà esecuzione per riprendere e congelare al prossimo *accesso in scrittura* ovunque all'interno tale aggregato selezionato, oa quello selezionato singola posizione variabile.

Quando si usa **Passo** o **Eseguire** , gli aggiornamenti dei valori variabile visualizzati vengono eseguiti in base alle impostazioni utente effettuate nel menu **VarAggiorna** - questo 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 mantenere l'esecuzione in ritardo rispetto a quello che altrimenti sarebbe eccessivo. **Area delle Variabili** finestra carica l'aggiornamento. quando **Animare** è in vigore, o se il **Cambiamenti Evidenziare** l'opzione di menu è selezionata, cambia il valore di una variabile durante **Esegui** risulterà nel suo valore visualizzato che viene aggiornato **subito** e diventa evidenziato - questo causerà il **Area delle Variabili** per scorrere (se necessario) sulla riga che contiene variabile e esecuzione non sarà più in tempo reale !.

Quando esecuzione si blocca dopo **Passo**, **Esegui Verso**, **Esegui Fino A**, o **Esegui** -poi- **Arresto**, il **Area delle Variabili** evidenzia la variabile corrispondente al **indirizzo (i) che è stato modificato** (se del caso) dal **ultima istruzione** durante quell'esecuzione (incluse le inizializzazioni della dichiarazione variabile). Se quella istruzione **completamente** riempito un **oggetto o matrice**, il **linea genitore (+) o (-)** per quell'aggregato diventa evidenziato. Se, invece, l'istruzione ha modificato a Posizione che è attualmente visibile, quindi diventa evidenziato. Ma se la (e) posizione (i) modificata (e) è attualmente nascosta all'interno un matrice o oggetto non espansi, quell'aggregato **linea genitore** ottiene un **evidenziazione dei caratteri in corsivo** come indicazione visiva che è stato scritto qualcosa al suo interno - facendo clic su espandere ne causerà quindi il suo **scorso** elemento o membro modificato da evidenziare.



Il **Modificare/Monitorare** finestra ti dà **la possibilità di seguire qualsiasi valore variabile durante esecuzione**, o a **cambia il suo valore nel mezzo di (fermato) programma esecuzione** (così puoi testare quale sarebbe l'effetto di continuare in futuro con quel nuovo valore). **Arresto** Prima esecuzione, quindi **lasciato doppio clic** sul variabile di cui desideri tenere traccia o modificare il valore. Per monitorare semplicemente il valore durante programma esecuzione, **lascia aperta la finestra di dialogo** e poi uno dei **Esegui** o **Passo** comandi -

il suo valore verrà aggiornato in **Modificare/Monitorare** secondo le stesse regole che governano gli aggiornamenti nel **Area delle Variabili**. **Per cambiare il valore variabile**, inserire il valore della casella di modifica e **Accettare**. Continua esecuzione (usando uno qualsiasi dei **Passo** o **Esegui** comandi) per utilizzare quel nuovo valore da quel punto in avanti (o è possibile **Ritornare** al valore precedente).

Su programma Caricare o Resetare nota che tutto il **valore non inizializzato-variabili viene ripristinato sul valore 0** e **tutti i puntatori non inizializzati-variabili vengono reimpostati su 0x0000**.

Area del Banco da Laboratorio

Il Area del Banco da Laboratorio mostra un 'Uno' scheda da 5 volt che è circondato da un set di 'I/O' Dispositivi che è possibile selezionare / personalizzare e collegare all'Uno pins desiderato.

Il 'Uno'

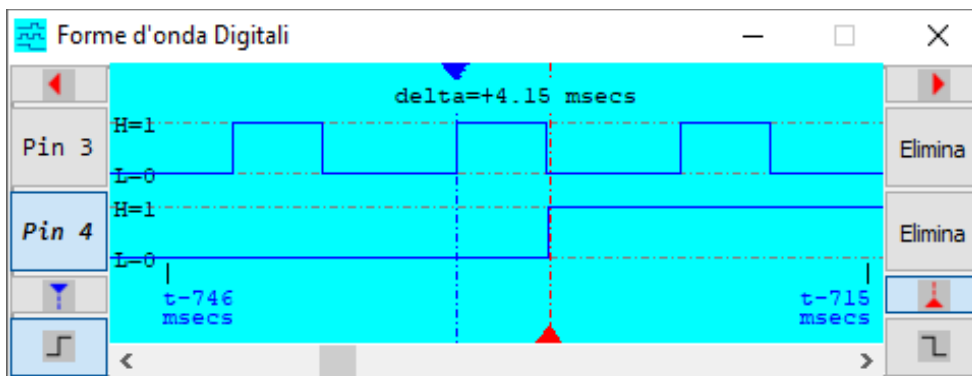
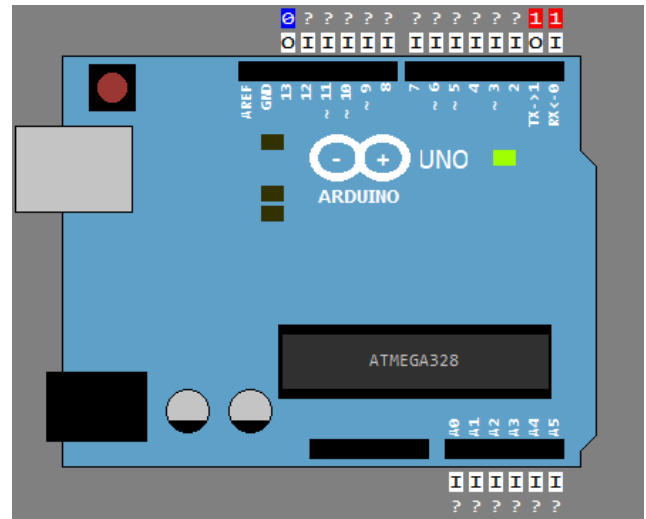
Questa è una rappresentazione di 'Uno' scheda e dei suoi LED integrati. Quando carichi un nuovo programma in UnoArduSim, se lo analizza correttamente subisce un "scaricamento simulato" sull'Uno che simula il comportamento di un 'Uno' scheda reale - vedrai lampeggiare la seriale RX e TX DEL (insieme all'attività su pins 1 e 0 quali sono *cablato per la comunicazione seriale con un computer host*). Questo è immediatamente seguito da un pin 13 DEL flash che indica scheda reset e (e UnoArduSim si fermano automaticamente) all'inizio del programma esecuzione caricato. È possibile evitare questa visualizzazione e il ritardo di caricamento associato deselectando **Mostra Scaricamento** a partire dal **Configurare | Preferenze**.

Il finestra consente di visualizzare i livelli logici del digitale su tutti i 20 'Uno' pins ('1' sul rosso per 'HIGH', '0' su blu per 'LOW', e '?' su grigio per una tensione indeterminata indefinita) e le direzioni programmate ('I' per 'INPUT', o 'O' per 'OUTPUT'). Per pins che viene pulsato usando PWM tramite 'analogWrite()' o da 'tone()' o da 'Servo.write()', il colore cambia in viola e il simbolo visualizzato diventa '^'.



Nota che **Digitale pins 0 e 1 sono cablati tramite resistori da 1-kOhm al chip USB per comunicazione seriale con un computer host.**







A parte: *Digitale pins 0-13 appaiono come simulatore pins 0-13 e analogico pins 0-5 appaiono come A0-A5. Per accedere a un analogico pin nel tuo programma, puoi fare riferimento al numero pin con 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()', una terza opzione è resa disponibile - puoi, per questa istruzione, rilasciare il 'A' prefisso dal numero pin e utilizzare semplicemente 0-5. Per accedere a pins 14-19 nel tuo programma usando 'digitalRead()' o 'digitalWrite()', puoi semplicemente fare riferimento a quel numero pin, oppure puoi usare gli alias A0-A5.*



Sinistra-clic su qualsiasi 'Uno' pin aprirà a **Forme d'Onda Digitali** finestra che mostra il passato **valore di un secondo di Attività a livello digitale** su quel pin. Puoi fare clic sull'altro pins per aggiungerli al display Forme d'Onda Digitali (fino a un massimo di 4 forme d'onda alla volta).



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

Una delle forme d'onda visualizzate sarà la **pin attivo** forma d'onda, indicato dal suo pulsante "Pin" che viene mostrato come depresso (come nella precedente acquisizione dello schermo Forme d'Onda Digitali). È possibile selezionare un forma d'onda facendo clic sul relativo pulsante numerico Pin, quindi selezionare la polarità del fronte di interesse facendo clic sul pulsante di selezione della polarità del fronte di salita / discesa appropriato, , o  o

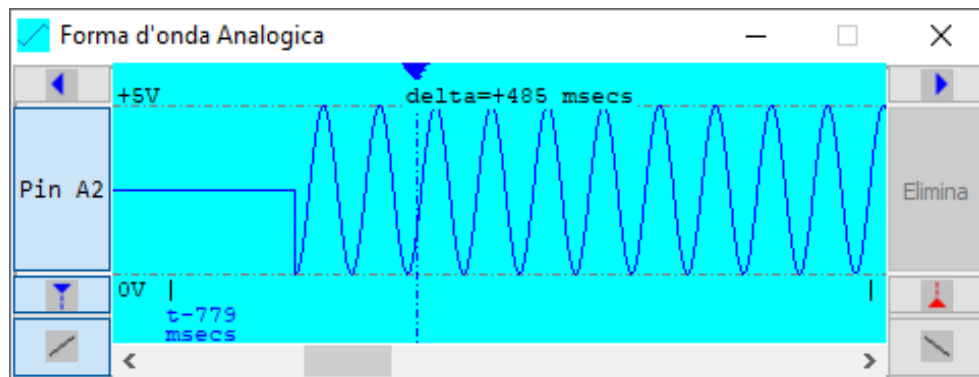
usando i tasti di scelta rapida **freccia su** e **Freccia in giù** . Puoi quindi **saltare** il cursore attivo (o le linee del cursore blu o rosso con il loro tempo di delta visualizzato) indietro o avanti rispetto al bordo digitale della polarità scelta di **questo pin attivo** forma d'onda usando i pulsanti del cursore ( ,  o  , ) (a seconda di quale cursore era 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  sopra riportati) - **questo salta anche 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 la scorciatoia **'Tab'** chiave.







Puoi **saltare** il cursore attualmente attivato da **fare clic con il tasto sinistro ovunque** nella regione di visualizzazione forma d'onda sullo schermo. In alternativa, è possibile selezionare la linea del cursore rossa o blu facendo clic su destra sopra di essa (per attivarla), quindi **trascinalo su una nuova posizione** e rilascio. Quando un cursore desiderato è attualmente fuori dallo schermo, puoi farlo **tasto destro del mouse ovunque** nell'ottica di saltarlo in quella nuova posizione sullo schermo. Se entrambi i cursori sono già sullo schermo, facendo clic con il pulsante destro si alterna semplicemente il cursore attivato.

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

Facendo invece a **tasto destro del mouse su qualsiasi 'Uno' pin** apre a **Forma d'Onde Analogica** finestra che visualizza il **oltre un secondo di valore** di **Attività a livello di analogico** su quel pin. A differenza del Forme d'Onda Digitali finestra, puoi farlo mostra l'attività analogico su un solo pin alla volta.



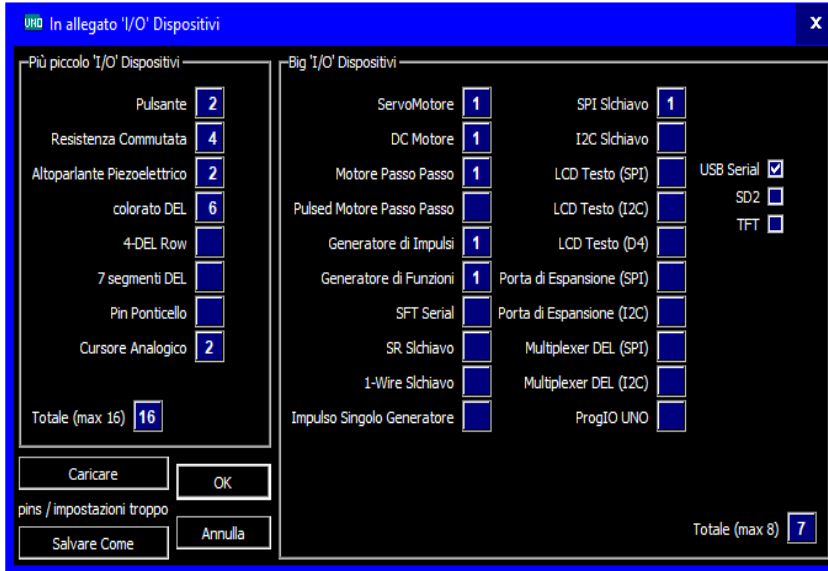
Clicca per visualizzare la pagina a sinistra oa destra, oppure usa i tasti Home, PgUp, PgDn, Fine

Puoi **saltare** il linee del cursore blu o rosso al successivo "punto di pendenza" in salita o in discesa usando i pulsanti freccia avanti o indietro ( ,  o  ,  , sempre in base al cursore attivato o utilizzare il ← e → tasti) di concerto con i pulsanti di selezione della pendenza in salita / discesa  ,  (il "punto di inclinazione" si verifica quando la tensione analogico passa attraverso la soglia del livello logico digitale di ATmega pin). In alternativa, puoi nuovamente fare clic per saltare o trascinare queste linee del cursore simili al loro comportamento nel Forme d'Onda Digitali finestra

urgente **'Ctrl-S'** dentro **finestra consente di salvare il forma d'onda (X, Y) dati** a un testo file di tua scelta, dove **X** è in microsecondi dal lato sinistro e **Y** è in bulloni.

'I/O' Dispositivi

Un certo numero di Dispositivi diversi circonda 'Uno' sul perimetro del **Area del Banco da Laboratorio** . Il "piccolo" 'I/O' Dispositivi (di cui è consentito un massimo di 16 in totale) risiede lungo i lati sinistro e destro del Area. "Large" 'I/O' Dispositivi (di cui sono consentiti fino a 8 in totale) hanno elementi "attivi" e risiedono lungo la parte superiore e inferiore del **Area del Banco da Laboratorio** . Il numero desiderato di ogni tipo di 'I/O' Dispositivo disponibile può essere impostato utilizzando il menu **Configurare | 'I/O' Dispositivi** .



Ogni 'I/O' Dispositivo ha uno o più allegati pin mostrati come **due cifra** Numero pin (00, 01, 02, ... 10,11,12, 13 e A0-A5 o 14-19, in seguito) in una casella di modifica corrispondente. Per i numeri Analizzatore dal 2 al 9 puoi semplicemente inserire il singolo cifra - lo 0 iniziale verrà fornito automaticamente, ma per pins 0 e 1 devi prima inserire lo 0 iniziale. Gli input sono *normalmente* sul lato sinistro di un 'I/O' Dispositivo e le uscite sono *normalmente* sulla destra (*spazio permettendo*). Tutti i 'I/O' Dispositivi risponderanno direttamente ai livelli pin e alle modifiche del livello pin, quindi risponderanno alla libreria moduli funzionali destinata al pins collegato o al programmato 'digitalWrite()' (per operazione "bit-banged").

Puoi collegare più Dispositivi allo stesso ATmega pin se non ne crea uno **conflitto elettrico**. Tale conflitto può essere creato da un 'Uno' pin come 'OUTPUT' guida contro un Dispositivo con conduzione forte (bassa impedenza) collegato (ad esempio, guida contro un'uscita 'FUNCGEN' o un 'MOTOR' **Enc** output), o da due Dispositivi connessi in competizione tra loro (ad esempio sia un 'PULSER' che un 'PUSH' - collegato allo stesso pin). Qualsiasi conflitto di questo tipo sarebbe disastroso in un'implementazione hardware reale e quindi non consentito, e verrà segnalato all'utente tramite una finestra di messaggio pop-up).

La finestra di dialogo può essere utilizzata per consentire all'utente di scegliere i tipi e i numeri desiderati di 'I/O' Dispositivi. Da questa finestra di dialogo puoi anche **Salvare** 'I/O' Dispositivi in un testo file e / o **Caricare** 'I/O' Dispositivi da un testo file precedentemente salvato (o modificato) (**include tutte le connessioni pin e le impostazioni selezionabili e tutti i valori inseriti nella casella di modifica**).

Si noti che dalla versione 1.6, i valori nelle caselle di modifica periodo, ritardo e larghezza impulso nel relativo IO Dispositivi possono avere il suffisso 'S' (o 's'). Ciò indica che dovrebbero essere scalato secondo la posizione di un globale 'I/O ____S' controllo del cursore che appare nel finestra principale **Strumento-Bar**. Con quel 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 ogni particolare 'I/O' Dispositivo). È possibile ridimensionare più di un valore di casella di modifica **contemporaneamente** usando questo cursore. Questa funzione ti consente di trascinare il cursore durante l'esecuzione per emulare facilmente variazioni di ampiezze, periodi e ritardi dell'impulso per quelli collegati 'I/O' Dispositivi.

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

Molti di questi Dispositivi **supporta il ridimensionamento dei valori digitati** usando il cursore sul finestra principale **Strumento-Bar**. Se il valore Dispositivo ha la lettera 'S' come suffisso, il suo valore sarà moltiplicato per un fattore di scala (tra 0,0 e 1,0) che è determinato dalla posizione del pollice scorrevole, soggetta al vincolo del valore minimo Dispositivo (1.0 è completamente a destra, 0.0 è completamente a sinistra) - vedere 'I/O ____S' sotto ciascuno dei tubi flessibili Dispositivi descritti di



seguito.

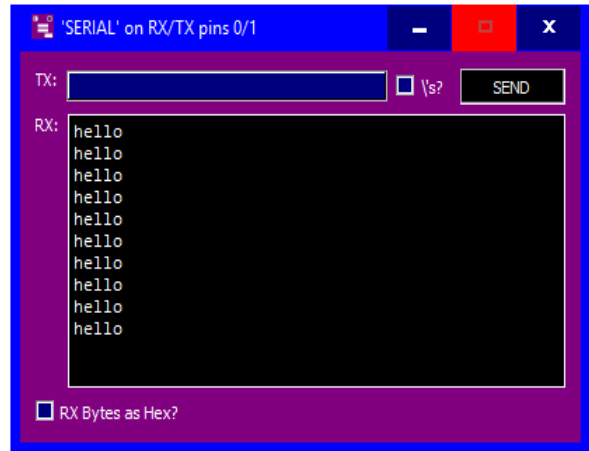
'Serial' Monitor ('SERIAL')



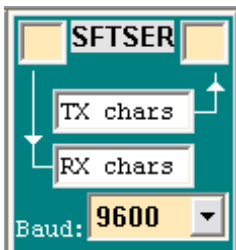
Questo 'I/O' Dispositivo consente l'ingresso e l'uscita seriale ATmega mediata dall'hardware (tramite il chip 'Uno' USB) su 'Uno' pins 0 e 1. Il velocità baud viene impostato utilizzando l'elenco a discesa in basso, il velocità baud selezionato **deve combaciare** il valore che il tuo programma passa al '`Serial.begin()`' modulo funzionale per una corretta trasmissione / ricezione. *La comunicazione seriale è fissata su 8 bit di dati, 1 bit di stop e nessun bit di parità.* Sei autorizzato a **disconnect** (Vuoto) **ma non sostituire** TX pin 00, ma non RX pin 01.

Per inviare l'input da tastiera al tuo programma, digita uno o più caratteri in alto (caratteri TX), modifica finestra e poi colpire il '**Enter**' tasto della tastiera. (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 il '`Serial.available()`' e '`Serial.read()`' moduli funzionali per leggere i caratteri nell'ordine in cui sono stati ricevuti nel buffer 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 all'uscita della console inferiore (caratteri RX) finestra chiamando l'Arduino '`print()`', '`println()`', o '`write()`' moduli funzionali.

Inoltre, **un finestra più grande per l'impostazione / visualizzazione dei caratteri TX e RX può essere aperto facendo doppio clic (o clic con il tasto destro) su questo 'SERIAL' Dispositivo**. Questo nuovo finestra ha una casella di modifica dei caratteri TX più grande e un pulsante 'Send' separato che può essere selezionato per inviare i caratteri TX all'"Uno" (su pin 0). C'è anche un'opzione di spunta per reinterpretare sequenze di caratteri con escape backslash come '`\n`' o '`\t`' per la visualizzazione non grezza.



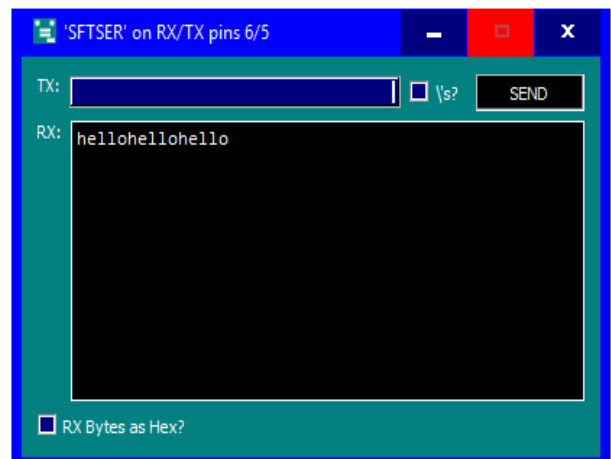
Seriale Software ('SFTSER')



Questo 'I/O' Dispositivo consente la libreria mediata dal software o, in alternativa, l'utente "bit-banged", l'input e l'output seriale su qualsiasi coppia di 'Uno' pins che si sceglie di compilare (**eccetto per** pins 0 e 1 dedicati all'hardware '`Serial`' comunicazione). Il tuo programma deve avere un '`#include <SoftwareSerial.h>`' linea in alto se si desidera utilizzare la funzionalità di tale libreria. Come con l'"SERIAL" Dispositivo basato su hardware, il velocità baud per 'SFTSER' viene impostato utilizzando l'elenco a discesa nella parte inferiore: il velocità baud selezionato deve corrispondere al valore che il programma passa al '`begin()`' modulo funzionale per una corretta trasmissione / ricezione. *La comunicazione seriale è fissata su 8 bit di dati, 1 bit di stop e nessun bit di parità.*

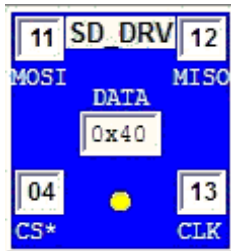
Inoltre, come con l'hardware basato '`SERIAL`', **un finestra più grande per l'impostazione / visualizzazione di TX e RX può essere aperto facendo doppio clic (o clic con il tasto destro) su SFTSER Dispositivo**.

Si noti che a differenza dell'implementazione hardware di '`Serial`', non è fornito Buffer TX supportato da operazioni di interrupt ATmega interne (solo un buffer RX), quindi quello '`write()`' (o '`print()`') le chiamate stanno bloccando (ovvero, il tuo programma non procederà finché non saranno completi).



Unità Disco SD ('SD_DRV')

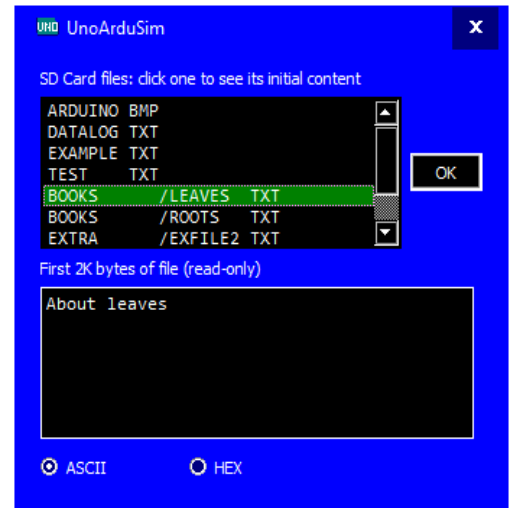
Questo 'I/O' Dispositivo consente la libreria mediata dal software (ma **non** "bit-banged") file operazioni di input e output sul 'Uno' SPI pins (puoi scegliere quale **CS *** pin che userai). Il tuo programma può semplicemente `'#include <SD.h>'` linea vicino alla cima, e puoi usare `'<SD.h>'` moduli funzionali O chiama direttamente `'SdFile'` moduli funzionali te stesso.



*Un finestra di grandi dimensioni che visualizza le directory e files (e il contenuto) può essere aperto facendo doppio clic (o clic con il tasto destro) su 'SD_DRV' Dispositivo . Tutto il contenuto del disco è **caricato da** un SD sottodirectory nella directory programma caricata (se esiste) a `'SdVolume::init()'` , e si **rispecchia** quello stesso SD sottodirectory su file `'close()'` , `'remove()'` e via `'makeDir()'` e*

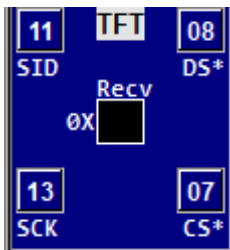
`'rmDir()'` .

Un DEL giallo lampeggia durante i trasferimenti SPI e 'DATA' mostra l'ultimo 'SD_DRV' **risposta** byte. Tutti i segnali SPI sono precisi e possono essere visualizzati in **Forma d'onda finestra**.



Display TFT ('TFT')

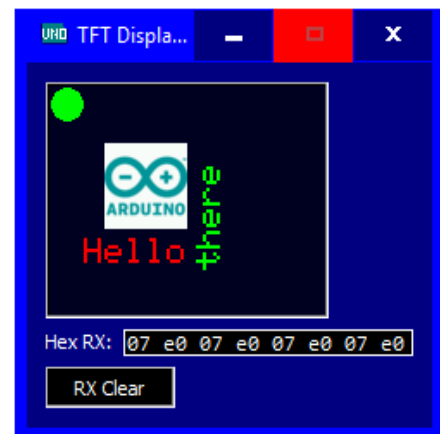
Questo 'I/O' Dispositivo emula un'Adafruit™ TFT di dimensioni 1280by-160 pixel (nella sua rotazione nativo = 0, ma quando si utilizza la libreria 'TFT.h', `'TFT.begin()'` insieme di inizializzazione per rotazione = 1 che fornisce una vista "paesaggio" di 160-by-128 pixel). È possibile spingere questo Dispositivo chiamando il moduli funzionali del `'TFT.h'` biblioteca (che richiede prima `'#include <TFT.h>'`), oppure è possibile utilizzare il sistema SPI per l'invio di propria sequenza di byte da spingere esso.



Il TFT è sempre collegato a 'SPI' 'MOSI' pins (per 'SID') e 'SCK' (per 'SCK') - quelli che non possono essere cambiati. Il 'DS*' pin è per dati / comando SELECT (modalità dati 'LOW' seleziona), e il 'CS*' pin è il chip-select attivo basso

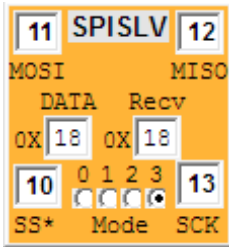
Non v'è alcun Resetare pin fornito in modo non si può fare un reset hardware questo Dispositivo guidando una bassa pin (come `'TFT::begin()'` modulo funzionale tenta di fare quando si è superato un certo numero 'reset' pin valida come terzo parametro al `'TFT(int cs, int ds, int rst)'` costruttore). Il Dispositivo tuttavia di una connessione nascosta alla linea Resetare sistema in modo che si resetta ogni volta che si fa clic sul principale UnoArduSim icona della barra degli strumenti Resetare, o il pulsante di reset 'Uno' scheda ..

Di **doppio clic** (o **tasto destro del mouse**) Su questo Dispositivo, un finestra maggiore è aperto per mostrare che la piena display LCD pixel 160-by-128, insieme ai più recenti ricevuto 8 byte (come mostrato sotto)



Configurabile SPI Schiavo ('SPISLV')

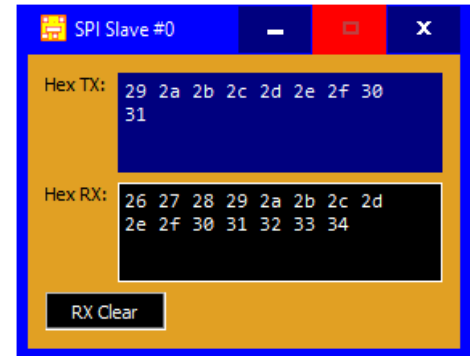
Questo 'I/O' Dispositivo emula uno slave SPI in modalità selezionata con un basso attivo **SS *** ("slave-select") pin che controlla il **MISO** uscita pin (quando **SS *** è alto, **MISO** non è spinto). Il tuo programma deve avere un `'#include <SPI.h>'` linea se si desidera utilizzare la funzionalità di incassato SPI Arduino oggetto e libreria. In alternativa, puoi scegliere di creare il tuo "bit-banged" **MOSI** e **SCK** segnala a spingere questo Dispositivo.



Il Dispositivo rileva le transizioni sui bordi **CLK** input secondo la modalità selezionata (`'MODE0'`, `'MODE1'`, `'MODE2'`, o `'MODE3'`), che deve essere scelto per abbinare la modalità SPI programmato del tuo programma.

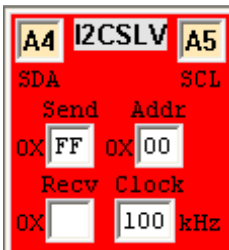
Facendo doppio clic (o clic con il tasto destro) sul Dispositivo puoi aprire un compagno finestra più grande quello invece

permette y o per riempire il buffer massimo di 32 byte (in modo da emulare SPI Dispositivi che restituisce automaticamente i loro dati) e per vedere gli ultimi 32 byte ricevuti (tutti come coppie esadecimali). **Nota che** il prossimo byte buffer TX è inviato automaticamente solo a 'DATA' **dopo** un pieno `'SPI.transfer()'` ha completato!



Due fili I2C Schiavo ('I2CSLV')

Questo 'I/O' Dispositivo emula solo a *modalità slave* Dispositivo. Al Dispositivo può essere assegnato un indirizzo di bus I2C usando una voce cifra a due esadecimali nella sua edit-box 'Addr' (risponderà solo a I2C transazioni di bus che coinvolgono il suo indirizzo assegnato). Il Dispositivo invia e riceve dati sul suo open-drain (solo pull-down) **SDA** pin e risponde al segnale di clock del bus sul suo drain aperto (solo verso il basso) **SCL** pin. Anche se 'Uno' sarà il maestro del bus responsabile della generazione di **SCL** segnale, questo slave Dispositivo tirerà anche **SCL** basso 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 casella di modifica 'Clock').

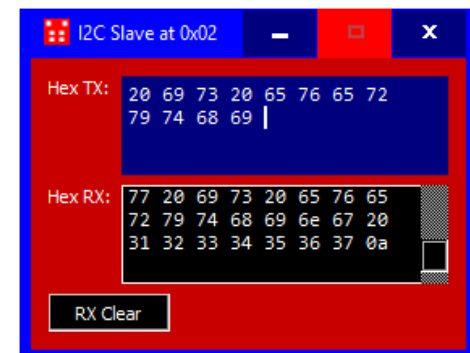


Il tuo programma deve avere un `'#include <Wire.h>'` linea se si desidera utilizzare la funzionalità del `'TwoWire'` libreria per interagire con questo Dispositivo. In alternativa, puoi scegliere di creare i tuoi dati bit-banged e i segnali di clock per spingere questo slave Dispositivo.

Un singolo byte per la trasmissione al master 'Uno' può essere impostato nella casella di modifica 'Send' e un singolo byte (ricevuto più di recente) può essere visualizzato nella sua casella di modifica 'Recv' (sola lettura). **Nota che** il valore della casella di modifica 'Send' riflette sempre il valore il prossimo byte per la trasmissione da questo buffer di dati interno Dispositivo.

Facendo doppio clic (o clic con il tasto destro) sul Dispositivo puoi aprire un compagno finestra più grande che invece consente di

riempire un buffer FIFO massimo di 32 byte (in modo da emulare TWI Dispositivi con tale funzionalità) e di visualizzare (fino a un massimo di 32) byte dei dati ricevuti più di recente (come display esadecimale cifra 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 dal momento che Arduino `'Wire.h'` libreria usa **cinque** tali buffer RAM nel suo codice di implementazione, che è costoso per la memoria RAM. Modificando l'installazione di Arduino `'Wire.h'` file per cambiare costante definita `'BUFFER_LENGTH'` (e anche modificando il compagno `'utility/twi.h'` file per cambiare TWI buffer length) sia per essere invece 16 o 8, un utente *poteva* ridurre in modo significativo l'overhead della memoria RAM di il 'Uno' in un target **implementazione hardware** - UnoArduSim rispecchia quindi questa possibilità del mondo reale attraverso **Configurare | Preferenze** .



LCD di Testo I2C ('LCDI2C')

Questo 'I/O' Dispositivo emula un 1,2, 04 4-line personaggio-LCD, in una delle tre modalità:

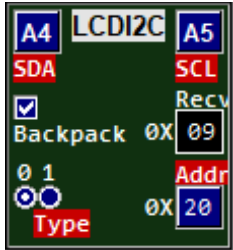
- a) zaino digitare 0 (espansione porta stile Adafruit con l'hardware con indirizzo bus I2C 0x20-0x27)
- b) tipo di zaino espansione porta 1 (stile DFRobot con l'hardware avere indirizzo bus I2C 0x20-0x27)
- c) senza zaino (modalità originale interfaccia I2C integrata avente indirizzo bus I2C 0x3C-0x3F)

Sostenere codice della libreria per ciascuna modalità Dispositivo è stata fornita all'interno della cartella

'include_3rdParty' della directory di installazione UnoArduSIm: 'Adafruit_LiquidCrystal.h',

'DFRobot_LiquidCrystal.h', e 'Native_LiquidCrystal.h',

Rispettivamente.



La Dispositivo può essere assegnato un indirizzo bus I2C utilizzando una voce di due esagoni-cifra nella sua edit-box 'Addr' (risponderà solo alle I2C le operazioni di autobus che coinvolgono l'indirizzo assegnato). Il Dispositivo riceve indirizzo bus e dati (e risponde con ACK = 0 o NAK = 1) sulla sua open-drain (pull-down-only) SDA pin. Puoi comandi LCD solo scrittura e dati DDRAM - si **non può** rilette i dati delle posizioni DDRAM scritte ..



Doppio click o tasto destro del mouse per aprire il monitor LCD finestra da cui è anche possibile impostare le dimensioni dello schermo e set di caratteri.

LCD di Testo SPI ('LCDSPI')

Questo 'I/O' Dispositivo emula un 1,2, 04 4-line personaggio-LCD, in uno dei due modi:

- a) aino (Porta SPI stile expander Adafruit)
- b) senza zaino (modalità nativa integrata SPI interfaccia - come mostrato di seguito)

Sostenere codice della libreria per ciascuna modalità Dispositivo è stata fornita all'interno della cartella

'include_3rdParty' della directory di installazione UnoArduSIm: 'Adafruit_LiquidCrystal.h', e

'Native_LiquidCrystal.h', Rispettivamente.



Pin 'SID' è dati seriali in, 'SS' è attivo basso Dispositivo-selezionare, 'SCK' è l'orologio pin, e 'RS' è il comando / dati pin. Puoi comandi LCD solo scrittura e dati DDRAM (tutte le transazioni sono SPI scrive) - **sinon può** rilette i dati delle posizioni DDRAM scritte.

Doppio click o tasto destro del mouse per aprire il monitor LCD finestra da cui è anche possibile impostare le dimensioni dello schermo e set di caratteri.



LCD di Testo D4 ('LCD_D4')

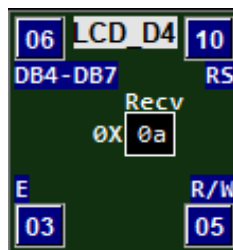
Questo 'I/O' Dispositivo emula un 1,2, 04 4-line carattere LCD avente un'interfaccia bus 4 bit-parallelo. byte di dati sono scritti / letti in **due metà** sui suoi dati 4 pins 'DB4-DB7' (in cui la casella di modifica contiene il *disponibilità numerati dei suoi 4 numeri pin consecutivi*), - dati un clock su fronti di discesa sul 'E' (abilitazione) pin, con direzione dei dati controllato dal 'R/W' pin, e dati LCD / modalità di comando dal 'RS' pin.

Supporto codice libreria è stata predisposta all'interno della

'include_3rdParty' cartella della directory di installazione UnoArduSIm:

'Adafruit_LiquidCrystal.h', , e

'Native_LiquidCrystal.h' sia il lavoro.

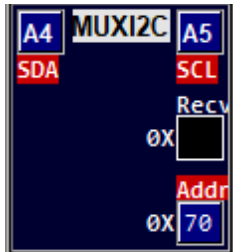


Doppio click o tasto destro del mouse per aprire il monitor LCD finestra da cui è anche possibile impostare le dimensioni dello schermo e set di caratteri.



Multiplexer DEL I2C ('MUXI2C')

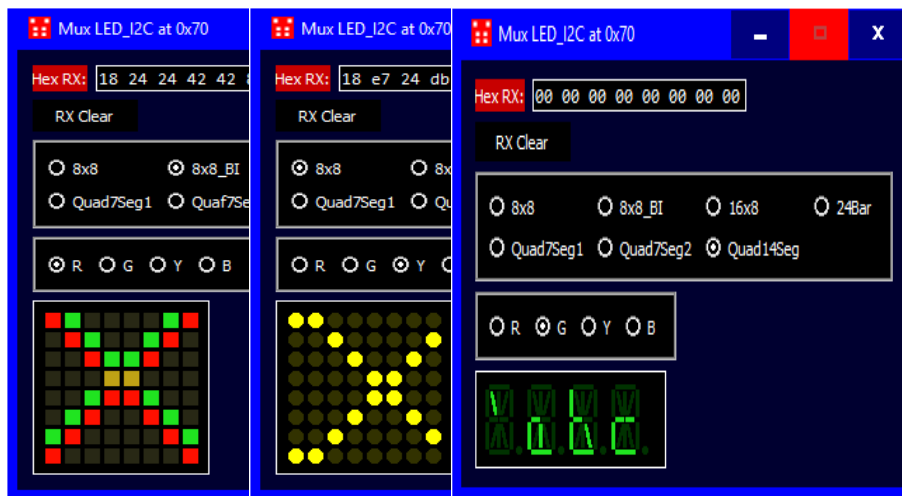
Questo 'I/O' Dispositivo emula un controller HT16K33 I2C-interfacciato (h I2C aving indirizzo bus 0x70-0x77) a cui uno dei diversi tipi di multiplati DEL display può essere attaccato:



- a) 8x8 o 16x8 DEL matrice
- b) 8x8 bicolore DEL matrice
- c) 24-bi-color-DEL bar
- d) due stili di 4 cifra display a 7 segmenti
- e) un 4-cifra 14 segmenti display alfanumerico

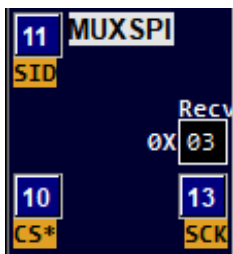
Tutti sono supportati dal 'Adafruit_LEDBackpack.h' codice previsto all'interno del 'include_3rdParty' cartella:

Doppio click (O tasto destro del mouse) per aprire una grande finestra di scegliere e vista uno dei numerosi DEL colorato visualizzata.

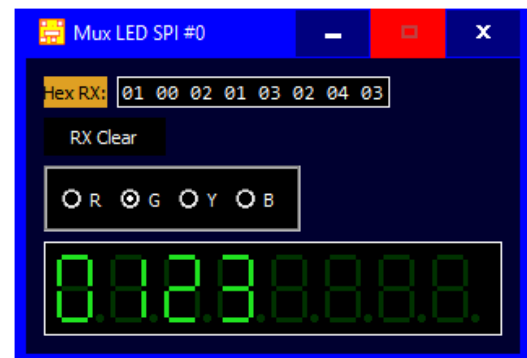


Multiplexer DEL SPI ('MUXSPI')

Un controller multiplexati-DEL basato sul MAX6219, con supporto 'MAX7219.h' il codice fornito all'interno della cartella 'include_3rdParty' per spingere fino a otto cifre a 7 segmenti.

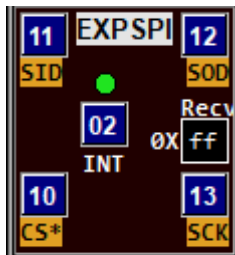


Doppio click (O tasto destro del mouse) per aprire una grande finestra vedere colorato 8-cifra visualizzazione 7-segment.

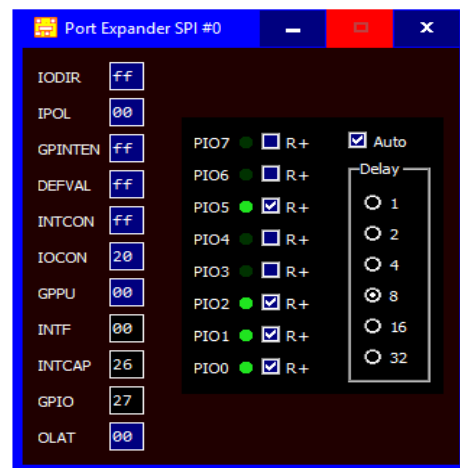


Porta di Espansione SPI ('EXPSPi')

Un 8-bit port expander basato sul MCP23008, con supporto 'MCP23008.h' codice previsto all'interno del 'include_3rdParty' cartella. È possibile scrivere a MCP23008 registri, e leggere di nuovo il GPIO pin livelli. Gli interrupt possono essere attivati su ogni cambiamento GPIO pin - un interrupt innescata sarà spingere il 'INT' pin.

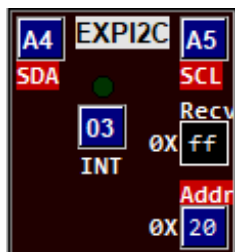


Doppio click (O tasto destro del mouse) aprire **una più grande finestra vedere il 8 linee di** porta GPIO, e le resistenze di pull-up allegate. È possibile modificare pull-up manualmente, facendo clic, o allegare un contatore che periodicamente li cambia in modo up-count. La velocità alla quale gli incrementi di conteggio è determinato dal ritardo scala-down fattore scelto dall'utente (fattore 1x corrisponde ad un incremento di circa ogni 30 millisecondi; fattori di ritardo superiori danno un lento up-count rate)

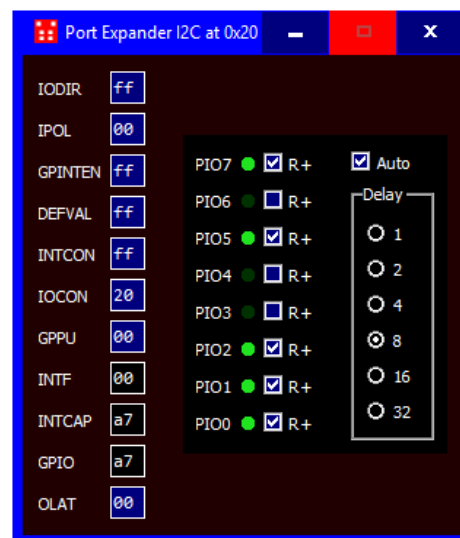


Porta di Espansione I2C ('EXPI2C')

Un 8-bit port expander basato sul MCP23008, con supporto 'MCP23008.h' codice previsto all'interno del 'include_3rdParty' cartella. Funzionalità corrispondono al 'EXPSPi' Dispositivo.

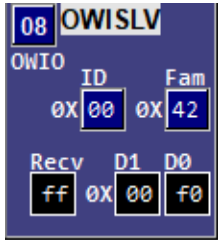


Doppio click (O tasto destro del mouse) per aprire una grande finestra come fro il 'EXPSPi' Dispositivo.



1-Wire' Slchiavo ('OWIISLV')

Questo 'I/O' Dispositivo emula uno di un piccolo set di bus '1-Wire' Dispositivi collegato a pin OWIO. È possibile creare un bus '1-Wire' (con uno o più di questi slave '1-Wire' Dispositivi) sul 'Uno' pin di propria scelta. Questa cabina Dispositivo può essere utilizzata chiamando il '**OneWire.h**' libreria moduli funzionali dopo aver posizionato un '**#include <OneWire.h>**' linea nella parte superiore del tuo programma. In alternativa puoi anche usare segnali bit-bang su OWIO su questo Dispositivo (anche se è molto complicato da fare correttamente senza causare un conflitto elettrico - tale conflitto è ancora possibile anche quando si usa il '**OneWire.h**' moduli funzionali, ma tali conflitti sono riportati in UnoArduSim).

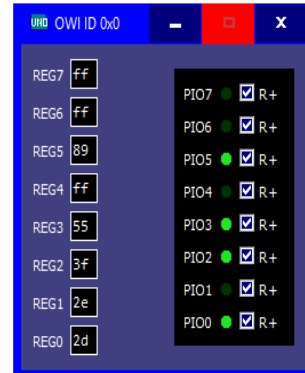
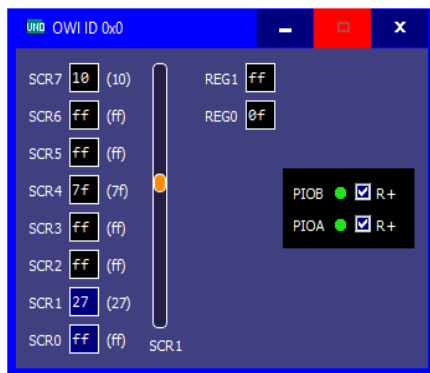


Ogni real-world OWISLV Dispositivo deve avere un unico 8 byte (64 bit!) numero seriale interno - in UnoArduSim questo è semplificato dall'utente che fornisce un breve esadecimale da 1 byte '**ID**' valore (che viene assegnato sequenzialmente per impostazione predefinita a Dispositivo carico / aggiunta), più il '**Fam**' Codice famiglia per quel Dispositivo. UnoArduSim riconosce un piccolo set di codici Famiglia a partire dalla V2.3 (0x28, 0x29, 0x3A, 0x42) che copre il sensore di temperatura, e il parallelo IO (PIO) Dispositivi (un codice famiglia non riconosciuto imposta il Dispositivo per diventare uno scratchpad generico da 8 byte Dispositivo con sensore generico).

Se la famiglia Dispositivo non ha registri PIO, registri **D0** e **D1** rappresentano i primi due byte del blocco note, altrimenti rappresentano il PIO Registro "status" (effettivi livelli pin) e registro dei dati di aggancio PIO pin, rispettivamente.

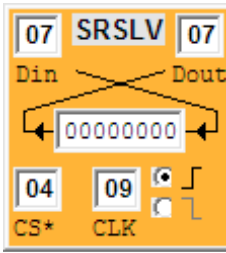
Di **doppio clic** (o **tasto destro del mouse**) sul Dispositivo, più grande **OWIMonitor** finestra è aperto. Da quel finestra più grande puoi controllare tutti i registri Dispositivo, cambiare le posizioni dei blocchi scratch SCR0 e SCR1 usando le modifiche e un cursore (di nuovo, SCR0 e SCR1 corrispondono solo a **D0** e **D1** se non è presente PIO) o impostare estensioni KIO pin esterne. Quando SCR0 e SCR1 vengono modificati, UnoArduSim memorizza questi valori modificati come "preferenza" dell'utente che rappresenta un valore iniziale (a partire da Resetare) che rappresenta l'output del valore firmato dal sensore Dispositivo; il cursore è resettato al 100% (un fattore di scala di 1,0) al momento della modifica. Quando il cursore viene successivamente spostato, il '**signed**' il valore in SCR1 viene ridimensionato in base alla posizione del cursore (fattore di scala da 1,0 a 0,0); la sua funzione consente di testare facilmente la risposta del tuo programma per modificare senza problemi i valori dei sensori. .

Per un Dispositivo con PIO pins, quando si impostano le caselle di controllo a livello pin, UnoArduSim memorizza questi valori controllati come gli attuali pull-up applicati esternamente ai pins. Questi valori di pull-up esterni vengono quindi utilizzati insieme ai dati del latch pin (registro **D1**) per determinare i livelli effettivi finali del pin e accendere o spegnere il DEL verde collegato al PIO pin (il pin va solo '**HIGH**' se viene applicato un pull-up esterno, **e** il corrispondente **D1** il chiavistello è un '**1**').



Registro di Cambio Schiavo ('SRSLV')

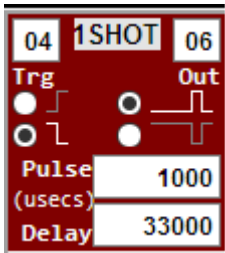
Questo 'I/O' Dispositivo emula un semplice registro a scorrimento Dispositivo con un basso attivo **SS *** ("slave-select") pin che controlla il '**Dout**' uscita pin (quando **SS *** è alto, '**Dout**' non è spinto). Il tuo programma potrebbe utilizzare la funzionalità del incassato SPI Arduino oggetto e la libreria. In alternativa, puoi scegliere di creare il tuo "bit-banged" '**Din**' e **CLK** segnala a spingere questo Dispositivo.



Il Dispositivo rileva le transizioni sui bordi **CLK** input che innescano lo spostamento del suo registro - la polarità del sensing **CLK** il bordo può essere scelto usando un controllo di pulsante radio. Su ogni **CLK** bordo (della polarità rilevata), il registro cattura il suo **frastuono** livello nella posizione del bit meno significativo (LSB) del registro a scorrimento, poiché i bit rimanenti vengono spostati simultaneamente a sinistra di una posizione verso la posizione MSB. Ogni volta **SS *** è basso, il valore corrente nella posizione MSB del registro a scorrimento è spinto su '**Dout**'.

Impulso Singolo ('1SHOT')

Questo 'I/O' Dispositivo emula uno scatto singolo digitale in grado di generare un impulso di polarità e ampiezza d'impulso scelte sul suo 'Out' pin, che si verifica dopo un ritardo specificato da un fronte di attivazione ricevuto sul suo **Trg** (trigger) input pin. Una volta ricevuto il fronte di attivazione specificato, sincronizzazione inizia e quindi un nuovo impulso di trigger non verrà riconosciuto fino a quando 'Out' l'impulso è stato prodotto (ed è completamente finito).



Un possibile utilizzo di questo Dispositivo è quello di simulare sensori di 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 i valori 'Delay' possono essere ridimensionati dal finestra principale **Strumento-Bar** Controllo del cursore fattore di scala 'I/O ____ S' aggiungendo il suffisso 'S' (o 's') a uno (o entrambi).

Un altro uso di questo Dispositivo è nel provare un programma che usa gli interrupt e ti piacerebbe vedere cosa succede se un **specifica istruzione programma** viene interrotto. Disconnetti temporaneamente l'I/O Dispositivo che hai collegato a pin 2 (o pin 3) e sostituiscilo con un '1SHOT' Dispositivo il cui 'Out' pin è collegato a 'Uno' pin 2 (o pin 3, rispettivamente), puoi quindi attivare il suo ingresso 'Trg' (supponendo che la sensibilità del fronte di salita sia impostata lì) inserendo la coppia di istruzioni { '**digitalWrite(LOW)**' , '**digitalWrite(HIGH)**' } **appena prima** all'istruzione all'interno della quale desideri che si verifichi l'interruzione. Impostare 1SHOT; s'Delay' per sincronizzare l'impulso prodotto su 'Out' all'interno dell'istruzione programma che segue questa coppia di istruzioni di attivazione. Si noti che alcune istruzioni mascherano gli interrupt (come '**SoftwareSerial.write(byte)**' , acosi non può essere interrotto.

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



Questo 'I/O' Dispositivo è in realtà un 'Uno' scheda spoglio che puoi programma (con un programma separato) per emulare un 'I/O' Dispositivo il cui comportamento è possibile definire completamente. È possibile scegliere fino a quattro pins (IO1, IO2, IO3 e IO4) che questo slave 'Uno' condividerà in comune con il master (principale) Uno che appare nel mezzo del proprio **Area del Banco da Laboratorio**. Come con altri Dispositivi, qualsiasi conflitto elettrico tra questo slave 'Uno' e il master 'Uno' verrà rilevato e contrassegnato. Si noti che tutti i pins condivisi sono **direttamente** collegato, **eccetto per pin 13** (dove si presuppone un resistore R-1K serie tra i due pins per impedire un conflitto elettrico a Resetare, questo 'Uno' slave non può

avere 'I/O' Dispositivi a parte. L'immagine a sinistra mostra le 4 connessioni pin come 4 pins del sistema SPI (**SS ***, **MISO**, **MOSI**, **SCK**) - questo ti permetterebbe di programma questo slave come uno SPI slave (o master) generico il cui comportamento tu definire.

Di **doppio clic** (o **tasto destro del mouse**) su questo Dispositivo, viene aperto un finestra più grande per dimostrare che questo slave 'Uno' ha il suo **Area di Codice** e associato **Area delle Variabili**, proprio come ha fatto il master 'Uno'. Ha anche il suo **Tool-Bar**, . Che puoi usare **caricare** e **controllo esecuzione** di uno slave programma - le azioni dell'icona hanno le stesse scorciatoie da tastiera di quelle del finestra principale. (**Caricare** è **Ctrl-L** , **Salvare** è **Ctrl-S** eccetera.). Una volta caricato, puoi eseguire da o il Main UnoArduSim finestra, o dall'interno di questo Schiavo

Monitor Slchiavo - in entrambi i casi il Main 'Uno' programma e il Slchiavo 'Uno' programma rimangono bloccati in sincronizzazione al passaggio del tempo reale mentre le loro esecuzioni avanzano. **Per scegliere il Area di Codice che avrà il caricamento, ricerca e messa a fuoco esecuzione**, clic sopra **la sua barra del titolo finestra - il Area di Codice non attivo ha quindi la sua barra degli strumenti azioni in grigio**.

Alcune idee possibili per lo slave Dispositivi che potrebbe essere programmato in questo 'PROGIO' Dispositivo sono elencate di seguito. Per l'emulazione seriale, I2C o SPI Dispositivo, è possibile utilizzare la codifica programma appropriata con matrici per i buffer di invio e ricezione, in ordine emulare il comportamento complesso di Dispositivo:

a) Un SPI master o slave Dispositivo. UnoArduSimV2.4 ha esteso il '**SPI.h**' libreria per consentire la modalità slave S {operazione PI attraverso un opzionale '**mode**' parametro in '**SPI.begin(int mode = SPI_MASTR)**'. Passa esplicitamente '**SPI_SLV**' per selezionare la modalità slave (invece di fare affidamento sulla modalità principale predefinita). Ora puoi anche definire un interrupt utente modulo funzionale (chiamiamolo '**onSPI**') in 'Uno' per trasferire i byte chiamando un'altra estensione aggiunta '**SPI.attachInterrupt(user_onSPI)**'. **Ora c** Allung '**rxbyte=SPI.transfer(tx_byte)**' da dentro il tuo '**user_onSPI**' modulo funzionale cancellerà la flag di interrupt e will **ritorno subito** con il byte appena ricevuto nel tuo variabile '**rxbyte**'. In alternativa, puoi evitare di attaccare un'interruzione SPI, e invece semplicemente chiama '**rxbyte=SPI.transfer(tx_byte)**' dall'interno del tuo programma principale, quella chiamata lo farà **bloccare esecuzione** fino a quando un byte SPI è stato trasferito, e sarà poi **ritorno** con il byte appena ricevuto all'interno '**rxbyte**'.

b) Un generico **seriale 'I/O'** Dispositivo. Avrai bisogno di comunicare tra uno '**SoftwareSerial**' definito all'interno del tuo master 'Uno' programma e un altro creato all'interno dello slave 'Uno' programma - questi devono usare **in modo opposto** definito '**txpin**' e '**rxpin**' valori in modo che lo slave 'Uno' '**SoftwareSerial**' riceve sul pin su cui il master '**SoftwareSerial**' trasmette (Non è possibile stabilire una connessione simile usando il '**Serial**' porta su pins 0 e 1 su entrambe le schede, come i due '**Serial**' i flussi voluto spingere i loro segnali TX l'uno contro l'altro).

c) Un generico 'I2C' master o slave Dispositivo. L'operazione Slchiavo è stata ora aggiunta per completare l'implementazione di UnoArduSim '**Wire.h**' libreria (moduli funzionali '**begin(address)**', '**onReceive()**' e '**onRequest**' sono stati ora implementati per supportare le operazioni in modalità slave).

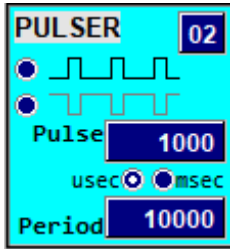
d) Un digitale generico **Pulser**. utilizzando '**delayMicroseconds()**' e '**digitalWrite()**' chiama dentro '**loop()**' nel tuo 'PROGIO' programma, puoi definire il '**HIGH**' e '**LOW**' intervalli di un treno di impulsi. Aggiungendo un separato '**delay()**' chiama dentro la tua '**setup()**' modulo funzionale, è possibile ritardare l'inizio di questo treno di impulsi. È anche possibile variare le larghezze dell'impulso col passare del tempo usando un contatore variabile. Puoi anche usare un separato '**IOx**' pin come trigger per avviare la sincronizzazione di un '1Shot' emulato (o doppio colpo, triplo scatto, ecc.) Dispositivo, ed è possibile controllare la larghezza dell'impulso prodotto per cambiare in qualsiasi modo si desideri con il passare del tempo.

e) Un segnalatore casuale. Questo è una variazione su a digitale **Pulser** che usa anche le chiamate a '**random()**' e '**delayMicroseconds()**' generare tempi casuali in cui a '**digitalWrite()**' un segnale su qualsiasi pin scelto condiviso con il master 'Uno'. Usando tutti e quattro '**IOx**' pins consentirebbe quattro segnali simultanei (e unici).

d) Un generico 'bit-banging' Dispositivo. Crea tutti i pattern di bit o di clock che desideri trasmettere ai segnali spingere su qualsiasi sottosistema sul master 'Uno'. E ricorda, **puoi usare qualsiasi schiavo 'Uno' programma istruzioni o sottosistemi che desideri**.

Generatore di Impulsi ('PULSER')

Questo 'I/O' Dispositivo emula un semplice digitale impulso generatore forma d'onda che produce un segnale periodico che può essere applicato a qualsiasi scelta 'Uno' pin.



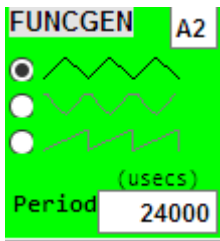
Le larghezze di periodo e ad impulsi (in microsecondi) possono essere impostati tramite modifica-scatole-periodo minimo consentito è 50 microsecondi, e l'impulso di larghezza minima è di 10 microsecondi. Si può scegliere tra i valori sincronizzazione in microsecondi ('usec') e millisecondi ('msec'), e questa scelta verranno salvate insieme con gli altri valori quando si 'Save' dalla Configurare | I dialogo / O Dispositivi.

La polarità può anche essere scelta: o impulsi positivi all'avanguardia (da 0 a 5 V) o negativi punta impulsi (5V a 0V).

valori 'Pulse' 'Period' e può essere scalata dalla principale **Strumento-Bar** 'I/O____S' fattore di scala del cursore aggiungendo come suffisso 'S' (o 's') per uno dei due (o entrambi). Questo permette poi di modificare il 'Pulse' o il valore 'Period' **dinamicamente** durante esecuzione.

Analogico Generatore di Funzioni ('FUNCGEN')

Questo 'I/O' Dispositivo emula un semplice generatore analogico forma d'onda 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. Il forma d'onda che crea può essere scelto come sinusoidale, triangolare o seghettato (per creare un'onda quadra, utilizzare invece un 'PULSER'). In periodi più brevi, vengono utilizzati meno campioni per ciclo per modellare il forma d'onda prodotto (solo 4 campioni per ciclo al periodo = 100 microsecondi).

'Period' il valore può essere ridimensionato dal finestra principale **Strumento-Bar** Controllo slider fattore di scala 'I/O____S' aggiungendo come suffisso la lettera 'S' (o 's').

Motore Passo Passo ('STEPR')

Questo 'I/O' Dispositivo emula un motore passo-passo bipolare o unipolare da 6 V con un controller controllore integrato spinto di **entrambi due** (sopra **P1** , **P2**) o **quattro** (sopra **P1** , **P2** , **P3** , **P4**) segnali di controllo. È inoltre possibile impostare il numero di passaggi per giro. Puoi usare il '**Stepper.h**' moduli funzionali '**setSpeed()**' e '**step()**' a spingere il 'STEPR'. In alternativa, lo farà 'STEPR' *anche rispondere* al tuo '**digitalWrite()**' " segnali "spingere" bitorzoluti.



Il motore è accuratamente modellato sia meccanicamente che elettricamente. Le cadute di tensione del motore controllore e la riluttanza e l'induttanza variabili sono modellate insieme a un momento di inerzia realistico rispetto alla coppia di ritenuta. L'avvolgimento del motore ha una resistenza modellata di $R = 6 \text{ ohm}$ e un'induttanza di $L = 6 \text{ milli-Henries}$ che crea una costante di tempo elettrica di 1,0 millisecondi. A causa della modellazione realistica si noterà che gli impulsi pin di controllo molto stretti *non si ottiene* il motore da calpestare, sia per il tempo di salita della corrente finito, sia per l'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**) motore controllore tra i fili del motore

e il 'Uno'!

Uno sfortunato errore nell'Arduino '**Stepper.h**' codice della libreria significa che al reset il motore Stepper non sarà in Passo posizione 1 (di quattro passi). Per superare questo, l'utente dovrebbe usare '**digitalWrite()**' in suo / a '**setup()**' routine per inizializzare i livelli di controllo pin su '**step(1)**' livelli appropriati per il controllo 2-pin (0,1) o 4-pin (1,0,1,0) e consentono al motore di muoversi di 10 millisecondi nella posizione motore iniziale di riferimento di 12-mezzogiorno desiderata.

AS di V2.6, questo Dispositivo ora include un 'sync' DEL (verde per sincronizzato, o rosso quando fuori da una o più fasi) .Inoltre, oltre al numero di passi per giro, due valori aggiuntivi (nascosti) possono facoltativamente specificata nella IODEvs.txt file per specificare il load meccanico per esempio, i valori 20, 50, 30 precisa 20 gradini per giro, un

momento d'inerzia del carico 50 volte quella del rotore motore stesso, ed una coppia di carico del 30 per cento di coppia completa presa del motore.

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

Pulsata Motore Passo Passo ('PSTEPR')

Questo 'I/O' Dispositivo emula un 6V **micro-stepping** bipolare Motore Passo Passo con un controller integrato controllore spinto da un **'Step' pulsato** pin, un attivo basso **'EN*'** (Abilitazione) pin, e un **'DIR'** (direzione) pin. Il numero di passi completi per rotazione può anche essere impostato direttamente, insieme al numero di micro-passi per intero stadio (1,2,4,8, o 16). In aggiunta a queste impostazioni, due valori aggiunti (nascosti) può opzionalmente specificata nella IODevs.txt file per specificare il load meccanico per esempio, i valori 20, 4, 50, 30 precisa 20 gradini per giro, 4 micro-passi al passo pieno, un momento d'inerzia del carico 50 volte quella del motore rotore stesso, ed una coppia di carico del 30 per cento della coppia piena presa del motore.

È necessario scrivere il codice per il controllo spingere pins appropriato.

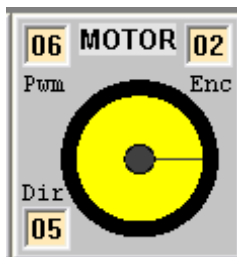


Il motore viene accuratamente modellata sia meccanicamente che elettricamente. Tensione motore-controller gocce e variando riluttanza e induttanza sono modellati con un momento di inerzia realistico rispetto alla coppia di tenuta. Il rotore avvolgimento del motore ha una resistenza modellato di $R = 6$ ohm, e un'induttanza di $L = 6$ milli-Henries che crea una costante di tempo elettrica di 1,0 millisecondi.

Questo Dispositivo comprende un'attività 'STEP' giallo DEL, e un 'sync' DEL (verde per sincronizzato, o rosso quando fuori da uno o più stadi).

DC Motore ('MOTOR')

Questo 'I/O' Dispositivo emula un motore CC a ingranaggi 100: 1 con alimentazione a 6 volt con un controller controllore integrato spinto mediante un segnale di modulazione della larghezza dell'impulso (sul suo **Pwm** input) e un segnale di controllo della direzione (sul suo **dir** ingresso). Il motore ha anche un'uscita encoder ruota che spinge è suo **Enc** uscita pin. Puoi usare `'analogWrite()'` a spingere il **Pwm** pin con un 490 Hz (su pins 3,9,10,11) o 980 Hz (su pins 5,6) PWM forma d'onda di duty cycle compreso tra 0,0 e 1,0 (`'analogWrite()'` valori da 0 a 255). In alternativa, 'MOTOR' lo farà *anche rispondere* al tuo `'digitalWrite()'` "segnali" spingere" bitorzoluti.



Il motore è accuratamente modellato sia meccanicamente che elettricamente.

Contabilizzando le cadute di tensione del transistor del motore controllore e la coppia di ingranaggi realistica senza carico si ottiene 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 rotore 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. A causa della modellazione realistica, noterete che gli impulsi PWM molto stretti *non si ottiene* il motore da girare - sia per il tempo di salita della corrente finito, sia per il 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 causerà il distacco del motore - questo concorda con quanto osservato quando si guida un motoriduttore reale da un 'Uno' con, ovviamente, un appropriato (**e richiesto**) modulo motore controllore tra il motore e 'Uno'!

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

ServoMotore ('SERVO')

Questo 'I/O' Dispositivo emula un servomotore CC a 6 volt PWM-spinto con posizione controllata. 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 millisecondi. Se la in basso a sinistra la casella di controllo è selezionata, il servo diventa a **continua rotazione** servo con la stessa velocità massima, ma ora la larghezza dell'impulso PWM imposta il **velocità** piuttosto che l'angolo



Il tuo programma deve avere un `'#include <Servo.h>'` linea prima di dichiarare il tuo `'Servo'` esempio (s) *se si sceglie di utilizzare la funzionalità della libreria 'Servo.h'*, per esempio `'Servo.write()'`, `'Servo.writeMicroseconds()'` In alternativa, 'SERVO' risponde anche a `'digitalWrite()'` Segnali "bit-banged". A causa dell'implementazione interna di UnoArduSim, sei limitato a 6 'SERVO' Dispositivi.

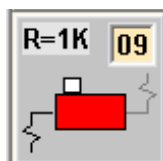
Altoparlante Piezoelettrico ('PIEZO')



Questo Dispositivo consente di "ascoltare" i segnali su qualsiasi 'Uno' pin scelto e può essere un utile complemento ai LED per il debug della tua operazione programma. Puoi anche divertirti un po' suonando le suonerie in modo appropriato `'tone()'` e `'delay()'` chiama (anche se non c'è alcun filtro del forma d'onda rettangolare, quindi non sentirai le note "pure").

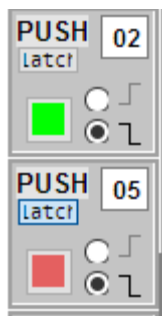
È inoltre possibile ascoltare un 'PULSER' o 'FUNCGEN' Dispositivo collegato collegando un 'PIEZO' al pin con Dispositivo spinge.

Resistenza di scorrimento ('R=1K')



Questo Dispositivo consente all'utente di connettersi a un 'Uno' pin o a un resistore di pull-up da 1 k-ohm a +5 V o a un resistore di pull-down da 1 k-ohm a terra. Ciò consente di simulare i carichi elettrici aggiunti a un vero hardware Dispositivo. Cliccando con il tasto sinistro sull'interruttore scorrevole **corpo** 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.

Pulsante ('PUSH')



Questo 'I/O' Dispositivo emula normalmente aperto **momentaneo o fermo** pulsante unipolare, a lancio singolo (SPST) con resistenza di pull-up (o pull-down) da 10 k-ohm. Se per il Dispositivo viene scelta una selezione di transizione del fronte di salita, i contatti del pulsante saranno cablati tra Dispositivo pin 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 saranno cablati tra Dispositivo pin e massa, con un pull-up da 10 k-Ohm a + 5V.

Facendo clic con il tasto sinistro sul pulsante o premendo un tasto qualsiasi, si chiude il contatto pulsante. Nel **momentaneo** modo, rimane chiuso fino a quando si tiene premuto il tasto o il tasto del mouse e in **chiavistello** modalità (abilitato facendo clic su 'latch' pulsante) 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 utilizzare il **barra spaziatrice** per premere il pulsante.

DEL colorato ('LED')



È possibile collegare un DEL tra l'Uno' pin scelto (tramite un resistore di limitazione di corrente incassato serie 1 nascosto da 1 k-Ohm) a terra o a +5 V: questo ti dà la possibilità di accendere il DEL quando il 'Uno' pin collegato è 'HIGH' o invece, quando è 'LOW'.
Il colore DEL può essere scelto per essere rosso ('R'), giallo ('Y'), verde ('G') o blu ('B') utilizzando la sua casella di modifica.

4-DEL Row ('LED4')



Puoi collegare questa fila di 4 LED colorati tra il set scelto di 'Uno' pins (ognuno ha una resistenza nascosta incassato serie 1 resistore di corrente k-Ohm) a terra o a + 5V - questo ti dà la possibilità di avere la luce dei LED quando è collegato il 'Uno' pin 'HIGH' o invece, quando è 'LOW'.

Il '1of4' La casella di modifica pin accetta un singolo numero pin che verrà preso in considerazione *il primo di quattro consecutivi* 'Uno' pins che si collegherà ai 4 LED.

Il colore DEL ('R', 'Y', 'G' o 'B') è un **opzione nascosta** Quello può essere **solo essere scelto da modificare il IODevices.txt file** (quale puoi creare usando **Salvare** dal **Configurare | I/O Dispositivi** la finestra di dialogo).

7 segmenti DEL Cifra ('7SEG')



È possibile collegare questo display Cifra DEL a 7 segmenti a a insieme scelto di **quattro 'Uno' pins consecutivi che danno il codice esadecimale** per il cifra visualizzato desiderato, (da '0' a 'F') e attivare o disattivare questo cifra utilizzando il CS * pin (attivo-BASSO per ON).

Questo Dispositivo include un decodificatore incassato che usa il **attivo alto** livelli sui quattro consecutivi '1of4' pins per determinare il esadecimale richiesto cifra da visualizzare. Te livello sul più basso numero pin (quello visualizzato nel '1of4' casella di modifica) rappresenta il bit meno significativo del codice esadecimale a 4 bit.

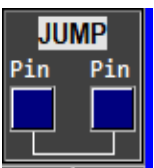
Il colore dei segmenti DEL ('R', 'Y', 'G' o 'B') è un **opzione nascosta** Quello può essere **solo essere scelto da modificare il IODevices.txt file** puoi creare usando **Salvare** dal **Configurare | I/O Dispositivi** la finestra di dialogo.

Cursore Analogico

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



Pin Ponticello ('JUMP')



Puoi connetterne due 'Uno' pins insieme usando questo Dispositivo (se viene rilevato qualsiasi conflitto elettrico quando si inserisce il secondo numero pin, la connessione scelta non è consentita e pin è disconnesso).

Questo jumper Dispositivo ha un'utilità limitata ed è molto utile se combinato con interrupt, per test programma, sperimentazione e scopi di apprendimento. **A partire da UnoArduSim V2.4 potresti trovare che usare un 'PROGIO' Dispositivo offre invece una maggiore flessibilità rispetto ai**

metodi interrupt-spinto sotto.

Tre possibili usi di questo Dispositivo sono i seguenti:

- 1) Puoi **crea un input digitale per testare il tuo programma** che ha una sincronizzazione più complessa di quella che può essere prodotta usando una qualsiasi delle serie di set fornite standard 'I/O' Dispositivi, come

segue:

Definisci un interrupt modulo funzionale (chiamiamolo `'myIntr'`) e fare `'attachInterrupt(0, myIntr, RISING)'` dentro il tuo `'setup()'`. Connetti a **Pulser** Dispositivo a pin2 - ora `'myIntr()'` sarà eseguire ogni volta a **Pulser** si verifica un fronte di salita. Il tuo `'myIntr()'` modulo funzionale può essere un algoritmo con programmato (usando il contatore globale variabili e forse anche `'random()'`) per produrre un forma d'onda di tuo disegno su qualsiasi disponibile `'OUTPUT'` pin (diciamo che è pin 9). Adesso **SALTARE** pin 9 sul 'Uno' desiderato 'INPUT' pin per applicare quello che ha generato digitale forma d'onda a quell'input pin (per testare la risposta del tuo programma a quel particolare forma d'onda). È possibile generare una sequenza di impulsi, o caratteri seriali, o semplicemente transizioni di bordo, tutta la complessità arbitraria e intervalli variabili. Si prega di notare che se il tuo principale programma chiama `'micros()'` (o chiama qualsiasi modulo funzionale che si basa su di esso), suo `'return'` valore **sarà aumentato** dal tempo trascorso dentro il tuo `'myIntr()'` modulo funzionale ogni volta che scatta l'interruzione. È possibile produrre una rapida raffica di spigoli accuratamente sincronizzati utilizzando le chiamate a `'delayMicroseconds()'` da dentro `'myIntr()'` (forse per generare un intero **byte** di una high-velocità baud transfer), o semplicemente generare una transizione per interruzione (forse per generare **un bit** di un trasferimento low-velocità baud) con il **Pulser** Dispositivo **'Period'** scelto appropriato per le tue esigenze sincronizzazione (ricordalo **Pulser** limita il suo minimo **'Period'** a 50 microsecondi).

2) Puoi *sperimentare i loopback del sottosistema*:

Ad esempio, scollegare il 'SERIAL' 'I/O' Dispositivo TX '00' pin (modificarlo in bianco), e poi **SALTARE** 'Uno' pin '01' di nuovo a 'Uno' pin '00' emulare un loopback hardware dell'ATmega `'Serial'` sottosistema. Ora nel tuo test programma, dentro `'setup()'` fare a **singolo** `'Serial.print()'` di una parola o carattere, e dentro il tuo `'loop()'` echo indietro qualsiasi carattere ricevuto (quando `'Serial.available()'`) facendo a `'Serial.read()'` seguito da a `'Serial.write()'` e poi guarda cosa succede. Si potrebbe osservare che un simile `'SoftwareSerial'` loop-back **avrà esito negativo** (come nella vita reale - il software non può fare due cose contemporaneamente).

Puoi anche provare **SPI** loop-back usando a **SALTARE** per collegare pin 11 (MOSI) a pin 12 (MISO).





3) Puoi *conta il numero e / o misura la spaziatura di transizioni di livello specifiche su qualsiasi 'Uno' uscita pin X* che si verificano come risultato di un complesso Istruzione o libreria Arduino modulo funzionale (come esempi: `'analogWrite()'`, o `'OneWire::reset()'`, o `'Servo::write()'`), come segue:

SALTARE pin X interrompere pin 2 e dentro il tuo `'myIntr()'` usare un `'digitalRead()'` e a `'micros()'` chiamata, e confronta con livelli e tempi salvati (da interruzioni precedenti). È possibile modificare la sensibilità del bordo per il successivo interrupt, se necessario, utilizzando `'detachInterrupt()'` e `'attachInterrupt()'` a partire dal **dentro** il tuo `'myIntr()'`. Nota che non sarai in grado di monitorare pin transizioni che si verificano troppo da vicino (più vicino del tempo totale di esecuzione di il tuo `'myIntr()'` modulo funzionale), come quelli che si verificano con trasferimenti I2C o SPI o con velocità baud alto `'Serial'` trasferimenti (anche se l'interrupt modulo funzionale non disturba il sincronizzazione inter-edge di questi trasferimenti prodotti via hardware). Si noti inoltre che i trasferimenti mediati dal software (come `'OneWire::write()'` e `'SoftwareSerial::write()'`) siamo deliberatamente protetto da interruzioni (con il loro codice di libreria che disabilita temporaneamente tutti gli interrupt, al fine di evitare interruzioni di sincronizzazione), quindi non è possibile misurare all'interno di quelli che utilizzano questo metodo.






Sebbene tu possa invece fare queste stesse misure di spaziatura dei bordi **visivamente** in un **Forme d'onda Digitale** finestra, se ti interessa la spaziatura minima o massima su un gran numero di transizioni, o nel conteggio delle transizioni, facendo così usando questo `'myIntr()'` -più- **SALTARE** la tecnica è più conveniente. E tu puoi misurare, ad esempio, le variazioni di spaziatura delle transizioni prodotte dal programma principale (a causa dell'effetto del tuo software che richiede vari percorsi esecuzione di diversi tempi esecuzione), fare un genere di programma "profilazione".

Menu









File:

<u>Caricare INO o PDE Prog (Ctrl-L)</u> 	Consente all'utente di scegliere un file file con l'estensione selezionata. Il programma riceve immediatamente un Analizzare
<u>Modificare/Esaminare (Ctrl-E)</u>	Apri il programma caricato per la visualizzazione / modifica.
<u>Salvare</u> 	Salvare il contenuto modificato del programma torna al programma originale file.
<u>Salvare Come</u>	Salvare il contenuto modificato di programma con un diverso nome file.
<u>Il prossimo ('#include')</u> 	Anticipa il Area di Codice per visualizzare il prossimo ' #include ' file
<u>Precedente</u> 	Restituisce il Area di Codice visualizzare il file precedente
<u>Esci</u>	Esce da UnoArduSim dopo aver ricordato all'utente di salvare qualsiasi file modificato.

Trova:

<u>Salire Mucchio di Chiamate</u> 	Passa alla funzione chiamante precedente nello stack di chiamate: il Area delle Variabili si adatterà a quella funzione
<u>Scendere Mucchio di Chiamate</u> 	Passa alla funzione chiamata successiva nello stack di chiamate: il Area delle Variabili si adatterà a quella funzione
<u>Imposta il testo Cerca (Ctrl-F)</u> 	Attiva il Strumento-Bar Trova edit-box per definire il testo da cercare successivamente (e aggiunge la prima parola dalla riga attualmente evidenziata nel Area di Codice o Area delle Variabili se uno di quelli lo ha l'attenzione).
<u>Trova Next Text</u> 	Passa alla successiva ricorrenza di testo nel Area di Codice (se ha il focus attivo), o alla successiva occorrenza del testo nel Area delle Variabili (se invece ha il focus attivo).
<u>Trova Testo precedente</u> 	Passa alla ricorrenza di testo precedente in Area di Codice (se ha il focus attivo), o alla precedente ricorrenza del testo nel Area delle Variabili (se invece ha il focus attivo).

Eseguire:

<u>Passo In (F4)</u>		Passa esecuzione avanti di una istruzione, o <i>in un modulo funzionale chiamato</i> .
<u>Passo Scavalcare (F5)</u>		Passa esecuzione avanti di una istruzione, o <i>con una chiamata completa modulo funzionale</i> .
<u>Passo Fuori (F6)</u>		Anticipo esecuzione di <i>quel tanto che basta per lasciare l'attuale modulo funzionale</i> .
<u>Eseguire Verso (F7)</u>		Esegue il programma, <i>fermandosi sulla linea programma desiderata</i> - devi prima fare clic su evidenziare sulla linea programma desiderata prima di utilizzare Eseguire Verso.
<u>Eseguire Fino A (F8)</u>		Esegue il programma finché non si verifica una scrittura sul variabile che aveva l'attuale evidenziare nel Area delle Variabili (fare clic su uno per stabilire l'iniziale evidenziare).
<u>Eseguire (F9)</u>		Esegue il programma.
<u>Arresto (F10)</u>		Arresta programma esecuzione (<i>e congela il tempo</i>).
<u>Resettare</u>		Reimposta il programma (tutti i valori-variabili vengono reimpostati sul valore 0 e tutti i puntatori-variabili vengono reimpostati su 0x0000).
<u>Animare</u>		Passa automaticamente le linee programma consecutive <i>con ritardo artificiale aggiunto</i> e l'evidenziazione dell'attuale riga di codice. Le operazioni e i suoni in tempo reale sono persi.
<u>Rallentatore</u>		Rallenta il tempo di un fattore di 10.

Opzioni:

<u>Passo Scavalcare Structors/ Operators</u>	Vola attraverso costruttori, distruttori e sovraccarico dell'operatore moduli funzionali durante qualsiasi passo (cioè non si fermerà all'interno di questi moduli funzionali).
<u>Registrati-assegnazione</u>	Assegnare i locals modulo funzionale ai registri ATmega libero invece che allo stack (genera un utilizzo della RAM piuttosto ridotto).
<u>Errore su non inizializzato</u>	Segnala come errore Analizzare ovunque il tuo programma tenti di usare un variabile senza aver prima inizializzato il suo valore (o almeno un valore all'interno di un matrice).
<u>Aggiunto 'loop()' 'Ritardo</u>	Aggiunge 1000 microsecondi di ritardo ogni volta 'loop()' viene chiamato (nel caso in cui non ci siano altre chiamate programma a 'delay()' ovunque) - utile per cercare di evitare di cadere troppo indietro rispetto al tempo reale.
<u>Consenti interrupt annidati</u>	Permetti di riattivare con 'interrupts()' dall'interno di una routine di servizio di interruzione utente.

Configurare:

<u>'I/O' Dispositivi</u>	Aprire una finestra di dialogo per consentire all'utente di scegliere il tipo (o i tipi) e i numeri desiderati di 'I/O' Dispositivi. Da questa finestra di dialogo puoi anche Salvare 'I/O' Dispositivi a un testo file e / o Caricare 'I/O' Dispositivi da un testo file precedentemente salvato (o modificato) (incluse tutte le connessioni pin e le impostazioni selezionabili e i valori digitati)
<u>Preferenze</u>	Aprire una finestra di dialogo per consentire all'utente di impostare le preferenze tra cui il rientro automatico delle linee programma di origine, consentendo la sintassi Esperto, la scelta del carattere carattere tipografico, optando per una dimensione di carattere maggiore, rafforzando i limiti matrice, permettendo le parole chiave dell'operatore logico, mostrando programma scaricamento , scelta della versione 'Uno' scheda e lunghezza del buffer TWI (per I2C Dispositivi).

VarAggiorna:

<u>Consenti Auto (-) Contrarsi</u>	Consenti a UnoArduSim di contrarsi di visualizzare espansi matrici / oggetti quando si trova in ritardo rispetto al tempo reale.
<u>Minimo</u>	Aggiorna solo il Area delle Variabili mostra 4 volte al secondo.
<u>Evidenziare I cambiamenti</u>	Evidenziare ha modificato i valori variabile durante l'esecuzione (può causare rallentamenti).

Finestre:

<u>'Serial' Monitor</u>	Collegare un I / O seriale Dispositivo a pins 0 e 1 (se nessuno) e sollevare un cavo più grande 'Serial' monitorare il testo TX / RX finestra.
<u>Ripristinare tutto</u>	Ripristina tutto il bambino ridotto al minimo finestre.
<u>Forme d'Onda Digitali</u>	Ripristina un Forme d'Onda Digitali Arresto ridotto al minimo.
<u>Forma d'Onde Analogica</u>	Ripristina un Forma d'Onde Analogica Forma d'Onde Analogica ridotto al minimo.

Aiuto:

<u>Aiuto veloce File</u>	Aprire il file OneArduSim_QuickHelp PDF file.
<u>Aiuto completo File</u>	Aprire il file OneArduSim_FullHelp PDF file.
<u>Correzioni Errore</u>	Visualizza le correzioni significative di errore dalla versione precedente.
<u>Modifica / Miglioramenti</u>	Visualizza modifiche e miglioramenti significativi rispetto alla versione precedente.
<u>Di</u>	Visualizza versione, copyright.

'Uno' Scheda e 'I/O' Dispositivi

L'"Uno" e l'"I/O" Dispositivi collegato sono tutti modellati elettronicamente, e sarete in grado di ottenere una buona idea a casa di come il vostro programmi si comporterà con l'hardware attuale, e tutti i pin elettrici conflitti saranno contrassegnati.

Sincronizzazione

UnoArduSim esegue abbastanza rapidamente su un PC o tablet che può (*nella maggior parte dei casi*) modellare le azioni di programma in tempo reale, **ma solo se incorpora il tuo programma** almeno un po' piccolo 'delay()' chiamate o altre chiamate che lo terranno naturalmente sincronizzato in tempo reale (vedi sotto).

Per fare ciò, UnoArduSim utilizza un timer di callback Finestre modulo funzionale, che consente di tenere traccia accurata del tempo reale. Il esecuzione di un numero di istruzioni programma viene simulato durante una porzione di timer e le istruzioni che richiedono un esecuzione più lungo (come le chiamate a 'delay()' potrebbe essere necessario utilizzare più sezioni del timer. Ogni iterazione del timer di callback modulo funzionale corregge l'ora del sistema utilizzando l'orologio hardware del sistema in modo tale che programma esecuzione sia costantemente regolato per rimanere in lock-step in tempo reale. *Le uniche volte il tasso esecuzione **dovere restare indietro in tempo reale*** è quando l'utente ha creato loop stretti **senza alcun ritardo aggiunto** oppure 'I/O' Dispositivi sono configurati per funzionare con frequenze 'I/O' Dispositivo molto elevate (e / o velocità baud) che genererebbero un numero eccessivo di eventi di cambiamento di livello pin e un sovraccarico di elaborazione associato. UnoArduSim affronta questo sovraccarico saltando alcuni intervalli del timer per compensare, e questo rallenta la progressione del programma in **sotto il tempo reale** .

Inoltre, programmi con grande matrici viene visualizzato, o ancora con anelli stretti **senza alcun ritardo aggiunto** può causare un'alta frequenza di chiamata modulo funzionale e generare un massimo **Area delle Variabili** mostra il carico di aggiornamento provocando un ritardo in tempo reale - UnoArduSim riduce automaticamente la frequenza di aggiornamento variabile per cercare di tenere il passo, ma quando è necessaria una riduzione ancora maggiore, scegli **Minimo**, dal **VarAggiorna** menu per specificare solo quattro aggiornamenti al secondo.

Modellazione accurata del tempo esecuzione al millisecondo per ogni istruzione o operazione programma **non è fatto** - solo stime molto approssimative per la maggior parte sono state adottate a fini di simulazione. Tuttavia, il sincronizzazione di 'delay()' , e 'delayMicroseconds()' moduli funzionali e moduli funzionali 'millis()' e 'micros()' sono tutti perfettamente accurati, e **a patto che si utilizzi almeno uno dei delay moduli funzionali** in un loop da qualche parte nel tuo programma, o usi un modulo funzionale che si lega naturalmente al funzionamento in tempo reale (come 'print()' che è legato al velocità baud scelto), quindi le prestazioni simulate del tuo programma saranno molto vicine al tempo reale (di nuovo, salvo eventi di cambiamento di livello pin ad alta frequenza palesemente eccessivi o eccessivi aggiornamenti Variabili permessi dall'utente che potrebbero rallentarlo).

Per vedere l'effetto delle singole istruzioni programma w *gallina in esecuzione* , potrebbe essere desiderabile essere in grado di rallentare le cose. Un fattore di rallentamento temporale di 10 può essere impostato dall'utente nel menu **Eseguire** .

'I/O' Dispositivo Sincronizzazione

Questi Dispositivi virtuali ricevono segnalazioni in tempo reale delle modifiche che si verificano sul loro ingresso pins e producono uscite corrispondenti sulla loro uscita pins che possono essere rilevate dall'"Uno" - sono quindi intrinsecamente sincronizzate con programma esecuzione. 'I/O' interno Dispositivo sincronizzazione viene impostato dall'utente (ad esempio tramite selezione velocità baud o Frequenza di clock) e gli eventi del simulatore sono impostati per tracciare il funzionamento interno in tempo reale.

Suoni

Ogni 'PIEZO' Dispositivo 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 su programma esecuzione, UnoArduSim avvia e interrompe la riproduzione di un buffer sonoro associato mentre esecuzione viene avviato / interrotto.

A partire dalla V2.0, il suono è stato modificato per utilizzare l'API audio Qt, purtroppo il suo QAudioOutput 'class'

non supporta il looping del buffer audio per evitare l'esaurimento dei campioni sonori (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 il finestra "attivo" (tranne quando è appena stato creato e attivato un nuovo figlio finestra), **e persino** quando UnoArduSim è il finestra principale "attivo" ma il puntatore del mouse è **al di fuori** di la sua principale area clienti finestra.

Nota che ciò implica che il suono sarà temporaneamente disattivato come re mentre il puntatore del mouse è in bilico **su un bambino finestra**, e verrà disattivato **Se quel bambino finestra viene cliccato per attivarlo** (finché non si fa di nuovo clic su UnoArduSim finestra per riattivarlo) .

Il suono può essere sempre disattivato facendo clic in qualsiasi punto all'interno dell'area client del KA193 principale di UnoArduSim.

A causa del buffering, sound 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

Files Incluso

A '<>' - tra parentesi '#include' di '<Servo.h>', '<Wire.h>', '<OneWire.h>', '<SoftwareSerial.h>', '<SPI.h>', '<EEPROM.h>' e '<SD.h>' è supportati ma questi sono solo emulati - non viene cercato il files effettivo; invece la loro funzionalità è direttamente "incorporata" in UnoArduSim e sono valide per la versione Arduino supportata fissa.

Qualsiasi citato '#include' (per esempio di "supp.ino", "Myutil.cpp", o "Mylib.h") è supportato, ma tutti questi files devono **risiedere nel stessa directory come il genitore programma file** quello contiene il loro '#include' (non è possibile effettuare ricerche in altre directory). Il '#include' La funzione può essere utile per ridurre al minimo la quantità di codice programma mostrato in **Area di Codice** in qualsiasi momento. Intestazione files con '#include' (cioè quelli che hanno un ".h" estensione) causerà anche il tentativo del simulatore di includere lo stesso file con a "Cpp" estensione (se esiste anche nella directory del genitore programma).

Allocazioni di Memoria Dinamica e RAM

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

L'utilizzo eccessivo della RAM per le dichiarazioni variabile viene contrassegnato all'ora Analizzare e l'overflow della memoria RAM viene contrassegnato durante programma esecuzione. Un voce sul menu **Opzioni** consente di emulare la normale allocazione del registro ATmega come farebbe l'AVR compilatore, o di modellare uno schema di compilazione alternativo che utilizza solo lo stack (come opzione di sicurezza nel caso in cui un errore si visualizzi nella modellazione dell'allocazione del registro). Se si dovesse utilizzare un puntatore per esaminare il contenuto dello stack, esso dovrebbe riflettere accuratamente ciò che apparirebbe in un'implementazione hardware effettiva.

'Flash' Allocazioni di Memoria

Memoria 'Flash' 'byte', 'int' e 'float' variabili / matrici e il loro corrispondente accesso in lettura moduli funzionali sono supportati. Qualunque 'F()' Chiamata modulo funzionale ('Flash'-macro) di qualsiasi stringa letterale è supportato, ma l'unica supportata 'Flash'-stringa di memoria accesso diretto moduli funzionali sono 'strcpy_P()' e 'memcpy_P()', quindi per usare altro moduli funzionali dovrai prima copiare la stringa 'Flash' su una RAM normale 'String' variabile , e quindi lavorare con quella RAM 'String' . Quando usi il 'PROGMEM' Parola chiave modificatore variabile, deve apparire **di fronte a** il nome variabile e variabile **deve anche essere dichiarato** come 'const' .

'String' Variabili

Il nativo 'String' la libreria è quasi completamente supportata con poche (e minori) eccezioni.

Il 'String' gli operatori supportati sono +, + =, <, <=, >, > =, ==, !=, e []. Nota che: 'concat()' prende un **singolo** argomento che è il 'String', o 'char', o 'int' da aggiungere all'originale 'String' oggetto, **non** due argomenti come erroneamente indicato nelle pagine Web di riferimento di Arduino).

Librerie Arduino

Solo 'Servo.h', 'SoftwareSerial.h', 'SPI.h', 'Wire.h', 'OneWire.h', 'Stepper.h', 'SD.h', 'TFT.h' e 'EEPROM.h' per il **V1.8.8 Arduino** rilasciare attualmente sono supportati in UnoArduSim. V2.6 introduce un meccanismo per 3rd supporto delle librerie interlocutore tramite files fornito nel 'include_3rdParty' cartella che possono essere trovati all'interno della directory di installazione UnoArduSim. Provando a '#include' il ".cpp" e ".h" files di altri non ancora supportato librerie sarà **non funziona** in quanto contengono le istruzioni di montaggio di basso livello e le direttive non supportati e non riconosciuta files!

Puntatori

Puntatori a tipi semplici, matrici o oggetti sono tutti supportati. Un puntatore può essere equiparato a un matrice dello stesso tipo (es 'iptr = intarray'), ma poi ci sarebbe *nessun successivo controllo dei limiti matrici* su un'espressione come 'iptr[index]'.

Moduli funzionali può restituire i puntatori o 'const' puntatori, ma qualsiasi livello successivo di 'const' sul puntatore restituito viene ignorato.

C'è **nessun supporto** per le chiamate modulo funzionale effettuate **puntatori modulo funzionale dichiarati dall'utente**.

'class' e 'struct' Oggetti

Sebbene il poliforfismo e l'ereditarietà (a qualsiasi profondità) siano supportati, a 'class' o 'struct' può essere definito solo per avere al massimo **uno** base 'class' (vale a dire **multipla** l'ereditarietà non è supportata). Le chiamate di inizializzazione del costruttore base 'class' (tramite notazione dei due punti) nelle righe di dichiarazione del costruttore sono supportate, ma **non** inizializzazioni dei membri utilizzando la stessa notazione dei due punti. Ciò significa che oggetti che contiene 'const' non- 'static' variabili, o tipo di riferimento variabili, 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 spostamento, ma la conversione oggetto definita dall'utente ("tipo-pessofuso") moduli funzionali non è supportata.

Scopo

Non c'è supporto per il 'using' parola chiave o per 'namespace', o per 'file' scopo. Tutte le dichiarazioni non locali sono per implementazione considerate globali.

Qualunque 'typedef', 'struct', o 'class' definizione (cioè che può essere utilizzato per future dichiarazioni), deve essere fatto **globale** scopo (**Locale** le definizioni di tali elementi all'interno di un modulo funzionale non sono supportate).

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

Il 'unsigned' prefisso funziona in tutti i normali contesti legali. Il 'const' parola chiave, quando usata, deve **precedere** il nome variabile o il nome modulo funzionale o 'typedef' nome dichiarato: posizionarlo dopo il nome causerà un errore Analizzare. Per Dichiarazioni modulo funzionale, può avere solo il ritorno puntatore moduli funzionali 'const' apparire nella loro dichiarazione

Tutti UnoArduSim variabili sono 'volatile' per implementazione, quindi il 'volatile' la parola chiave viene

semplicemente ignorata in tutte le dichiarazioni variabile. Moduli funzionali non può essere dichiarato **'volatile'**, né gli argomenti della chiamata modulo funzionale.

Il **'static'** la parola chiave è consentita per il normale variabili e per i membri oggetto e per il membro moduli funzionali, ma non è esplicitamente consentito per le istanze oggetto (**'class'** / **'struct'**), per moduli funzionali non membro e per tutti gli argomenti modulo funzionale.

Direttive Compilatore

'#include' e regolare **'#define'** sono entrambi supportati, ma **non macro '#define'**. Il **'#pragma'** direttive di inclusione direttiva e condizionale (**'#ifdef'**, **'#ifndef'**, **'#if'**, **'#endif'**, **'#else'** e **'#elif'**) sono anche **non supportato**. Il **'#line'**, **'#error'** e macro predefinite (come **'_LINE_'**, **'_FILE_'**, **'_DATE_'**, e **'_TIME_'**) sono anche **non supportato**.

Elementi di Lingua Arduino

Tutti gli elementi linguistici nativi di Arduino sono supportati con l'eccezione del dubbio **'goto'** istruzione (l'unico uso ragionevole a cui posso pensare potrebbe essere un salto (ad un ciclo infinito di salvataggio e chiusura senza fine) 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" che salvano i bit per i membri nelle definizioni di struttura sono **non supportato**.

'union' è **non supportato**.

Lo strano "operatore virgola" è **non supportato** (quindi non è possibile eseguire più espressioni separate da virgole quando è normalmente prevista una sola espressione, ad esempio in **'while()'** e **'for(; ;)'** costrutti).

Modelli Modulo Funzionale

moduli funzionali definito dall'utente che utilizza la parola chiave "modello" per consentirgli di accettare argomenti di tipo "generico" **non supportato**.

Emulazione in Tempo Reale

Come notato sopra, i tempi esecuzione delle molte diverse possibili istruzioni Arduino programma sono **non** modellato in modo accurato, in modo che, al fine di eseguire in tempo reale il tuo programma avrà bisogno di una sorta di dominante **'delay()'** istruzione (almeno una volta per **'loop()'**), o un'istruzione che è naturalmente sincronizzata con le modifiche a livello di pin in tempo reale (come, **'pulseIn()'**, **'shiftIn()'**, **'Serial.read()'**, **'Serial.print()'**, **'Serial.flush()'** eccetera.).

Vedere **Sincronizzazione** e **Suoni** sopra per maggiori dettagli sulle limitazioni.

Note di Rilascio

Correzioni Errore

V2.7.0- Mar. 2020

- 1) Quando il (Finestre-default) è stato adottato il tema luce OS, il **Area di Codice** non mostrava il colore evidenziazione introdotto in V2.6 (invece solo una evidenziare grigio risultante da una sostituzione sistema).
- 2) La versione 2.6 inavvertitamente rotto auto-trattino-scheda Formattazione alla prima `'switch()'` costruire.
- 3) La nuova funzionalità di navigazione mucchio di chiamate introdotta nella versione 2.6 ha mostrato **valori errati** per locali variabili quando non all'interno del attualmente in esecuzione modulo funzionale, e fallito con chiamate modulo funzionale membri nidificato.
- 4) `'TFT :: text ()'` funzionava, ma le funzioni `'TFT :: print ()'` non funzionavano (semplicemente bloccate per sempre). Inoltre, `'TFT :: loadImage ()'` non è riuscito se `'Serial.begin ()'` era stato fatto in precedenza (che è il caso normale, e ora è richiesto).
- 5) La versione 2.6 ha introdotto un errore che ha visualizzato il valore corretto per 'RX' presente e passato byte per 'I2CSLV', 'SPISLV', 'TFT', 'LCDI2C' and 'LCDSPI' Dispositivi (e il loro monitoraggio finestre).
- 6) Una modifica apportata in V2.4 causato Esegui | Animare evidenziando per saltare molte code-linee eseguito.
- 7) Poiché V2.4, de-asserendo 'SS*' 'CS*' o su un 'I/O' Dispositivo nell'istruzione immediatamente a seguito di un `'SPI.transfer()'` causerebbe che Dispositivo di sicuro per ricevere il byte di dati trasferiti. Inoltre, byte ricezione in `'SPI_MODE1'` e `'SPI_MODE3'` non è stato contrassegnato fino all'inizio del prossimo byte inviato dal master (e il byte era completamente perso se il Dispositivo 'CS*' è stato de-selezionato prima di allora).
- 8) Nel nuovo `'SPI_SLV'` modalità consentita dal V2.4, `'bval = SPI.transfer()'` tornato solo il valore corretto per `'bval'` se il trasferimento di byte era già completo e in attesa quando `'transfer()'` è stato chiamato.
- 9) La casella di 'DATA' edit on 'SPISLV' Dispositivi ora ottiene il default 0xFF quando non ci sono più byte per rispondere con.
- 10) La sincronizzazione stato DEL era corretto per 'PSTEPR' Dispositivi avere più di 1 micro-passo a passo pieno.
- 11) Elettrico conflitti causato da 'I/O' Dispositivi reagire alle transizioni sull'orologio 'SPI', un segnale 'PWM', o `'tone'` di segnale, non sono stati segnalati, e potrebbe portare a inspiegabile (danneggiato) ricezione dei dati.
- 12) Quando l'intervallo tra gli interrupt era troppo piccolo (meno di 250 microsecondi), una (guasto) variazione V2.4 alterato la sincronizzazione di incassato moduli funzionali che utilizzano uno sistema temporizzatori o loop di istruzioni, per generare ritardi (esempi di ciascuna sono `'delay()'` e `'delayMicroseconds()'`). Una successiva modifica V2.5 causato disallineamento di `'shiftOut()'` dati e segnali di clock quando un interrupt accaduto tra bit.
- 13) Accettare un testo auto-completamento incassato modulo funzionale tramite il tasto Invio non è riuscito a nudo fuori tipi di parametri dal testo del bando modulo funzionale inserito.
- 14) Caricamento di un nuovo (user-interrupt-spinto) programma quando un programma precedentemente in esecuzione aveva ancora un interrupt in attesa potrebbe causare un crash durante il download (a causa di guasto tentato esecuzione della nuova routine di interrupt).
- 15) auto-completamenti Oggetto-membri (accessibile tramite 'ALT'-freccia destra) per oggetti all'interno `'#include'` files sono ora accessibili non appena il loro `'#include'` file è successo analizzato.
- 16) Una dichiarazione modulo funzionale avere uno spazio all'interno di una involontaria nome del parametro modulo funzionale causato un messaggio di errore Analizzare chiaro.
- 17) Quando esecuzione fermato in un diverso modulo dal principale programma, il File | azione precedente non è riuscito a diventare abilitato.
- 18) Single-citato parentesi graffe (`'{'` e `'}'`) Erano ancora contati (erroneamente) come parentesi scopo nel Analizzare, e anche confusa formattazione auto-tab-trattino ..
- 19) `'OneWire::readBytes(byte* buf, int count)'` era stato fallendo per aggiornare immediatamente la

visualizzazione 'buf' contenuto della Area delle Variabili.

20) Ottale-latch 'OWISLV' Dispositivi mostrato uscita pin livelli ritardato da una scrittura latch-registro.

V2.6.0- Jan 2020

- 1) Un errore introdotto in V2.3 portato ad un crash quando il valore aggiunto **Vicino** pulsante è stato utilizzato nel **Trova / Sostituire** dialogo (piuttosto che la sua Esci Pulsante titolo-bar).
- 2) Se un utente ha fatto programma '#include' di altri utenti files, la **Salvare** pulsante interno **Modificare/Esaminare** avrebbe mancato di effettivamente risparmiare un file modificato se c'è stato un errore Analizzare o Esecuzione esistente contrassegnato all'interno di una diversa file.
- 3) UN **Annulla** dopo un **Salvare** potrebbe anche essere confusione - per questi motivi la **Salvare** e **Annulla** funzionalità dei pulsanti sono stati modificati (vedi **Modifiche e miglioramenti**).
- 4) parentesi sbilanciati all'interno di un 'class' definizione potrebbe causare un blocco dal V2.5.
- 5) test logica diretta su 'long' valori restituiti 'false' se sono stati fissati nessuno dei 16 bit più bassi.
- 6) UnoArduSim era stato Contrassegno di un errore quando un puntatore variabile è stato dichiarato come il ciclo variabile all'interno delle parentesi di un 'for()' dichiarazione.
- 7) UnoArduSim era stato non permettere il test logici che i puntatori rispetto a 'NULL' o '0'.
- 8) UnoArduSim era stato non permettere puntatori coinvolge un intero variabile (solo numeri interi costanti erano stati lasciati).
- 9) Interrupt impostato con 'attachInterrupt(pin, name_func, LOW)' era stato sentito solo su un **transizione** per 'LOW'.
- 10) Quando si utilizza più di un dispositivo 'I2CSLV', uno slave non indirizzato potrebbe interpretare i dati bus successivi come corrispondenti al suo indirizzo bus (o chiamata globale 0x00) e quindi segnalare falsamente ACK, corrompendo il livello ACK del bus e appendendo una 'requestFrom()'.
- 11) Passando valore numerico '0' (o 'NULL') Come argomento modulo funzionale a un puntatore in una chiamata modulo funzionale è ora consentito.
- 12) Il livello di rientro tabulazione dopo un annidamento di 'switch()' costrutti era troppo bassa quando la scelta di 'auto-indent formatting' **Configurare | Preferenze** era usato.
- 13) Sottrazione di due puntatori compatibili oggi si traduce in un tipo di 'int'.
- 14) UnoArduSim si aspettava un costruttore di default definito dall'utente per un membro oggetto, anche se non fosse stato dichiarato come 'const'.
- 15) In una pausa esecuzione, la posizione disegnata di una 'STEPR', 'SERVO' o 'MOTOR' motore potrebbe ritardo fino a 30 millisecondi di moto dietro sua attuale posizione di ultimo-calcolata.
- 16) 'Stepper::setSpeed(0)' stava causando un arresto a causa di una divisione per zero.
- 17) Linea singola 'if()', 'for()', e 'else' non è più causa di uno troppe schede di auto-indentazione costrutti.

V2.5.0- ottobre 2019

- 1) A errore introdotto in V2.4 inizializzazione della scheda 'SD' rotto (causato un crash).
- 2) Utilizzando il sottosistema 'SPI' nella nuova 'SPI_SLV' Modalità lavorato in modo improprio in 'SPI_MODE1' e 'SPI_MODE3'.
- 3) Completamento automatico pop-up (come richiesto dalla 'ALT-right=arrow') sono stati fissati per moduli funzionali avere oggetto parametri; la lista pop-up ora anche includono ereditato i membri della classe (Base-), e auto-completamenti appaiono ora anche per 'Serial'.
- 4) Dal momento che V2.4 il valore restituito per 'SPI.transfer()' e 'SPI.transfer16()' era corretto se una routine di interrupt utente sparato durante quel trasferimento.

5) In V2.4, veloci forme d'onda periodiche state mostrate come aventi una durata superiore loro durata effettiva evidente se visto da zoom più elevato.

6) il costruttore `'File::File(SdFile &sdf, char *fname)'` non funzionava, così `'File::openNextFile()'` (Che si basa su quel costruttore) era anche non funziona.

7) UnoArduSim è stato erroneamente dichiarando un errore di Analizzare su oggetto-variabili, e oggetto-ritorno moduli funzionali, dichiarato come `'static'`.

8) istruzioni di assegnazione con `'Servo'`, `'Stepper'`, o `'OneWire'` oggetto variabile sul lato sinistro, e un oggetto-ritorno modulo funzionale o il costruttore sul RHS, causato un miscount interna del numero di associati oggetti, portando ad un eventuale bloccarsi.

9) test booleani su oggetti di tipo `'File'` sono stati sempre il ritorno `'true'` anche se il file non era aperto.

V2.4- Maggio 2019

1) I punti di guardia Eseguiere Fino A potrebbero essere falsamente attivati da una scrittura in un variabile adiacente (indirizzo inferiore di 1 byte).

2) Le selezioni della velocità di trasmissione potrebbero essere rilevate quando il mouse si trovava all'interno di un `'SERIAL'` o `'SFTSER'` Casella di riepilogo a discesa baud Dispositivo (anche quando non è stato effettivamente fatto clic su baud rate).

3) Dalla V2.2, gli errori di ricezione seriale si sono verificati con una velocità di trasmissione di 38400.

4) Una modifica apportata nella V2.3 ha causato **'SoftwareSerial'** a volte segnalare erroneamente un'interruzione disabilitata (e quindi fallita alla prima ricezione RX).

5) Una modifica apportata nella V2.3 ha causato l'interpretazione errata di SPISLV Dispositivi nei valori esadecimali immessi direttamente nella loro casella di modifica `'DATA'`.

6) Una modifica apportata nella V2.3 ha causato SRSLV Dispositivi a volte falsamente, e silenziosamente, rileva un conflitto elettrico sul loro `'Dout'` pin, e quindi non consente l'assegnazione di un pin Ii. I ripetuti tentativi di collegare un pin a `'Dout'` potrebbero quindi portare a un eventuale arresto anomalo una volta rimosso il Dispositivo.

7) Il tentativo di collegare un `'LED4'` Dispositivo dopo pin 16 causerebbe un arresto anomalo.

8) Un errore attivato da rapide chiamate ripetute a `'analogWrite(255)'` per **'MOTOR'** il controllo nell'utente programma ha causato il risultato **'MOTOR'** le velocità non sono corrette (molto troppo lente).

9) A partire dalla V2.3, non è stato possibile assegnare a SRSLV Dispositivi un `'Dout'` pin a causa di un rilevamento elettrico conflitto difettoso (e quindi non consentire l'assegnazione del pin).

10) Gli slave SPI ora reimpostano la loro logica di trasmettitore e ricevitore quando sono loro **'SS'** pin va **'HIGH'**.

11) chiamata `'Wire.h'` moduli funzionali `'endTransmission()'` o `'requestFrom()'` quando gli interrupt sono correntemente \ disabilitati ora genera un errore esecuzione (`'Wire.h'` ha bisogno di interrupt abilitati per funzionare).

12) `'Ctrl-Home'` e `'Ctrl-End'` ora funzionano come previsto in Modificare/Esaminare.

13) Il `'OneWire'` comando del bus `0x33 ('ROM_READ')` non funzionava e ha appeso l'autobus.

V2.3- Dec. 2018

1) Un errore introdotto in V2.2 ha reso impossibile modificare il valore di un elemento matrice all'interno **Modificare/Monitorare**.

2) Dalla versione 2.0, il testo nel `'RAM free'` Il controllo barra degli strumenti era visibile solo se si utilizzava un tema scuro di Finestre-OS.

3) Sopra **File | Caricare** su I / O Dispositivo file **Caricare**, i filtri file (come `'*.ino'` e `'*.txt'`) non funzionavano - sono stati invece mostrati files di tutti i tipi.

4) Lo stato "modificato" di un file è stato perso dopo averlo fatto **Accettare** o **Compilare** in un successivo **File | Modificare/Esaminare se non sono state apportate ulteriori modifiche** (**Salvare** è diventato disabilitato e non è stato richiesto automaticamente **Salvare** su programma **Esci**).

5) operatori `'/='` e `'%='` ha dato solo risultati corretti per `'unsigned'` lato sinistro variabili

- 6) Il condizionale ternario `'(bval) ? S1:S2'` ha dato un risultato errato quando `'S1'` o `'S2'` era un'espressione locale invece di un variabile.
- 7) Modulo funzionale `'noTone()'` è stato corretto per diventare `'noTone(uint8_t pin)'`.
- 8) Un errore di vecchia data ha causato un crash dopo aver proceduto da un **Resettare** quando quel Resettare è stato fatto nel mezzo di un modulo funzionale chiamato con uno dei suoi parametri mancanti (e quindi ricevendo un valore di iniziatore predefinito).
- 9) Espressioni dei membri (es `'myobj.var'` o `'myobj.func()'`) non stavano ereditando il `'unsigned'` proprietà della loro destra (`'var'` o `'func()'`) e quindi non potrebbe essere direttamente confrontato o combinato con altri `'unsigned'` tipi - un incarico intermedio a un `'unsigned'` variabile è stato inizialmente richiesto.
- 10) UnoArduSim insisteva sul fatto che se una definizione di modulo funzionale avesse un parametro qualsiasi con un iniziatore di default, che il modulo funzionale abbia dichiarato un precedente prototipo.
- 11) Chiama a `'print(byte bvar, base)'` promosso erroneamente `'bvar'` ad `'unsigned long'`, e così stampato troppe cifre.
- 12) `'String(unsigned var)'` e `'concat(unsigned var)'` e operatori `'+=(unsigned)'` e `'+(unsigned)'` creato in modo errato `'signed'` stringhe invece.
- 13) Un 'R=1K' Dispositivo caricato da un **IODevices.txt** file con posizione `'U'` è stato erroneamente disegnato con il suo cursore (sempre) nel **posizione opposta** dalla sua vera posizione elettrica.
- 14) Tentativo di fare affidamento sul valore predefinito `'inverted=false'` argomento quando si dichiara a `'SoftwareSerial()'` oggetto ha causato un arresto anomalo e il passaggio `'inverted=true'` ha funzionato solo se l'utente programma ne ha fatto un successivo `'digitalWrite(txpin, LOW)'` per stabilire innanzitutto il minimo necessario `'LOW'` livello sul `'TX'` pin.
- 15) 'I2CSLV' Dispositivi non ha risposto alle modifiche nelle loro caselle di modifica pin (le impostazioni predefinite A4 e A5 sono rimaste in vigore).
- 16) 'I2CSLV' e 'SPISLV' Dispositivi non hanno rilevato e corretto le modifiche parziali quando il mouse ha lasciato i bordi
- 17) I valori Pin per Dispositivi che seguivano uno SPISLV o SRSLV sono stati salvati in modo non corretto **IODevs.txt** file come esadecimali.
- 18) Cercando di connettere più di uno SPISLV Dispositivo MISO a pin 12 ha sempre generato un errore elettrico Conflitto.
- 19) Cambiare un pin da `'OUTPUT'` di nuovo a **Modalità** `'INPUT'` impossibile ripristinare il livello di latch dei dati di pin su `'LOW'`.
- 20) utilizzando `'sendStop=false'` in chiamate a `'Wire.endTransmission()'` o `'Wire.requestFrom()'` ha causato un errore.
- 21) UnoArduSim ha permesso impropriamente a `'SoftwareSerial'` la ricezione si verifica simultaneamente con a `'SoftwareSerial'` trasmissione.
- 22) Variabili dichiarate con `'enum'` al tipo non è stato possibile assegnare un nuovo valore dopo la riga di dichiarazione e UnoArduSim non riconosceva i membri 'enum' quando si fa riferimento a un oggetto (legale) `'enumname::'` prefisso.

V2.2- giu. 2018

- 1) Chiamando un modulo funzionale con un numero di argomenti inferiore a quello necessario per la sua definizione (quando quel modulo funzionale era "forward-defined", cioè quando non aveva una precedente riga di dichiarazione prototipo) causava una violazione della memoria e un arresto anomalo.
- 2) Dalla V2.1, **Forme d'onda** non era stato aggiornato durante **Esegui** (Solo a **Arresto** dopo a **Passo**) - Inoltre, **Area delle Variabili** i valori non venivano aggiornati durante il lungo periodo **Passo** operazioni.
- 3) Alcuni minori **Forma d'onda** i problemi di scorrimento e zoom che sono esistiti dalla versione 2.0 sono stati risolti.
- 4) Anche nelle prime versioni, il Resettare at = 0 con un PULSER o FUNCGEN, il cui periodo sarebbe il suo

ultimo ciclo prima di $t = 0$ essere solo un **parziale** ciclo, ha provocato il suo **Forma d'onda** dopo $t = 0$ che viene spostato dalla sua posizione reale da questo (o dal restante) importo del ciclo frazionario, a destra o a sinistra (rispettivamente).

5) Risolti alcuni problemi con l'evidenziazione del colore della sintassi in **Modificare/Esaminare** .

5) Dalla V2.0, facendo clic su espandere un oggetto in un matrice di oggetti non funzionava correttamente.

6) '**delay(long)**' è stato corretto per essere '**delay(unsigned long)**' e '**delayMicroseconds(long)**' è stato corretto per essere '**delayMicroseconds(unsigned int)**' .

7) A partire dalla V2.0, moduli funzionali collegato usando '**attachInterrupt()**' non venivano controllati come moduli funzionali validi per quello scopo (es '**void**' ritorno e senza parametri di chiamata).

8) L'impatto di '**noInterrupts()**' su moduli funzionali '**micros()**', '**mills()**', '**delay()**', '**pulseInLong()**'; il suo impatto su '**Stepper::step()**' e su '**read()**' e '**peek()**' timeout; su tutta la ricezione seriale RX e in seguito '**Serial**' trasmissione, è ora riprodotta accuratamente.

9) Il tempo trascorso all'interno delle routine di interrupt utente è ora considerato nel valore restituito da '**pulseIn()**', il ritardo prodotto da '**delayMicroseconds()**' e nella posizione dei bordi visualizzati **Forme d'Onda Digitali** .

10) Chiama a **oggetto-membro** moduli funzionali che facevano parte di espressioni complesse più grandi, o si trovavano all'interno di chiamate modulo funzionale con più argomenti complessi, e, g, '**myobj.memberfunc1()** + **count/2**' o '**myfunc(myobj.func1(), count/3)**', i valori errati sarebbero stati calcolati / passati al tempo di esecuzione a causa di allocazioni errate dello spazio dello stack.

11) Matrici del puntatore variabili funzionava correttamente, ma presentava valori di visualizzazione errati mostrati in **Area delle Variabili**.

12) Quando sono stati creati i matrici dinamici di tipo semplice con '**new**', solo il primo elemento aveva ottenuto un'inizializzazione (predefinita) per il valore 0 - ora tutti gli elementi lo fanno.

13) '**noTone()**', o la fine di un tono finito, non resetta più il pin (rimane '**OUTPUT**' e va '**LOW**').

14) La rotazione continua 'SERVO' Dispositivi ora è perfettamente stazionaria con una larghezza d'impulso di 1500 microsecondi.

15) La chiamata a **SdFile::ls()** (elenco di directory della scheda SD) funzionava correttamente, ma mostrava scorrettamente alcuni trasferimenti SPI a blocchi duplicati nelle forme d'onda finestra.

V2.1.1- Mar. 2018

1) Risolte incoerenze nelle impostazioni locali non inglesi con la lingua salvata in '**myArduPrefs.txt**', con i pulsanti della lingua radio visualizzati nel **Preferenze** finestra di dialogo e con la corrispondenza alle linee tradotte in '**myArduPrefs.txt**'.

2) Allocazione con '**new**' ora accetta una dimensione matrice intera che non è una costante.

3) Cliccando nel **Area delle Variabili** a espandere un matrice multidimensionale mostrerebbe un vuoto superfluo '**[]**' Staffa-pair .

4) Riferimenti a elementi Matrice con caratteri superflui finali (es '**y[2]12**') non sono stati catturati come errori al tempo Analizzare (i personaggi extra venivano semplicemente ignorati).

V2.1- Mar. 2018

1) Un errore nelle nuove versioni V2.0.x ha causato la crescita dell'heap Finestre con ogni aggiornamento nel **Area delle Variabili** -- dopo milioni di aggiornamenti (molti minuti del valore di esecuzione), potrebbe causare un arresto anomalo.

2) Chiamate verso 'static'membro moduli funzionali usando doppio-colon '**::**' notazione non riuscita a Analizzare quando all'interno '**if()**', '**while()**', '**for()**', e '**switch()**' parentesi quadre e quando le espressioni interne vengono utilizzate come argomenti di chiamata modulo funzionale o indici matrice.

V2.0.2 febbraio 2018

- 1) Un errore introdotto nella V 2.0 ha causato a **File | Caricare** crash se un `'#include'` riferito a un file mancante o vuoto
- 2) All'interno di **IOdevs.txt** file, lui **'I/O'** il nome 'One-Shot' era previsto al posto del vecchio 'Oneshot'; entrambi sono ora accettati.

V2.0.1- Gen. 2018

- 3) In locali di lingua non inglese, **'en'** è stato erroneamente mostrato come selezionato in **Preferenze**, rendendo inopportuno tornare all'inglese (richiedendo la deselezionazione e la ri-selezione).
- 4) Era possibile per l'utente lasciare un valore di modifica contrarsi pin in uno stato incompleto (come 'A_') e lasciare incompleti i bit 'DATA' di un 'SRS:V'.
- 5) Il numero massimo di cursori Analogico era stato limitato a 4 (corretto ora per essere 6).
- 6) UnoArduSim non insiste più su `'='` visualizzato in un'inizializzazione di aggregati matrice.
- 7) UnoArduSim aveva insistito sull'argomento "inverted_logic" `'SoftwareSerial()'`.
- 8) Le operazioni di bit-shift ora consentono turni più lunghi della dimensione del variabile spostato.

V2.0- Dic. 2017

- 1) Tutti i moduli funzionali dichiarati come **'unsigned'** avevano comunque restituito valori come se lo fossero **'signed'**. Ciò non ha avuto alcun effetto se il **'return'** valore è stato assegnato a un **'unsigned'** variabile, ma avrebbe causato un errore interpretazione negativa se avesse `MSB == 1`, ed è stato quindi assegnato a a **'signed'** variabile, o testato in una disuguaglianza.
- 2) I cursori Analogico stavano raggiungendo il massimo `'analogRead()'` valore di 1022, non il 1023 corretto.
- 3) Un errore inavvertitamente introdotto in V1.7.0 nella logica utilizzata per accelerare la gestione del sistema SPI SCK pin ha causato trasferimenti SPI per `'SPI_MODE1'` e `'SPI_MODE3'` fallire dopo il primo byte trasferito (un extra spurio Transizione SCK seguita da ciascun byte). Anche gli aggiornamenti di un box 'DATA' di 'SPISLV' edit per byte trasferiti sono stati ritardati,
- 4) Il DEL Dispositivo colorato non stava elencando 'B' (per il blu) come opzione di colore (anche se è stato accettato).
- 5) Le impostazioni per 'SPISLV' e 'I2CSLV' Dispositivi non venivano salvate all'utente **'I/O' Dispositivi** file.
- 6) Copia **'Servo'** istanze non riuscite a causa di un errore `'Servo::Servo(Servo &toCopy)'` implementazione del copy-constructor.
- 7) Fuori portata `'Servo.writeMicroseconds()'` i valori sono stati rilevati correttamente come errore, ma i valori limite indicati che accompagnavano il testo del messaggio di errore erano errati.
- 8) Non è stata accettata una velocità baud legale di 115200 quando caricato da un **'I/O' Dispositivi** testo file.
- 9) Il pin elettrico conflitti causato da un Cursore Analogico Dispositivo collegato non è stato sempre rilevato.
- 10) In rari casi, passando un puntatore stringa difettoso (con la stringa 0-terminatore mancante) a a **'String'** modulo funzionale potrebbe causare l'arresto anomalo di UnoArduSim.
- 11) **Area di Codice** potrebbe evidenziare l'attuale linea di errore Analizzare nel **sbagliato** Modulo programma (quando `'#include'` era usato).
- 12) Caricamento di un KL Dispositivi Dispositivi 'I/O' con un Dispositivo che avrebbe (in modo improprio) spingere contro 'Uno' pin 13 causato un blocco programma nel popup del messaggio di errore.
- 13) UnoArduSim aveva erroneamente permesso l'utente per incollare caratteri non esadecimali nel buffer espansi TX finestre per SPISLV e I2CSLV.
- 14) Inizializzazioni della linea di dichiarazione fallito quando il valore del lato destro era il **'return'** valore da un membro oggetto-modulo funzionale (come in `'int angle = myservo1.read();'`).
- 15) **'static'** membro variabili avendo esplicito `'ClassName::'` i prefissi non venivano riconosciuti se apparivano all'inizio di una riga (ad esempio, in un incarico a una base- 'class' variabile),

16) chiamata `'delete'` su un puntatore creato da `'new'` è stato riconosciuto solo se è stata utilizzata la notazione modulo funzionale parentesi, come in `'delete(pptr)'`.

17) Implementazione di UnoArduSim di `'noTone()'` insisteva erroneamente che fosse fornito un argomento pin.

18) Cambiamenti che aumentavano i byte 'RAM' globali in un programma che veniva usato `'String'` variabili (via **Modificare/Esaminare o File | Caricare**), potrebbe portare alla corruzione in quello spazio globale 'Uno' a causa dell'eliminazione dell'heap del `'String'` oggetti appartenente al vecchio programma durante l'utilizzo (non corretto) dell'heap appartenente al nuovo programma. In alcune circostanze questo potrebbe portare a un arresto programma. Sebbene un secondo Caricare o Analizzare abbia risolto il problema, questo errore è stato finalmente risolto.

19) I valori di ritorno per `'Wire.endTransmission()'` e `'Wire.requestFrom()'` erano stati entrambi bloccati a 0 - questi ora sono stati risolti.

V1.7.2- Feb. 2017

1) Anche gli interrupt su pin 2 sono stati attivati (inavvertitamente) dall'attività del segnale su pin 3 (e viceversa).

V1.7.1- Feb. 2017

1) Modulo funzionale `'delayMicroseconds()'` stava producendo un ritardo in *milli*-secondi (1000 volte troppo grandi).

2) tipo-pessofuso esplicito di un `'unsigned'` variabile a un tipo intero più lungo ha restituito un valore errato (`'signed'`) risultato.

3) Valori esadecimali superiori a `0x7FFF` sono ora `'long'` per definizione, e così così ora genererà `'long'` risultanti espressioni aritmetiche in cui vengono coinvolte.

4) Un errore introdotto inavvertitamente dalla V1.7.0 ha impedito il tipo-pessofuso 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 dall'utente programma.

6) Il variabili globale dichiarato dall'utente non occupa più spazio nella RAM 'Uno' se non viene mai effettivamente utilizzato.

7) Singolo variabili dichiarato come `'const'`, `'enum'` membri, e puntatori a stringhe letterali, non occupano più spazio nella RAM 'Uno' (per concordare con la compilazione Arduino),

8) Richiesti RAM per `'#include'` le librerie incorporate ora corrispondono strettamente ai risultati della compilazione condizionale Arduino.

9) utilizzando `'new'` su un puntatore la riga di dichiarazione effettiva era fallita (solo una più tardi `'new'` assegnazione al puntatore funzionante).

10) Risolto un errore in cui uno show "in sospeso" di una directory del disco SD poteva causare un blocco di 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 un Arduino modulo funzionale blocchi durante l'attesa (come `'pulseIn()'`, `'shiftIn()'`, `'SPI.transfer()'`, `'flush()'`, e `'write()'`) aveva causato un errore nel flusso esecuzione al ritorno dell'interrupt

b) copie multiple del variabili locale di qualsiasi modulo funzionale interrotto apparso nel **Area delle Variabili** (una copia per interrupt-return) e questo è stato risolto in V1.6.3, ma gli altri problemi di interruzione sono rimasti).

c) Modulo funzionale `'delayMicroseconds()'` non stava creando alcun ritardo se chiamato da una routine di interruzione utente.

d) Chiamate al blocco di moduli funzionali come `'pulseIn()'` a partire dal **dentro** un'interruzione la routine non

aveva funzionato.

1) Un errore introdotto nella V1.6.3 ha causato la perdita di aggiornamento del valore nel file **Area delle Variabili** durante l'esecuzione quando i valori effettivamente stavano cambiando (questo è successo solo dopo due o più **Arresto** o menu **VarAggiorna** azioni dell'utente). Inoltre, quando un **Esegui Verso** è stato fatto dopo **Consentire la riduzione** era stato innescato, il **Area delle Variabili** occasionalmente non è stato ridisegnato (quindi i vecchi valori e il locale variabili potrebbero essere apparsi lì fino al prossimo Passo).

2) Il **Area di Codice** evidenziando il comportamento del **Passo Scavalcare** comando potrebbe apparire fuorviante in `'if() -else'` Catene - che ora è stato corretto (anche se la funzionalità di stepping effettiva era corretta).

3) Modulo funzionale `'pulseIn()'` ha impostato erroneamente il timeout in millisecondi anziché in microsecondi, inoltre è stato riavviato in modo errato il timeout quando le transizioni ai livelli inattivo e attivo sono state viste per la prima volta.

4) Usando i valori letterali HEX tra `0x8000` e `0xFFFF` in incarichi o aritmetica con `'long'` intero variabili ha dato risultati errati a causa di estensione di segno non spuntata.

5) Passare o tornare a un `'float'` da qualsiasi `'unsigned'` il tipo intero che ha un valore con MSB = 1 ha dato risultati errati a causa di un errore `'signed'` interpretazione.

6) Tutti `'bit_()'` moduli funzionali ora accetta anche le operazioni su `'long'` -dimensioni variabili, e UnoArduSim verifica posizioni di bit non valide (che non rientrano nella dimensione variabile).

7) Un input non valido per la 'Pulse' (larghezza) edit-box su un 'PULSER' Dispositivo ha causato il danneggiamento del valore 'Period' (finché non è stato corretto dalla successiva voce di modifica 'Period' dell'utente).

8) Cancellare un 'PULSER' o 'FUNCGEN' Dispositivo usando il menu Configurare non stava rimuovendo il suo segnale periodico dal pin che stava guidando (un Resetare non è più necessario).

9) La possibilità di inizializzare un 1-D `'char'` Manca matrice con una stringa citata, (ad es `'char strg[] = "hello"; '`).

10) Visualizzazione esadecimale nel monitor espansi 'SERIAL' o 'SFTSER' finestre mostrava il carattere più significativo errato per valori di byte superiori a 127.

11) Il Forma d'onda finestre non rifletteva le modifiche programmatiche dell'utente effettuate da `'analogWrite()'` quando il un nuovo valore era 0% o 100% duty-cycle.

12) L'implementazione di `'Serial.end()'` è stato risolto

13) UN `'myArduPrefs.txt'` file con più di 3 parole su una riga (o spazi nel file **'I/O' Dispositivi** Nome file) potrebbe causare un arresto anomalo a causa di un puntatore interno difettoso.

14) La linea finale di un **'I/O' Dispositivi** file non è stato accettato se non fosse così **terminare con un avanzamento di riga**.

15) L'aggiunta di più di quattro cursori Analogico ha causato un errore silenzioso che ha sovrascritto i puntatori DEL 'I/O' Dispositivo

16) A partire dalla V1.6.0, i campioni analogico forma d'onda per prima metà di ciascuna **triangolo** forma d'onda erano tutti zero (a causa di un errore nel calcolo della tabella forma d'onda).

17) Fare un ripetuto **Esegui Verso** quando su una linea punto di arresto non richiede più clic multipli per anticipo.

18) Il passaggio delle espressioni di indirizzo a un parametro matrice matrice non è stato accettato da Analizzatore.

19) Il moduli funzionali ricorsivo che restituiva espressioni contenenti il puntatore o i de-riferimenti di matrice forniva risultati errati a causa del non ripristino dei flag "pronti" su quelle espressioni componente.

20) chiamata `'class'` membro-moduli funzionali attraverso **qualsiasi puntatore oggetto variabile o espressione del puntatore** non funzionava

21) L'utente moduli funzionali che ha restituito il valore-by oggetti ha restituito il proprio valore solo alla prima chiamata modulo funzionale **Se** hanno restituito un oggetto senza nome costruito (come `'String("dog")'`) - nelle chiamate successive il ritorno è stato saltato a causa di una bandiera "pronta" bloccata.

22) Non c'era stata alcuna salvaguardia per impedire il comando **Finestre | 'Serial' Monitor** from aggiungendo un

nuovo '**SERIAL**' Dispositivo quando in realtà non c'era spazio per questo.

23) Se aggiungendo un Dispositivo pins fisso (come 'SPISLV') causato un messaggio pop-up pin conflitto, il **Area del Banco da Laboratorio** il ridisegno potrebbe mostrare un duplicato "fantasma" Dispositivo sovrapponendo il più a destra 'I/O' Dispositivo (fino al successivo ridisegno).

24) Risolti alcuni problemi con suoni 'PIEZO' inaffidabili per segnali pin non periodici.

25) '**PROGMEM**' Anche variabili deve essere esplicitamente dichiarato come '**const**' essere d'accordo con Arduino.

26) "Nessuno spazio heap" veniva erroneamente contrassegnato come errore esecuzione quando un '**SD.open()**' impossibile trovare il nome file o un '**openNextFile()**' raggiunto l'ultimo file nella directory.

27) Un Analizzatore errore aveva accettato impropriamente un parentesi graffa di chiusura fuori dal luogo '}'.

28) Un errore con **Area delle Variabili** le rimozioni sul membro-oggetto - il ritorno del costruttore è stato corretto (il errore si applicava solo per oggetti che essi stessi contengono altri oggetti come membri).

V1.6.3- Settembre 2016

1) Il variabili locale di qualsiasi modulo funzionale interrotto non veniva rimosso dal **Area delle Variabili** sull'interruzione della voce modulo funzionale, che porta a più copie apparire lì sul ritorno dell'interrupt-modulo funzionale (e un possibile eventuale errore esecuzione o un crash).

2) Il Forma d'onda finestre non rifletteva i cambiamenti programmatici in '**analogWrite()**' a un nuovo ciclo di lavoro di 0% o 100%.

3) Visualizzazione esadecimale nel monitor espansi 'SERIAL' o 'SFTSER' finestra mostrava il carattere MSB errato per valori di byte superiori a 127.

V1.6.2- Settembre 2016

1) Le chiamate Modulo funzionale effettuate con il numero o il tipo di argomenti errati non avevano generato un messaggio di errore Analizzare appropriato (solo il è apparso un messaggio generico di "non un identificatore valido").

2) Il **Strumento-Bar** il pulsante di ripristino ora funziona in modo identico al pulsante di ripristino KL71 'Uno'.

3) Il testo dell'errore Analizzare non viene più tagliato dopo 16 caratteri senza mostrare i puntini di sospensione.

V1.6.1- agosto 2016

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

2) Cambiare il valore di un variabile facendo doppio clic sul file **Area delle Variabili** potrebbe causare errori di errore "nessuna allocazione di memoria" pop-up (per programmi con qualsiasi definito dall'utente '**class**').

3) '**SoftwareSerial**' non ha permesso l'accesso a '**write(char* ptr)**' e '**write(byte* ptr, int size)**' moduli funzionali a causa di un rilevamento di sovraccarico modulo funzionale difettoso.

4) Risolto il problema con l'inclusione automatica del corrispondente ".cpp" file per una libreria ".h" isolata '**#include**'.

V1.6 - Giugno 2016

1) In V1.5 rientro automatico sulla chiave 'Enter' in **Modificare/Esaminare** (quando si entra in una nuova linea) era stato perso.

2) Rilevato pin è in conflitto con collegato esterno a conduzione forte 'I/O' Dispositivi ora è stato aggiunto '**Serial**' pin 1, su SPI pins SS, MOSI e SCK, su I2C pins SCL e SDA (tutto quando il corrispondente '**begin()**' è chiamato), e su qualsiasi dichiarato '**SoftwareSerial**' TX pin.

V1.5.1- giugno 2016

1) In V1.5 i nuovi colori Syntax Evidenziare adattabili al tema non venivano reimpostati correttamente ogni volta

Modificare/Esaminare è stato aperto, e così (con un tema sfondo bianco) erano solo corrette ogni seconda volta.

- 2) Interrompere '**RISING**' e '**FALLING**' la sensibilità era stata di fronte all'attuale polarità del fronte di innesco.

V1.5 - Maggio 2016

- 1) Un errore introdotto in V1.4.1 ha impedito il passaggio di valori letterali stringa nulli a membro moduli funzionali che si aspettava a '**String**' oggetto, come in '**mystring1.startsWith("Hey")**' .
- 2) Un errore nell'originale **SD** l'implementazione di UnoArduSim è consentita solo **SD** accesso usando le chiamate a '**read()**' e '**write()**' (accesso via '**Stream**' moduli funzionali è stato prevenuto).
- 3) Gli interruttori a scorrimento 'R=1K' non venivano ridisegnati correttamente quando si spostava il cursore.
- 4) **Annulla** nella finestra di dialogo Confirm-Salvare file dovrebbe essere impedita l'uscita dall'applicazione.
- 5) Manca una chiusura o una chiusura '>' -parentesi su un utente file '**#include**' causerebbe un blocco.
- 6) Risolto un problema con errore nell'evidenziazione della sintassi di '**String**' e utente '**class**' o '**struct**' e l'evidenziazione estesa da includere chiamate modulo funzionale del costruttore.
- 7) Risolti alcuni problemi minori in **Modificare/Esaminare** con le modifiche / evidenziazione del testo e il **Disfare** pulsante.

V1.4.3 - Aprile 2016

- 1) utilizzando **Configurare** | '**I/O Dispositivi**' aggiungere il nuovo Dispositivi, e successivamente rimuovere uno di questi Dispositivi appena aggiunti potrebbe causare un arresto anomalo al reset o un altro Dispositivo smettere di funzionare.
- 2) Modifica a '**String**' variabile facendo doppio clic sul file **Area delle Variabili** fallito (il nuovo '**String**' è stato letto impropriamente).
- 3) Le modifiche Pin su '**FUNCGEN**' e '**PULSER**' Dispositivi non sono state riconosciute fino a quando non è stato effettuato prima un reset.

V1.4.2 - marzo 2016

- 1) V1.4.1 aveva introdotto uno sfortunato Analizzare errore che ha impedito incarichi che coinvolgono qualsiasi '**class**' oggetti (incluso '**String**').
- 2) Una correzione errore incompleta fatta in V1.4.1 ha causato '**unsigned**' valori di tipo '**char**' per stampare come caratteri ASCII anziché come valori interi.
- 3) Gli argomenti di chiamata complessi modulo funzionale di espressioni membro non venivano sempre riconosciuti come corrispondenze valide per il parametro modulo funzionale.
- 4) **Tutti** i letterali interi e le espressioni erano troppo generose (per '**long**') e quindi esecuzione non riflette il **effettivo** overflow (a negativo) che possono verificarsi in Arduino su operazioni di aggiunta / moltiplicazione coinvolgenti '**int**' valori dimensionali.
- 5) Espressioni che implicano un mix di '**signed**' e '**unsigned**' i tipi interi non venivano sempre gestiti correttamente (il '**signed**' valore sarebbe visto impropriamente come '**unsigned**').
- 6) Nei casi pin-conflitto, i messaggi di errore "value =" potrebbero mostrare valori pin stantii anche dopo un Resetare da un conflitto precedente che l'utente aveva già cancellato.

V1.4.1 - Gennaio 2016

- 1) Chiamate a '**print(char)**' ora stampa correttamente come caratteri ASCII (piuttosto che valori numerici).
- 2) La risposta all'interrupt è ora abilitata di default quando '**attachInterrupt()**' viene chiamato, quindi non c'è più alcun bisogno nel tuo '**setup()**' chiamare l'abilitazione modulo funzionale '**interrupts()**' .
- 3) Multiple '**#include**' istanze di user-files da all'interno di un file ora vengono gestiti correttamente.

V1.4 - Dicembre 2015

- 1) UN **Di vecchia data** errore erroneamente contrassegnato a **dividere per zero** condizione quando si divide per un valore frazionale inferiore all'unità.
- 2) Fisso '**SoftwareSerial**' (che è stato inavvertitamente rotto da un aggiunto '**class**' -verifica della convalida dei membri in V1.3 release).
- 3) Le chiamate modulo funzionale di fine linea con un punto e virgola mancante non sono state rilevate e hanno causato il Analizzatore saltare la riga successiva.
- 4) Un cattivo formato '**I/O** **Dispositivi**' il testo file ha dato un messaggio di errore improprio.
- 5) Analizzare L'errore di evidenziazione della riga errata (adiacente) nelle espressioni e istruzioni multilinea è stato corretto
- 6) Test logici di puntatori usando il '**not**' (**!**) l'operatore è stato invertito.

V1.3 - ottobre 2015

- 1) La manipolazione interna impropria dello scratchpad variabili ha causato occasionali " **superamento della profondità massima di annidamento del scratchpad** "Errori Analizzare.
- 2) parentesi *all'interno di virgolette singole*, parentesi graffe, punto e virgola, parentheses all'interno di stringhe tra virgolette e caratteri sfuggiti sono stati gestiti in modo improprio.
- 3) Un Matrice con una dimensione vuota e nessun elenco di inizializzazione ha causato un blocco RESET e matrici con un solo elemento non è stato disabilitato (e ha causato la loro interpretazione errata come un puntatore inizializzato non valido).
- 4) Gli errori Analizzare a volte a volte evidenziare avevano la linea sbagliata (adiacente).
- 5) Passare un puntatore a un non '**const**' a un modulo funzionale che accetta un puntatore a a '**const**' era stato respinto (invece del contrario).
- 6) Le espressioni di inizializzazione erano erroneamente ereditate '**PROGMEM**' qualificatori dal variabile in fase di inizializzazione
- 7) '**PROGMEM**' dichiarato che variabili aveva contato erroneamente la loro dimensione in byte **due volte** contro la loro allocazione di memoria 'Flash' durante il processo Analizzare.
- 8) Digitando nella casella di modifica 'Send' di un 'I2CSLV' a volte causerebbe un arresto a causa di '**sscanf**' errore.
- 9) Caricamento di un nuovo programma con un nuovo '**I/O** **Dispositivi**' file nella sua directory potrebbe causare pin è in conflitto con irrilevante **vecchio** Indicazioni pin.
- 10) La gestione dei caratteri seriali di escape è stata applicata impropriamente alle sequenze di caratteri ricevute, piuttosto che trasmesse, nel (più grande) '**Serial**' **Monitor** buffer finestra.
- 11) '**while()**' e '**for()**' loop con corpi completamente vuoti, come ad esempio '**while(true);**' o '**for(int k=1;k<=100;k++);**' passato il Analizzatore (con un messaggio di avvertimento) ma fallito al tempo esecuzione.

V1.2 - Giugno 2015

- 1) Il più semplice utente moduli funzionali che ha effettuato chiamate a entrambi '**digitalRead()**' o a '**analogRead()**' o '**bit()**' potrebbe aver corrotto il loro (molto prima) dichiarato variabile locale (se presente) a causa dello spazio scaricabile del modulo funzionale scaricabile (se solo due blocchi scratchpad sono stati assegnati all'inizio dello stack modulo funzionale). Qualsiasi espressione numerica all'interno di un modulo funzionale è sufficiente per causare un'assegnazione dello scratchpad di 4 byte, evitando così il problema. Questo sfortunato errore è stato in circolazione dalla versione originale V1.0.
- 2) Moduli funzionali che sono '**void**' con un esplicito precoce '**return**' e non '**void**' moduli funzionali con più di uno '**return**' dichiarazione, vedrebbe esecuzione fall-through al **chiusura parentesi graffa** (se è stato raggiunto).
- 3) Qualunque '**return**' dichiarazioni dentro '**if()**' i contesti che mancavano a parentesi graffe portavano a un target return-to-call difettoso.

- 4) **'PULSER'** e gli impulsi **'FUNCGEN'** oi periodi di valore 0 potrebbero causare un arresto anomalo (0 non è ora consentito).
- 5) Dove non c'erano parentesi graffe, **'else'** continuazioni dopo un **'if()'** non ha funzionato se hanno seguito un **'break'**, **'continue'**, o **'return'**.
- 6) quando **multiple 'enum' utente- le dichiarazioni sono state fatte**, costanti definite in tutto ma il primissimo **'enum'** generato difettoso " **'enum'** mancata corrispondenza" Errori Analizzare (questo errore è stato introdotto in V1.1).
- 7) Un identificatore nullo per l'ultimo parametro di un modulo funzionale prototipo ha causato un errore Analizzare.
- 8) **Eseguire Verso** i punti di interruzione 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 su Resetare (così potrebbero apparire i caratteri rimanenti dell'ultima volta).
- 10) Il Analizzatore non è riuscito a verificare la presenza di bit-flipping illegali **'float'** e aritmetica del puntatore tentata con operatori illegali.

V1.1 - Marzo 2015

- 1) Indici Matrice che erano **'byte'** o **'char'** variabili dimensionato ha causato offset matrice errati (se un 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) Qualunque **'return'** dichiarazioni incorporate all'interno **'for()'** o **'while()'** i loop erano mal gestiti.
- 4) Gli elenchi di inizializzazione aggregati per matrici di oggetti, o oggetti contenente altri oggetti / matrici, o elenchi di inizializzazione completamente vuoti, non venivano gestiti correttamente.
- 5) Accesso di **'enum'** i valori dei membri usando un **'enumname: :'** il prefisso non era supportato.
- 6) Inizializzazione della linea di dichiarazione di a **'char[]'** matrice con una stringa letterale quotata non funzionava.
- 7) Un matrice passato a un modulo funzionale senza un'inizializzazione precedente è stato erroneamente contrassegnato con un errore "usato ma non inizializzato".
- 8) Le espressioni di puntatore che coinvolgono i nomi matrice erano mal gestite.
- 9) Parametri Modulo funzionale dichiarati come **'const'** non sono stati accettati
- 10) Il Forma d'Onde Analogica finestra non visualizzava i segnali PWM (**'servo.write()'** e **'analogWrite()'**).
- 11) Il membro moduli funzionali a cui si è acceduto tramite un puntatore oggetto ha fornito accessi membri difettosi.
- 12) Le forme d'onda non venivano aggiornate quando a **Eseguire Verso** punto di arresto è stato raggiunto.
- 13) La modellizzazione dell'allocazione del registro poteva fallire quando un parametro modulo funzionale veniva usato direttamente come argomento per un'altra chiamata modulo funzionale

V1.0.2 - Aug. 2014

Ordine fisso di A0-A5 pins sul perimetro del 'Uno' scheda.

V1.0.1 - Giu. 2014

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

V1.0 - prima pubblicazione maggio 2014

Modifiche / Miglioramenti

V2.7.0- Mar. 2020

- 1) Oltre al codice-riga corrente (verde se pronto per l'esecuzione, rosso se errore), UnoArduSim ora mantiene, per ogni modulo, l'ultimo codice-linea dall'utente cliccato o stack di navigazione (evidenziata con uno sfondo scuro oliva), making facilitare l'impostazione e trovare linee punto di arresto temporanei (uno per ogni modulo è ora consentito, ma solo quello nel modulo attualmente visualizzati sono a tutti gli effetti in un 'Run-To').
- 2) Aggiunta una nuova 'I/O' Dispositivi (e sostenendo codice della libreria 3rd-party), tra cui 'SPI' e 'I2C' **Port Expander** 'SPI' e 'I2C' **Multiplexer DEL** Controllori e display (DEL matrici, 4-alfanumerici, e 4-cifra o 8-cifra display a 7 segmenti).
- 3) **'Wire'** operazioni non sono più consentiti dall'utente all'interno interrupt routine (supporta interrupt esterno da un 'I2C' espansione porta).
- 4) Digitale forme d'onda ora mostrano un livello intermedio (tra **'HIGH'** e **'LOW'**) Quando il pin non è essere spinto.
- 5) Per evitare confusione quando si passa sopra singola **'SPI.transfer()'** istruzioni, UnoArduSim ora fa in modo che in allegato 'I/O' Dispositivi ora ricevono la loro definitiva fronte di clock 'SCK' (logica-ritardato) prima che i rendimenti modulo funzionale.
- 6) Quando l'auto-tab-formattazione **Preferenza** è abilitata, digitando un parentesi graffa chiusura **'}'** nel **Modificare/Esaminare** ora provoca un salto alla posizione trattino scheda del suo corrispondente apertura parentesi graffa **'{'** compagno.
- 7) UN **Re-Format** Pulsante è stato aggiunto alla **Modificare/Esaminare** (A causa immediata auto-tab-rientro ri-formattazione) - questo pulsante è abilitato solo quando la preferenza Auto-tab-trattino è abilitata.
- 8) Un messaggio di errore più chiaro ora si verifica quando una parola chiave prefisso (come **'const'**, **'unsigned'**, o **'PROGMEM'**) segue un identificatore in una dichiarazione (di cui ha bisogno per precedere l'identificatore).
- 9) Initialized variabili globale, anche quando non utilizzato in seguito, sono ora sempre assegnato un indirizzo di memoria, e così apparirà visibile.

V2.6.0 gennaio 2020

- 1) Aggiunta visualizzazione dei caratteri-LCD Dispositivi avere 'SPI', 'I2C', e l'interfaccia 4-bi-parallelo. Sostenere codice sorgente della libreria è stato aggiunto alla cartella 'include_3rdParty' nuova installazione (ed è possibile accedervi utilizzando un normale **#include** direttiva) - Gli utenti possono inoltre scegliere di scrivere invece il proprio moduli funzionali al spingere il Dispositivo LCD.
- 2) **Area di Codice** evidenziazione è stata migliorata, con colori evidenziare separate per un codice-ready, per un codice di linea di errore, e per qualsiasi altro codice-linea.
- 3) Il **Trova** menu e tool-bar 'func' azioni (precedente-up e il prossimo-down) non salta più al / precedente successivo modulo funzionale linea di partenza, e invece ora salire (o discesa) la chiamata stack, evidenziando il relativo codice linea a chiamante (o chiamato) modulo funzionale, rispettivamente, dove la **Area delle Variabili** contenuto è rettificato per variabili per la modulo funzionale contenente il codice linea attualmente evidenziata.
- 4) Per evitare confusione, un **Salvare** all'interno done **Modificare/Esaminare** provoca un immediato ri-**Compilare**, E se la Salvare ha avuto successo, con una successiva Annulla o Esci sarà ora tornerà solo il testo a tale ultimo salvato testo.
- 5) Aggiunto un pulsato ingresso Motore Passo Passo ('PSTEPR') con 'STEP' (impulso), 'EN*' (abilitazione), e ingressi 'DIR' (direzione), e un ambiente micro-passi-per-passo (1,2,4,8, o 16) .
- 6) Sia 'STEPR' e 'PSTEPR' Dispositivi hanno ora una 'sync' DEL (VERDE per sincronizzato, o rosso quando fuori da uno o più passi.)

- 7) 'PULSER' Dispositivi ora hanno una scelta tra microsecondi e millisecondi per 'Period' e 'Pulse'.
- 8) Incassato-modulo funzionale auto-completamenti conservano non è più il tipo di parametro davanti al nome del parametro.
- 9) Quando si passa di nuovo ad un precedente **Area di Codice**, La sua linea precedentemente evidenziata viene nuovamente evidenziato.
- 10) Come aiuto per impostare un break-point temporaneo, utilizzando Annulla Esci o da **Modificare/Esaminare** lascia il evidenziare nel **Area di Codice** sulla linea ultima visita dal cursore in **Modificare/Esaminare**.
- 11) Un definito dall'utente (o 3a parte) '**class**' è ora consentito l'uso '**Print**' o '**Stream**' come la sua classe base. A sostegno di questa, è stata aggiunta una nuova cartella 'include_Sys' (nella cartella di installazione UnoArduSim), che fornisce il codice sorgente per ogni base '**class**'. In questo caso, le chiamate a tale base- '**class**' moduli funzionali saranno trattati in modo identico a utente codice (che) può essere intensificato in), anziché come modulo funzionale incassato che non può essere entrò (ad esempio '**Serial.print()**').
- 12) Gli-modulo funzionale auto-completamenti ora includono il nome del parametro **invece di** suo tipo.
- 13) L'UnoArduSim Analizzare consente ora un nome di oggetto in una dichiarazione variabile essere preceduto dal suo facoltativa (e la congruenza) '**struct**' o '**class**' parola chiave, seguita dal '**struct**' o '**class**' nome.

V2.5.0 ottobre 2019

- 1) Aggiunto il supporto per '**TFT.h**' biblioteca (con l'eccezione di '**drawBitmap()**'), E aggiunto un 'TFT' associato 'I/O' Dispositivo (128 x 160 pixel). Si noti che, al fine di evitare eccessivi ritardi in tempo reale durante la grande '**fillXXX()**' trasferimento, porzione sa dei trasferimenti '**SPI**' **nel mezzo del riempimento** sarà assente dal bus '**SPI**'.
- 2) Durante grandi file trasferimenti attraverso '**SD**', un parte dei trasferimenti '**SPI**' nel mezzo della sequenza di byte sarà similmente essere assente dal bus '**SPI**'.
- 3) Diminuzione '**Stream**' sovraccarico -usage byte in modo che '**RAM free**' il valore più si avvicina Arduino compilazione.
- 4) UnoArduSim ora avvisa l'utente quando un '**class**' ha più membri dichiarati su una riga della dichiarazione.
- 5) utilizzando 'File | Save As' Ora imposta la directory corrente che quella salvati-nella directory.
- 6) I due dispersi '**remove()**' membro moduli funzionali sono stati aggiunti alla '**String**' classe.
- 7) UnoArduSim costruttore ora esclude dal finanziamento di base-chiamate in un costruttore-modulo funzionale prototipo a meno che la definizione completa del corpo modulo funzionale segue immediatamente (in modo da concordare con l'Arduino compilatore).
- 8) Edtempo di transizione ge di forme d'onda digitale è stato ridotto a supportare la visualizzazione di segnali '**SPI**' veloci a più alto zoom.
- 9) un oArduSim consente ora alcuni costruttori da dichiarare '**private**' o '**protected**' (Per l'uso classe interna).

V2.4 maggio 2019

- 1) Tutti i file di dispositivo 'I/O' vengono ora salvati nella forma tradotta dalla lingua e, insieme al file delle **Preferenze**, vengono ora tutti salvati nella codifica del testo UTF-8 per evitare errori di corrispondenza nella lettura successiva.

- 2) Aggiunto un nuovo **'PROGIO'** Dispositivo che è uno scheletro programmabile 'Uno' scheda che condivide fino a 4 pins in comune con **Area del Banco da Laboratorio** master 'Uno', - può avere uno slave 'Uno' **no** 'I/O' Dispositivi a parte.
- 3) È ora possibile eliminare qualsiasi 'I/O' Dispositivo facendo clic su di esso mentre si preme il tasto 'Ctrl'.
- 4) Nel **Modificare/Esaminare** , il completamento automatico del testo è stato aggiunto per i membri globali, built-in e membri variabili e moduli funzionali (utilizzare ALT-freccia destra per richiedere un completamento, oppure **accedere** se l'elenco dei Built-in sta attualmente evidenziando una selezione corrispondente).
- 5) Nel **Preferenze** , una nuova scelta consente l'inserimento automatico di un punto e virgola che termina la linea su n **accedere** premere il tasto (se la riga corrente è un'istruzione eseguibile che sembra autonoma e completa).
- 6) urgente **'Ctrl-S'** da un **Forma d'onda** finestra ti consente di salvare su un file tutto (X, Y) punta lungo la sezione visualizzata di ciascun forma d'onda (dove X è microsecondi dal forma d'onda più a sinistra punto, e Y è volt).
- 7) A 'SFTSER' 'Dispositivo ora ha un nascosto (opzionale) 'inverted' valore (*si applica sia a TX che a RX*) che può essere aggiunto dopo il suo valore di baud rate alla fine della sua linea in un **IODevs.txt** file.
- 8) Aggiunto il 'SPISettings' classe, moduli funzionali 'SPI.transfer16()' , 'SPI.transfer(byte* buf, int count)' , 'SPI.beginTransaction()' , e 'SPI.endTransaction()' , così come 'SPI.usingInterrupt()' e 'SPI.notUsingInterrupt()' .
- 9) Aggiunta libreria SPI moduli funzionali 'SPI.detachInterrupt()' insieme a un'estensione della libreria SPI 'SPI.attachInterrupt(void myISRfunc)' (invece della vera libreria modulo funzionale 'SPI.attachInterrupt(void)' (in modo da evitare di dover riconoscere generici 'ISR(int vector)' interrupt di basso livello con dichiarazioni modulo funzionale).
- 10) Il sistema SPI può ora essere utilizzato in modalità slave, sia facendo il **'SS'** pin (pin 10) a **'INPUT'** pin e guidandolo **'LOW'** dopo **'SPI.begin()'** o specificando **'SPI_SLV'** come facoltativo **'mode'** parametro in **'SPI.begin(int mode=SPI_MSTR)'** (un'altra estensione di UnoArduSim a **'SPI.h'**). I byte ricevuti possono quindi essere raccolti utilizzando **'rxbyte = SPI.transfer(tx_byte)'** all'interno di un modulo funzionale non-SPI-interrupt o all'interno di un servizio di interruzione utente modulo funzionale precedentemente collegato da **'SPI.attachInterrupt(myISRfunc)'** . In modalità slave, **'transfer()'** aspetta fino a quando un byte di dati è pronto in SPDR (quindi normalmente bloccherà in attesa di una ricezione completa di byte, ma in una routine di interrupt **ritorno** immediatamente perché il byte SPI ricevuto è già lì). In ogni caso, **'tx_byte'** viene inserito nell'SPDR, quindi verrà ricevuto dallo SPI master allegato al successivo **'transfer()'** .
- 11) Il supporto alla modalità Schiavo è stato aggiunto all'implementazione UnoArduSim di 'Wire.h'. Modulo funzionale **'begin(uint8_t slave_address)'** è ora disponibile, così come sono **'onReceive(void*)'** e **'onRequest(void*)'** .
- 12) **'Wire.end()'** e **'Wire.setClock(freq)'** ora può essere chiamato; il secondo per impostare la frequenza SCL con a **'freq'** valore di 100.000 (la frequenza SCL in modalità standard predefinita) o 400.000 (modalità veloce).
- 13) 'I2CSLV' Dispositivi ora rispondono tutti al 0x00 general-call bus-address, e così via 0x00 non può più essere scelto come indirizzo bus I2C univoco per uno di questi slave.
- 14) I ritardi modellati di esecuzione delle operazioni di intero e di assegnazione di base e matrice e le operazioni del puntatore sono state ridotte e ora vengono aggiunti 4 microsecondi per ogni operazione in virgola mobile.

V2.3 dicembre 2018

- 1) Il monitoraggio è ora abilitato su **Strumento-Bar** 'I/O____S' **cursore** per continuo e liscio ridimensionamento dei valori 'I/O' Dispositivo che l'utente ha aggiunto il suffisso 'S'.
- 2) Una nuova **'LED4'** 'I/O' Dispositivo (fila di 4 LED accesi **4 numeri pin consecutivi**) è stato aggiunto.
- 3) Una nuova **'7SEG'** 'I/O' Dispositivo (7-segmenti DEL cifra con codice esadecimale attivo **4 numeri pin**

consecutivi e con attivo-basso **CS** * seleziona input), è stato aggiunto.

- 4) Una nuova '**JUMP**' È stato aggiunto 'I/O' Dispositivo che funge da ponticello tra due 'Uno' pins. Questo permette a '**OUTPUT**' pin da collegare a un '**INPUT**' pin (vedi sopra il Dispositivo per i possibili usi di questa nuova funzione).
- 5) Una nuova '**OWISLV**' È stato aggiunto 'I/O' Dispositivo e la terza parte '**<OneWire.h>**' la libreria può ora essere utilizzata con '**#include**' in modo che l'utente programmi possa testare l'interfacciamento con un piccolo sottoinsieme del bus '1-Wire' Dispositivi.
- 6) Il **Esegui** menu **Reset** il comando è ora connesso al **Reset** pulsante.
- 7) Per maggiore chiarezza, quando **Ritardo artificiale 'loop()'** è selezionato sotto il **Opzioni** menu, un esplicito '**delay(1)**' la chiamata viene aggiunta alla parte inferiore del ciclo all'interno '**main()**' - questo è ora un vero ritardo che può essere interrotto dagli interrupt utente collegati 'Uno' pins 2 e 3.
- 8) pin è in conflitto con elettrico open-drain, o CS-selected, 'I/O' Dispositivi (es. I2CLV o SPISLV) sono ora dichiarati **solo quando si verifica un vero conflitto al tempo esecuzione**, piuttosto che causare un errore immediato quando viene collegato per la prima volta il Dispositivo.
- 9) Modulo funzionale '**pulseInLong()**' ora è accurato a 4-8 microsecondi per concordare con Arduino (la precisione precedente era di 250 microsecondi).
- 10) Errori segnalati durante l'inizializzazione di un variabile globale ora evidenziano che variabile nel **Area di Codice**.

V2.2 giugno 2018

- 1) Sopra **Salvare** da entrambi **Preferenze** finestra di dialogo o da **Configurare | I/O Dispositivi**, il '**myArduPrefs.txt**' file è ora salvato nella directory del programma attualmente caricato - ogni successivo **File | Caricare** quindi carica automaticamente il file, insieme al suo IODEvs file specificato, dalla stessa directory programma.
- 2) Modulo funzionale '**pulseInLong()**' **era stato** mancante, ma ora è stato aggiunto (si basa su '**micros()**' per le sue misure).
- 3) Quando un utente programma fa un '**#include**' di una '***.h**' file, UnoArduSim ora tenta anche automaticamente di caricare il corrispondente '***.c**' file Se un corrispondente '***.cpp**' file non è stato trovato.
- 4) Inserimento automatico di una chiusura parentesi graffa '}' (dopo ogni open-parentesi graffa '{') è stato aggiunto a **Preferenze**.
- 5) Una nuova **Opzioni** la scelta del menu ora lo consente '**interrupts()**' essere chiamato dall'interno di una routine di interruzione utente - questo è solo a scopo didattico, poiché la nidificazione degli interrupt dovrebbe essere evitata nella pratica.
- 6) Tipo-pessofuso di puntatori ad un '**int**' il valore è ora supportato (ma verrà visualizzato un messaggio di avviso).
- 7) UnoArduSim ora supporta linee programma etichettate (es '**LabelName: count++;**' per comodità dell'utente (ma '**goto**' è ancora **annullato**)
- 8) Gli avvertimenti Esecuzione ora si verificano quando una chiamata a '**tone()**' potrebbe interferire con PWM attivo su pins 3 o 11, quando '**analogWrite()**' interferirebbe con un Servo già attivo sullo stesso pin, quando un arrivo di un carattere seriale viene perso perché gli interrupt sono attualmente disabilitati e quando gli interrupt arriveranno così velocemente che UnoArduSim perderà alcuni di essi.

V2.1 marzo 2018

- 1) mostrato **Area delle Variabili** i valori sono ora aggiornati solo ogni 30 millisecondi (e l'opzione Minima può ancora ridurre ulteriormente quella frequenza di aggiornamento), ma il **VarAggiorna** l'opzione di menu per disabilitare la riduzione degli aggiornamenti è stata rimossa.
- 2) Operazioni che mirano solo a una parte dei byte di un valore variabile (come quelli fatti tramite i puntatori) ora causare la modifica a quella che il valore variabile deve essere riflesso nel **Area delle Variabili** display.

V2.0.1 gennaio 2018

- 1) Arduino moduli funzionali non documentato '`exp()`' e '`log()`' sono stati aggiunti ora
- 2) 'SERVO' Dispositivi può ora essere ruotato in modo continuo (quindi la larghezza dell'impulso controlla la velocità anziché l'angolo).
- 3) Nel **Modificare/Esaminare**, una chiusura parentesi graffa '}' viene ora aggiunto automaticamente quando si digita un parentesi graffa di apertura '{' se lo hai selezionato **Preferenza**.
- 4) Se si fa clic sul **Modificare/Esaminare** finestra barra del titolo 'x' per uscire, ti viene data la possibilità di interrompere se hai modificato, ma non salvato, il programma file visualizzato.

V2.0 settembre 2017

- 1) L'implementazione è stata trasferita a QtCreator, quindi la GUI presenta alcune piccole differenze visive, ma nessuna differenza funzionale oltre a qualche miglioramento:
 - a) La messaggistica sulla riga di stato nella parte inferiore del finestra principale e all'interno del **Modificare/Esaminare** finestra di dialogo, è stata migliorata e aggiunta una codifica a colori evidenziare.
 - b) Lo spazio verticale assegnato tra il **Area di Codice** e **Area delle Variabili** è ora regolabile tramite una barra di separazione drag-able (ma non visibile) al loro bordo condiviso.
 - c) I valori della casella di modifica 'I/O' Dispositivo vengono ora convalidati solo dopo che l'utente ha spostato il puntatore del mouse all'esterno del Dispositivo, evitando così modifiche automatiche per applicare 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 una traduzione personalizzata *.qm file per quella lingua).
- 3) Il suono è ora stato modificato per utilizzare l'API audio Qt - questo ha richiesto il silenziamento in determinate circostanze al fine di evitare fastidiosi rumori e clic durante i lunghi tempi operativi della finestra del sistema operativo causati dai normali clic del mouse dell'utente, consultare la sezione -Sound per maggiori dettagli su questo.
- 4) Per comodità dell'utente, gli spazi vuoti vengono ora utilizzati per rappresentare un valore 0 nelle caselle di modifica del conteggio Dispositivo in **Configurare | 'I/O' Dispositivi** (così ora puoi usare la barra spaziatrice per rimuovere Dispositivi).
- 5) Il non ridimensionato Il qualificatore (U) è ora opzionale su 'PULSER', 'FUNCGEN' e '1SHOT' Dispositivi (è l'impostazione predefinita presunta).
- 6) UnoArduSim ora consente (oltre ai valori numerici letterali) '**const**' variabili con valore intero, e '**enum**'

V1.7.2- Feb. 2017

- 1) La scelta del colore blu (B) è stata aggiunta per DEL Dispositivi.

V1.7.1 - Febbraio 2017

- 1) suffissi 'L' e 'o' 'U' sono ora accettati alla fine delle costanti numeriche letterali (per definirle come '`long`' e /

o 'unsigned'), e ('0b' o '0B' prefisso) Le costanti binario sono ora accettate. Qualsiasi costante numerica ogni decimale **iniziando con a '0'** ora è considerato come un **ottale** valore. (essere d'accordo con Arduino).

2) Quando si esegue un ciclo stretto da cui non c'è via di fuga (per esempio `'while(x); x++;'` dove `x` è sempre vero), facendo clic **Arresto** una seconda volta ora garantisce che il esecuzione esecuzione si fermi effettivamente (e su quella linea programma difettosa).

V1.7.0- dic. 2016

1) Una nuova **Strumento-Bar** è stata aggiunta una funzione che mostra i byte della RAM libero durante programma esecuzione (che tiene conto dei byte totali utilizzati dal variabili globale, dalle allocazioni dell'heap e dallo stack locale variabili).

2) L'interrupt utente moduli funzionali potrebbe ora chiamare anche il blocco di Arduino moduli funzionali `'pulseIn()'` (ma questo dovrebbe essere usato solo con cautela, poiché l'interrupt modulo funzionale non ritornerà fino a quando il blocco modulo funzionale non sarà completo).

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 di Arduino moduli funzionali che può essere interrotto (come `'delay()'` e `'pulseIn()'`) e i messaggi della barra di stato sono stati aumentati per mostrare quando si è raggiunto un interrupt punto di arresto all'interno di tale modulo funzionale (o quando si fa clic su Arresto quando esecuzione è attualmente all'interno di tale modulo funzionale).

5) Una nuova **Eseguire Fino A** comando (e **Strumento-Bar** oggetto) è stato aggiunto - clic singolo su qualsiasi **Area delle Variabili** variabile (può essere semplice, un aggregato matrice o oggetto o un elemento matrice o membro oggetto) a evidenziare esso, quindi faccia **Eseguire Fino A** - esecuzione si bloccherà al successivo **accesso in scrittura** all'interno di quell'aggregato variabile, o in quella singola posizione.

6) Quando esecuzione si blocca dopo a **Passo**, **Eseguire Verso**, **Eseguire Fino A**, o **Eseguire-poi-Arresto** azione, il **Area delle Variabili** ora evidenzia il variabile che corrisponde al **indirizzo (i) che è stato modificato** (se del caso) dal **all'ultimo istruzione durante quel esecuzione** - se quella posizione è attualmente nascosta all'interno di un matrice o oggetto non espansi, facendo clic su espandere farà in modo che l'elemento o il membro modificato per ultimo vengano evidenziati.

7) L'utente può ora mantenere un orologio speciale sul valore di uno specifico **Variabile Area** variabile / member / element durante l'esecuzione - fai doppio clic su quella linea nel file **Area delle Variabili** per aprire il **Modificare/Monitorare Valore Variabile** finestra, quindi fai uno dei **Eseguire** o **Passo** comandi - il valore visualizzato verrà aggiornato durante esecuzione in base alle stesse regole che regolano gli aggiornamenti in **Area delle Variabili**. Dopo aver bloccato esecuzione, è possibile immettere un nuovo valore e **Accettare** prima di riprendere esecuzione (e può **Ritornare** al pre**Accettare** valore se cambi idea prima di allora).

8) I tasti di accelerazione F4-F10 sono stati impostati in modo da corrispondere al menu Eseguire **Strumento-Bar** comandi (da sinistra a destra).

9) Oltre a fare doppio clic su di essi, fare clic con il pulsante destro su 'SERIAL', 'SFTSER', 'SPISLV', 'I2CSLV' Dispositivi ora visualizzerà anche un TX / RX byte / caratteri finestra di grandi dimensioni (e su 'SD_DRV', un files-monitoraggio finestra).

10) La casella di modifica TX in 'SERIAL' o 'SFTSER' non è più disabilitata durante una trasmissione di caratteri attiva (quindi ora è possibile aggiungere o sostituire ciò che è presente), ma un ritorno a capo (o il pulsante 'Send' fare clic sul pulsante 'Seriale; finestra) verrà ignorata fino a quando la trasmissione ritorna allo stato di riposo (i caratteri ora sono mostrati in corsivo quando la trasmissione è pronta per iniziare, è attiva). Inoltre, l'utente è ora avvisato su un flusso seriale `'begin()'` se avessero già iniziato prima l'allegato Dispositivo (ora in corso) le trasmissioni, in quanto non ci sarebbe quindi alcuna sincronizzazione di framing, portando a errori di ricezione.

11) Il valore aggiunto è stato aggiunto `'loop()'` il ritardo è stato aumentato da 250 microsecondi a un millisecondo, in modo da non scendere molto indietro rispetto al tempo reale quando l'utente trascura di includere alcuni `'delay()'` (esplicito o naturale) da qualche parte all'interno `'loop()'` o all'interno di un modulo

funzionale che chiama.

12) Matrici e tipi semplici sono stati aggiunti al supporto per l'allocazione dell'heap `'new'` istruzioni.

13) Verifiche più estese (e messaggi di errore associati) sono stati aggiunti per gli accessi agli indirizzi fuori campo programma dell'utente (cioè al di fuori della RAM 'Uno' o al di fuori di 'Flash' per `'PROGMEM'` accessi).

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

15) L'utente `'myArduPrefs.txt'` file è ora caricato a tutti **File | Caricare**, non solo al lancio di UnoArduSim.

16) Un errore Analizzare è ora contrassegnato quando si tenta di `'attachInterrupt()'` a un utente modulo funzionale che non lo è `'void'` ritorno, o che ha parametri modulo funzionale, o che non è stato dichiarato da qualche parte prima `'attachInterrupt()'`.

17) `'static'` membro-variabili sono ora visualizzati nella parte superiore del **Area delle Variabili** come globals, piuttosto che apparire all'interno di ogni istanza di un oggetto (espansi).

18) Modulo funzionale `'availableForWrite()'` è stato aggiunto all'implementazione di `'Serial'`.

19) Tutto speciale `'PROGMEM'`, `'typedef'` piace `'prog_char'` e `'prog_int16'` sono stati rimossi (sono stati deprecati in Arduino).

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

21) La dimensione programma massima consentita è stata aumentata.

V1.6.3- Sett. 2016

1) Aggiunto un messaggio di errore migliorato analizzare quando `'attachInterrupt()'` si riferisce ad un interrupt-modulo funzionale che non lo era ***prototipato prima***.

2) Aggiunto un messaggio di errore Analizzare migliorato per gli elenchi di inizializzazione matrice multidimensionali.

V1.6.2- Sett. 2016

1) Aggiunto a **Trova-Text** modifica il controllo per **Strumento-Bar** per semplificare la ricerca di testo (nel file **Area di Codice** e **Area delle Variabili**).

2) Il **Strumento-Bar** Il pulsante Resetare ora funziona in modo identico al pulsante LK1041 scheda Resetare.

V1.6.1- agosto 2016

Aggiunto un controllo per evitare il caricamento e l'analisi duplicati di quelli già precedenti `'#include'` files, .

V1.6 - Giugno 2016

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

2) Aggiunta una nuova funzione che rende i valori della casella di modifica 'I/O' Dispositivo facilmente **scalato** durante l'esecuzione trascinando un cursore di scala 'I/O_____S' globale sul main **Strumento-Bar** (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 la libreria EEPROM moduli funzionali `'update()'`, `'put()'` e `'get()'`, e per l'accesso ai byte tramite notazione matrice, ad es `'EEPROM[k]'`.

2) **Consenti automatico (-) Contrarsi** è stato aggiunto al menu **VarAggiorna** per consentire un controllo esplicito sull'opportunità o meno di espansi matrici / oggetti auto-contrattato quando esecuzione è in ritardo rispetto al tempo reale.

3) I personaggi di una **'String'** È ora possibile accedere anche a variabile tramite notazione matrice, per esempio **'mystring[k]'** .

V1.5 - Maggio 2016

1) **Modificare/Esaminare** ora ha una scorciatoia ctrl-E, e ha un nuovo pulsante per **Compilare** (ctrl-R), oltre a una incassato Analizzare-error box, per consentire il test delle modifiche senza la necessità di chiudere il finestra.

2) **Modificare/Esaminare** ora ora supporta anche **Rifaree** ha una nuova **Salvare** (ctrl-S) (equivalente a **Accettare** più un successivo finestra principale **Salvare**), e ora dà una scelta di **'Tab'** dimensione (una nuova preferenza che può essere salvata usando **Configurare | Preferenze**).

3) Tutte le caselle di modifica scrivibili ora seguono i colori del tema OS Finestre scelti e, per contrasto, tutte le caselle di modifica 'RECV' di sola lettura utilizzano testo bianco su sfondo nero. Il **Modificare/Esaminare** i colori di sfondo e sintassi-evidenziare ora si adattano anche al tema scelto.

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

5) Valori letterali binario predefiniti Arduino (come **'B01011011'**) sono ora consentiti.

6) Le sequenze di caratteri citate con esadecimale esadecimale, ottale e 4-cifra possono ora essere utilizzate come valori letterali numerici.

7) Dopo aver fatto un clic del mouse su un pulsante K10105 '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 controlla più **'Stepper'** e **'Servo'** pin conflitti, ovvero l'utente difettoso programma tenta di collegarsi a pins già collegato a prima **'Stepper'** o **'Servo'** variabili.

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

11) L'inutilizzato **'String'** classe **'flags'** il membro variabile è stato rimosso per concordare con Arduino V1.6.6. UN **'String'** oggetto ora occupa 6 byte (più l'allocazione dell'heap dei caratteri).

V1.4.2 - marzo 2016

1) moduli funzionali definito in avanti (ovvero quelli senza dichiarazione prototipo prima della loro prima chiamata) ora generano solo avvertimenti (non errori analizzare) quando la successiva definizione di tipo modulo funzionale restituisce una corrispondenza errata tra il tipo dedotto dal primo utilizzo.

2) Matrici con una dimensione uguale a 1 non è più rifiutato (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 Finestre in uso.

4) **'SERIAL'**, **'SFTSER'**, **'SPISLV'** e **'I2CSLV'** Dispositivo espansi Monitor finestre (aperto con doppio clic) ora adotta il colore di sfondo del loro genitore **'I/O'** Dispositivo.

V1.4 - Dicembre 2015

1) **'Stepper.h'** sono state aggiunte funzionalità di libreria e **'I/O'** Dispositivi associati.

- 2) **Tutte le impostazioni e i valori di 'I/O' Dispositivo** (in aggiunta al pins selezionato) ora vengono salvati anche come parte dell'utente scelto 'I/O' Dispositivi testo file per il successivo ricarico.
- 3) **DEL** È ora possibile impostare il colore LK105 Dispositivo come rosso, giallo o verde utilizzando una casella di modifica sul Dispositivo.
- 4) Gli inizializzatori di dichiarazione Variabile ora possono attraversare più righe.
- 5) Gli indici Matrice ora possono essere essi stessi elementi matrice.
- 6) **Configurare | Preferenze** ora include una casella di controllo per consentire **'and'** , **'or'** , **'not'** parole chiave da utilizzare al posto dello standard C **'&&'** , **'||'** , e **'!'** operatori logici.
- 7) "Mostra Programma Scaricamento" è stato spostato in **Configurare | Preferenze**

V1.3 - ottobre 2015

- 1) Il **'PUSH'** Dispositivo ora ha un check-box "push-like" etichettato 'latch' per renderli "latching" (invece di "momentary"), cioè, si bloccano nella posizione chiusa (e cambiano colore) quando vengono premuti, finché non vengono premuti di nuovo per rilasciare i contatti.
- 2) Le funzionalità complete 'SPISLV' Dispositivi sono state aggiunte con la selezione del nodo (**'MODE0'** , **'MODE1'** , **'MODE2'** , o **'MODE3'**). Il doppio clic apre i buffer TX / RX finestra dove possono essere definiti i prossimi REPLY (TX) byte e per la visualizzazione dei byte ricevuti (RX) passati. Il semplice slave a registro a scorrimento Dispositivo della versione precedente è stato rinominato come 'SRSLV' Dispositivo.
- 3) **Grassetto** carattere tipografico può ora essere scelto per **Area di Codice** e **Area delle Variabili** (dal menu **Opzioni**), e **Evidenziazione grassetto di parole chiave e operatori** ora può essere attivato / disattivato in **Modificare/Esaminare** .
- 4) UnoArduSim ora consente **'bool'** come sinonimo di **'boolean'** .
- 5) Per chiarezza nella segnalazione degli errori, le dichiarazioni variabile non sono più consentite per estendersi su più righe (eccetto per matrici con gli elenchi di inizializzatori).
- 6) Sintassi velocità di colorazione in **Modificare/Esaminare** è stato migliorato (questo sarà evidente con programmi più grande).
- 7) Un overhead opzionale da 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 l'utente programma non ha aggiunto **'delay ()'** ovunque (vedi discussione Sincronizzazione sotto).

V1.2 giugno 2015

- 1) La libreria SD è ora completamente implementata e un (piccolo) 8Mbyte SD Disk 'I/O' Dispositivo ('SD_DRV') è stato aggiunto (e funzionalità testate su tutti i campioni di Arduino SD programmi).
- 2) Come Arduino, UnoArduSim convertirà automaticamente un argomento modulo funzionale nel suo indirizzo quando chiama un modulo funzionale in attesa di un puntatore da passare.
- 3) I messaggi di errore Analizzare sono ora più appropriati quando mancano i punti e virgola e dopo le dichiarazioni non riconosciute.
- 4) stantio **Area delle Variabili** i punti salienti della linea ora vengono rimossi sulla chiamata / ritorno modulo funzionale.

V1.1 - Marzo 2015

- 1) Il finestra principale ora può essere ingrandito o ridimensionato per fare il **Area di Codice** e **Area delle Variabili**

più ampio (per schermi più grandi).

2) Un nuovo menu Trova (con **Pulsanti della barra degli strumenti**) sono stati aggiunti per consentire una navigazione più veloce in **Area di Codice** e **Area delle Variabili** (PgUp e PgDown, o ricerca di testo con freccia su, freccia giù).

3) Il **Modificare/Esaminare** finestra ora consente i salti di navigazione ctrl-PgUp e ctrl-PgDn (alla prossima riga vuota) e ha aumentato la velocità **Trova / Sostituire** funzionalità.

4) UN nuovo elemento è stato aggiunto al menu **VarAggiorna** per consentire all'utente di selezionare un approccio di risparmio di calcolo sotto pesante **Area delle Variabili** aggiorna i carichi.

5) 'Uno' pins e DEL allegato ora riflettono tutte le modifiche apportate a 'I/O' Dispositivi anche quando il tempo è congelato (ovvero, anche quando esecuzione viene fermato).

6) L'altro utente moduli funzionali può ora essere richiamato da un interrupt utente modulo funzionale (in conformità con l'aggiornamento ad Arduino 1.06).

7) UN **carattere più grande** ora può essere scelto dal menu **Opzioni**.

V1.0.1 - Giu. 2014

Forma d'onda finestre ora etichetta analogico pins come A0-A5 invece di 14-19.

V1.0 - prima pubblicazione maggio 2014