PROTEUS DESIGN SUITE

Visual Designer Help

Labcenter Electronics

www.labcenter.com

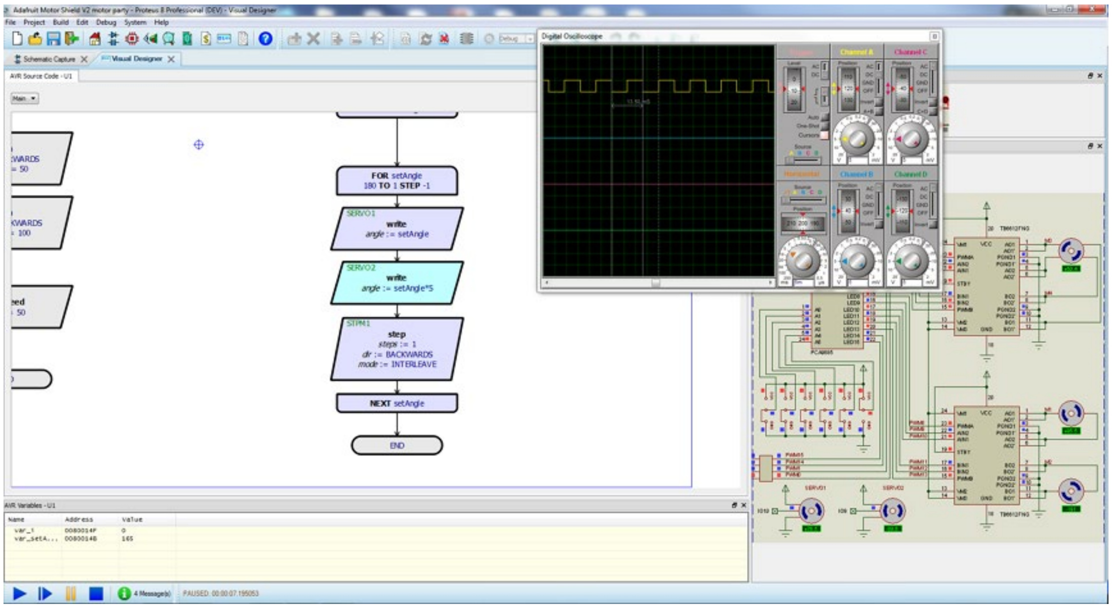
COPYRIGHT NOTICE

TABLE OF CONTENTS

VISUAL DESIGNER TUTORIAL

INTRODUCTION

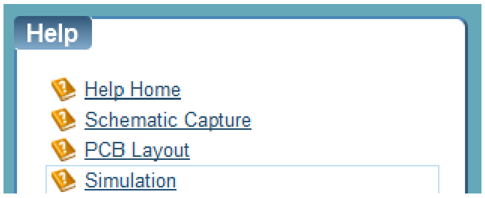
Visual Designer is a unique development tool that uses flowchart blocks together with schematic clips of Arduino™ shields or Raspberry Pi hats to allow drag and drop creation of real-world Arduino and rPi embedded systems.



```
Name        Address    Value
var_i       0080014F   0
var_setA... 0080014B   165
```

Single step debugging in Visual Designer with the AdaFruit Motor Shield

This document explains the various features of Visual Designer and includes several practical example projects and walkthroughs. Other modules in the Proteus system such as the schematic capture and general simulation have their own help files and are companion resources to this document. You'll find these help files either on the help menu in the schematic capture module or via the Help section on the Proteus home page.


Help

Help Home
Schematic Capture
PCB Layout
Simulation

Launchpad for other help files.

What is Arduino™ / Genuino™?

Arduino is an open-source computer hardware and software company, project and user community that designs and manufactures microcontroller-based kits for building digital devices and interactive objects that can sense and control objects in the physical world.

The project is based on microcontroller board designs, manufactured by several vendors, using various microcontrollers. These systems provide sets of digital and analog I/O pins that can be interfaced to various expansion boards ("shields") and other circuits.

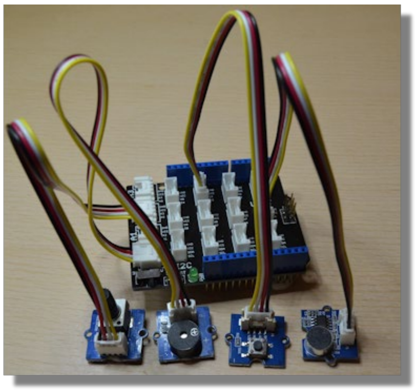
See Also: Arduino Website

The Arduino and Genuino names and logos are registered trademarks of Arduino, LLC.The Visual Designer for Arduino simulation product is not directly connected or endorsed by Arduino.

What is Raspberry Pi ?

Raspberry Pi is a series of small, low cost, single board computers. The Raspberry Pi is still a complete Linux computer but is still often used to control external electronics.

What is Grove ?



Grove is a modular electronic platform for quick prototyping. Every module has one function, such as touch sensing, creating audio effect and so on. Just plug the modules you need to the base shield, then you are ready to test your ideas.

This Grove Starter Kit is a great way for beginners and students to get started with Arduino. The Grove Base Shield drops onto your existing Arduino and lets you connect up to 16 Grove modules without any fuss. The Arduino headers are still broken out on the top of the shield making it very easy to connect other breakout boards and sensors to your Arduino.

See Also: Grove Starter Kit (Website)

Raspberry Pi: Limitations of Simulation

The Raspberry Pi is a high level behavioural model in terms of simulation. Since the rPi runs a full copy of Linux and arbitrary code can be executed a full simulation model is not possible. Specifically, the following libraries are supported:

Smbus
Pygame
Wiringpi
RPi.GPIO
Spidev

There is also a set of Python libraries, which are supplied by us in the Raspberry Pi drivers directory:

Adafruit_GPIO
Adafruit_I2C
Adafruit_MCP230xx
Adafruit_MCP3008
Adafruit_MotorHAT
Adafruit_PCA9685
Adafruit_PureIO
Automationhat
Explorerhat
ADS1x15
cap1xx
grove_128_64_oled
grove_rgb_lcd
grovepi
max31855
pcf8574
pcf8591
piglow
sn3218

If you are writing your firmware with Visual Designer flowchart blocks then the Raspberry Pi simulation should work with no problems. However, if you convert your project to write your program in Python then of course you can write code that is based on other Python libraries. In such cases you may not be able to simulate your project because your source code may pull in libraries unsupported in simulation.

GUIDED TOUR

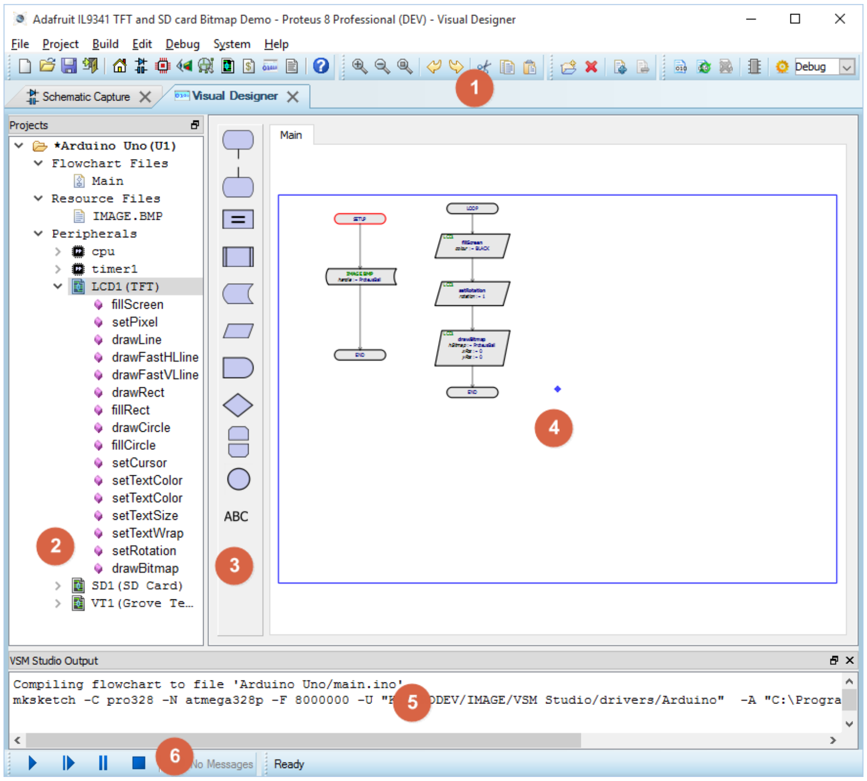
Visual Designer is a tool for designing embedded systems via a simple flowchart interface and then simulating and debugging the entire embedded system in software. It makes sense therefore that the IDE changes and provides both an Editing Layout and a Debugging Layout.

This topic introduces both environments and includes links to more detailed topics where appropriate.

Visual Designer Environment

Editing Layout

At design time, we are creating our embedded system, adding both hardware shields and embedded control logic. The Visual Designer Environment breaks down into six main areas as shown below



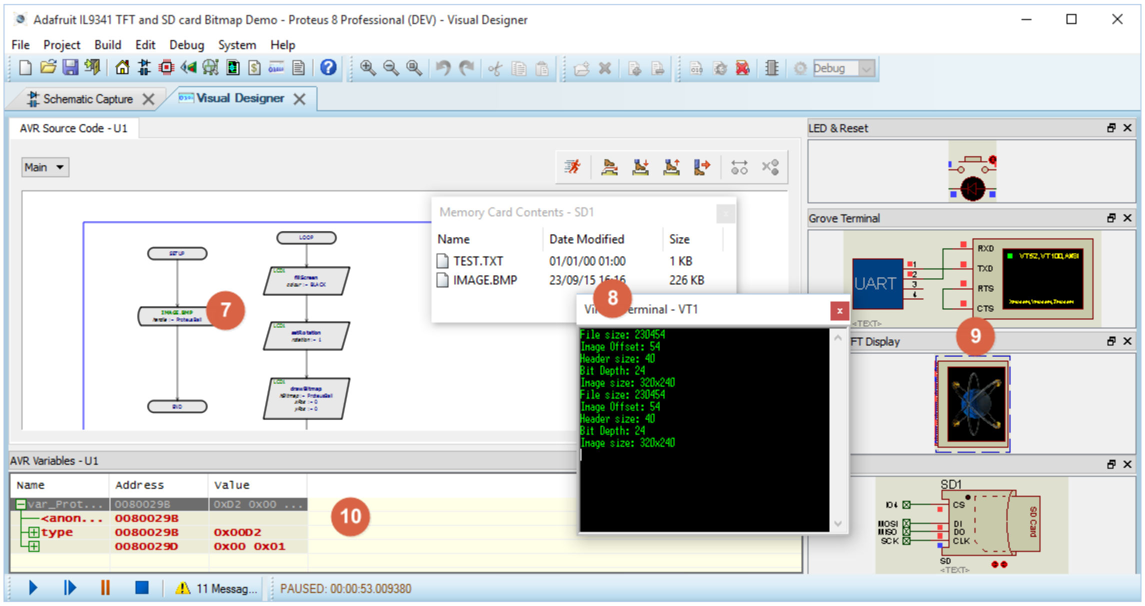
```
Compiling flowchart to file 'Arduino Uno/main.ino'
mksketch -C pro328 -N atmega328p -F 8000000 -U "F DEV/IMAGE/VSM Studio/drivers/Arduino"  -A "C:\Progra
```

The Visual Designer IDE at Design Time.

1) Menus, Icons and Tab	2) Project Tree
3) Flowchart Blocks	4) Editing Window
5) Output Window	6) Simulation Control Panel

Debugging Layout

During simulation and debug, the environment changes to provide relevant tools.



Memory Card Contents - SD1

Name	Date Modified	Size
TEST.TXT	01/01/00 01:00	1 KB
IMAGE.BMP	23/09/15 16:16	226 KB

```
File size: 230454
Image Offset: 54
Header size: 40
Bit Depth: 24
Image size: 320x240
File size: 230454
Image Offset: 54
Header size: 40
Bit Depth: 24
Image size: 320x240
```

AVR Variables - U1

Name	Address	Value
var_Prot...	0080029B	0xD2 0x00 ...
<anon...	0080029B	
type	0080029B	0x00D2
	0080029D	0x00 0x01

The Visual Designer IDE when single step debugging a simulation.

7) Source Code Window	8) Pop-up Peripheral Windows
9) Active Popups	10) Variables Window / Debug Windows

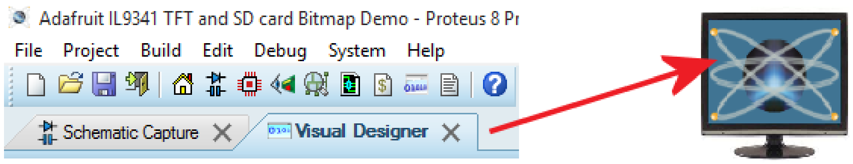
These items are all discussed in the following sections and are also covered in a more practical way in the tutorials.

1) Menu's, Icons and Tabs

The top menu commands will be familiar to most users and specific functionality is discussed where relevant in the tutorial and other areas of the documentation. Some of the iconography may be new however and each icon is therefore listed below along with its use:

ICON	DESCRIPTION	ICON	DESCRIPTION	ICON	DESCRIPTION
📄	New Proteus Project	0101	Open VSM Studio / Visual Designer	➕	Create New Firmware Project
📂	Open Project	📄	Open Project Notes	❌	Delete Existing Firmware Project
💾	Save Project	❓	Open the Help files	➕	Add Source Files to the Existing Project
🚪	Close Project	⊕	Zoom In	➖	Remove Source Files to the Existing Project
🏠	Open Proteus Home Page Open Proteus Home Page	⊖	Zoom Out	010	Build / Compile Project
	Open Schematic Capture Program		Zoom All	🔄	Rebuild / Compile Project
	Open the PCB Layout Program	↶	Undo	❌	Stop Build
	Open the 3D Viewer	↷	Redo		Attached to IDE / Hardware
	Open the Gerber Editor	✂	Cut	⚙	Project Settings
	Open the Design Explorer		Copy	Debug	Type of Build
$	Open the Bill of Materials	📋	Paste		

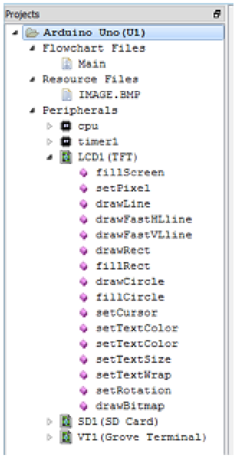
The tabs allow you to switch between different areas of the Proteus Design Suite. We will be working primarily inside Visual Designer but we are building an embedded system and that creates a schematic design as well. The schematic will be automatically drawn as we add shields and peripherals to Visual Designer but you can switch to the Schematic tab at any time to look at your 'virtual hardware'. You can even drag and drop a tab onto a different monitor if you want to look at both modules simultaneously.


Adafruit IL9341 TFT and SD card Bitmap Demo - Proteus 8 Pr
File Project Build Edit Debug System Help
Schematic Capture
Visual Designer

Drag and Drop Tabs onto a second monitor to split screen.

Having virtual hardware in the form of a schematic really shows its power during simulation as you can take measurements anywhere you please. You simply place and wire instruments such as oscilloscopes and logic analyzers to inspect waveforms. See the tutorial for more information and examples.

2) Project Tree



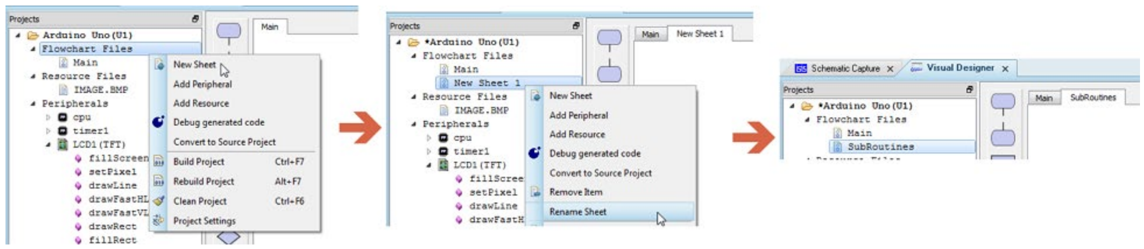
```
Projects
Arduino Uno (U1)
  Flowchart Files
    Main
  Resource Files
    IMAGE.BMP
  Peripherals
    cpu
    timer1
    LCD1 (TFT)
      fillScreen
      setPixel
      drawLine
      drawFastHLline
      drawFastVLline
      drawRect
      fillRect
      drawCircle
      fillCircle
      setCursor
      setTextColor
      setTextColor
      setTextSize
      setTextWrap
      setRotation
      drawBitmap
    SD1 (SD Card)
    VT1 (Grove Terminal)
```

The project tree in Visual Designer performs three main roles:

- Control of Flowchart Sheets
- Control of Resources in the Embedded System
- Control of Hardware Peripherals in the Embedded System

Control of Flowchart Sheets

When you start a new project you will have by default a single sheet in the Editing Window called 'Main'. You can add more sheets (for example a subroutines sheet) if your program is likely to become quite large.



Adding and naming a second sheet in the Flowchart.

When you have more than one sheet in your project you can quickly move between sheets by double clicking on the sheet name in the Project Tree. Alternatively, you can select the sheet you wish to navigate to from the sheet tabs at the top of the Editing window.

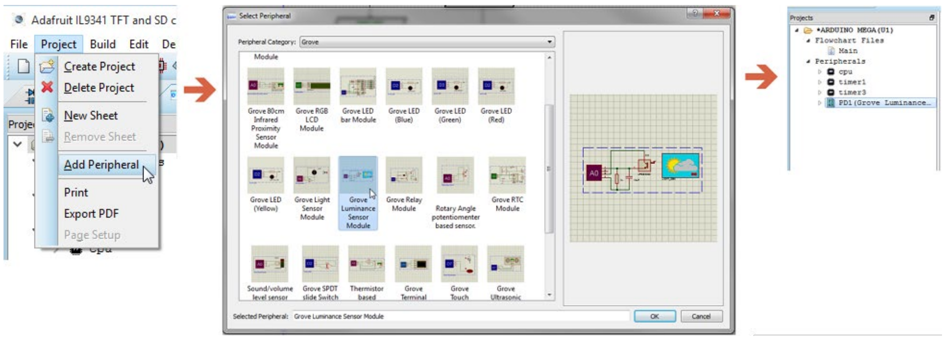
You can also find the sheet commands on the Project Menu at the top of the Visual Designer IDE.

Control of Resource Files

Resource files provide a great way to include pictures and audio files into your project. You can add /remove resource files from the right click context menu on the Project Tree. Once you have a resource file you can drag and drop it straight into the setup routine of your flowchart program to assign it a handle.

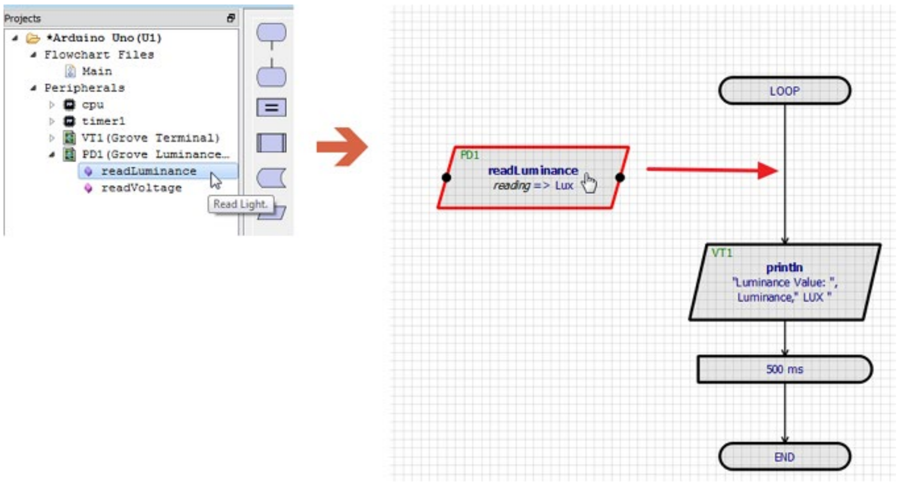
Control of Hardware Peripherals

Visual Designer is a development environment for a complete embedded system and so the peripherals block includes both on-board peripherals of the CPU and also external peripherals (supported Arduino shields or Grove Sensors) that you add to build up your hardware design. When you start a new project you will see two or three peripherals such as CPU core and Timer1 related to the processor core and you add extra peripherals via the right click context menu on the Project Tree or the command on the Project Menu.



Adding a LUX sensor to the current Flowchart Project

Once added, you can see all of the methods available to you. These are the means by which you interact with the hardware and you simply drag and drop the method into your flowchart program where it will appear as an I/O Peripheral Block.



Drag and Drop the method you want straight into place in the Flowchart Editor.

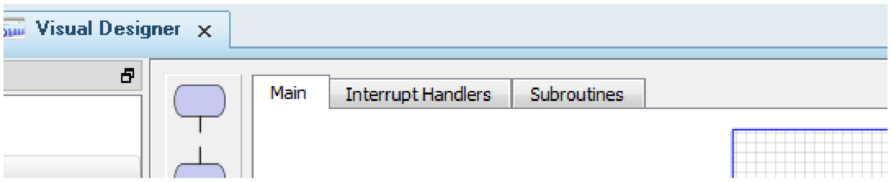
You can also rename Peripherals once added. This is particularly useful for example, if you have several buttons or LED's where it is more transparent to name them according to their job. Simply right click on the peripheral and select the rename option from the resulting context menu.

3) Flowchart Blocks

The Flowchart Blocks are the building blocks of your program. While you will often drag and drop straight from the Project Tree you can also drag and drop flowchart objects from this column onto the Editing Window. Indeed, some object types such as Delay Blocks, Loop Constructs and Event Handlers are only available from this column.

4) Editing Window

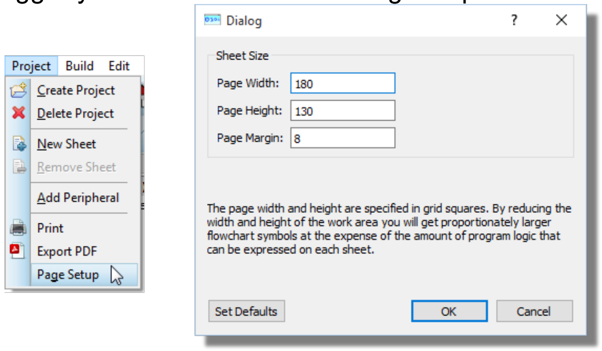
The Editing Window is the place where you actual design your flowchart and create your programs. For programs with multiple sheets the Editing Window displays the current sheet and provides a small tab-strip at the top for easy navigation to other sheets.



Three sheets in a Visual Designer Program with the Editing Window showing the Main Sheet.

The sheet size you see is simple a set number of grid squares wide and high. Flowchart blocks and other object are then scaled to also be a set number of grid squares. You can, from the

Page Setup command on the Project Menu, adjust the number of visible grid squares. This allows you to change the relative size of the flowchart blocks on the screen; for example, if you want the blocks to be bigger you reduce the number of grid squares on the screen.



Sheet Size

Page Width: 180

Page Height: 130

Page Margin: 8

The page width and height are specified in grid squares. By reducing the width and height of the work area you will get proportionately larger flowchart symbols at the expense of the amount of program logic that can be expressed on each sheet.

When you print to physical paper the chart will automatically fit to page.

You can place flowchart objects on the editing window by drag and drop from either the Project Tree or from the Flowchart bar on the left hand size.

You can zoom in and out the Editing Window via either the middle mouse wheel, the F6 and F7 keys or the icons on the toolbar.

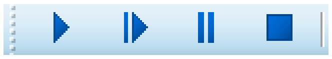
5) Output Window

The output window provides a dump of status information and will list any errors when you build your flowchart or program the physical hardware.

If you close the output window the Editing Window will be larger and you will 'zoom in' a little more on the flowchart. The output window will come back automatically when it is needed (i.e. when you build your program)

6) Simulation Control Panel

Interactive simulations are controlled from a simple panel that behaves just like a normal remote control. This control is situated at the bottom left of the screen by default. There are four buttons that you use to control your simulation.



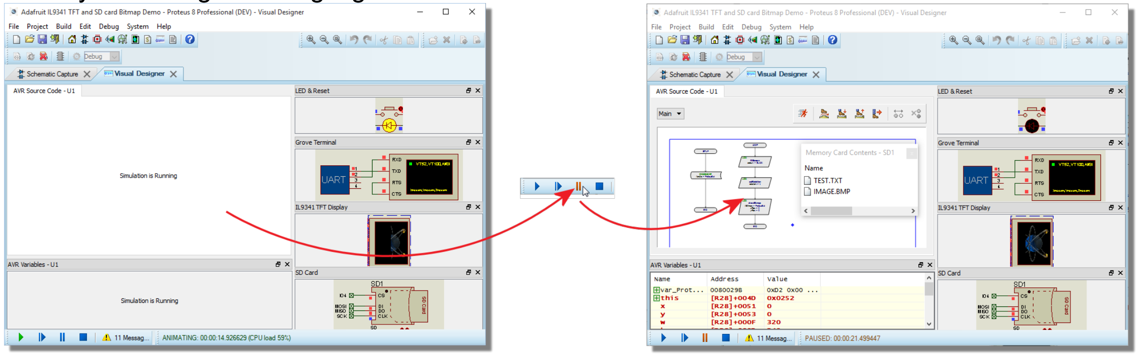
- The PLAY button is used to start the simulator.

The STEP button allows you to step through the animation at a defined rate. If the button is pressed and released then the simulation advances by one time step; if the button is held down then the animation advances continuously until the button is released. The single step time increment may be adjusted from the Animated Circuit Configuration dialog box on the System Menu.

- The step time capability is useful for monitoring the circuits more closely and seeing in slow motion what is affecting what.
- The PAUSE button suspends the animation which can then be resumed either by clicking the PAUSE button again, or single stepped by pressing the STEP button. The simulator will also enter the paused state if an breakpoint is encountered.
- The STOP button tells the system to stop doing a real time simulation. All animation is stopped and the simulator is unloaded from memory. All the indicators are reset to their inactive states but the actuators (switches etc.) retain their existing settings.

7) Source Code Window

The source code window is the primary tool for debugging your designs in Proteus and allows you to single step your program, advancing the entire embedded system as you debug your software. When the simulation is paused the source window will display the flowchart with the currently executing block highlighted in red.



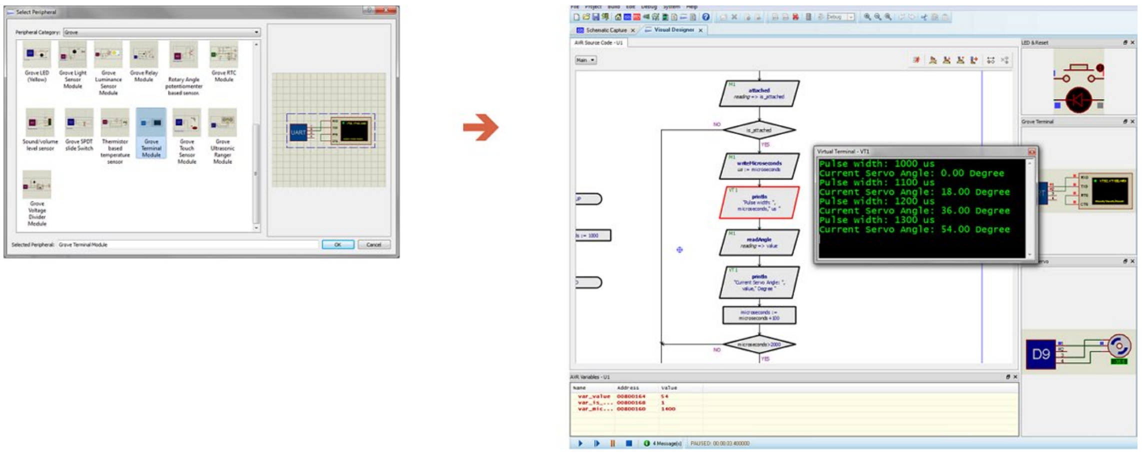
```
Name          Address        Value
var_Prot...   0080029B       0xD2 0x00 ...
this          [R28]+004D     0x0252
  x           [R28]+0051     0
  y           [R28]+0053     0
  w           [R28]+000F     320
```

Simulation running and then displaying the program for single step debug when paused.

You can even choose to debug the source code rather than the flowchart by first selecting the Debug Generated Code option from the project tree context menu.

8) Peripheral Popup Windows

Some peripherals present a popup window during simulation. For example, if you add the Grove Terminal Module then you will see a virtual terminal popup during simulation that you can use to read and write text.



```
Pulse width: 1000 us
Current Servo Angle: 0.00 Degree
Pulse width: 1100 us
Current Servo Angle: 18.00 Degree
Pulse width: 1200 us
Current Servo Angle: 36.00 Degree
Pulse width: 1300 us
Current Servo Angle: 54.00 Degree
```

Virtual Terminal acting as a console with info dump on servo motor position.

Peripheral popups are typically visible during running simulation so that you can interact with them. They can be closed at any time from the cross at top right and then re-opened from the Debug menu when the simulation is paused.

9) Active Popups

Active popups are simply areas of interest on the schematic that appear as docked widgets on the right hand side of the of the Visual Designer debugging layout. They have two main roles :

1) You can see relevant parts of the hardware while debugging the software (E.g. text appearing on an LCD).

2) You can interact with relevant parts of the hardware while debugging the software (E.g. press a button or adjust a sensor).

The end result of using active popups is that you don't have to constantly switch tabs between the schematic and the firmware in a debug session. They allow all relevant information to be displayed on a single tab.

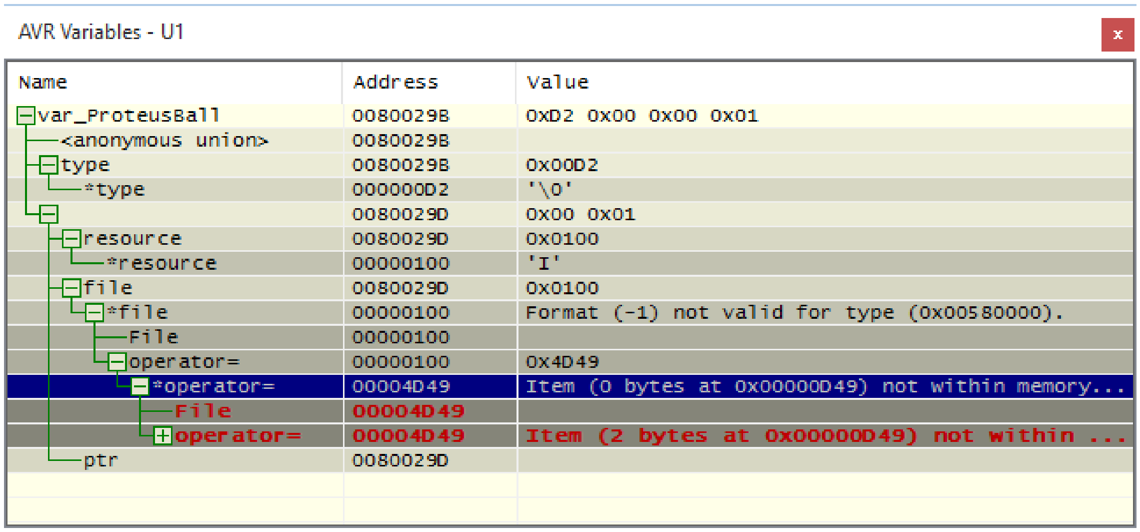
You can, if you prefer, work with two tabs and switch between schematic and Visual Designer as required. You can even drag and drop into space if you are at a workstation with multiple monitors.

10) Variables Window

The variables window is a debugging tool that lists program variables during a debug session. It includes several powerful features to help you when debugging.

Expansion of compound types

The variables window will seamlessly handle compound types (structs, arrays, enums) and pointers, providing an expandable tree displaying the contents of the type or the de-referencing of the pointer.

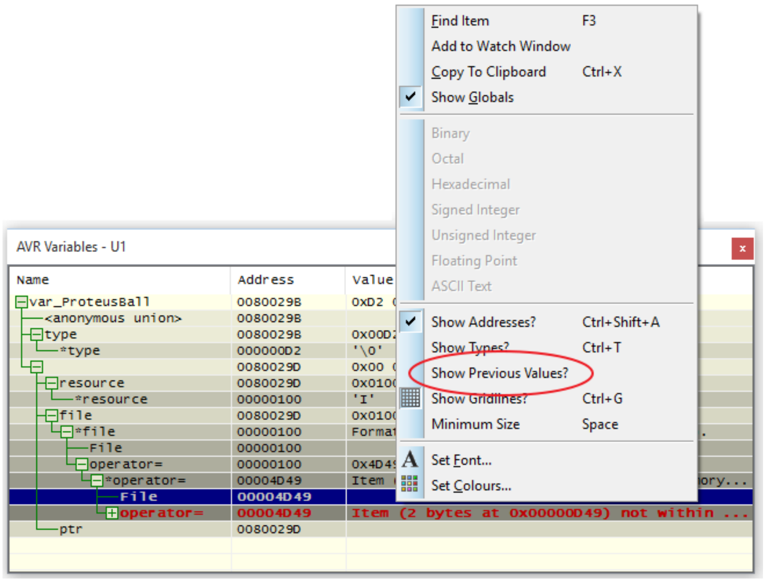


AVR Variables - U1

Name	Address	Value
var_ProteusBall	0080029B	0xD2 0x00 0x00 0x01
<anonymous union>	0080029B	
type	0080029B	0x00D2
*type	000000D2	'\0'
	0080029D	0x00 0x01
resource	0080029D	0x0100
*resource	00000100	'I'
file	0080029D	0x0100
*file	00000100	Format (-1) not valid for type (0x00580000).
File	00000100	
operator=	00000100	0x4D49
*operator=	00004D49	Item (0 bytes at 0x00000D49) not within memory...
File	00004D49	
operator=	00004D49	Item (2 bytes at 0x00000D49) not within ...
ptr	0080029D	

Change Notification and Previous Values

A variable in the variables window will highlight when the value of that variable changes and the simulation is paused. You can also view the previous value of the variable by selecting the Show Previous Values option from the context menu (right click on the variables window).



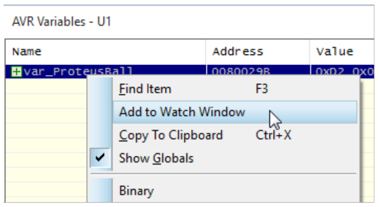
AVR Variables - U1

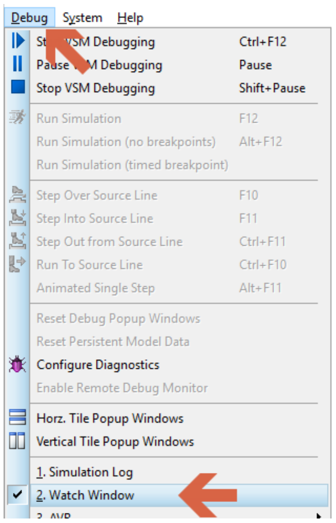
Name	Address	Value
var_ProteusBall	0080029B	0xD2
<anonymous union>	0080029B	
type	0080029B	0x00D
*type	000000D2	'\0'
	0080029D	0x00
resource	0080029D	0x010
*resource	00000100	'I'
file	0080029D	0x010
*file	00000100	Format
File	00000100	
operator=	00000100	0x4D4
*operator=	00004D49	Item ory...
File	00004D49	
operator=	00004D49	Item (2 bytes at 0x00000D49) not within ...
ptr	0080029D	

If you have a compound type (such as a struct) and an element of the type changes then only that element will highlight. This means that, unless the type is expanded, you will not see a change in the variables window at the point the element changes.

Add to Watch Window

The variables window is not visible during free running simulation but it's companion the Watch Window is. You can add a variable to the watch window by the right click context menu and you can then open the watch window from the Debug Menu.




Debug System Help
Start VSM Debugging Ctrl+F12
Pause VSM Debugging Pause
Stop VSM Debugging Shift+Pause
Run Simulation F12
Run Simulation (no breakpoints) Alt+F12
Run Simulation (timed breakpoint)
Step Over Source Line F10
Step Into Source Line F11
Step Out from Source Line Ctrl+F11
Run To Source Line Ctrl+F10
Animated Single Step Alt+F11
Reset Debug Popup Windows
Reset Persistent Model Data
Configure Diagnostics
Enable Remote Debug Monitor
Horz. Tile Popup Windows
Vertical Tile Popup Windows
1. Simulation Log
2. Watch Window
3. AVR

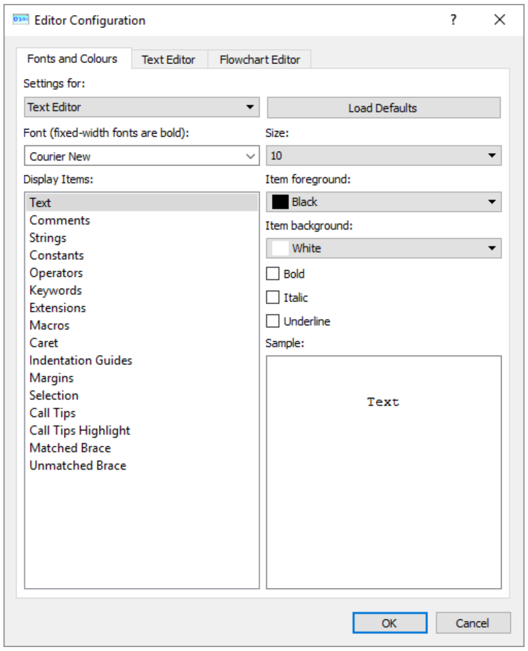
⚠ If you drag a variable to the watch window and you then recompile the program and re-simulate then there is no guarantee that you will be viewing the contents of the variable in the watch window – this depends on how the compiler re-uses memory. The watch window simply monitors an address in memory. Be careful.

In addition to the Variables and Watch Windows there are many more available debug windows which provide more detailed information about the MCU. These can all be launched from the Debug Menu when the simulation is paused and are documented in full in the Proteus VSM help files.

Colours, Fonts and Flowcart Editor

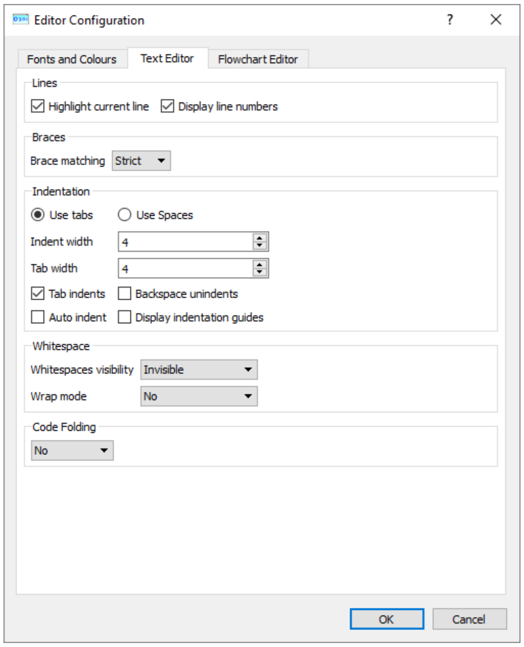
In Visual Designer you can edit the fonts, colours and the style of the Flowchart editor to your own preference. To open the dialog go to the System Menu > Editor Configuration

Fonts and Colours



Settings for: Text Editor
Font (fixed-width fonts are bold): Courier New
Size: 10
Display Items: Text, Comments, Strings, Constants, Operators, Keywords, Extensions, Macros, Caret, Indentation Guides, Margins, Selection, Call Tips, Call Tips Highlight, Matched Brace, Unmatched Brace
Item foreground: Black
Item background: White
☐ Bold
☐ Italic
☐ Underline
Sample: Text

Text Editor



Editor Configuration

Fonts and Colours | Text Editor | Flowchart Editor

Lines

☑ Highlight current line ☑ Display line numbers

Braces

Brace matching: Strict

Indentation

● Use tabs ○ Use Spaces

Indent width: 4

Tab width: 4

☑ Tab indents ☐ Backspace unindents

☐ Auto indent ☐ Display indentation guides

Whitespace

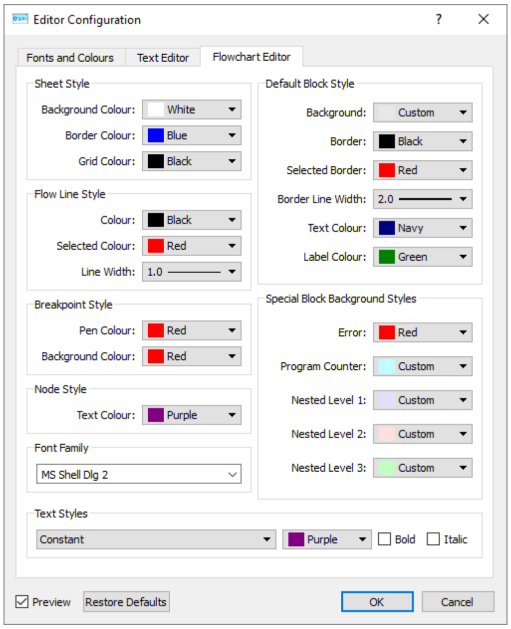
Whitespaces visibility: Invisible

Wrap mode: No

Code Folding

No

Flowchart Editor



Editor Configuration

Fonts and Colours | Text Editor | Flowchart Editor

Sheet Style

Background Colour: ⬜ White

Border Colour: 🟦 Blue

Grid Colour: ⬛ Black

Flow Line Style

Colour: ⬛ Black

Selected Colour: 🟥 Red

Line Width: 1.0

Breakpoint Style

Pen Colour: 🟥 Red

Background Colour: 🟥 Red

Node Style

Text Colour: 🟪 Purple

Font Family

MS Shell Dlg 2

Default Block Style

Background: Custom

Border: ⬛ Black

Selected Border: 🟥 Red

Border Line Width: 2.0

Text Colour: 🟦 Navy

Label Colour: 🟩 Green

Special Block Background Styles

Error: 🟥 Red

Program Counter: Custom

Nested Level 1: Custom

Nested Level 2: Custom

Nested Level 3: Custom

Text Styles

Constant 🟪 Purple ☐ Bold ☐ Italic

☑ Preview

Editing Skills

Overview

This topic explains how to interact with the flowchart editor. The focus here is on learning the skills needed to drive the software and create a program; simulation and debugging are covered separately and we will link to more detailed topics where appropriate.

Navigation

To zoom in and out on the Editing Window

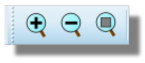
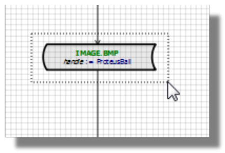
You can either:

1) Roll the middle mouse wheel in and out. This will zoom the Editing Window around the mouse position.

2) Use the zoom icons. This will zoom around the center of the Editing Window.

3) Press the F6 key to zoom in and the F7 key to zoom out. This will zoom the Editing Window around the current mouse position. The F8 key will reset the zoom levels to show the entire sheet.

4) Hold SHIFT key and left drag a box with the mouse. When you release, the Editing Window will zoom to the area drawn out by the box.

To pan across the sheet in the Editing Window

You can either:

1) Roll the middle mouse wheel backwards (Zoom out), move the mouse to desired place and roll the middle mouse wheel forwards to zoom in.

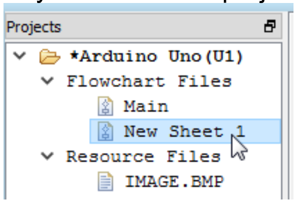
2) Hold the SHIFT key down and bump the mouse against the side of the Editing Window. This will pan the sheet across the Editing Window.

To switch between sheets in the Editing Window

You can either:

1) Click on the sheet that you want on the small tabholder directly above the main Editing window display.

2) Double click on the sheet that you want in the project tree.

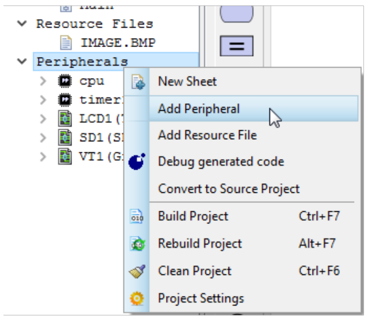


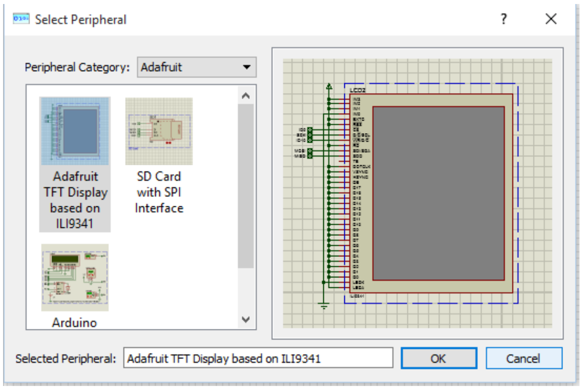
Placing, Selecting and Deleting Blocks

To place a flowchart block

Peripheral Operation Blocks:

Add the peripheral via the right click context menu on the Project Tree and then drag and drop the method onto the Editing Window.



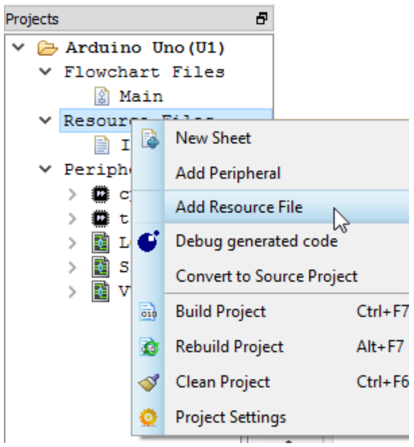

Select Peripheral
Adafruit TFT Display based on ILI9341
SD Card with SPI Interface
Arduino
OK
Cancel

Peripheral Category: Adafruit

Selected Peripheral: Adafruit TFT Display based on ILI9341

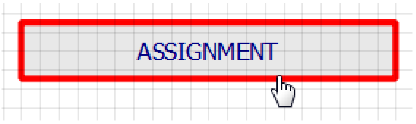
Storage Block Types:

Add the resource via the right click context menu on the Project Tree and then drag and drop onto the Editing Window



Any block type:

Drag and drop from the appropriate flowchart icon across into the Editing Window.



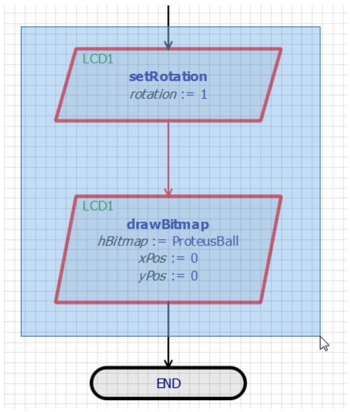
To select a flowchart block:

Left click the mouse on the block.

To select multiple flowchart blocks:

You can either:

1) Drag a box with the left mouse button over the blocks you want to select.



2) Hold SHIFT key down and left click on blocks individually.

Note that many actions are specific to a single flowchart object (e.g Edit). So, while you can for example move selected objects as a group, you will be prevented from performing other block operations.

To delete a flowchart block:

You can either:

1) Right click on the block and select delete from the resulting context menu.

2) Left click to select the flowchart block and press the delete key on the keyboard.

To delete multiple flowchart blocks:

Select the blocks to delete and then press the delete button on the keyboard to remove them.

Attaching, Moving and Detaching Blocks

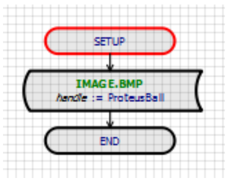
To move an unconnected block:

Select the block by left clicking and then drag the block.

To move a connected block:

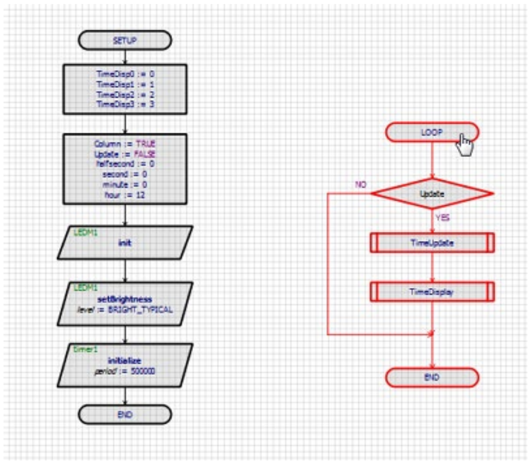
Select and then drag the block. This will bump the adjacent block in the direction of motion and result in a concertina affect. This is an important technique because it is the method you use to make space on a flowchart for inserting a block.

You can quickly 'tidy' a flowchart vertically by moving the top or bottom blocks to make the spacing between all of the blocks the same.



To move a series of connected blocks:

If the set of blocks you want to move includes both a start and end block then you can simply drag a box around it and move it to a better place on the sheet.

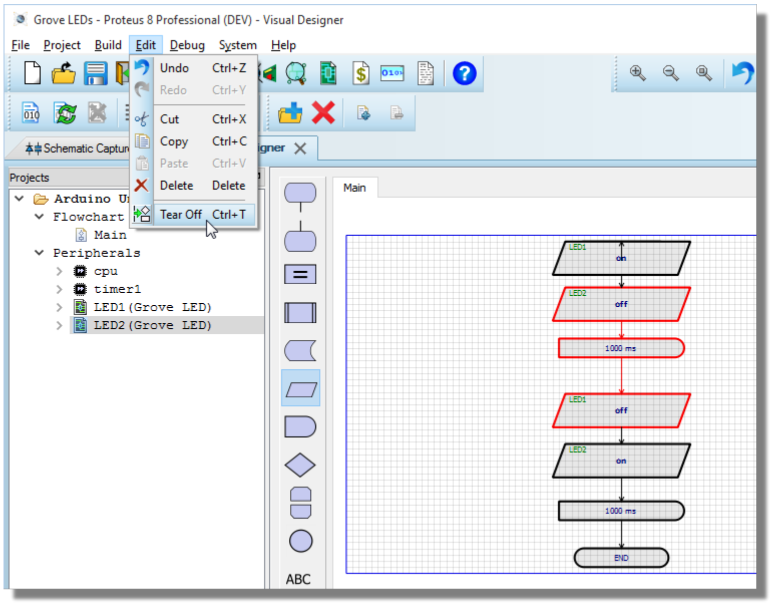


Block move of an entire routine

If you want to move the routine onto a different sheet you can use the clipboard cut and paste across the two sheets.

If you want to move a series of connected blocks that form part of a routine, but not the full routine, then you must either:

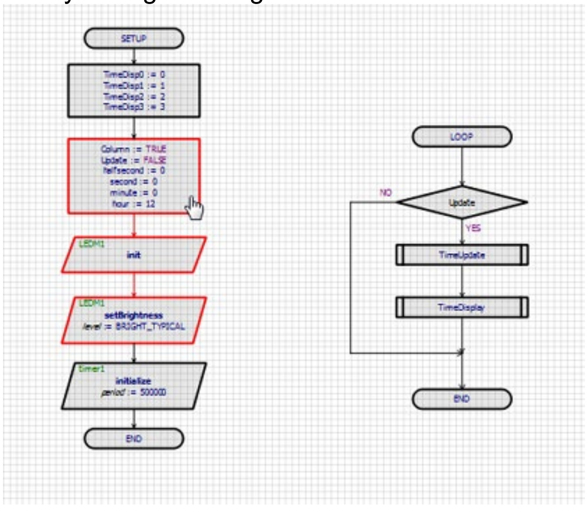
1) Tear off the series of blocks and move them elsewhere.



2) Move them in the context of the current routine.

- If you move vertically this will bump other blocks closer together but it will not allow you to change the position of the selected blocks in the flowchart.
- If you move horizontally it will move the entire routine horizontally.

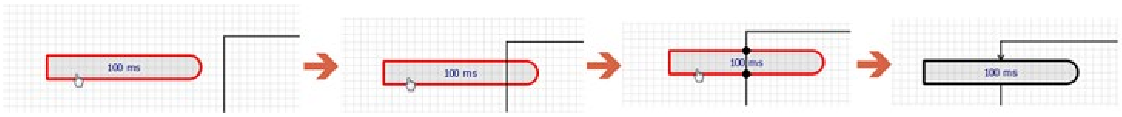
Also, it can lead to some very strange looking charts!



Block Move of part of a routine.

To attach a block to a flowchart

When you want to insert a block you do so by dragging and dropping. As you drag the flowchart block you will see connection nodes appear when the block is correctly positioned over the wire. Releasing at this point will connect the block to the flowchart.

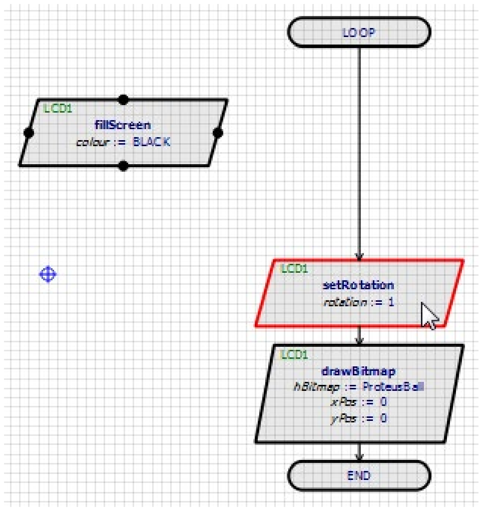


Connection Nodes tell you when to release the mouse - this will auto-connect the block to the chart

You can do this as you initially place the block or by selecting a block that is placed but unattached.

Note that some block can only be connected at top and bottom (e.g. delay block) while others also allow sideways entry (e.g. assignment block).

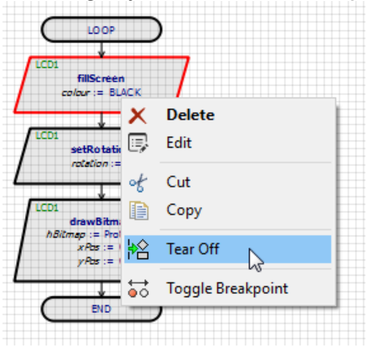
If you are inserting a block you may need to first make some space. You can do this easily by dragging one of the blocks in the flowchart up or down to clear an area for the new block to go.



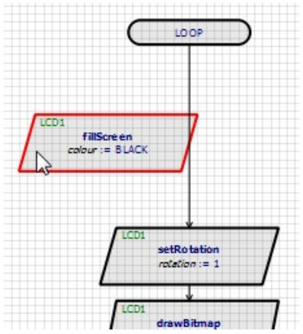
To detach a block from a flowchart

You can detach or tear off a block from a flowchart by either:

1) Right click on the block and select tear off from the resulting context menu. This will detach the block and leave it slightly to the side for re-positioning.



2) Select the block and hold the CTRL button down as you move the block away. This will tear the block from the current chart and you can then re-position as required.



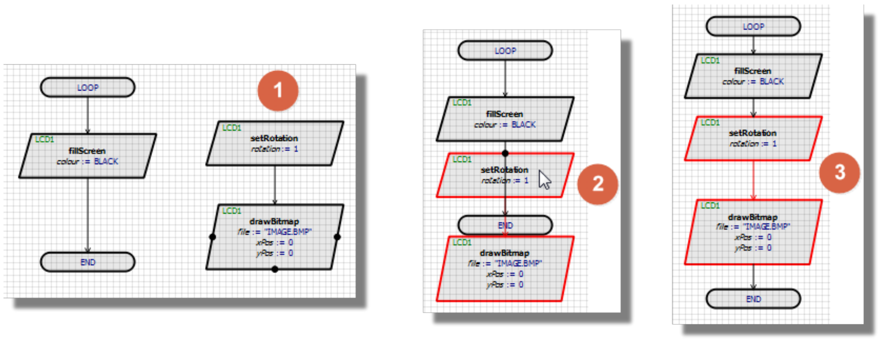
To attach a series of flowchart blocks

1) Make some room at the place in the chart you want to insert.

2) Select the series of 'connected' blocks you want to insert.

3) Move them into place, watching the top connection nodes of the uppermost block. When this appears, check that the bottom connection node is also visible.

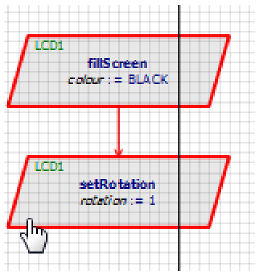
4) Release the mouse to insert.



To detach a series of flowchart blocks

1) Select the series of blocks you want to insert.

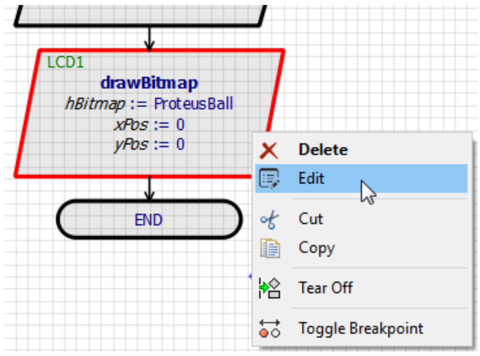
2) Hold the CTRL button down and drag the blocks off the flowchart or use the tear off command on the Edit Menu.



Commands on the context menu apply to a single block whereas commands on the Edit menu apply to all selected blocks. This is an important distinction as some commands (e.g. Edit) are meaningless for multiple blocks.

Editing a Block

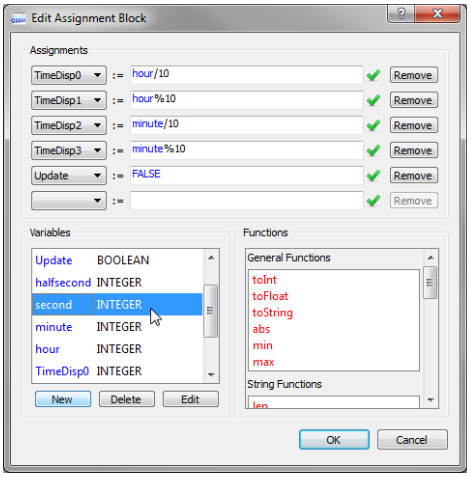
To edit a block you can either right click and select Edit from the resulting context menu or you can double click on the block.



Regardless of the type of block you are editing you are almost certain to want to enter an expression to give the block meaning. In some cases such as calling a subroutine this will be extremely simple but for other block types it can be a little more complex. Fortunately, Visual Designer can help in several different ways:

Variable and Function Listing

Where appropriate, all current variables are displayed in the bottom left of the dialogue form and standard library functions are displayed in the bottom right of the dialogue form. Double clicking on variable/function will add it to the current expression.
You can easily create, edit and remove variables via the buttons at the bottom of the variables display.


Edit Assignment Block

Assignments

TimeDisp0	:=	hour/10	Remove
TimeDisp1	:=	hour%10	Remove
TimeDisp2	:=	minute/10	Remove
TimeDisp3	:=	minute%10	Remove
Update	:=	FALSE	Remove
	:=		Remove

Variables

Update	BOOLEAN
halfsecond	INTEGER
second	INTEGER
minute	INTEGER
hour	INTEGER
TimeDisp0	INTEGER

New Delete Edit

Functions

General Functions
toInt
toFloat
toString
abs
min
max
String Functions
len

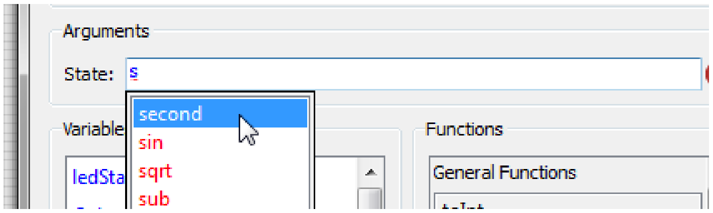
OK Cancel

Multiple Expression Assignments using Variables

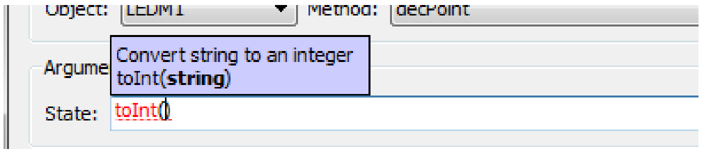
Combo-boxes will be used to restrict selection to variables of the correct type where possible. For example, the loop variable in a For-Next loop must be of type INTEGER.

Auto Complete

The auto-complete system will both prompt you with function hinting during entry and also automatically complete named variables when typing. Press TAB or ENTER to auto-complete.



Auto-completion

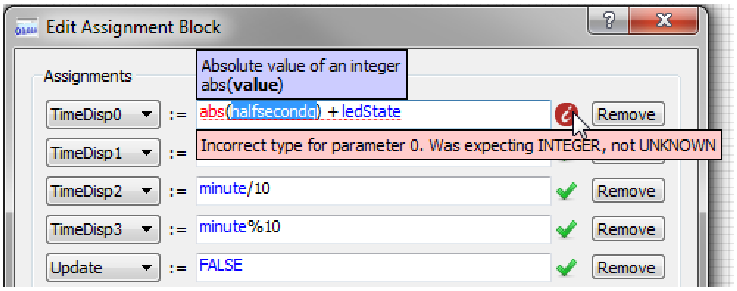


Convert string to an integer
toInt(**string**)

Function hinting

Syntax and Type Checking

Comprehensive syntax and type checking takes place as you enter the expression. The expression status is indicated at the right hand side and clicking on the red info icon will provide detail on any errors in the expression.



Absolute value of an integer
abs(**value**)

TimeDisp0	:=	abs(halfsecondq) + ledState	Remove
TimeDisp1	:=		
TimeDisp2	:=	minute/10	Remove
TimeDisp3	:=	minute%10	Remove
Update	:=	FALSE	Remove

Incorrect type for parameter 0. Was expecting INTEGER, not UNKNOWN

Get more information on incorrect expressions.

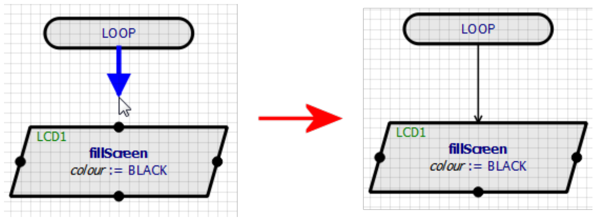
Refer to specifics on individual block types for more detailed information.

Wiring blocks together

Blocks are connected together with flowlines and Visual Designer provides an point-to-point router to help with the task. The golden rule is that you must connect from an output to an input.

To join blocks together with a flowline

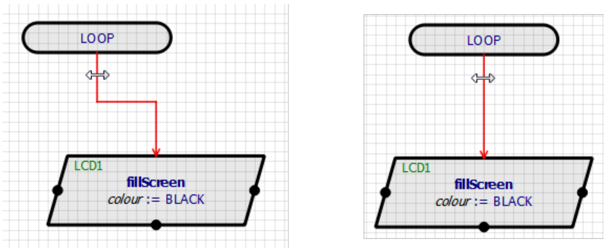
1) Click on the output connection node.
2) Move the mouse to the destination (input) connection node.
3) Click to complete the flowline.



It's good practise to connect flowlines from top downwards to bottom because it makes loops much clearer. Try to avoid connecting from bottom to top in normal flow.

To adjust / move a placed flowline

1) Click to select the flowline.
2) The mouse cursor will change to show you the available direction of movement.
3) Left drag with the mouse to drag.



To delete a flowline

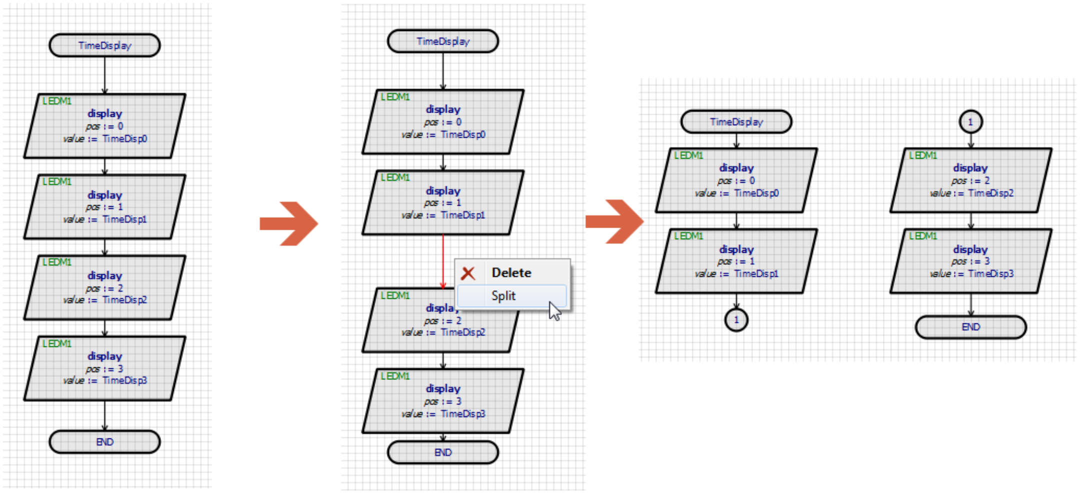
1) Right click on the flowline and select the delete option from the context menu.

Splitting the flowchart into columns

If the flowchart is getting to the bottom of the page inside a single routine you may need to split the flow with interconnect blocks.

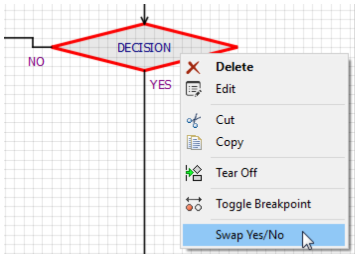
1) Right click on the flowline near the bottom of the page and select split from the resulting context menu.

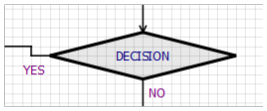
2) Block Move the bottom portion up to the top of the page at the right hand side.
3) Make some space.
4) Continue inserting blocks.



To change the flow of decision blocks

1) Place two flowlines corresponding to the output paths of the decision.
2) If the YES and NO paths are the wrong way around, right click on the decision and swap them.



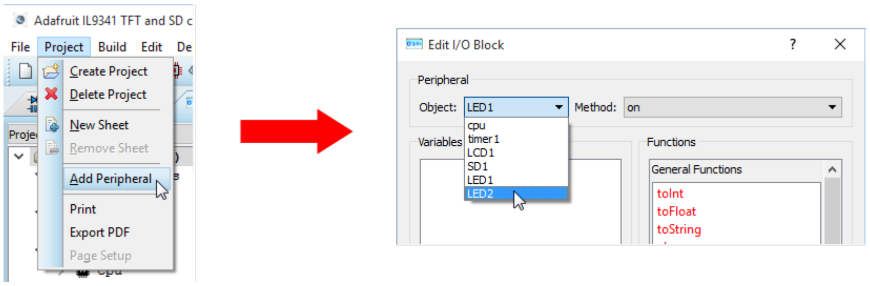


Clipboard Commands

Clipboard commands in Visual Designer work in exactly the same way as other Windows applications and can by invoked in the normal way.

- CTRL+X : Cut to Clipboard.
- CTRL+C : Copy to Clipboard.
- CTRL+V : Paste from Clipboard.
- Menu Icons.
- Commands on Edit Menu.

The most common use for the clipboard in Visual Designer is for organizing flowchart routines in the Editing Window and, in particular, for moving routines between sheets.
Remember when copying and pasting that you may be pasting blocks that include peripherals (e.g. hardware shields). In this case the paste will duplicate the flowchart blocks and assignments but will not create new hardware on the schematic. If that is what you want to do, you need to add a peripheral, edit the relevant pasted blocks and assign the method to the correct hardware.

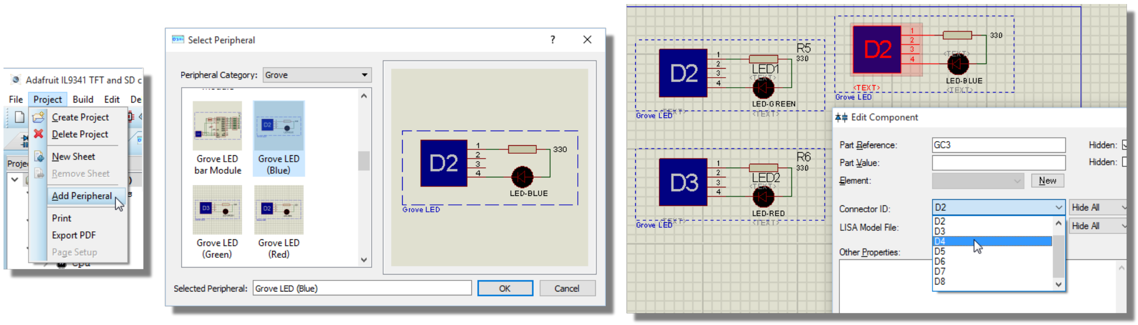


Object: LED1
Method: on

Adding, Using and Removing Peripherals

To add a peripheral device

1) Right click on Project Tree and select Add Peripheral from the resulting context menu.
2) Select the shield or Grove peripheral from the Browser.
3) If you are using Grove peripherals you may need to switch to the schematic module, double click on the label and change the connector to a unique ID.

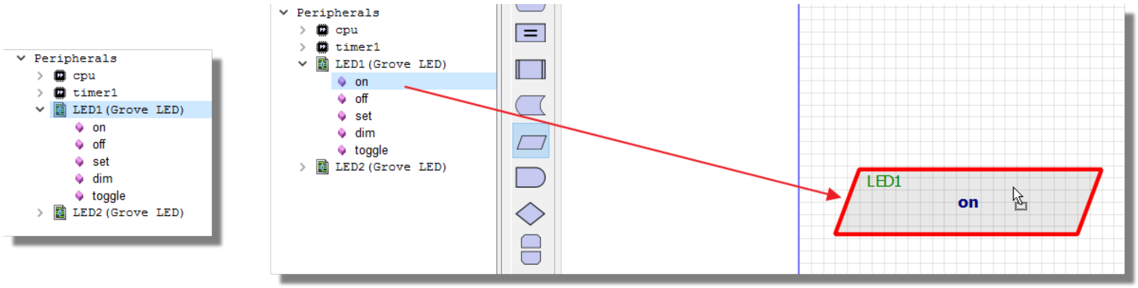


Peripheral Category: Grove
Selected Peripheral: Grove LED (Blue)
Part Reference: GC3
Connector ID: D2

With Arduino, it is quite easy to add shields that are incompatible with each other. For example, if two shields use the same CPU timer, the program is unlikely to work. You need to check this compatibility yourself as Visual Designer does not interfere with the Arduino drivers.

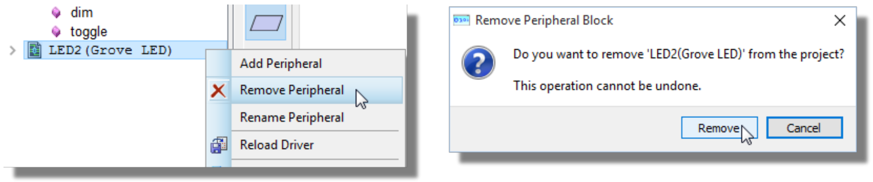
To use a peripheral device

1) Expand the menu in the Project tree to access the methods of the peripheral.
2) Drag and drop the method onto the flowchart.



To remove a peripheral device

1) Right click on the peripheral in the Project tree and select Remove Peripheral from the context menu.
2) Confirm from the resulting dialogue box.



Do you want to remove 'LED2(Grove LED)' from the project?

This operation cannot be undone.

If you remove a peripheral device, it will not remove any flowblocks that referenced the device. You need to adjust or delete all such blocks manually.

Adding, Using and Removing Variables

Variables are created and manipulated inside the object editing dialogue forms and are always global to the project. The following types are supported:

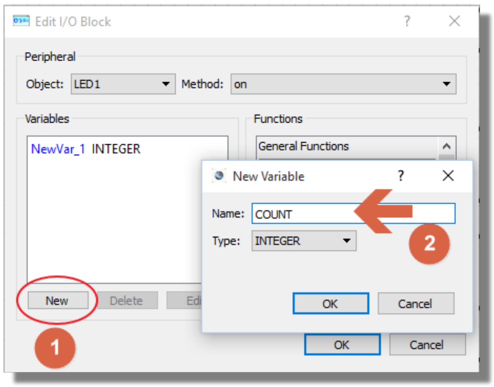
- Boolean (TRUE / FALSE)
- Integer (Integer number)
- Float (Floating point number)
- String (String of characters)
- Handle (Handle to a resource).

The expression editor will type-check assignments automatically.

To create a new variable

1) Edit the block you are working on.
2) At the bottom left, use the New button to launch the Create Variable dialogue.

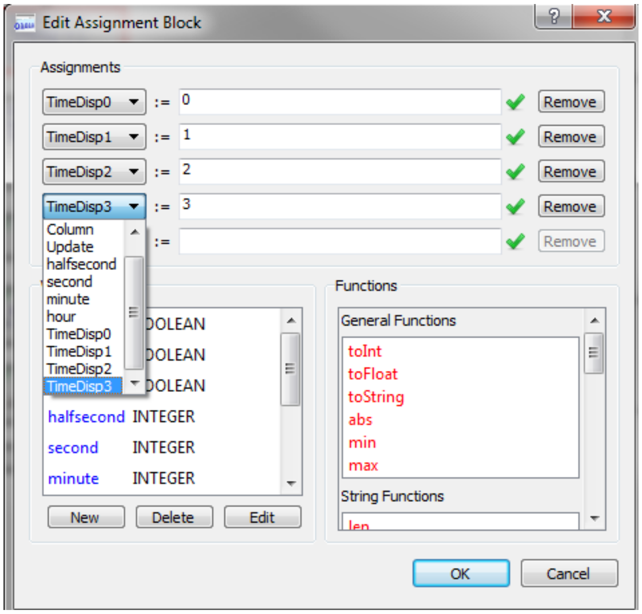
3) Enter the name of the variable and assign it a type.
4) Click OK to exit the dialogue.



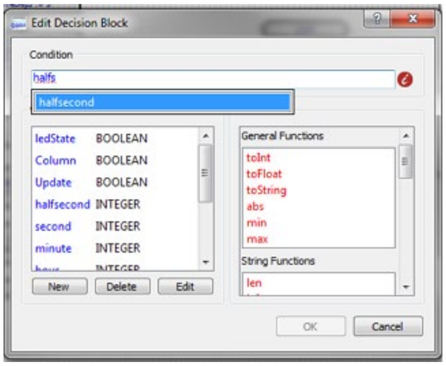
1, Click New. 2, Set the Variable and click OK.

To use an existing variable

Edit the block you are working on and then either:
1) Use the variable drop down to select the variable.



2) Type the variable name and use the auto-complete.



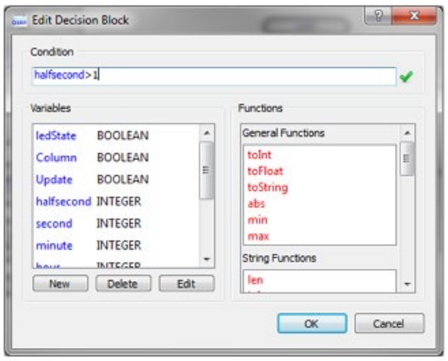
Condition: halfs

halfsecond

ledState	BOOLEAN
Column	BOOLEAN
Update	BOOLEAN
halfsecond	INTEGER
second	INTEGER
minute	INTEGER

General Functions: toInt, toFloat, toString, abs, min, max

String Functions: len



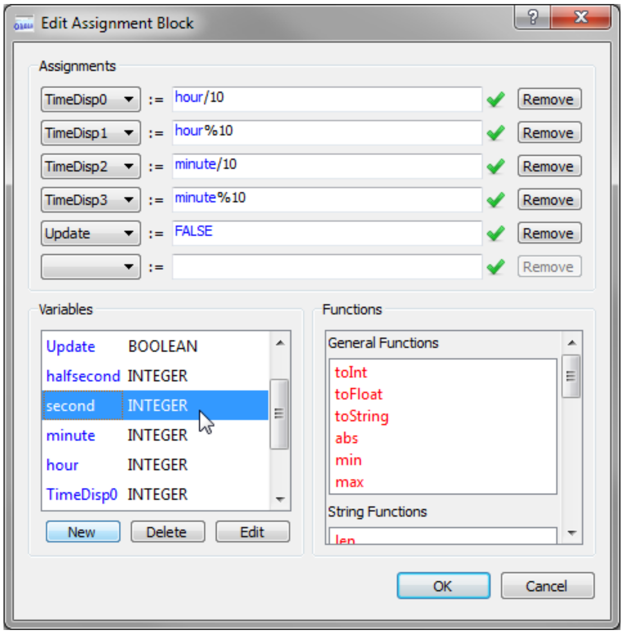
Condition: halfsecond>1

ledState	BOOLEAN
Column	BOOLEAN
Update	BOOLEAN
halfsecond	INTEGER
second	INTEGER
minute	INTEGER

General Functions: toInt, toFloat, toString, abs, min, max

String Functions: len

3) Double click the variable name from the list to insert into the expression.



TimeDisp0	:=	hour/10
TimeDisp1	:=	hour%10
TimeDisp2	:=	minute/10
TimeDisp3	:=	minute%10
Update	:=	FALSE
	:=	

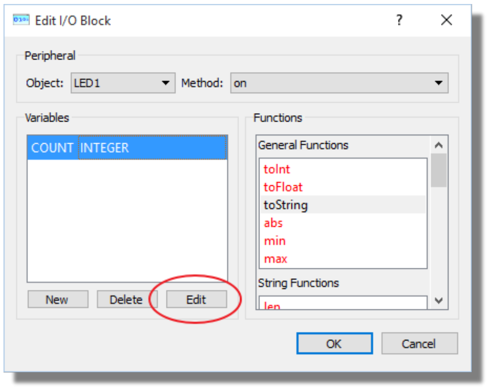
Update	BOOLEAN
halfsecond	INTEGER
second	INTEGER
minute	INTEGER
hour	INTEGER
TimeDisp0	INTEGER

General Functions: toInt, toFloat, toString, abs, min, max

String Functions: len

To edit a variable

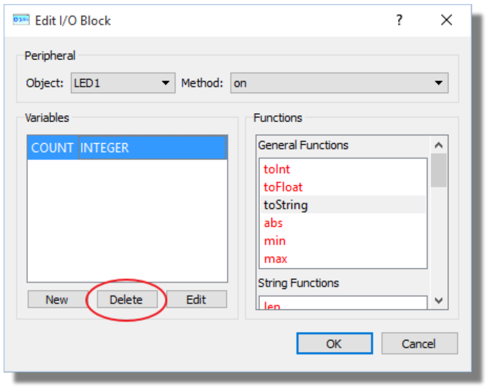
1) Edit one of the blocks on the flowchart.
2) Select the variable from the list on the left hand side.
3) Click the edit button and change the name and/or type of the variable.



Object: LED1
Method: on
COUNT INTEGER

To remove a variable

1) Edit one of the blocks on the flowchart.
2) Select the variable from the list on the left hand side.
3) Click delete to remove the variable



Object: LED1
Method: on
COUNT INTEGER

If you edit or remove a variable you need to remember to check and change all the other flowchart blocks in which the variable has been used.

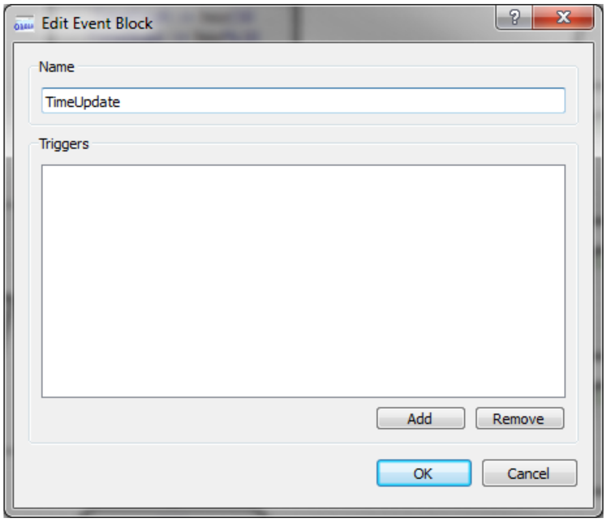
Flowchart Blocks

Visual Designer includes a small set of flowchart blocks which are used as the programming constructs in your firmware. These will be familiar to many users and are listed below in full.

ICON	DESCRIPTION	ICON	DECRIPTION
⬭	Event Start Block: Used together with an end block to define a sub-routine or event handler.	◗	Time Delay: A time delay in program execution.
⬭	End Block: Used together with an event start block to define a sub-routine or event handler.	◇	Decision Block: A boolean decision in the flowchart.
⊟	Assignment Block: Used to assign a value to a variable.	⬓	Loop Construct: Used to simplify the configuration of different types of loop.
▭	Subroutine Call: Used to call a sub-routine.	○	Interconnector : Used in pairs to connect longer sequences of blocks into separate columns on the editor.
◖	Stored Data Block: Used to specify an action on a storage object (such as an SD Card).	ABC	Comment: Allows you to add a textual comment on the flowchart.
▱	I/O (Peripheral) Operation: Used to specify an action on the hardware, either internally to the CPU (e.g. timer) or external (e.g. LCD).		

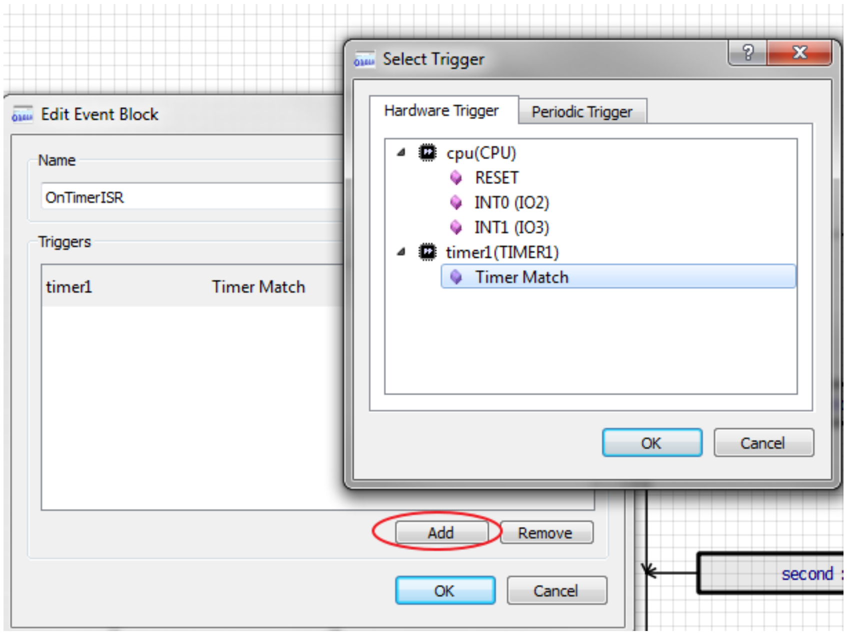
Event / Subroutine Block

The event block is used together with the end block to define the start and end of sub-routines (e.g. to write to a display) and event handlers (e.g. to handle a timer interrupt).
If you are creating a subroutine then all that is required is a name for the block. This is the name of the routine and will be selectable when placing and editing subroutine call blocks.



Simple Subroutine Config - Only the name is required.

If you are creating a routine to handle a triggerable event such as an interrupt handler then you need to also specify the trigger



Setting up a timer interrupt service routine based on a timer1 match event.

The periodic trigger provides a very handy way to get a repeatable call at regular time intervals. It does however make use of one of the CPU timers so be careful not to use it elsewhere if you configure it here for use here.

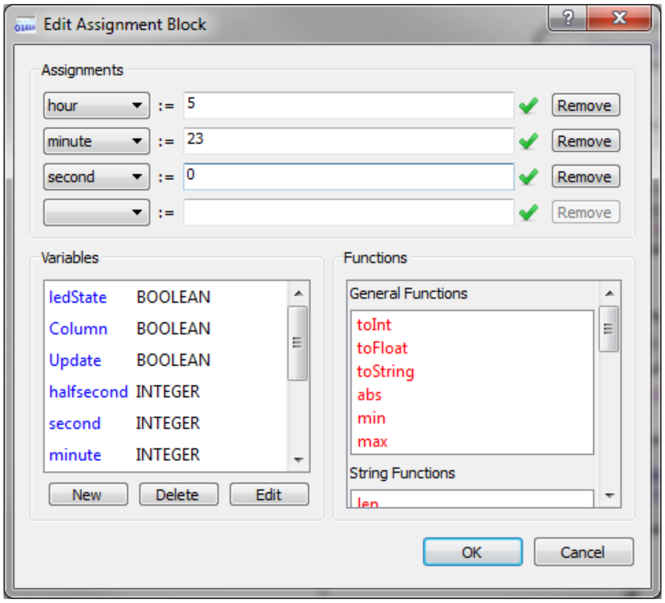
End Block

The end block is used to terminate a routine or event handler and is normally placed together the event block.

A typical Arduino project will initialize with two event and end blocks, one for program setup and the other for main program execution.

Assignment Block

This is the standard block for assigning values to variables. It is also the place to create new, edit and delete variables.


Edit Assignment Block

Assignments

hour	:=	5	Remove
minute	:=	23	Remove
second	:=	0	Remove
	:=		Remove

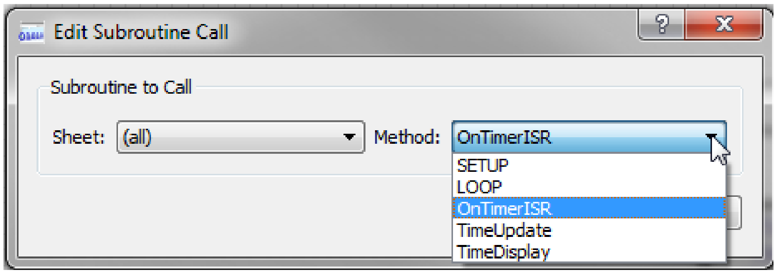
Variables

ledState	BOOLEAN
Column	BOOLEAN
Update	BOOLEAN
halfsecond	INTEGER
second	INTEGER
minute	INTEGER

Assigning values to variables for the time.

Subroutine Call Block

The subroutine block lets you call a function defined elsewhere on the flowchart.



Edit Subroutine Call

Subroutine to Call

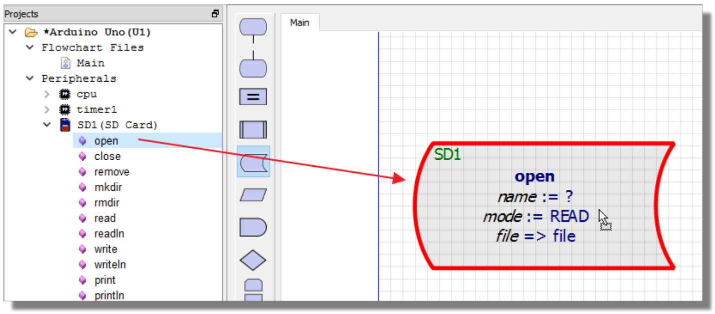
Sheet: (all) Method: OnTimerISR

Specifying the subroutine to call.

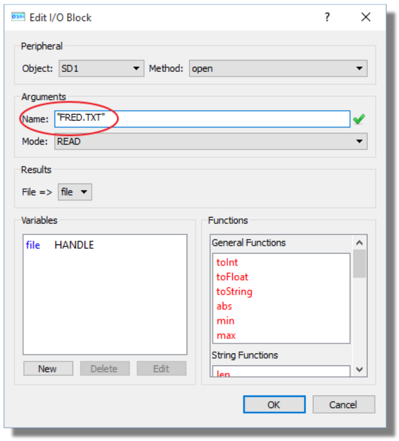
You must first create and name the subroutine with an event and end block. The combo-box on the subroutine dialogue lists all named routines.

Stored Data Block

The storage block is used to denote an action or method on a storage object (SD Card). The methods are then typically used with resources (e.g a text file) to manipulate the filestore.



SD1
open
name := ?
mode := READ
file => file



Edit I/O Block

Peripheral

Object: SD1 Method: open

Arguments

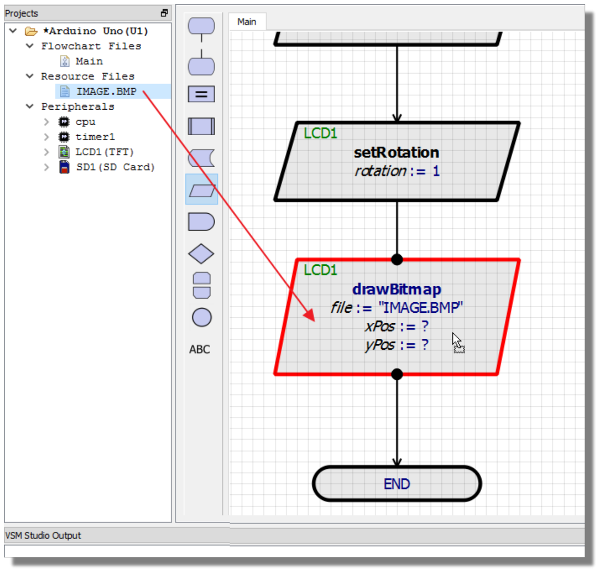
Name: "FRED.TXT"

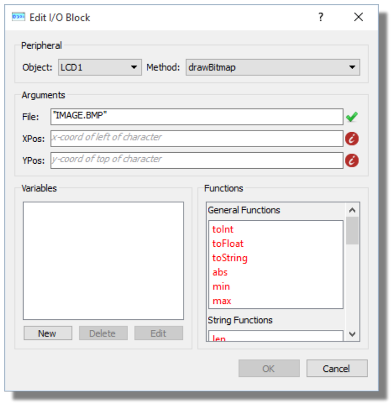
Mode: READ

Results

File => file

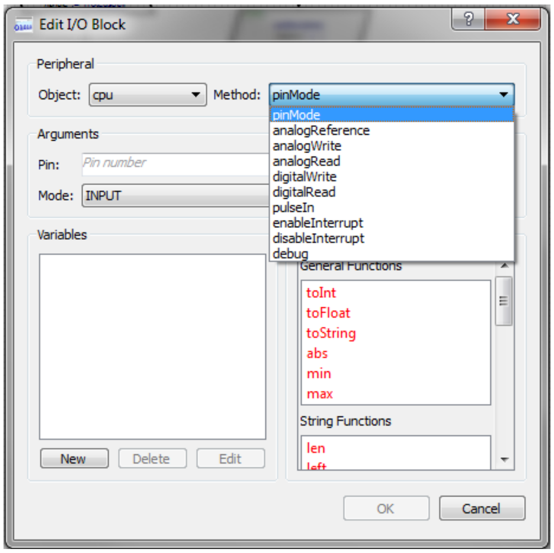
When you add a shield that contains an SD card it will almost always have a method for using the contents of the SD Card. For example, a TFT shield has a DrawBitmap() method and a Wave shield has a Play() method. In these cases, you should not use the storage data blocks but rather just drag and drop the resource onto the chart





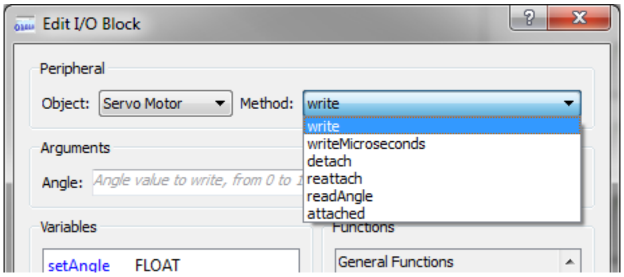
Peripheral Operation Block

A peripheral operation block basically allows you to perform an action on the hardware. With a new project the hardware will consist of only the processor and the available actions are therefore limited.



Available actions on the processor.

However, as you add peripheral shields to your project you will be able to interact and control their I/O via this block type.


Edit I/O Block

Peripheral

Object: Servo Motor Method: write

write
writeMicroseconds
detach
reattach
readAngle
attached

Arguments

Angle: Angle value to write, from 0 to 1

Variables

setAngle FLOAT

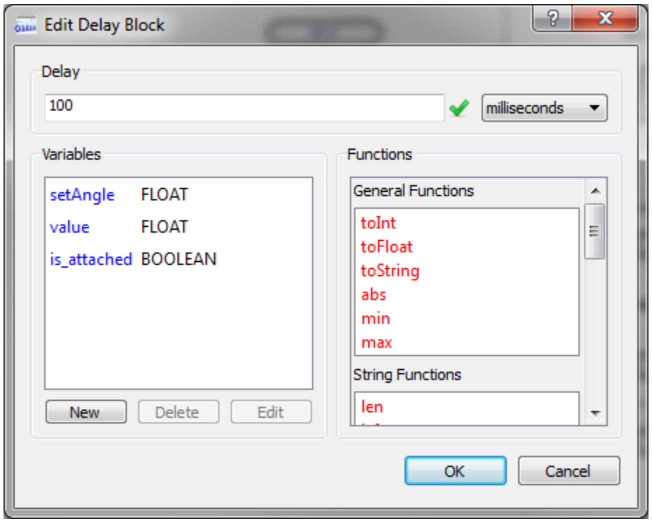
Functions

General Functions

Available methods on the Grove Servo Motor Shield.

Time Delay Block

This block is used to introduce a specific delay in the program.


Edit Delay Block

Delay

100 ✓ milliseconds

Variables

setAngle	FLOAT
value	FLOAT
is_attached	BOOLEAN

Functions

General Functions

toInt
toFloat
toString
abs
min
max

String Functions

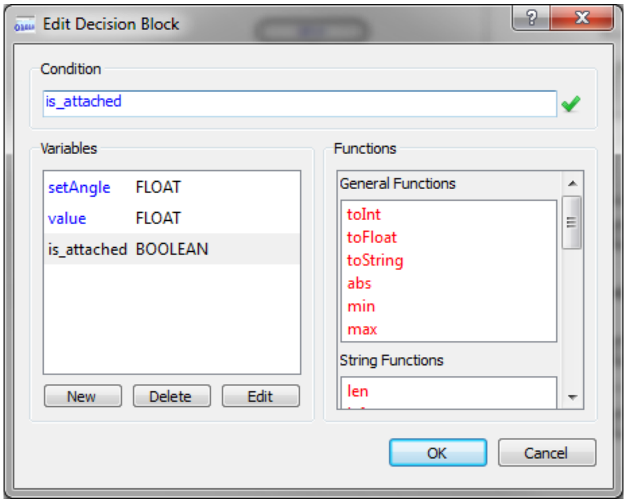
len

Specifying a 100ms Delay in the program.

In Arduino, no other reading of sensors, mathematical calculations, or pin manipulation can go on during the delay function, so in effect, it brings most other activity to a halt. Interrupts and other specific functions will however still work. Refer to the Arduino documentation on Delay() for more detail.

Decision Block

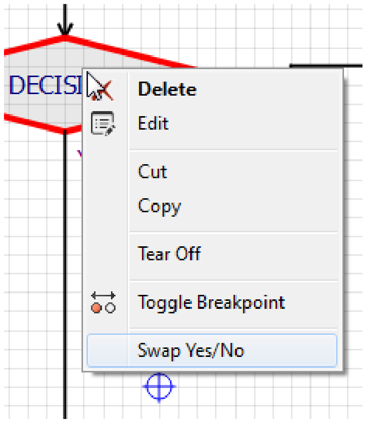
The decision block is the basic yes/no question that determines desired program flow. The dialogue form is fairly self-explanatory and requires a boolean expression.



setAngle	FLOAT
value	FLOAT
is_attached	BOOLEAN

Testing the is_attached variable in a decision block.

When a decision block is linked on the flowchart, the YES and NO labels are placed in default locations. You can swap them over via the context menu if that better suits the flowchart.



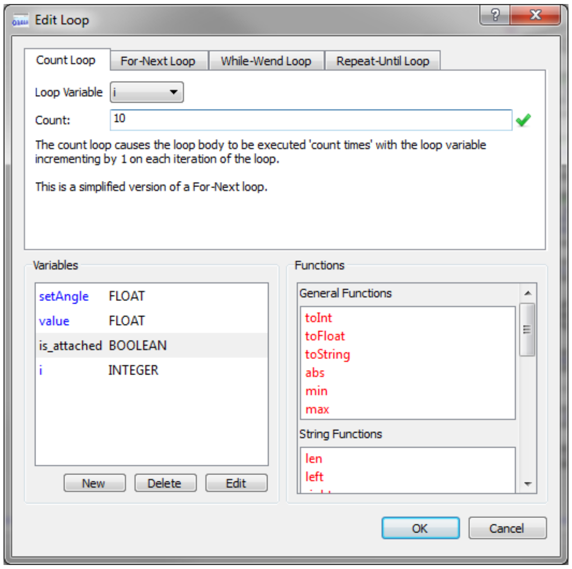
Swap the YES and NO flowlines on a decision block.

Loop Construct Block

The loop construct provides a simple, dialogue driven way to configure some common types of program loop. It is placed as a pair and the required logic blocks are then attached inside the loop. The edit loop dialogue then provides several options:

Count Loop

The count loop is simply a way of executing the loop body a specified number of times.



Loop Variable: i

Count: 10

The count loop causes the loop body to be executed 'count times' with the loop variable incrementing by 1 on each iteration of the loop.

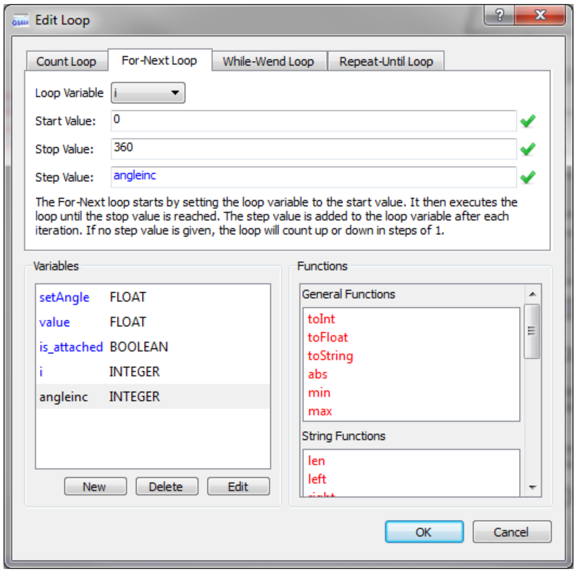
This is a simplified version of a For-Next loop.

setAngle	FLOAT
value	FLOAT
is_attached	BOOLEAN
i	INTEGER

Count Loop

For-Next Loop

A For-Next Loop is an extension of the count loop as you need to specify the start, stop and step to determine how often the loop body will execute.



Loop Variable: i

Start Value: 0

Stop Value: 360

Step Value: angleinc

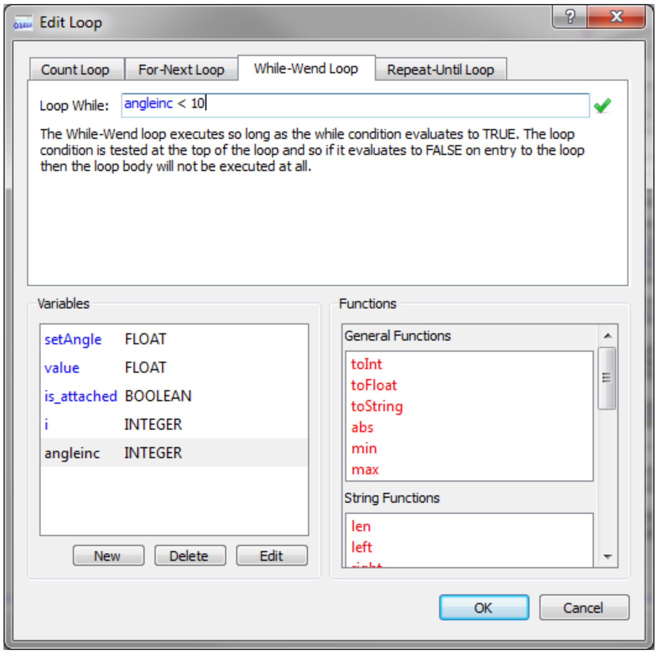
The For-Next loop starts by setting the loop variable to the start value. It then executes the loop until the stop value is reached. The step value is added to the loop variable after each iteration. If no step value is given, the loop will count up or down in steps of 1.

setAngle	FLOAT
value	FLOAT
is_attached	BOOLEAN
i	INTEGER
angleinc	INTEGER

For-Next Loop.

While-Wend Loop

This construct executes the body of the loop while the tested condition evaluates as TRUE.



Loop While: angleinc < 10

The While-Wend loop executes so long as the while condition evaluates to TRUE. The loop condition is tested at the top of the loop and so if it evaluates to FALSE on entry to the loop then the loop body will not be executed at all.

setAngle	FLOAT
value	FLOAT
is_attached	BOOLEAN
i	INTEGER
angleinc	INTEGER

General Functions: toInt, toFloat, toString, abs, min, max

String Functions: len, left

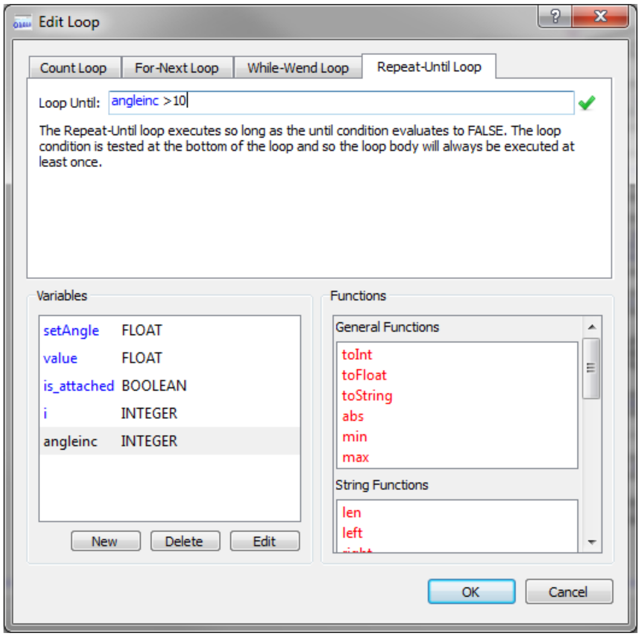
While-Wend Loop

Note that the test takes place at the top of the loop and so if the test is FALSE to begin with the loop contents will not be executed at all

Repeat-Until Loop

Repeat-Until is very similar to While-Wend except:

- It executes until the tested condition evaluates to FALSE
- The test takes place at the bottom of the loop and so the loop body always executes at least once.



Edit Loop

Count Loop | For-Next Loop | While-Wend Loop | Repeat-Until Loop

Loop Until: angleinc >10

The Repeat-Until loop executes so long as the until condition evaluates to FALSE. The loop condition is tested at the bottom of the loop and so the loop body will always be executed at least once.

Variables

setAngle	FLOAT
value	FLOAT
is_attached	BOOLEAN
i	INTEGER
angleinc	INTEGER

New Delete Edit

Functions

General Functions
toInt
toFloat
toString
abs
min
max

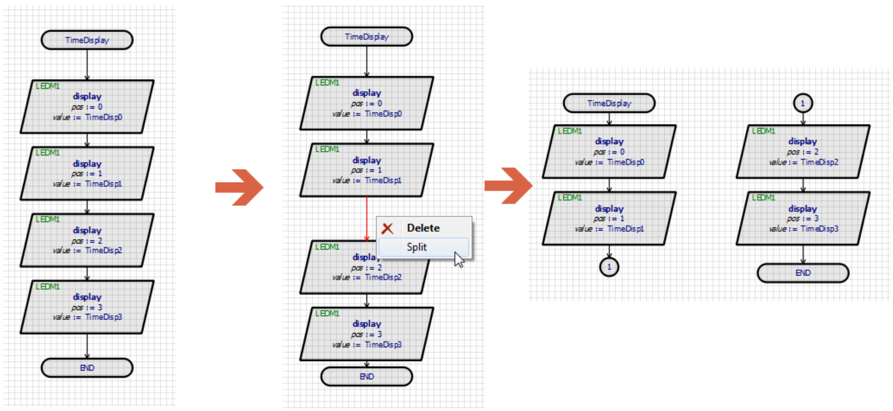
String Functions
len
left

OK Cancel

Repeat-Until Loop

Interconnector Block

The interconnect blocks are basically 'virtual connections' and need to be in pairs. If you have two interconnect blocks with the same number then you can imagine an invisible wire joining them together. The purpose of the interconnect blocks is to let you split flowchart logic into multiple columns.
You can either drag and drop two interconnect blocks, link them on the flowchart and then re-number appropriately, or you can simply right click and split a flowline.

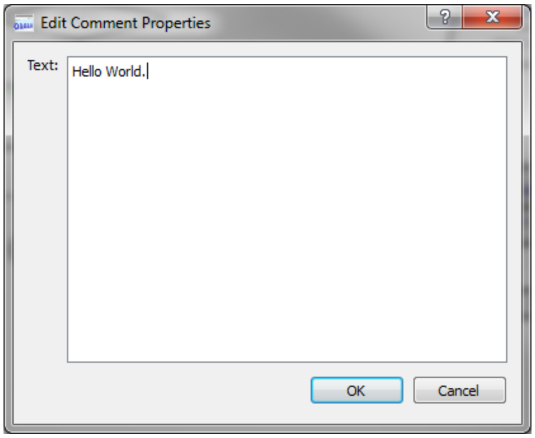


Splitting into two separate lines with interconnect

You can join two interconnects together into a single line by dragging one over the top of the other.

Comment Block

The comment block enables a free format block for the entry of descriptive text.



Comment Block Dialog

ARDUINO HARDWARE

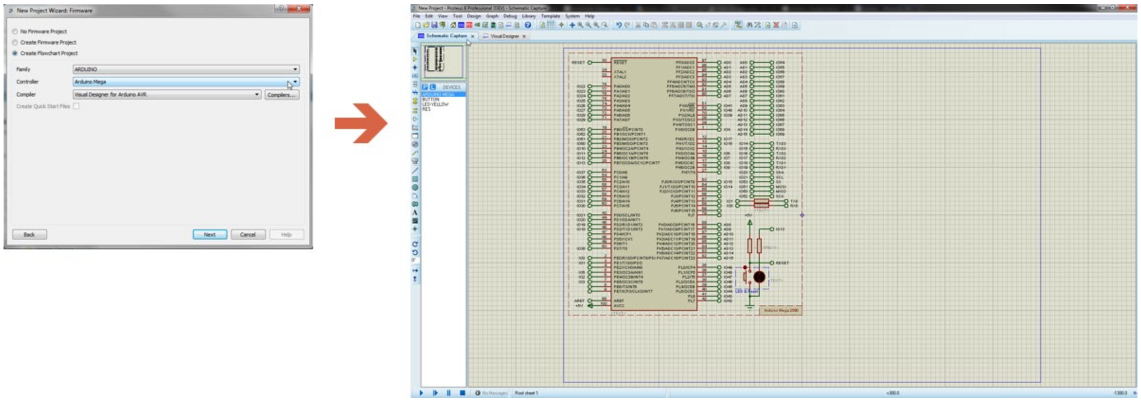
Introduction

Designing 'virtual hardware' for your project is greatly simplified in Visual Designer. The various options available to you are described in detail below.

Arduino Base Board

In hardware, Arduino is a system consisting of a fairly simple base board with an AVR microcontroller, some supporting electronics and Arduino pin headers (connectors). There are then a wealth of peripheral boards (shields) which plug into the pin headers to form the hardware for a particular project.

In Visual Designer for Arduino, the creation of the project is the point at which you make your decision on the base board (E.g. Arduino Uno, Arduino Mega). This is automatically placed on the schematic when the project is created.


New Project Wizard: Firmware

No Firmware Project
Create Firmware Project
[✓ Create Flowchart Project]

Family: ARDUINO
Controller: Arduino Mega
Compiler: Visual Designer for Arduino AVR
Create Quick Start Files

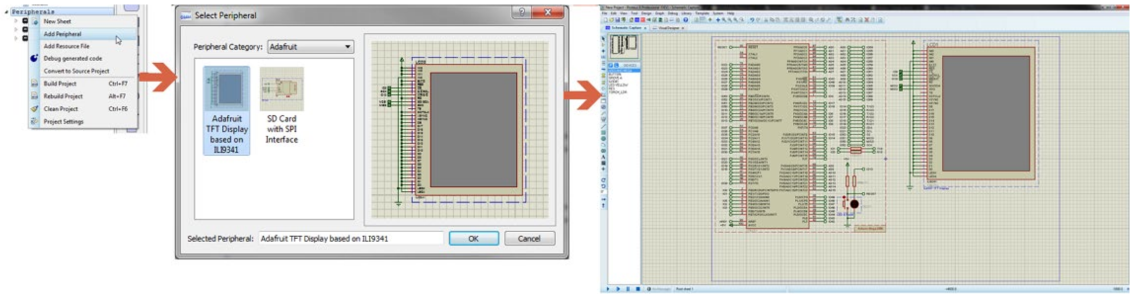
Automatic placement of the Arduino Mega on the Schematic when the Project is created.

Arduino Shields

Arduino shields are modular circuit boards that piggyback onto your Arduino base-board to instill it with extra functionality. They are the simplest way to add hardware functionality to your project because the circuitry and connectivity on the schematic happens automatically.

Adding a Shield

When you then want to add a shield in Visual Designer you right click on the Project Tree and select 'Add Peripheral' from the resulting context menu. Select the 'Adafruit' category on the peripheral browser and then select the shield you want to add. This will auto-place the shield electronics on the schematic.

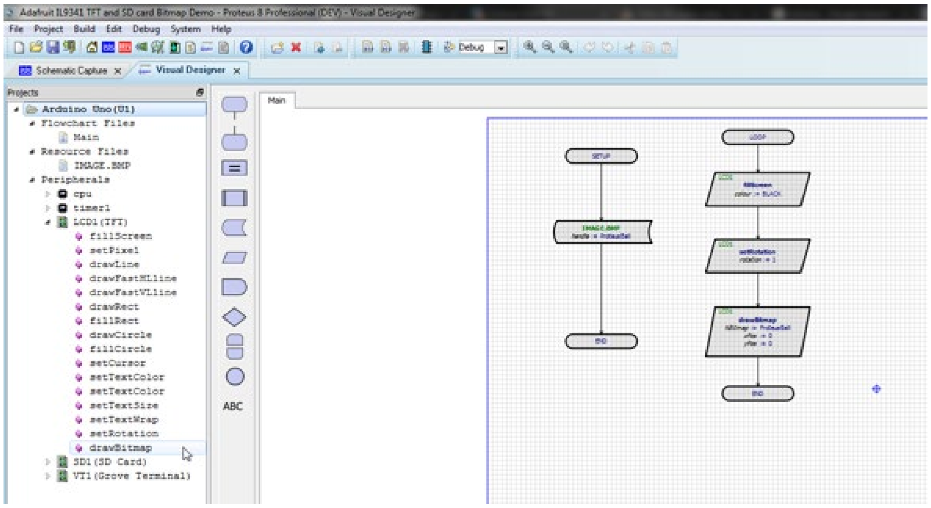


Peripheral Category: Adafruit
Selected Peripheral: Adafruit TFT Display based on ILI9341

Adding a Peripheral will auto-place and auto-connect on the schematic.

Writing a Program to Control a Shield

Shields appear in the project tree in Visual Designer once picked from the Peripheral Gallery. They include high level methods that allow you to quickly control the connected electronics via flowchart blocks.



Drag and Drop methods make it simple to drive the shields.

Potential Pitfalls

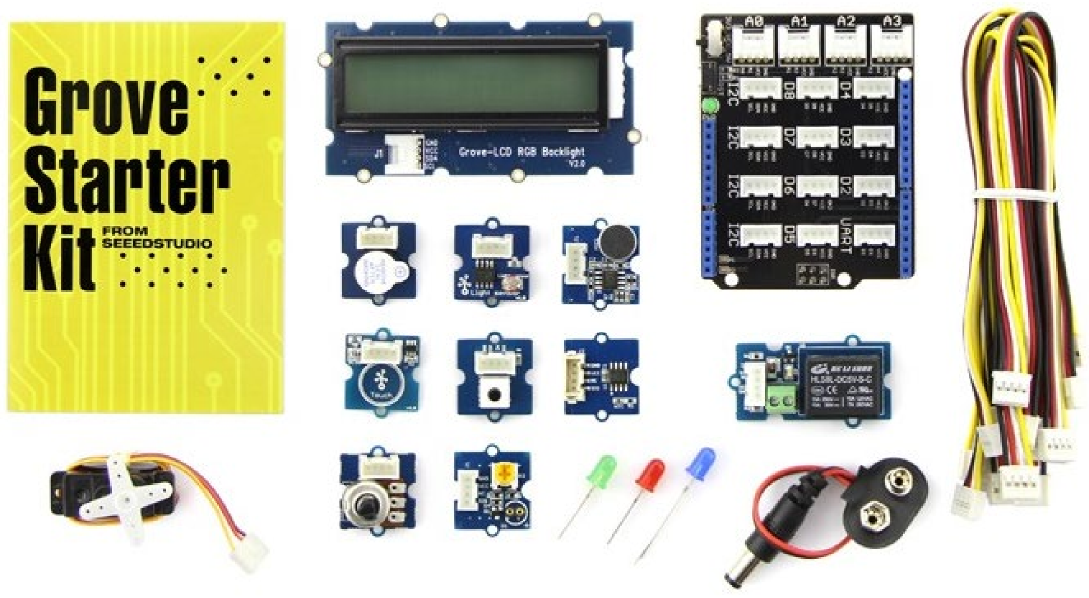
Working with Virtual Shields is the simplest method to quickly create projects in Visual Designer. You do however still have to take care with what hardware you choose to use. For example:

- You need to make sure that your use of Arduino shields is compatible with hardware. For example, some shields are incompatible because they make use of the same resource on the CPU (e.g. Timer1). Others are pin incompatible.
- You need to make sure that your target Arduino base board has enough memory to handle the shields you are adding to it. In particular, with an Arduino Uno you can run out of memory if you add shields with large and complex software stacks.

Grove Sensors and Modules

Grove is a modular electronic platform for quick prototyping. Every module has one function, such as touch sensing, creating audio effect and so on. You simply plug the modules you need into the base shield and then connect the base shield to the Arduino base board in the normal way.

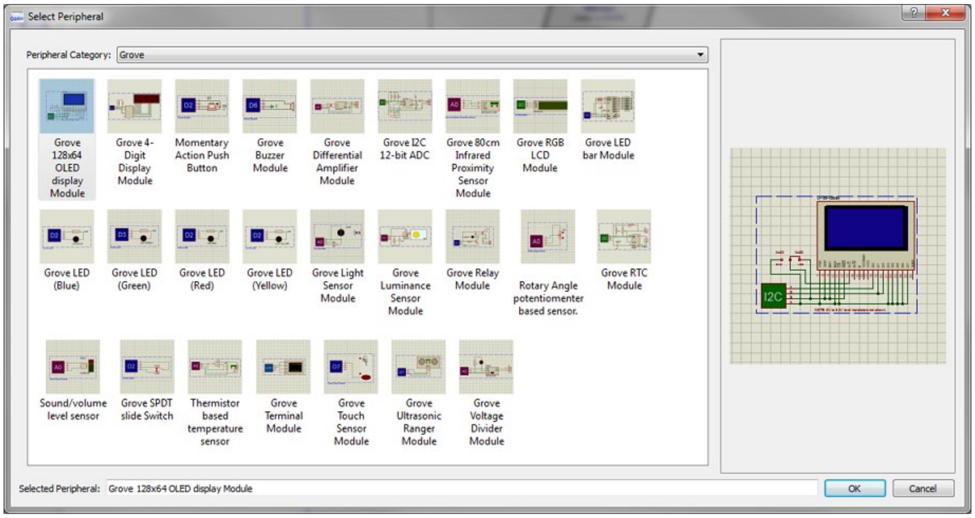
The Grove Starter Kit Plus is a great way for beginners and students to get started with Arduino. The Grove Base Shield drops onto your existing Arduino and lets you connect up to 16 Grove modules without any fuss. The Arduino headers are still broken out on the top of the shield making it very easy to connect other breakout boards and sensors to your Arduino.


Grove Starter Kit FROM SEEEDSTUDIO
Grove-LCD RGB Backlight V2.0
A0
A1
A2
A3
I2C
D8
D7
D6
D5
D4
D3
D2
UART

Grove Starter Kit Hardware.

Adding a Grove Module

You can add a Grove module in the same way as adding a shield, except that you select the Grove category in the Peripheral browser.


Select Peripheral
Grove 128x64 OLED display Module
Grove 4-Digit Display Module
Momentary Action Push Button
Grove Buzzer Module
Grove Differential Amplifier Module
Grove I2C 12-bit ADC
Grove 80cm Infrared Proximity Sensor Module
Grove RGB LCD Module
Grove LED bar Module
Grove LED (Blue)
Grove LED (Green)
Grove LED (Red)
Grove LED (Yellow)
Grove Light Sensor Module
Grove Luminance Sensor Module
Grove Relay Module
Rotary Angle potentiometer based sensor.
Grove RTC Module
Sound/volume level sensor
Grove SPDT slide Switch
Thermistor based temperature sensor
Grove Terminal Module
Grove Touch Sensor Module
Grove Ultrasonic Ranger Module
Grove Voltage Divider Module
I2C
OK
Cancel

Peripheral Category: Grove

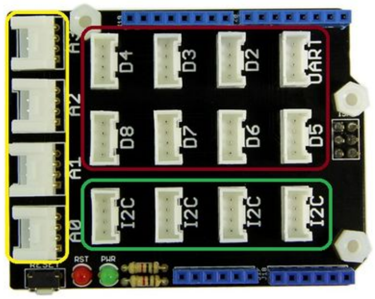
Selected Peripheral: Grove 128x64 OLED display Module

Adding Grove Starter Kit modules in Visual Designer.

You can then control the Grove peripherals via the high level methods available from the Project Tree. Multiple peripherals and sensors can easily be added to a single project which provides an extra level of flexibility over using standard Arduino shields.

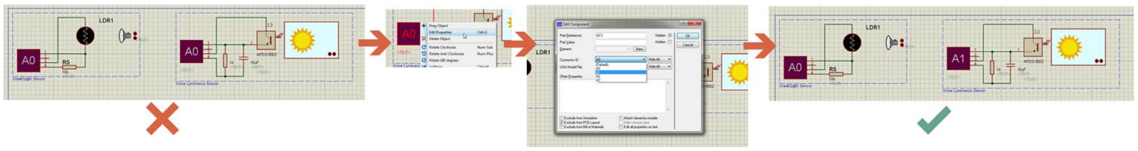
Potential Pitfalls

When you are working with Grove modules in Visual Designer the process is identical to working with complete Arduino shields. However, since the modules themselves plug into an Arduino shield, you need to take care not to re-use a connector on the Grove board. The real Grove base-board shield will look something like the following:



Grove Shield - the sensor modules plug-in to the exposed header pins.

When you add a Grove peripheral it will be assigned an ID on the schematic corresponding to a connector on the real board. If you get a duplicate, you'll need to edit and rename one of the connectors to a free slot.

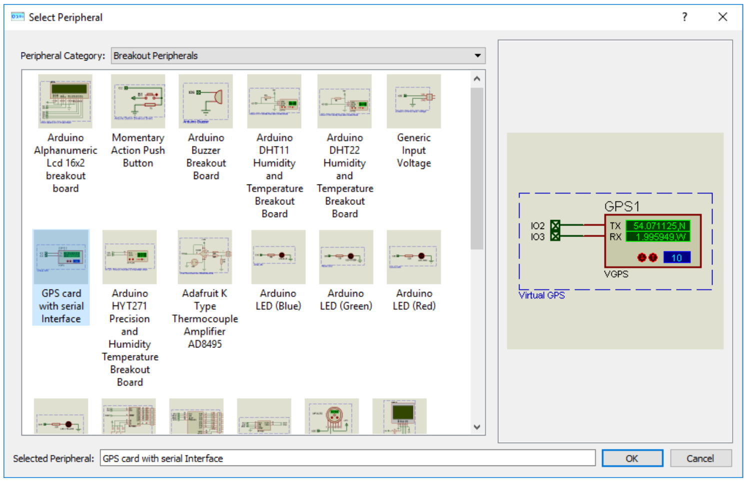


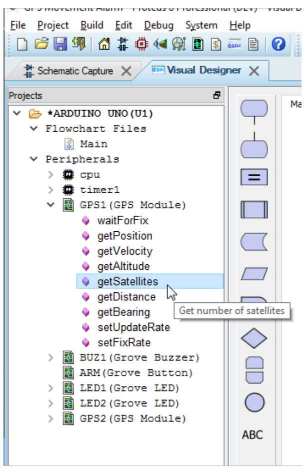
If two grove modules have the same connector, you need to edit one and assign to an unused header slot on the Grove Shield.

⚠ You need to make sure that your use of Grove modules is compatible with hardware. Since the pins used in each socket overlap it is quite easy to mistakenly use the same pin twice. A good example of this is discussed in Tutorial 2

Breakout Boards and Peripherals

The third category of hardware that you will find in the Peripheral Gallery is breakout boards. These are useful and interesting pieces of electronics (often single components) for which we have provided control methods inside Visual Designer. So, when you add a breakout peripheral to your project, you can drive it from the methods it adds to the Project Tree.


Select Peripheral
Peripheral Category: Breakout Peripherals
Arduino Alphanumeric Lcd 16x2 breakout board
Momentary Action Push Button
Arduino Buzzer Breakout Board
Arduino DHT11 Humidity and Temperature Breakout Board
Arduino DHT22 Humidity and Temperature Breakout Board
Generic Input Voltage
GPS card with serial Interface
Arduino HYT271 Precision and Humidity Temperature Breakout Board
Adafruit K Type Thermocouple Amplifier AD8495
Arduino LED (Blue)
Arduino LED (Green)
Arduino LED (Red)
GPS1
IO2
IO3
TX
RX
54.071125 N
1.995949 W
10
VGPS
Virtual GPS
Selected Peripheral: GPS card with serial Interface
OK
Cancel



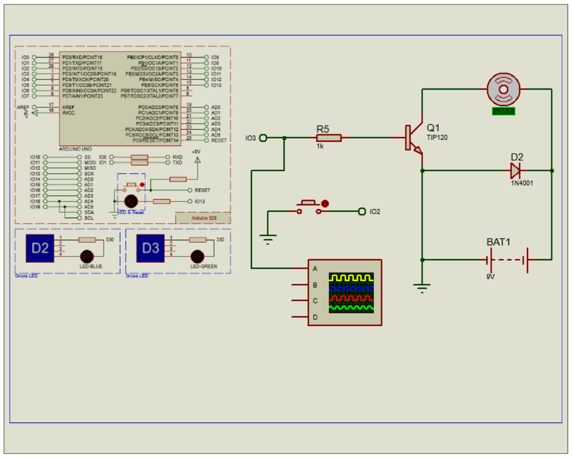
Breakout boards provide even more flexibility on the schematic. Indeed, they often connect to other schematic components that you want to place and wire yourself.

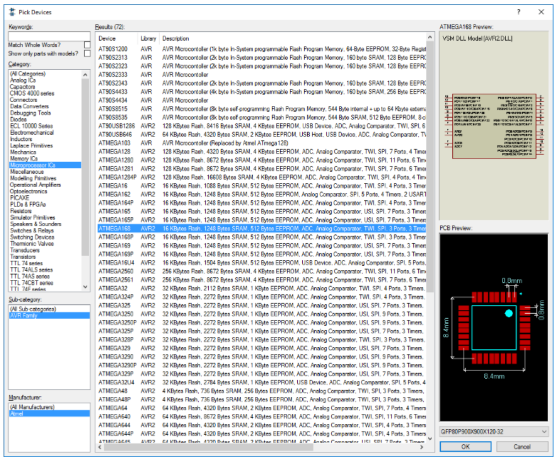
Potential Pitfalls

CPU resources and pin assignments need careful consideration.

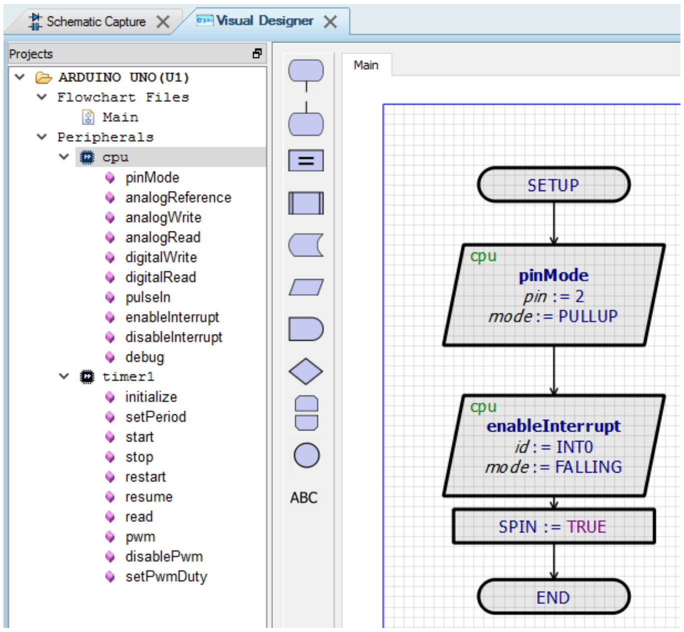
Schematic Design

With Visual Designer you have access to the full power of the Proteus VSM simulation environment. This means that you have a professional schematic capture tool with tens of thousands of embedded components that you can place, wire and simulate.





This is the most flexible design method of all but involves a significantly higher learning curve. The programming in Visual Designer will have to be done via the CPU methods rather than specific methods for peripherals.



Potential Pitfalls

Schematic design for simulation is a large and relatively complex topic. In addition to the basic of placement and connectivity there will also be power considerations and you will need to program for interconnect protocols where appropriate. At this stage, the transition from Visual Designer to Proteus VSM should be considered as there is a point beyond which C programming is required to effectively design the embedded system.

RASPBERRY PI HARDWARE

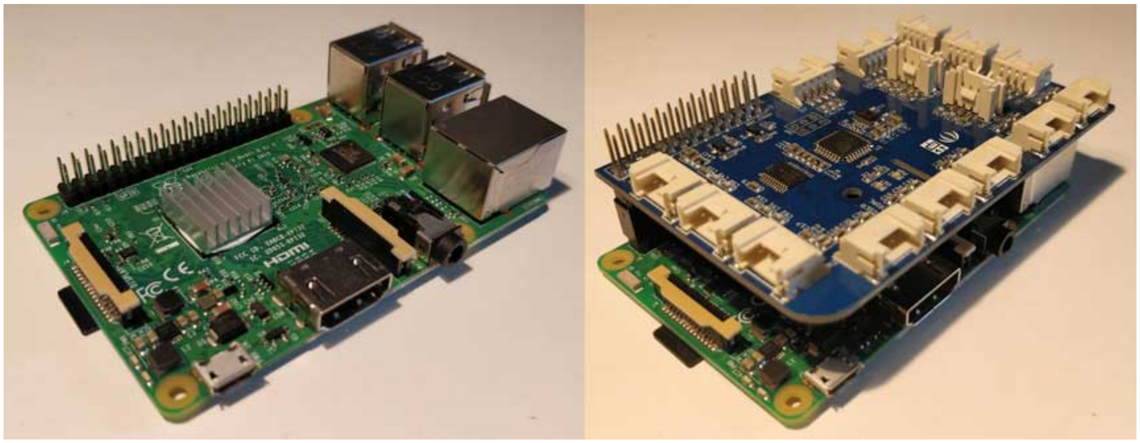
Introduction

Designing 'virtual hardware' for your project is greatly simplified in Visual Designer. The various options available to you are described in detail below.

Raspberry Pi Base Board

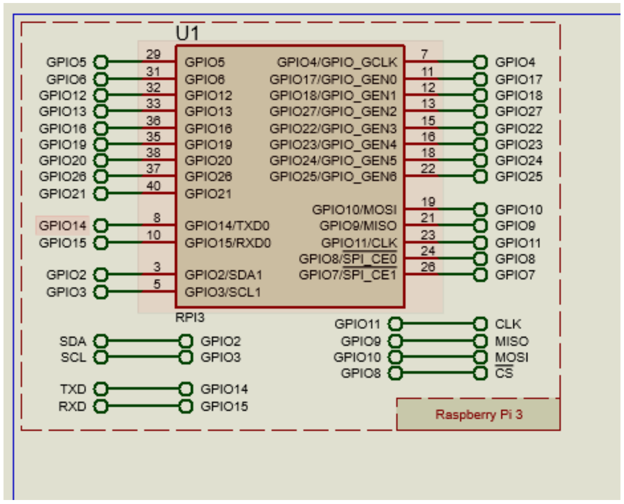
In hardware, Raspberry Pi projects start with the Pi computer board and then sometimes will have additional electronics attached via the header block. These can be either complete functional electronics blocks (known in the Pi world as 'hats') or loose electronics components such as LED's or buttons.

In Visual Designer for Raspberry Pi the creation of the project is the point at which you make your decision on the base board. You can choose between the Raspberry Pi 3 or the Raspberry Pi 3 with Grove. The latter can be thought of as the Pi3 with the Grove hat stacked on top of it



Equivalent Hardware: Raspberry Pi on the Left and Raspberry Pi with Grove Hat on the right

Your choice will be automatically placed on the schematic when the project is created.



Automatic placement of the Raspberry Pi 3 on the Schematic when the Project is created.

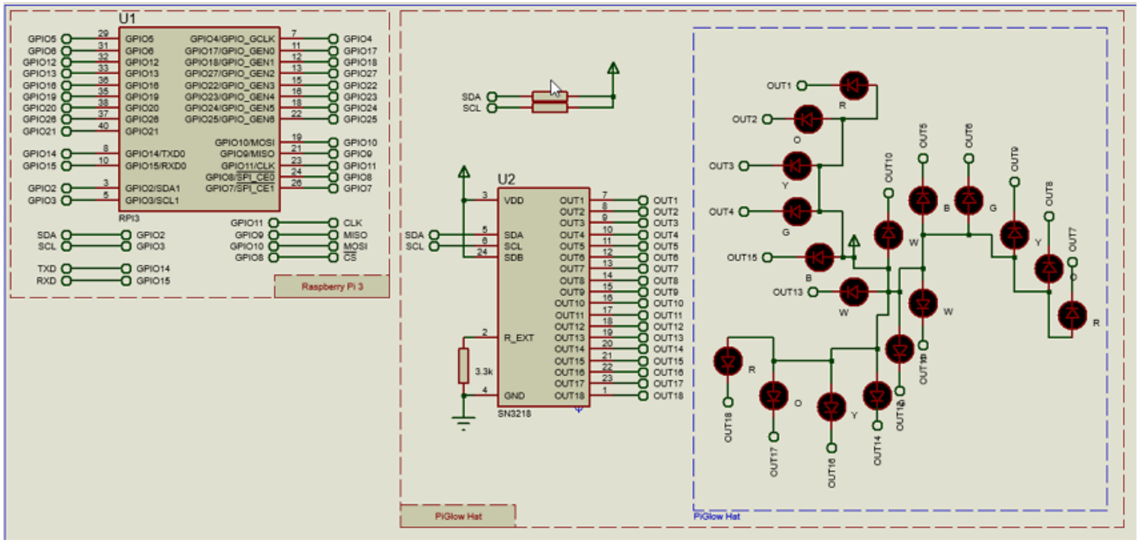
In theory, the Raspberry Pi Zero W should also work as the pinout is identical and it contains the wi-fi chip. If you are working with this device select Pi 3 in the configurator. Raspberry Pi 2 will not work with Visual Designer because the pinout is completely different.

Raspberry Pi Hats

Raspberry Pi Hats are modular circuit boards that sit on your Pi3 base-board to instill it with extra functionality. They are the simplest way to add hardware functionality to your project because the circuitry and connectivity on the schematic happens automatically.

Adding a Hat

When you then want to add a hat in Visual Designer you right click on the Project Tree and select 'Add Peripheral' from the resulting context menu. Select either the 'Adafruit' or the Pimoroni category on the peripheral browser and then select the hat you want to add. This will auto-place the electronics on the schematic and connect it to the Pi baseboard.

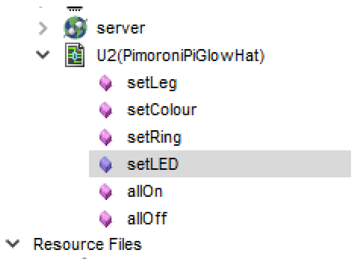


Adding a Peripheral will auto-place *<u>and</u>* *auto-connect on the schematic.*

The connections are made by terminal name. Two terminals with the same name can be thought of as being connected with an invisible wire.

Writing a Program to Control a Hat

The Raspberry Pi Hat will appear in the project tree in Visual Designer once picked from the Peripheral Gallery. They include high level methods that allow you to quickly control the connected electronics via flowchart blocks.



Drag and Drop methods make it simple to drive the hats.

Potential Pitfalls

Working with Virtual Shields is the simplest method to quickly create projects in Visual Designer. You do however still have to take care with what hardware you choose to use. For example:

- You need to make sure that your use of hats is compatible with hardware. For example, some hats are incompatible because they make use of the same resource on the CPU (e.g. Timer1). Others are pin incompatible.

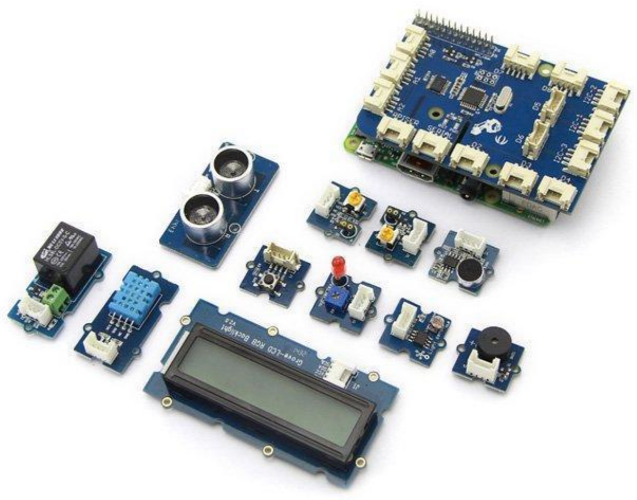
Grove Sensors and Modules

Grove is a modular electronic platform for quick prototyping. Every module has one function, such as touch sensing, creating audio effect and so on. You simply plug the modules you need into the base shield and then connect the base shield to the Raspberry Pi base board in the normal way.

The GrovePi+ is the Pi Hat which attaches to the Pi 3 baseboard and includes 15 Grove 4-pin interfaces that then let you directly connect Grove sensors to the Raspberry Pi.

In Visual Designer if you want to want to work with Grove and Raspberry Pi you should choose the Raspberry Pi 3 with Grove option during project creation.

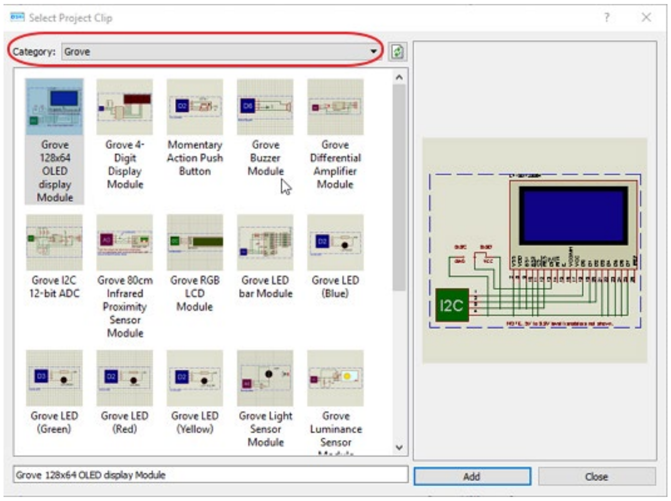
Grove Pi+ Starter Kit is a great way for beginners and students to get started with Raspberry Pi. The Grove Base Shield drops onto your existing Pi and lets you connect up to 16 Grove modules without any fuss. The headers are still broken out on the top of the shield making it very easy to connect other breakout boards and sensors to your Pi.



Grove Pi+ Starter Kit Hardware.

Adding a Grove Module

You can add a Grove module in the same way as adding a shield, except that you select the Grove category in the Peripheral browser.

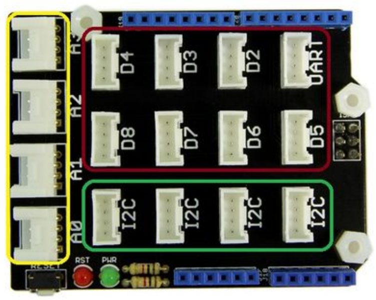

Select Project Clip
Category: Grove
Grove 128x64 OLED display Module
Grove 4-Digit Display Module
Momentary Action Push Button
Grove Buzzer Module
Grove Differential Amplifier Module
Grove I2C 12-bit ADC
Grove 80cm Infrared Proximity Sensor Module
Grove RGB LCD Module
Grove LED bar Module
Grove LED (Blue)
Grove LED (Green)
Grove LED (Red)
Grove LED (Yellow)
Grove Light Sensor Module
Grove Luminance Sensor
I2C
Grove 128x64 OLED display Module
Add
Close

Adding Grove Starter Kit modules in Visual Designer.

You can then control the Grove peripherals via the high level methods available from the Project Tree. Multiple peripherals and sensors can easily be added to a single project which provides an extra level of flexibility over using standard Raspberry Pi shields.

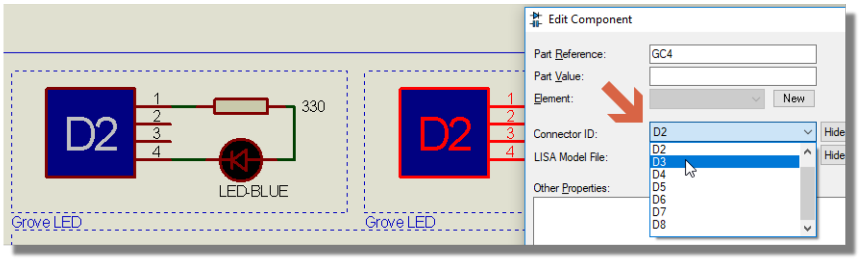
Potential Pitfalls

When you are working with Grove modules in Visual Designer the process is identical to working with complete hats. However, since the modules themselves plug into a hat, you need to take care not to re-use a connector on the Grove board. The real Grove base-board shield will look something like the following:



Grove Shield - the sensor modules plug-in to the exposed header pins.

When you add a Grove peripheral it will be assigned an ID on the schematic corresponding to a connector on the real board. If you get a duplicate, you'll need to edit and rename one of the connectors to a free slot.



Part Reference: GC4
Part Value:
Element:
Connector ID: D2
LISA Model File:
Other Properties:

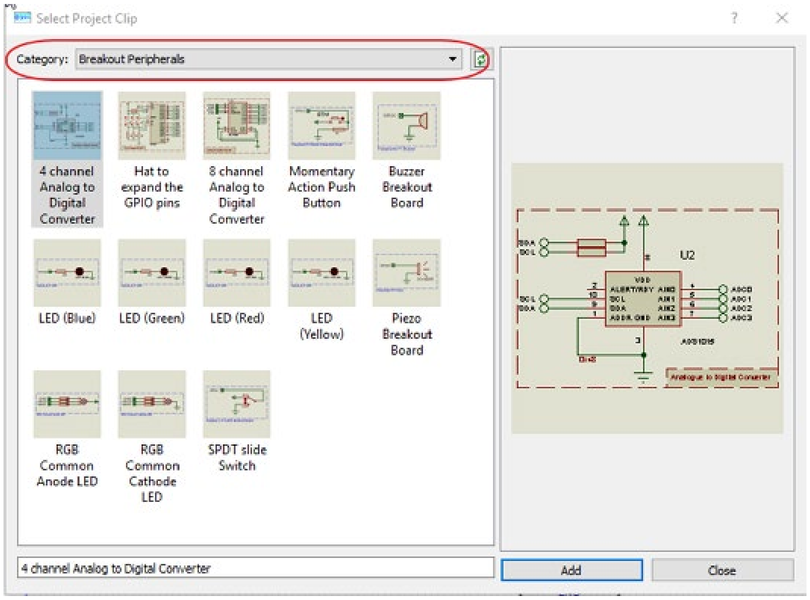
D2, D3, D4, D5, D6, D7, D8

If two grove modules have the same connector, you need to edit one and assign to an unused header slot on the Grove Shield.

⚠ You need to make sure that your use of Grove modules is compatible with hardware. Since the pins used in each socket overlap it is quite easy to mistakenly use the same pin twice.

Breakout Boards and Peripherals

The third category of hardware that you will find in the Peripheral Gallery is breakout boards. These are useful and interesting pieces of electronics (often single components) for which we have provided control methods inside Visual Designer. So, when you add a breakout peripheral to your project, you can drive it from the methods it adds to the Project Tree.



Category: Breakout Peripherals

4 channel Analog to Digital Converter

Breakout boards provide even more flexibility on the schematic. Indeed, they often connect to other schematic components that you want to place and wire yourself.

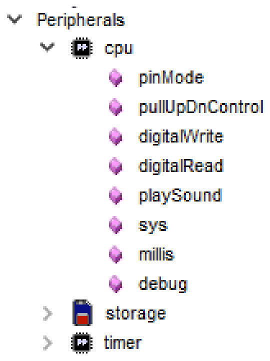
Potential Pitfalls

CPU resources and pin assignments need careful consideration.

Schematic Design

With Visual Designer you have access to the full power of the Proteus VSM simulation environment. This means that you have a professional schematic capture tool with tens of thousands of embedded components that you can place, wire and simulate.

This is the most flexible design method of all but involves a significantly higher learning curve. The programming in Visual Designer will have to be done via the CPU methods rather than specific methods for peripherals.



Potential Pitfalls

Schematic design for simulation is a large and relatively complex topic. In addition to the basic of placement and connectivity there will also be power considerations and you will need to program for interconnect protocols where appropriate. It should also be noted that the Raspberry Pi simulation model in Proteus is a high level behavioural model. Since the real rPi runs a full operating system it is quite easy to run Python code that is not supported in simulation.

DEBUGGING SKILLS

Overview

When developing your embedded system you may want to test your program as it evolves and you will certainly want to test it when it is complete. It is a rare occasion when everything works perfectly first time and you'll therefore also need to debug your program in order to find and fix errors.

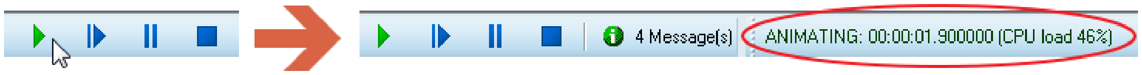
Visual Designer makes this process extremely simple because it all happens in software. The following topics summarize the essential skills needed to simulate, debug and measure with Visual Designer. It's assumed at this stage that you are familiar with all the required editing skills and have a project ready for testing.

System Level Simulation

When you add peripherals and control them in your flowchart program you are building an embedded system. Loosely speaking, there will be the microcontroller board and some hardware pieces representing the peripherals. Running a system level simulation is the real power of Proteus because you are simulating the microcontroller together with all of the analog and digital electronics connected to it. This allows you to interact with, test and debug your project entirely inside the Proteus software suite.

To start a simulation

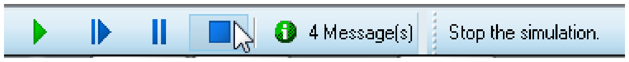
To begin simulating either press the F12 shortcut key or the play button on the animation control panel. Your program will compile and simulation progress will be reported on the status bar.



When you start the simulation the time elapsed is shown on the status bar.

To stop a simulation

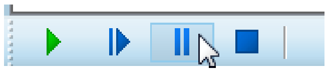
To end a simulation either press the ESCAPE shortcut key or the stop button on the animation control panel. The program will stop and Visual Designer will return to the Editing Layout



Stop the simulation via the Control Panel.

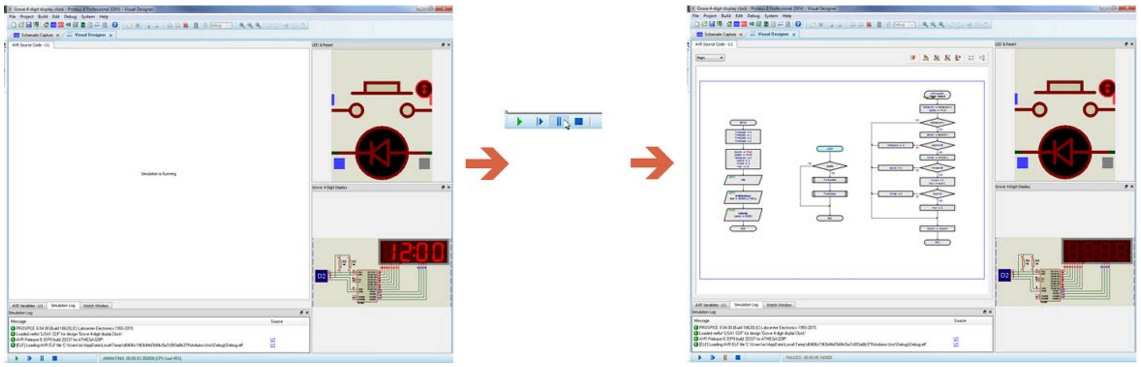
To pause a simulation

Pausing the Proteus system level simulation is an important concept; you are effectively stopping time. For example, your capacitors wont discharge and your motors will maintain their angular position and their momentum. This allows you to inspect both your program and your virtual hardware and when you single step debug the entire system will advance in time synchronously. Your embedded system works to the clock on the status bar, not the clock on the wall and you are in charge of the clock ! It's enormously powerful and we'll see many uses in both these topics and more practically in the tutorials.
 To pause a running simulation either press the PAUSE button on the keyboard or click the pause button on the animation control panel.



Pausing a Simulation

A simulation will also be automatically paused at a breakpoint. Breakpoints are discussed in the Debugging Skills topic below and demonstrated in the various tutorial topics.
Whenever the simulation is paused, Visual Designer will switch to full debugging layout and both the flowchart and the various debug windows will become available for inspection and interaction.



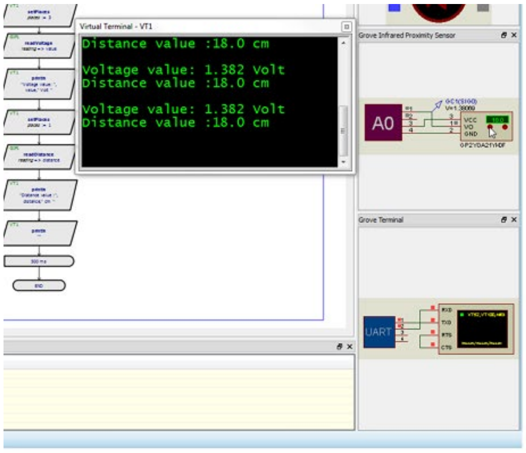
Pausing the Simulation prepares Visual Designer for debugging.

You don't see the flowchart during free running simulation because the program is executing so quickly that actually showing it's current position would be impossible.

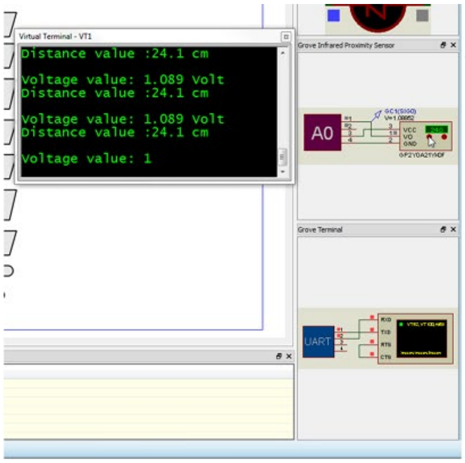
To interact with a simulation

When the simulation is running you will see Visual Designer in its debugging layout, with all the interactive elements aligned on the right hand side of the application. These will almost certainly be a mix of indicators (led's, LCD's, etc.) that provide you with feedback and actuators (buttons, switches, keyboards, etc.) that you click to provide stimuli to the simulation. Here's an example where we adjust an infra-red proximity sensor:

- Start the Simulation
- Move the mouse over the adjustment arrows on the sensor.
- Click up or down as required.



```
Distance value :18.0 cm

Voltage value: 1.382 Volt
Distance value :18.0 cm

Voltage value: 1.382 Volt
Distance value :18.0 cm
```



```
Distance value :24.1 cm

Voltage value: 1.089 Volt
Distance value :24.1 cm

Voltage value: 1.089 Volt
Distance value :24.1 cm

Voltage value: 1
```

Clicking on the IR sensor adjustment controls mimics the behaviour of a physical test.

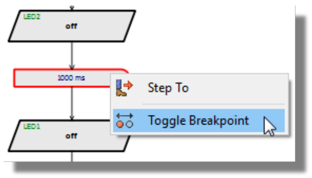
You can just as easily switch to the schematic tab and interact on that screen if you prefer. However, it's handy to have everything in one place and particularly so during a debugging session when you want to view hardware while single stepping your flowchart.

The virtual terminal is slightly different as you can simply click and type when the simulation is running. This means you can use the terminal as a command interface to your program which has many, many uses.

Debugging Skills

To set a software breakpoint

Right click on the flowchart block either before simulation or when the simulation is paused and select the toggle breakpoint command from the context menu.



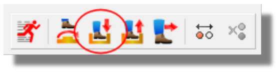
To single step debug your flowchart

When the simulation is paused, use the step commands at the top of the window or the F10 and F11 shortcut keys to single step through your flowcode.

To watch the program flow it is often helpful to try a simulation run by pressing pause and then the animated single step command from the Debug Menu. This is slow enough that you can see the path your program executes through decision blocks and is often a quick way to spot a problem.

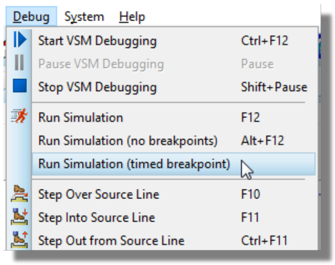
To step to a block on the flowchart

To step to a specific block on the flowchart, first pause the simulation, then right click on the block and select the Step to command from the context menu.



To set a timed breakpoint

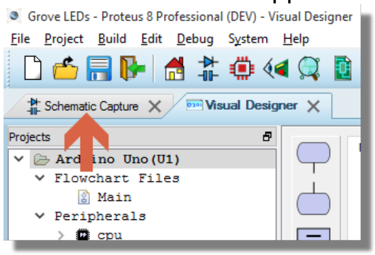
It's occasionally useful to pause a simulation after a period of time rather than on a particular flowblock. You can do this by selecting the Run Simulation with Timed Breakpoint command from the Debug Menu.


Debug
System
Help
Start VSM Debugging Ctrl+F12
Pause VSM Debugging Pause
Stop VSM Debugging Shift+Pause
Run Simulation F12
Run Simulation (no breakpoints) Alt+F12
Run Simulation (timed breakpoint)
Step Over Source Line F10
Step Into Source Line F11
Step Out from Source Line Ctrl+F11

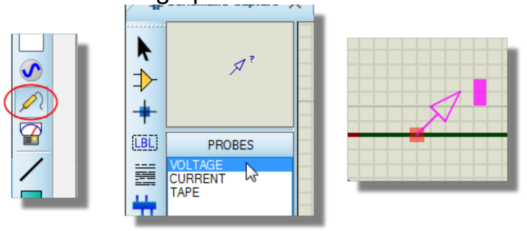
To set a hardware breakpoint

You can also set a breakpoint on a hardware condition as opposed to a software condition. The procedure is:

1) With the simulation stopped switch to the schematic capture tab.



2) Place a voltage probe on the wire of interest.



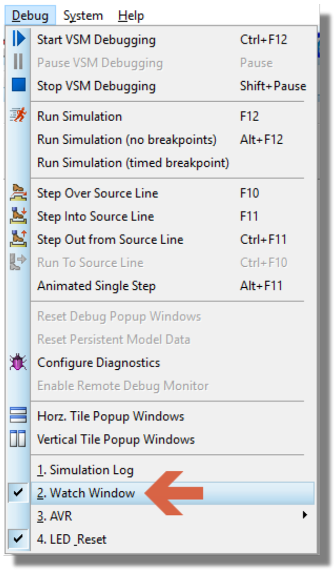
Terminals with the same name are considered connected by an invisible wire. It is a simple way to keep shields and breakouts self-contained and not have wires criss-crossing all over the screen.

3) Edit the probe and configure the Real Time Breakpoint settings.
4) Run the simulation

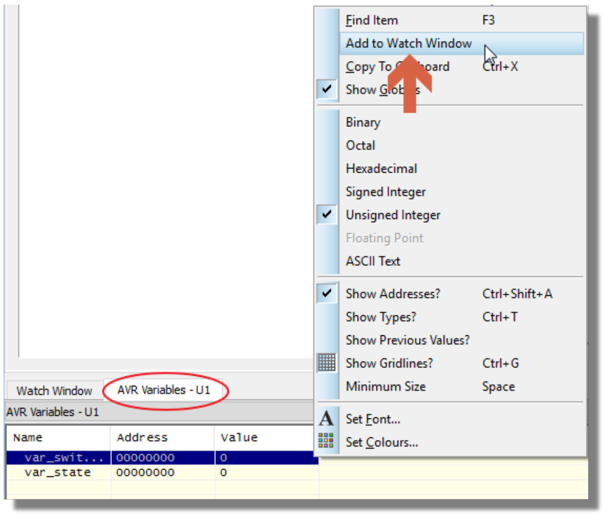
To add a variable to the watch window

The watch window is the only debugging window available during a free running simulation. You may therefore choose to add a program variable to the watch window in order to monitor the value as the program executes. To do this:

1) With the simulation paused, open the watch window from the Debug Menu. By default, the window will dock beside the variables window.

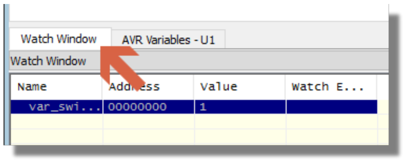

Debug
System
Help
Start VSM Debugging Ctrl+F12
Pause VSM Debugging Pause
Stop VSM Debugging Shift+Pause
Run Simulation F12
Run Simulation (no breakpoints) Alt+F12
Run Simulation (timed breakpoint)
Step Over Source Line F10
Step Into Source Line F11
Step Out from Source Line Ctrl+F11
Run To Source Line Ctrl+F10
Animated Single Step Alt+F11
Reset Debug Popup Windows
Reset Persistent Model Data
Configure Diagnostics
Enable Remote Debug Monitor
Horz. Tile Popup Windows
Vertical Tile Popup Windows
1. Simulation Log
2. Watch Window
3. AVR
4. LED_Reset

2) Switch to the variables window, right click on the variable of interest and select the 'Add to Watch Window' command.



Name	Address	Value
var_swit...	00000000	0
var_state	00000000	0

3) Switch back to the Watch Window and run the simulation.



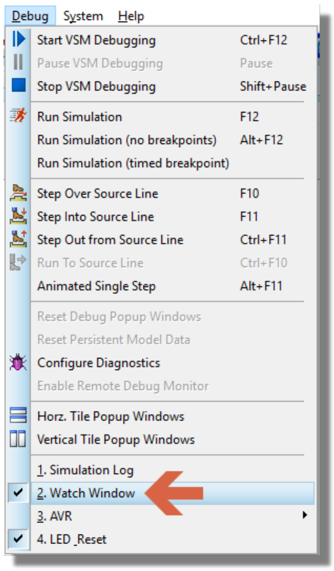
Name	Address	Value	Watch E...
var_swi...	00000000	1	

⚠ Variables are identified in the Watch Window by address. If you change your program between simulation runs there is a chance that the compiler will assign a different address to the variable and you will not be monitoring the variable on subsequent simulation runs. This can be extremely confusing and it's best to remove and re-add the variable when debugging after programming changes.

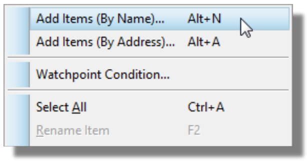
To add other items to the watch window

The watch window is simply a monitor on memory so you can add AVR registers or other program locations. For example, to add a register:

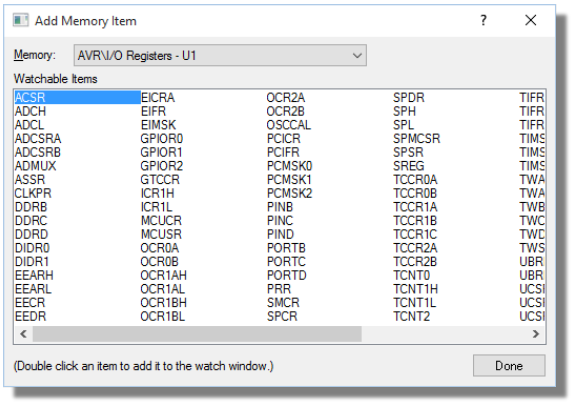
1) With the simulation paused make sure the Watch Window is open from the Debug Menu.


Debug
System
Help
Start VSM Debugging Ctrl+F12
Pause VSM Debugging Pause
Stop VSM Debugging Shift+Pause
Run Simulation F12
Run Simulation (no breakpoints) Alt+F12
Run Simulation (timed breakpoint)
Step Over Source Line F10
Step Into Source Line F11
Step Out from Source Line Ctrl+F11
Run To Source Line Ctrl+F10
Animated Single Step Alt+F11
Reset Debug Popup Windows
Reset Persistent Model Data
Configure Diagnostics
Enable Remote Debug Monitor
Horz. Tile Popup Windows
Vertical Tile Popup Windows
1. Simulation Log
2. Watch Window
3. AVR
4. LED _Reset

2) Right click on the Watch Window and select add item by name.



3) Select the register you want to watch and add it.


Add Memory Item

Memory: AVR\I/O Registers - U1

Watchable Items

ACSR	EICRA	OCR2A	SPDR	TIFR
ADCH	EIFR	OCR2B	SPH	TIFR
ADCL	EIMSK	OSCCAL	SPL	TIFR
ADCSRA	GPIOR0	PCICR	SPMCSR	TIMS
ADCSRB	GPIOR1	PCIFR	SPSR	TIMS
ADMUX	GPIOR2	PCMSK0	SREG	TIMS
ASSR	GTCCR	PCMSK1	TCCR0A	TWA
CLKPR	ICR1H	PCMSK2	TCCR0B	TWA
DDRB	ICR1L	PINB	TCCR1A	TWB
DDRC	MCUCR	PINC	TCCR1B	TWC
DDRD	MCUSR	PIND	TCCR1C	TWD
DIDR0	OCR0A	PORTB	TCCR2A	TWS
DIDR1	OCR0B	PORTC	TCCR2B	UBR
EEARH	OCR1AH	PORTD	TCNT0	UBR
EEARL	OCR1AL	PRR	TCNT1H	UCSI
EECR	OCR1BH	SMCR	TCNT1L	UCSI
EEDR	OCR1BL	SPCR	TCNT2	UCSI

(Double click an item to add it to the watch window.)

Done

The process is very similar when adding memory locations by address.

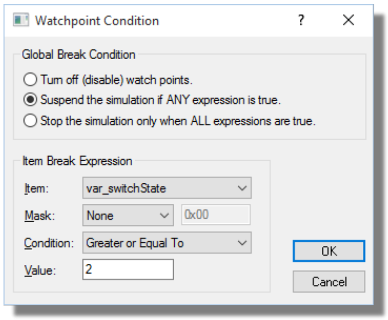
To set a conditional breakpoint

A conditional breakpoint (a watchpoint) is where you set a breakpoint to occur when a preset condition is met on an item in the watch window. It is often under used as a debug tool - for example, it is the perfect way to pause the simulation every time a timer overflows. To set a watchpoint condition:

1) Use the methods described above to add either a variable or a register to the watch window.

2) Right click on the item in the Watch Window and select Watchpoint Condition from the context menu.

3) Configure as required.



Global Break Condition

- ○ Turn off (disable) watch points.
- ● Suspend the simulation if ANY expression is true.
- ○ Stop the simulation only when ALL expressions are true.

Item Break Expression

Item:	var_switchState
Mask:	None 0x00
Condition:	Greater or Equal To
Value:	2

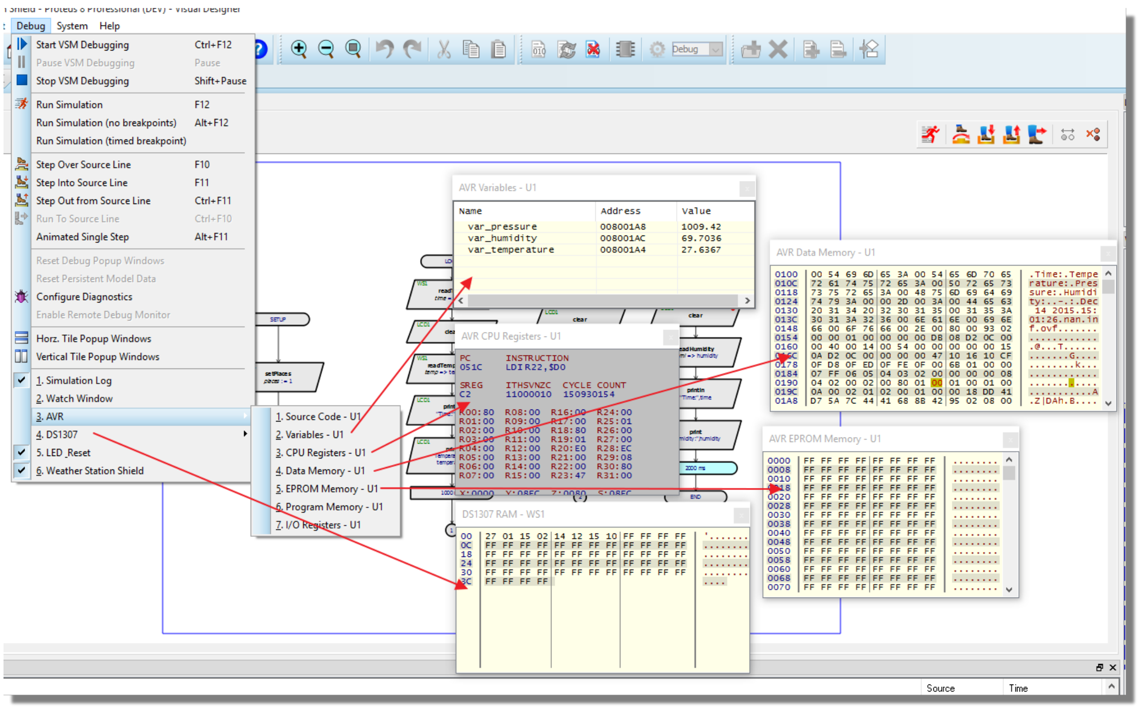
4) Run the simulation.

There is an option on the dialogue to specify whether the watchpoint is a Global Break Condition. This just determines whether the simulation suspends when any item expression is true, or only when all watchpoints are true.

To open a debug window

In addition to the source, variables and watch windows there are a host of additional debug windows that we encourage users to explore. After all, the more familiar you are with the available tools the better at finding problems you will become and the debugging toolset in Proteus is unparalleled. Often, you can even open debug windows on the shield peripheral models as well as the processor model. The process is:

1) Pause the simulation.
2) Launch the debug window you want from the Debug Menu.


Debug
System
Help
Start VSM Debugging Ctrl+F12
Pause VSM Debugging Pause
Stop VSM Debugging Shift+Pause
Run Simulation F12
Run Simulation (no breakpoints) Alt+F12
Run Simulation (timed breakpoint)
Step Over Source Line F10
Step Into Source Line F11
Step Out from Source Line Ctrl+F11
Run To Source Line Ctrl+F10
Animated Single Step Alt+F11
Reset Debug Popup Windows
Reset Persistent Model Data
Configure Diagnostics
Enable Remote Debug Monitor
Horz. Tile Popup Windows
Vertical Tile Popup Windows
1. Simulation Log
2. Watch Window
3. AVR
4. DS1307
5. LED_Reset
6. Weather Station Shield
1. Source Code - U1
2. Variables - U1
3. CPU Registers - U1
4. Data Memory - U1
5. EPROM Memory - U1
6. Program Memory - U1
7. I/O Registers - U1
Source
Time

AVR Variables - U1

Name	Address	Value
var_pressure	008001A8	1009.42
var_humidity	008001AC	69.7036
var_temperature	008001A4	27.6367

AVR CPU Registers - U1

```
PC        INSTRUCTION
051C      LDI R22,$D0

SREG      ITHSVNZC   CYCLE COUNT
C2        11000010   150930154

R00:80   R08:00   R16:00   R24:00
R01:00   R09:00   R17:00   R25:01
R02:00   R10:00   R18:80   R26:00
R03:00   R11:00   R19:01   R27:00
R04:00   R12:00   R20:E0   R28:EC
R05:00   R13:00   R21:00   R29:08
R06:00   R14:00   R22:00   R30:80
R07:00   R15:00   R23:47   R31:00

X:0000   Y:08EC   Z:0080   S:08EC
```

AVR Data Memory - U1

```
0100 | 00 54 69 6D | 65 3A 00 54 | 65 6D 70 65 | .Time:.Tempe
010C | 72 61 74 75 | 72 65 3A 00 | 50 72 65 73 | rature:.Pres
0118 | 73 75 72 65 | 3A 00 48 75 | 6D 69 64 69 | sure:.Humidi
0124 | 74 79 3A 00 | 00 2D 00 3A | 00 44 65 63 | ty:..-.:.Dec
0130 | 20 31 34 20 | 32 30 31 35 | 00 31 35 3A |  14 2015.15:
013C | 30 31 3A 32 | 36 00 6E 61 | 6E 00 69 6E | 01:26.nan.in
0148 | 66 00 6F 76 | 66 00 2E 00 | 80 00 93 02 | f.ovf.......
0154 | 00 00 01 00 | 00 00 00 DB | 08 D2 0C 00 | ............
0160 | 00 40 00 14 | 00 54 00 00 | 00 00 00 15 | .@...T......
016C | 0A D2 0C 00 | 00 00 00 47 | 10 16 10 CF | .......G....
0178 | 0F D8 0F ED | 0F FE 0F 00 | 6B 01 00 00 | ........k...
0184 | 07 FF 06 05 | 04 03 02 00 | 00 00 00 08 | ............
0190 | 04 02 00 02 | 00 80 01 00 | 01 00 01 00 | ............
019C | 0A 00 02 01 | 02 00 01 00 | 00 18 DD 41 | ...........A
01A8 | D7 5A 7C 44 | 41 68 8B 42 | 95 02 08 00 | .Z|DAh.B....
```

AVR EPROM Memory - U1

```
0000 | FF FF FF FF | FF FF FF FF | ........
0008 | FF FF FF FF | FF FF FF FF | ........
0010 | FF FF FF FF | FF FF FF FF | ........
0018 | FF FF FF FF | FF FF FF FF | ........
0020 | FF FF FF FF | FF FF FF FF | ........
0028 | FF FF FF FF | FF FF FF FF | ........
0030 | FF FF FF FF | FF FF FF FF | ........
0038 | FF FF FF FF | FF FF FF FF | ........
0040 | FF FF FF FF | FF FF FF FF | ........
0048 | FF FF FF FF | FF FF FF FF | ........
0050 | FF FF FF FF | FF FF FF FF | ........
0058 | FF FF FF FF | FF FF FF FF | ........
0060 | FF FF FF FF | FF FF FF FF | ........
0068 | FF FF FF FF | FF FF FF FF | ........
0070 | FF FF FF FF | FF FF FF FF | ........
```

DS1307 RAM - WS1

```
00 | 27 01 15 02 | 14 12 15 10 | FF FF FF FF | '...........
0C | FF FF FF FF | FF FF FF FF | FF FF FF FF | ............
18 | FF FF FF FF | FF FF FF FF | FF FF FF FF | ............
24 | FF FF FF FF | FF FF FF FF | FF FF FF FF | ............
30 | FF FF FF FF | FF FF FF FF | FF FF FF FF | ............
3C | FF FF FF FF |                           | ....
```

If you find yourself with a very busy screen you can close debug windows and re-open them when needed from the Debug Window. Additionally, with the exception of the Watch Window, they will all disappear during free running simulation.

RESOURCE FILES & DATA STORAGE

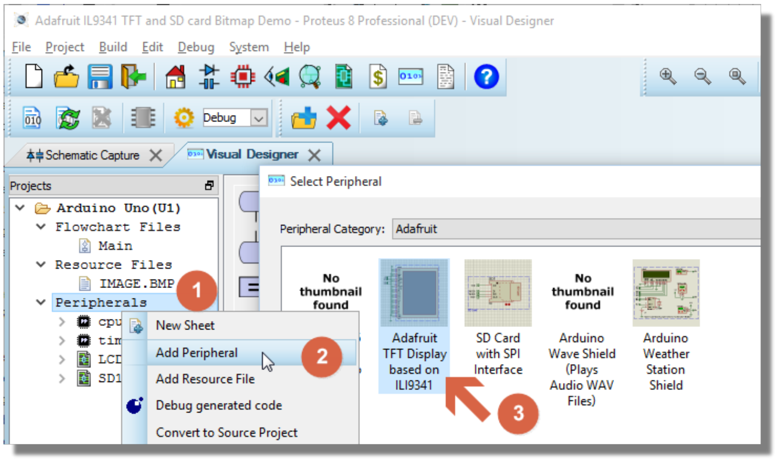
Introduction

Visual Designer includes a very powerful yet simple storage model. This takes the form of resource files which can be added to the current project directly from the Project menu or via the context menu in the Project Tree. Resource files can be anything from a text file to a picture to an audio clip.

How it Works

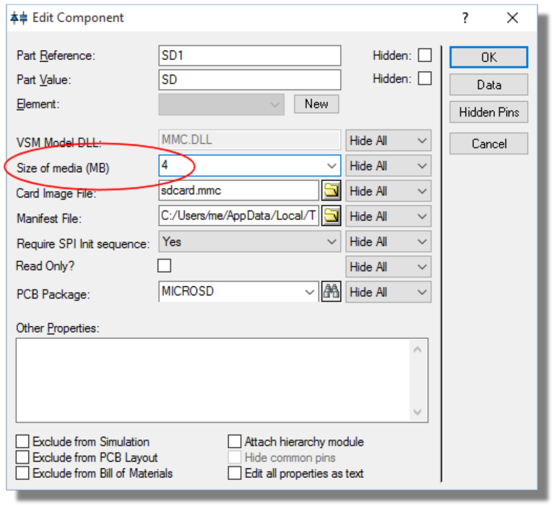
It is very common to have a shield containing both an SD Card and other electronics for using the contents of the SD Card. Obvious examples include the TFT shield which can render pictures stored on the SD card and the Wave shield which can play WAV files stored on the SD Card. Like everything else in Visual Designer the process for working with these shields is fairly straightforward:

1) Add the shield containing the resource file (e.g. Wave shield, TFT Shield)


Adafruit IL9341 TFT and SD card Bitmap Demo - Proteus 8 Professional (DEV) - Visual Designer
File Project Build Edit Debug System Help
Debug
Schematic Capture
Visual Designer
Projects
Arduino Uno (U1)
Flowchart Files
Main
Resource Files
IMAGE.BMP
Peripherals
cpu
tim
LCD
SD1
1
New Sheet
Add Peripheral
2
Add Resource File
Debug generated code
Convert to Source Project
Select Peripheral
Peripheral Category: Adafruit
No thumbnail found
Adafruit TFT Display based on ILI9341
SD Card with SPI Interface
No thumbnail found
Arduino Wave Shield (Plays Audio WAV Files)
Arduino Weather Station Shield
3

1, right click and on Peripherals, 2, Add Peripheral, 3, Select Peripheral.

2) Check that the SD Card image size is big enough for your intended purposes. Clearly, the SD Card image size for wave files needs to be larger than that for pictures.



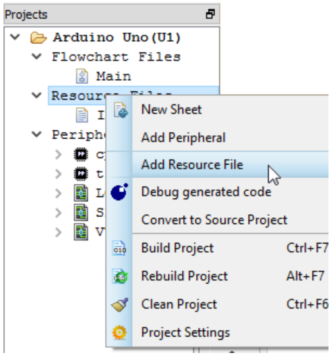
Part Reference:	SD1	Hidden: ☐	
Part Value:	SD	Hidden: ☐	
Element:		New	
VSM Model DLL:	MMC.DLL	Hide All	
Size of media (MB)	4	Hide All	
Card Image File:	sdcard.mmc	Hide All	
Manifest File:	C:/Users/me/AppData/Local/T	Hide All	
Require SPI Init sequence:	Yes	Hide All	
Read Only?	☐	Hide All	
PCB Package:	MICROSD	Hide All	

☐ Exclude from Simulation
☐ Exclude from PCB Layout
☐ Exclude from Bill of Materials
☐ Attach hierarchy module
☐ Hide common pins
☐ Edit all properties as text

Change the size to cope with what you are loading. i.e. a WAV file will be larger than an image file.

In software, Proteus works some magic here and will create a FAT image for you of the correct size. If you are programming the real hardware you'll need to make sure you have a FAT16 or FAT32 SD Card of a suitable size in the SD Card slot of the shield before you hit the program button!

3) Add the resource file(s)



⚠ Be careful with the naming of resource files. The Arduino SD stack by default supports only 8.3 filenames in order to support FAT16 cards. This means that the namestem must be 8 characters or less and the extension must be 3 characters or less. For example, a picture called ProteusLogo.bmp will not load whereas renaming it to logo.bmp or image.bmp will work fine.

4) Drag and drop the resource file into your program. Visual Designer will automatically:

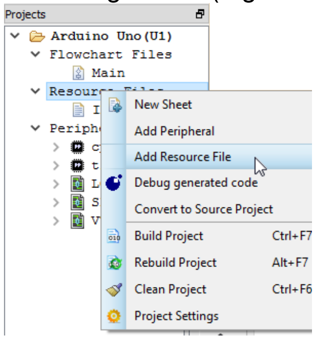
- Detect a destination IO routine for the resource (e.g. DrawBitmap() or Play())
- Set the source for the routine to be the resource.

⚠ If you add a resource file there **must** be either an SD Card in an included shield or an SD Card breakout peripheral otherwise you have no place to store the resource in hardware !

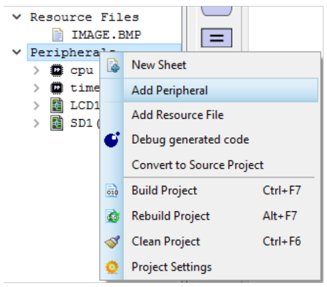
The Filestore

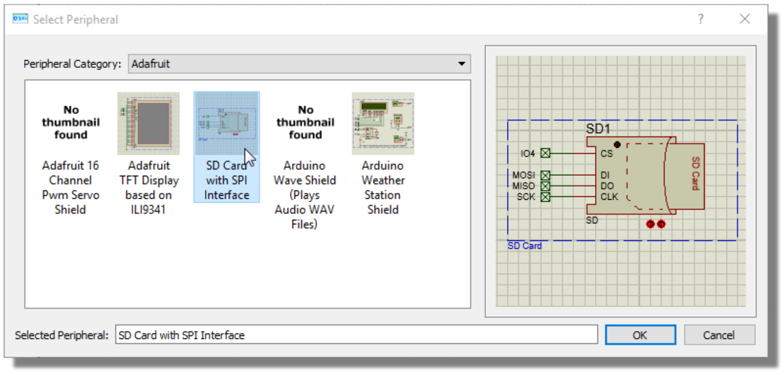
Sometimes you just need to store and manipulate data in your program. The filestore gives you this capability. To use the filestore:

1) Add the resource(s). This is the data being stored (e.g. text file).



2) Add the SD Card peripheral. This is the filestore.

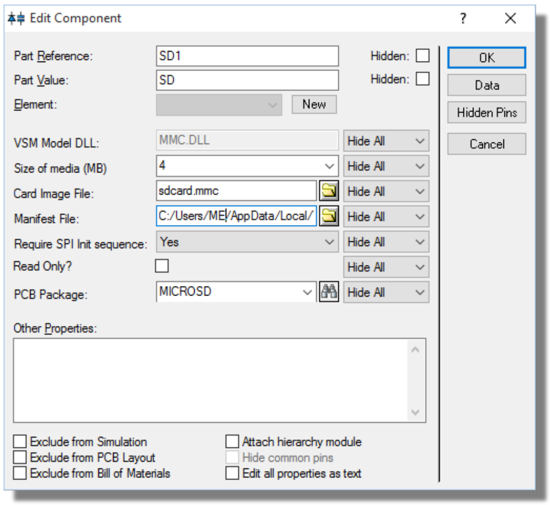



Select Peripheral
No thumbnail found
Adafruit 16 Channel Pwm Servo Shield
Adafruit TFT Display based on ILI9341
SD Card with SPI Interface
No thumbnail found
Arduino Wave Shield (Plays Audio WAV Files)
Arduino Weather Station Shield
SD1
IO4
MOSI
MISO
SCK
CS
DI
DO
CLK
SD
SD Card
OK
Cancel

Peripheral Category: Adafruit

Selected Peripheral: SD Card with SPI Interface

3) Use the methods on the filestore (storage data blocks) to open, read and otherwise manipulate the data in your program.



Part Reference: SD1 Hidden: ☐

Part Value: SD Hidden: ☐

Element:

VSM Model DLL: MMC.DLL

Size of media (MB): 4

Card Image File: sdcard.mmc

Manifest File: C:/Users/ME/AppData/Local/

Require SPI Init sequence: Yes

Read Only? ☐

PCB Package: MICROSD

Other Properties:

☐ Exclude from Simulation
☐ Exclude from PCB Layout
☐ Exclude from Bill of Materials
☐ Attach hierarchy module
☐ Hide common pins
☐ Edit all properties as text

When you program the physical hardware the programmer will handle the transfer of the resources to the SD Card (as long as there is a suitable SD card in the hardware!).

PROGRAMMING THE PHYSICAL HARDWARE

Introduction

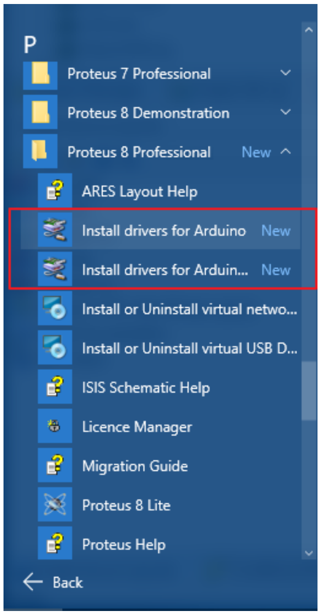
This topic discusses how to program the Arduino hardware from inside Proteus Visual Designer so that you can see your projects working in the real world.
For technical support or problems with programming please refer to either the troubleshooting guide below or to the following Arduino resources:

- https://www.arduino.cc/en/Main/FAQ
- https://forum.arduino.cc
- https://forum.arduino.cc/index.php?topic=261445.0

A Quick Guide to Programming

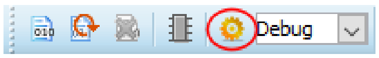
For convenience, Proteus Visual Designer includes an interface to the Arduino AVR programmer. To program your hardware from inside Proteus you will need to :

1. Make sure you have the Arduino drivers installed. If you don't you can install them from the Labcenter Program Group or by installing the Arduino IDE from their website.


P
Proteus 7 Professional
Proteus 8 Demonstration
Proteus 8 Professional New
ARES Layout Help
Install drivers for Arduino New
Install drivers for Arduin... New
Install or Uninstall virtual netwo...
Install or Uninstall virtual USB D...
ISIS Schematic Help
Licence Manager
Migration Guide
Proteus 8 Lite
Proteus Help
Back

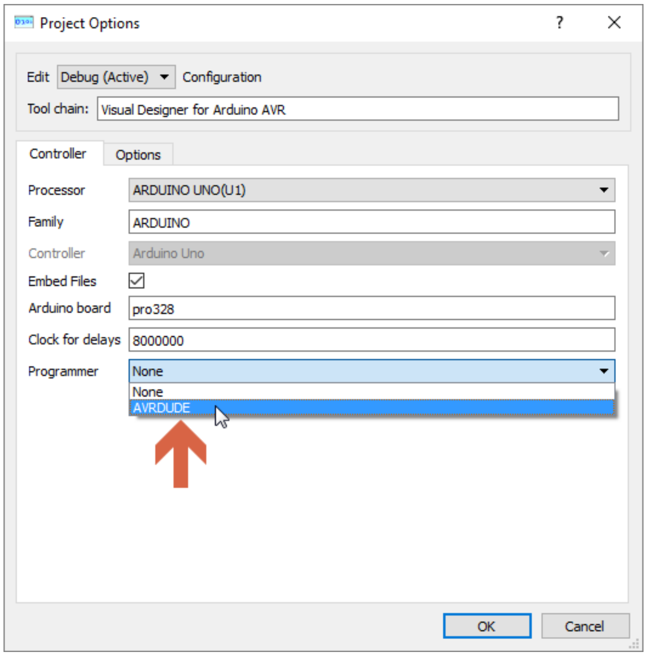
Installing Drivers from the Labcenter Program Group in the Start Menu.

2. Go to the Project Settings dialogue



Launching the Project Settings Dialog in Visual Designer

3. Specify the programmer (AVRDUDE) and the board (eg. UNO)


Project Options

Edit: Debug (Active) Configuration

Tool chain: Visual Designer for Arduino AVR

Controller | Options

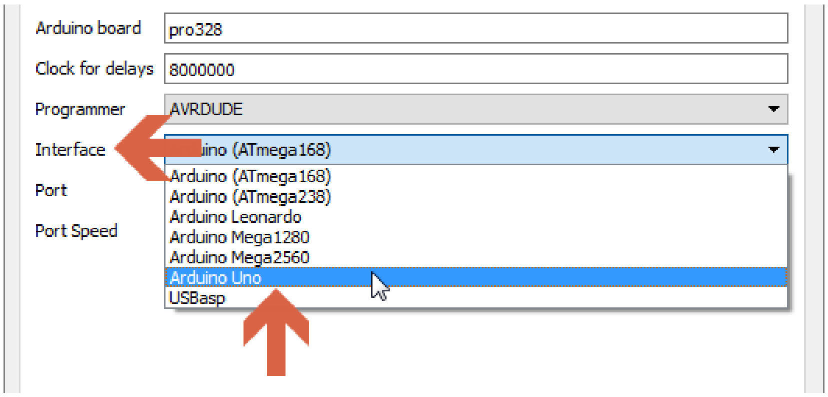
Processor	ARDUINO UNO(U1)
Family	ARDUINO
Controller	Arduino Uno
Embed Files	☑
Arduino board	pro328
Clock for delays	8000000
Programmer	None

None
AVRDUDE

OK | Cancel

Specify the Programmer.

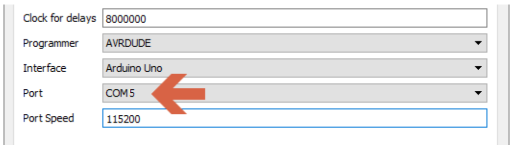
4. Specify the correct Interface based on the physical hardware you have connected.


Arduino board	pro328
Clock for delays	8000000
Programmer	AVRDUDE
Interface	Arduino (ATmega168)
Port	
Port Speed	

Arduino (ATmega168)
Arduino (ATmega238)
Arduino Leonardo
Arduino Mega1280
Arduino Mega2560
Arduino Uno
USBasp

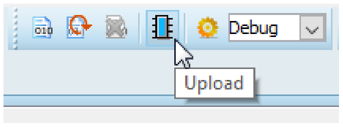
Specifying the Interface.

5. Plug in the hardware, check device manager and enter the COM Port that your PC is using for the USB here. Refer to the instructions below if you are unsure what to select


Clock for delays	8000000
Programmer	AVRDUDE
Interface	Arduino Uno
Port	COM5
Port Speed	115200

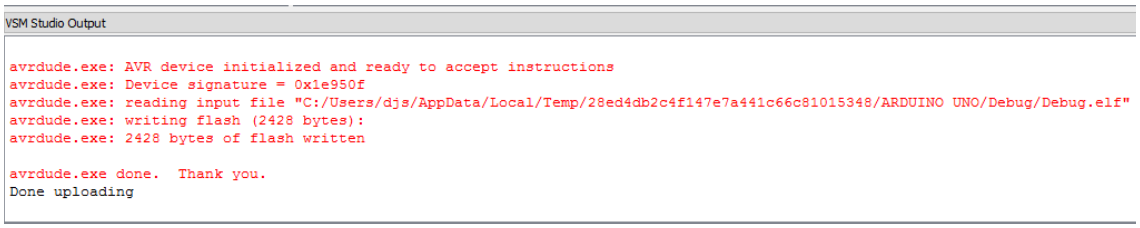
Set the Correct COM Port.

6. Compile your program and press the program button to transfer to the real hardware



Build and Program the Hardware.

You should receive the following message:



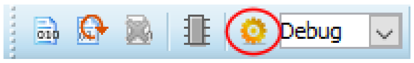
VSM Studio Output

```
avrdude.exe: AVR device initialized and ready to accept instructions
avrdude.exe: Device signature = 0x1e950f
avrdude.exe: reading input file "C:/Users/djs/AppData/Local/Temp/28ed4db2c4f147e7a441c66c81015348/ARDUINO UNO/Debug/Debug.elf"
avrdude.exe: writing flash (2428 bytes):
avrdude.exe: 2428 bytes of flash written

avrdude.exe done.  Thank you.
Done uploading
```

Completed Upload to Hardware.

The Controller Dialog

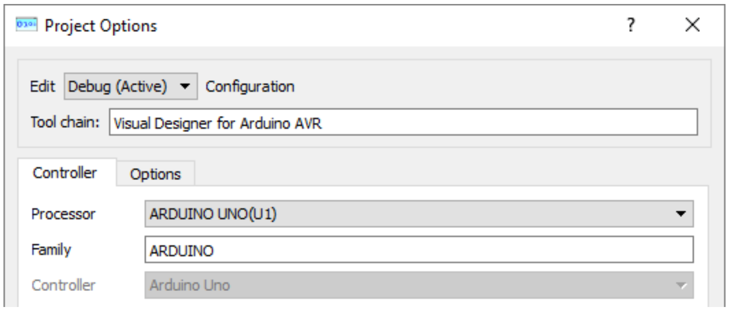
Apart from the basic programming options there are a number of additional options on the dialogue form which are discussed in full below.



Launch the Project Settings Dialogue Form.

Processor, Family and Controller

The Processor, Family and Controller fields are set when you create the project and will be automatically populated for you in this dialogue form.



Project Options

Edit Debug (Active) Configuration

Tool chain: Visual Designer for Arduino AVR

Controller | Options

Processor	ARDUINO UNO(U1)
Family	ARDUINO
Controller	Arduino Uno

Processor, Family and Controller Options are preset when you create the project.

Embed Files Option

The Embed Files Option keeps all source and debug files inside the .PDSPRJ file container. This avoids common problems with relative paths to libraries and files as well as making the project easily portable. We do not recommend turning this option off without good reason.

Embed Files ✓

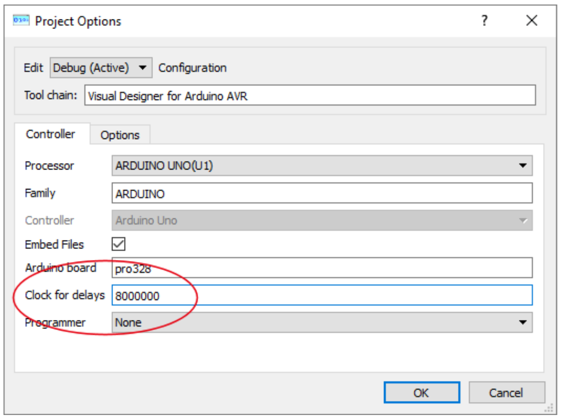
Arduino Board Option

The Arduino board option is a string which is passed to, and used by the Arduino compiler and linker to identify the processor on the board. Do not change this option unless you are confident that you know what you are doing.

Arduino board: pro328

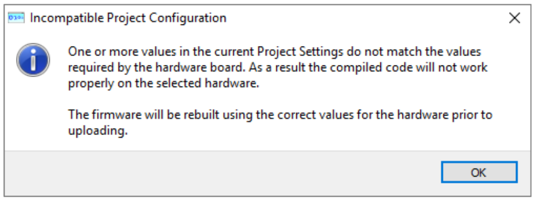
Clock for Delays Option

The clock for delays option is basically the processor clock frequency. For simulation, this is halved to 8MHz to ensure good performance even on modest computers. For the real hardware it needs to be changed to 16MHz but there are two ways to do this.
We recommend that you leave the value in the field alone. Each time you upload you will get a message explaining the discrepancy and Visual Designer will then program the real board with the correct 16MHz clock. This option give the best of both worlds as you will simulate with an 8MHz clock and program with a 16MHz clock.


Project Options

Edit: Debug (Active) Configuration
Tool chain: Visual Designer for Arduino AVR

Processor: ARDUINO UNO(U1)
Family: ARDUINO
Controller: Arduino Uno
Embed Files: ✓
Arduino board: pro328
Clock for delays: 8000000
Programmer: None

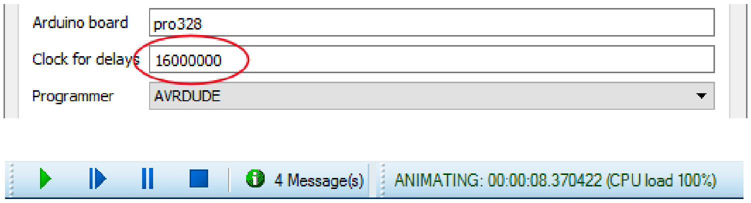

Incompatible Project Configuration

One or more values in the current Project Settings do not match the values required by the hardware board. As a result the compiled code will not work properly on the selected hardware.

The firmware will be rebuilt using the correct values for the hardware prior to uploading.

Leave the clock option alone for best simulation performance while maintaining correct hardware compilation.

If you prefer, you can change the value in the field to 16MHz. This will be correct for the real hardware but may impact simulation performance.



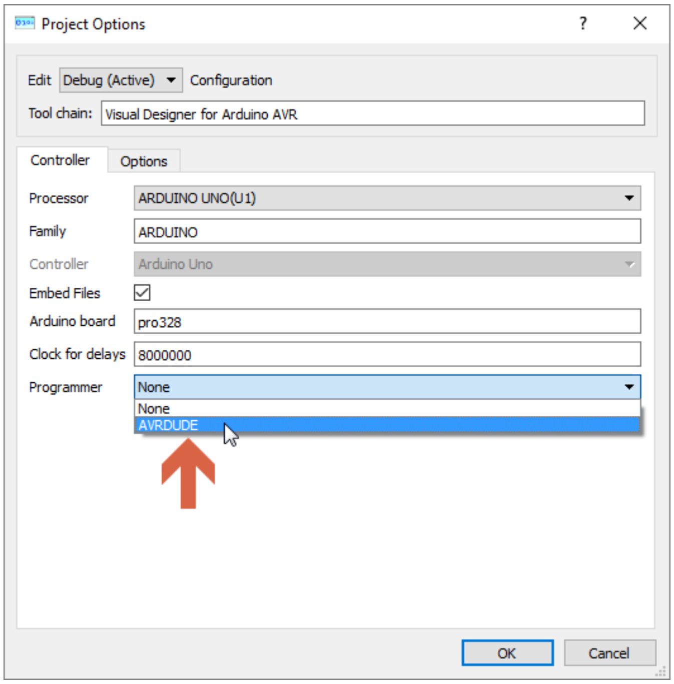
Arduino board: pro328
Clock for delays: 16000000
Programmer: AVRDUDE

4 Message(s) ANIMATING: 00:00:08.370422 (CPU load 100%)

Change the clock for correct hardware compilation at the expense of degraded simulation performance.

Programmer Option

The programmer field should be set to the AVRDUDE programmer. If this is not available/installed then you can easily get it by installing the Arduino IDE from their website.



Project Options

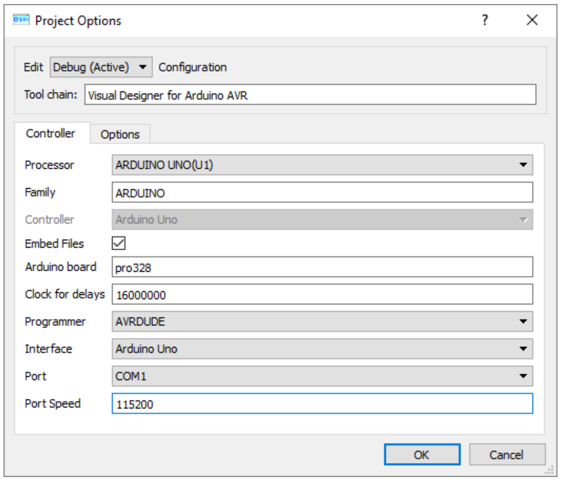
Edit: Debug (Active) Configuration
Tool chain: Visual Designer for Arduino AVR

Controller | Options

Processor: ARDUINO UNO(U1)
Family: ARDUINO
Controller: Arduino Uno
Embed Files: ☑
Arduino board: pro328
Clock for delays: 8000000
Programmer: None
None
AVRDUDE

Interface Option

The interface field should then be set according to the real hardware (e.g Arduino Uno for the Uno board).



Edit: Debug (Active) Configuration
Tool chain: Visual Designer for Arduino AVR

Processor	ARDUINO UNO(U1)
Family	ARDUINO
Controller	Arduino Uno
Embed Files	☑
Arduino board	pro328
Clock for delays	16000000
Programmer	AVRDUDE
Interface	Arduino Uno
Port	COM1
Port Speed	115200

COM Port Option

The COM port field is likely to cause the most confusion. This is determined by your PC when you connect the Arduino hardware to the computer and it is vital that you provide the correct port number in this field. The easiest way to make sure you have the correct number is to

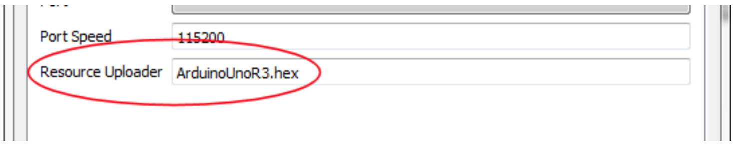
a) Open Device Manager.
b) Plug in the Hardware
c) Make a note of the COMx number that appears and select it in the Port field.

Port Speed Option

The Port Speed or baud rate option is defined in the Arduino datasheets (e.g. 115200 for the Arduino Uno) and should not be changed unless you are sure that you know what you are doing.

Resource Uploader

Finally, the Resource Uploader file is used by Labcenter to automatically transfer Visual Designer resources such as WAV files and Bitmaps to the SD card. You should **not** modify this field.



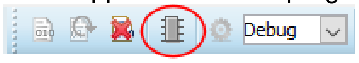
Port Speed	115200
Resource Uploader	ArduinoUnoR3.hex

Troubleshooting Guide

Program Button is Disabled

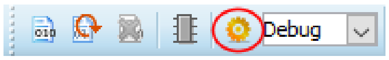
Problem

This happens when the programmer is not installed or set.



Solution

Plug the USB cable in to the board and in to the PC. Make sure that the board has power (LEDs will light up on the board). Visual Designer does not automatically detect that a board has been plugged in so the Upload icon may still be greyed out. Click on the Project Settings icon:



We need to let Proteus know which programmer we wish to use. In this case it is the AVRDUDE If there is nothing listed you will need to install Arduino IDE from:
https://www.arduino.cc/en/Main/Software

Programming takes a long time and then reports multiple errors

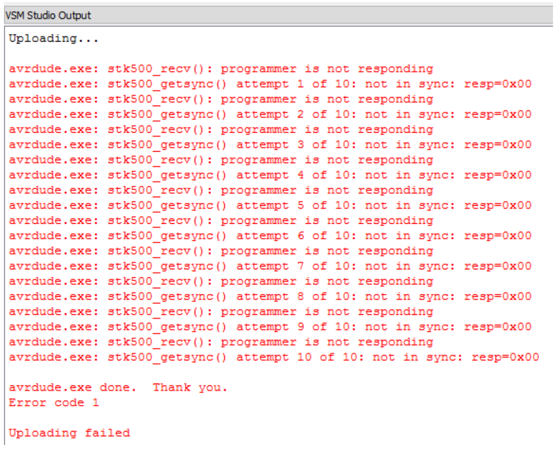
You can stop a build or program at any time by clicking on the Stop Build icon on the toolbar

There can be several different causes for this...

Cause 1

Problem

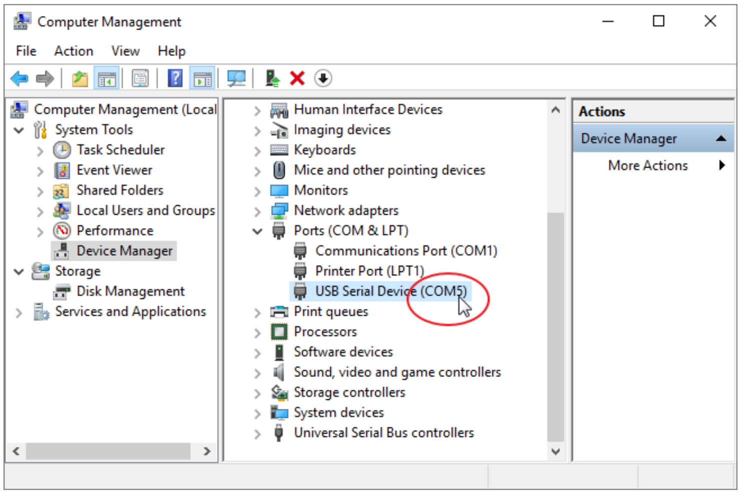
The wrong COM port has been selected. Go back to the Project settings and select the correct COM port.


VSM Studio Output

```
Uploading...

avrdude.exe: stk500_recv(): programmer is not responding
avrdude.exe: stk500_getsync() attempt 1 of 10: not in sync: resp=0x00
avrdude.exe: stk500_recv(): programmer is not responding
avrdude.exe: stk500_getsync() attempt 2 of 10: not in sync: resp=0x00
avrdude.exe: stk500_recv(): programmer is not responding
avrdude.exe: stk500_getsync() attempt 3 of 10: not in sync: resp=0x00
avrdude.exe: stk500_recv(): programmer is not responding
avrdude.exe: stk500_getsync() attempt 4 of 10: not in sync: resp=0x00
avrdude.exe: stk500_recv(): programmer is not responding
avrdude.exe: stk500_getsync() attempt 5 of 10: not in sync: resp=0x00
avrdude.exe: stk500_recv(): programmer is not responding
avrdude.exe: stk500_getsync() attempt 6 of 10: not in sync: resp=0x00
avrdude.exe: stk500_recv(): programmer is not responding
avrdude.exe: stk500_getsync() attempt 7 of 10: not in sync: resp=0x00
avrdude.exe: stk500_recv(): programmer is not responding
avrdude.exe: stk500_getsync() attempt 8 of 10: not in sync: resp=0x00
avrdude.exe: stk500_recv(): programmer is not responding
avrdude.exe: stk500_getsync() attempt 9 of 10: not in sync: resp=0x00
avrdude.exe: stk500_recv(): programmer is not responding
avrdude.exe: stk500_getsync() attempt 10 of 10: not in sync: resp=0x00

avrdude.exe done.  Thank you.
Error code 1

Uploading failed
```

Solution

1. Making sure the board is plugged in
2. Open the Windows Device Manager
3. Select 'Windows Device Manager > Ports (COM & LPT1) > USB Serial Device (COMx)'.
4. Use whatever COM port number is stated there.



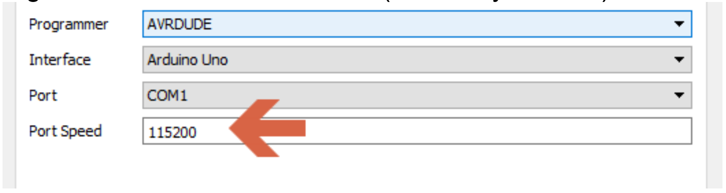
Cause 2

Problem

Incorrect Port Speed.

Solution

Have a look on the data sheet to find out what the Port Speed should be and in the project Settings form, change it to be the correct number (Generally 115200).

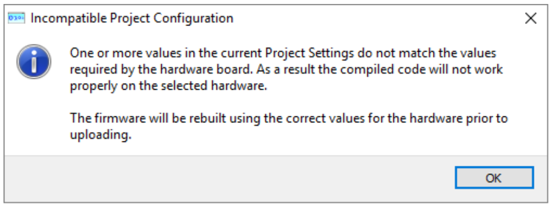


Programmer	AVRDUDE
Interface	Arduino Uno
Port	COM1
Port Speed	115200

Clock for Delays Error

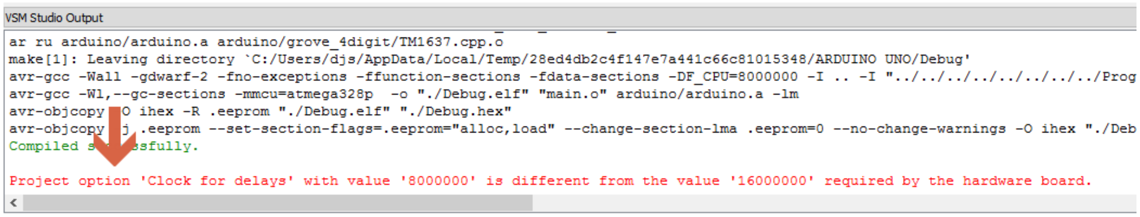
Problem

You get the following error messages:


Incompatible Project Configuration

One or more values in the current Project Settings do not match the values required by the hardware board. As a result the compiled code will not work properly on the selected hardware.

The firmware will be rebuilt using the correct values for the hardware prior to uploading.

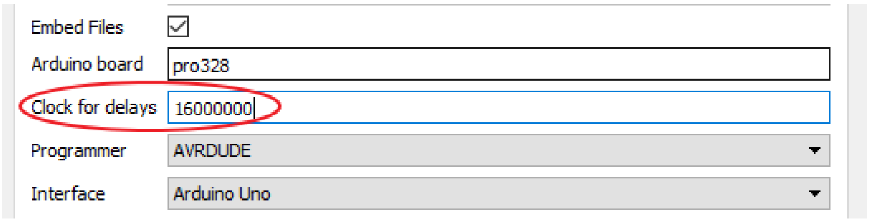

VSM Studio Output

```
ar ru arduino/arduino.a arduino/grove_4digit/TM1637.cpp.o
make[1]: Leaving directory `C:/Users/djs/AppData/Local/Temp/28ed4db2c4f147e7a441c66c81015348/ARDUINO UNO/Debug'
avr-gcc -Wall -gdwarf-2 -fno-exceptions -ffunction-sections -fdata-sections -DF_CPU=8000000 -I .. -I "../../../../../../../../Prog
avr-gcc -Wl,--gc-sections -mmcu=atmega328p  -o "./Debug.elf" "main.o" arduino/arduino.a -lm
avr-objcopy -O ihex -R .eeprom "./Debug.elf" "./Debug.hex"
avr-objcopy -j .eeprom --set-section-flags=.eeprom="alloc,load" --change-section-lma .eeprom=0 --no-change-warnings -O ihex "./Deb
Compiled successfully.

Project option 'Clock for delays' with value '8000000' is different from the value '16000000' required by the hardware board.
```

This will generally be because the clock speed / 'Clock for Delays' value on the board for this particular variant is different than that of the program. In the error message in the output window, it will show the 'set' value in the in the Project Settings dialog, and the 'expected' value required by the board. The software will compile the code with the 'expected' and upload the program. But it will not change it in the settings, so this error will always appear every time the Upload button is pressed.

Solution

This is not necessarily a problem. Indeed, for good simulation performance on modest computers it is almost essential that the clock speed for simulation is different from the clock speed of the real hardware. However, if you do want to change the field to be consistent you can do so by going to to the Project Settings dialog and set the Clock For Delays to be the 'Expected' value or read the datasheet for that variant to find out what the clock speed should be..



Embed Files	✓
Arduino board	pro328
Clock for delays	16000000
Programmer	AVRDUDE
Interface	Arduino Uno

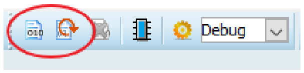
ELF File not found

Problem

There is no compiled firmware to upload

Solution

Compile the firmware via the build button and then try again.



CONFIGURING RASPBERRY PI

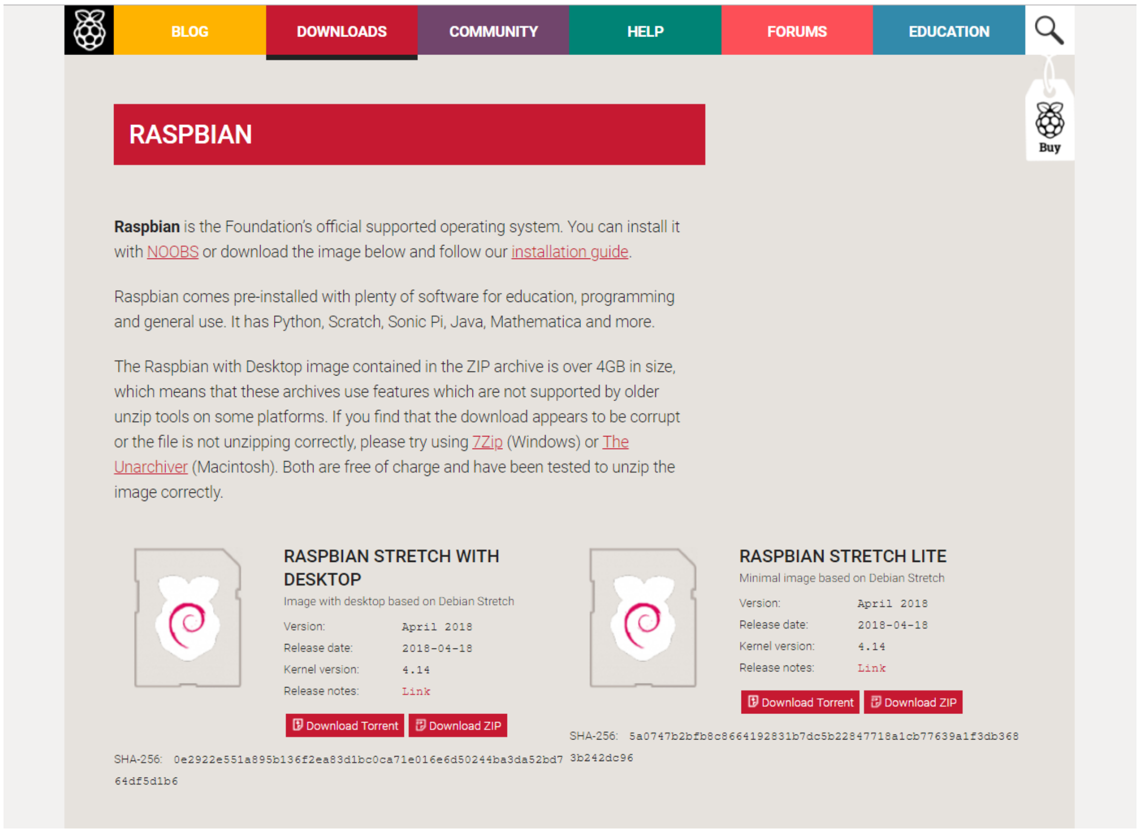
Introduction

Unlike the Arduino, the Raspberry Pi needs to be configured to work with Proteus before it can be programmed. This is normally a one time configuration but can be quite involved. There are three main stages:

1) Prepare the SD Card.
2) Find the Raspberry Pi.
3) Configure the Raspberry Pi for use with Proteus.

Preparing the SD Card

Before you start with your Raspberry Pi you will need a copy of Raspbian installed on an SD card. If you don't know how to do this there is a link below which will walk you through it.
https://www.raspberrypi.org/downloads/raspbian/



RASPBIAN

Raspbian is the Foundation's official supported operating system. You can install it with NOOBS or download the image below and follow our installation guide.

Raspbian comes pre-installed with plenty of software for education, programming and general use. It has Python, Scratch, Sonic Pi, Java, Mathematica and more.

The Raspbian with Desktop image contained in the ZIP archive is over 4GB in size, which means that these archives use features which are not supported by older unzip tools on some platforms. If you find that the download appears to be corrupt or the file is not unzipping correctly, please try using 7Zip (Windows) or The Unarchiver (Macintosh). Both are free of charge and have been tested to unzip the image correctly.

RASPBIAN STRETCH WITH DESKTOP

Image with desktop based on Debian Stretch

Version:	April 2018
Release date:	2018-04-18
Kernel version:	4.14
Release notes:	Link

Download Torrent Download ZIP

SHA-256: 0e2922e551a895b136f2ea83d1bc0ca71e016e6d50244ba3da52bd7 64df5d1b6

RASPBIAN STRETCH LITE

Minimal image based on Debian Stretch

Version:	April 2018
Release date:	2018-04-18
Kernel version:	4.14
Release notes:	Link

Download Torrent Download ZIP

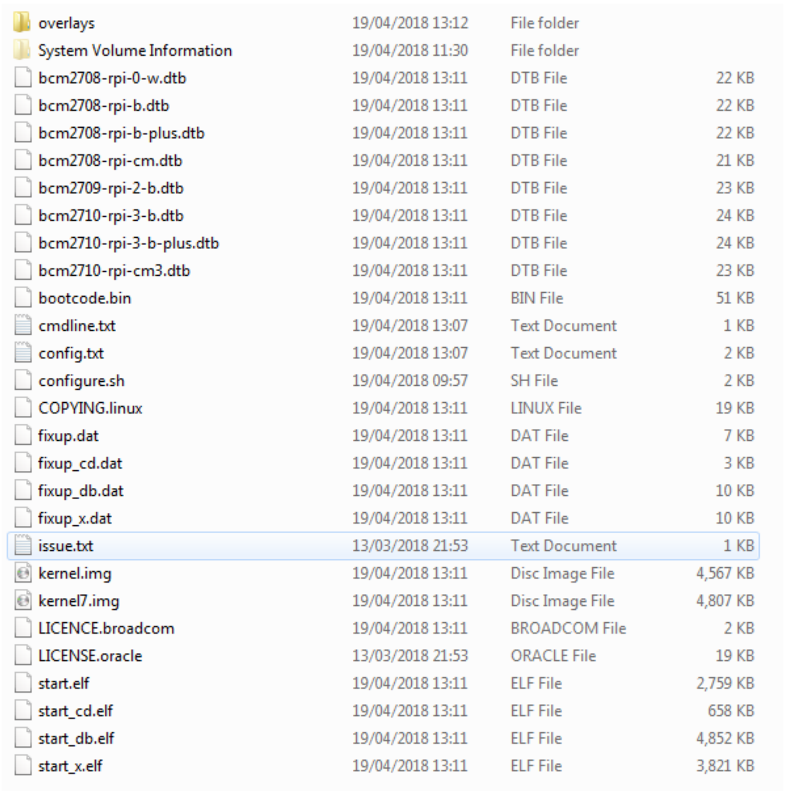
SHA-256: 5a0747b2bfb8c8664192831b7dc5b22847718a1cb77639a1f3db368 3b242dc96

Download Raspbian onto an SD Card.

Proteus will work with both Raspbian and Raspbian Light depending on your preference however Raspbian Light will boot up much faster.

Copying Files

Now that you have Raspbian on an SD card you'll need to open it on your main computer. Don't plug the SD card into the Raspberry Pi yet. Instead, remove it and re-insert it into your computer. Once you open the SD card in your computer you should see a list of files like this.

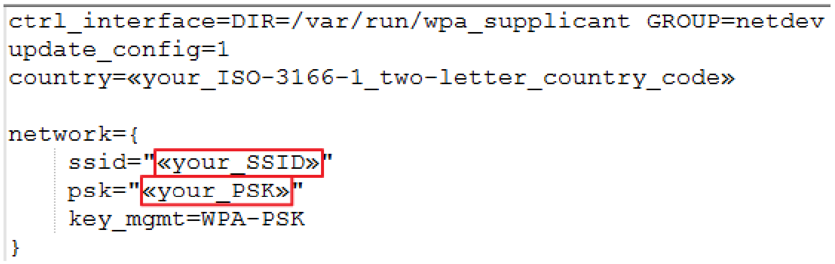


overlays	19/04/2018 13:12	File folder	
System Volume Information	19/04/2018 11:30	File folder	
bcm2708-rpi-0-w.dtb	19/04/2018 13:11	DTB File	22 KB
bcm2708-rpi-b.dtb	19/04/2018 13:11	DTB File	22 KB
bcm2708-rpi-b-plus.dtb	19/04/2018 13:11	DTB File	22 KB
bcm2708-rpi-cm.dtb	19/04/2018 13:11	DTB File	21 KB
bcm2709-rpi-2-b.dtb	19/04/2018 13:11	DTB File	23 KB
bcm2710-rpi-3-b.dtb	19/04/2018 13:11	DTB File	24 KB
bcm2710-rpi-3-b-plus.dtb	19/04/2018 13:11	DTB File	24 KB
bcm2710-rpi-cm3.dtb	19/04/2018 13:11	DTB File	23 KB
bootcode.bin	19/04/2018 13:11	BIN File	51 KB
cmdline.txt	19/04/2018 13:07	Text Document	1 KB
config.txt	19/04/2018 13:07	Text Document	2 KB
configure.sh	19/04/2018 09:57	SH File	2 KB
COPYING.linux	19/04/2018 13:11	LINUX File	19 KB
fixup.dat	19/04/2018 13:11	DAT File	7 KB
fixup_cd.dat	19/04/2018 13:11	DAT File	3 KB
fixup_db.dat	19/04/2018 13:11	DAT File	10 KB
fixup_x.dat	19/04/2018 13:11	DAT File	10 KB
issue.txt	13/03/2018 21:53	Text Document	1 KB
kernel.img	19/04/2018 13:11	Disc Image File	4,567 KB
kernel7.img	19/04/2018 13:11	Disc Image File	4,807 KB
LICENCE.broadcom	19/04/2018 13:11	BROADCOM File	2 KB
LICENSE.oracle	13/03/2018 21:53	ORACLE File	19 KB
start.elf	19/04/2018 13:11	ELF File	2,759 KB
start_cd.elf	19/04/2018 13:11	ELF File	658 KB
start_db.elf	19/04/2018 13:11	ELF File	4,852 KB
start_x.elf	19/04/2018 13:11	ELF File	3,821 KB

*SD Card Files on your computer (*Not* the Raspberry Pi)*

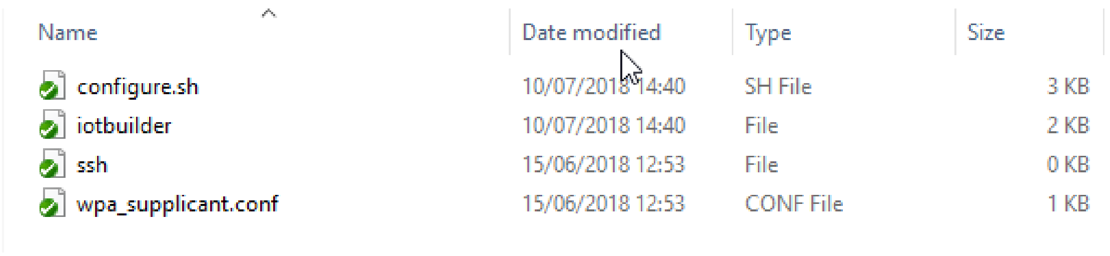
Now collect all the files from the drivers directory of your Proteus installation:
..\VSM Studio\drivers\RaspberryPi\Configure
And insert them into the SD card. If you're not using ethernet you will need to edit wpa_supplicant.conf which will allow you to connect to the network on bootup. Simply input the network name and password and the Raspberry Pi will connect on bootup.

Also at this stage edit the configure.sh file and provide a unique hostname for the Raspberry Pi.



```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=«your_ISO-3166-1_two-letter_country_code»

network={
    ssid="«your_SSID»"
    psk="«your_PSK»"
    key_mgmt=WPA-PSK
}
```

Enter wireless access details in the wpa_supplicant file.



Name	Date modified	Type	Size
configure.sh	10/07/2018 14:40	SH File	3 KB
iotbuilder	10/07/2018 14:40	File	2 KB
ssh	15/06/2018 12:53	File	0 KB
wpa_supplicant.conf	15/06/2018 12:53	CONF File	1 KB

Files to be copied onto the SD Card from your computer.

Now safely eject the card from the computer.

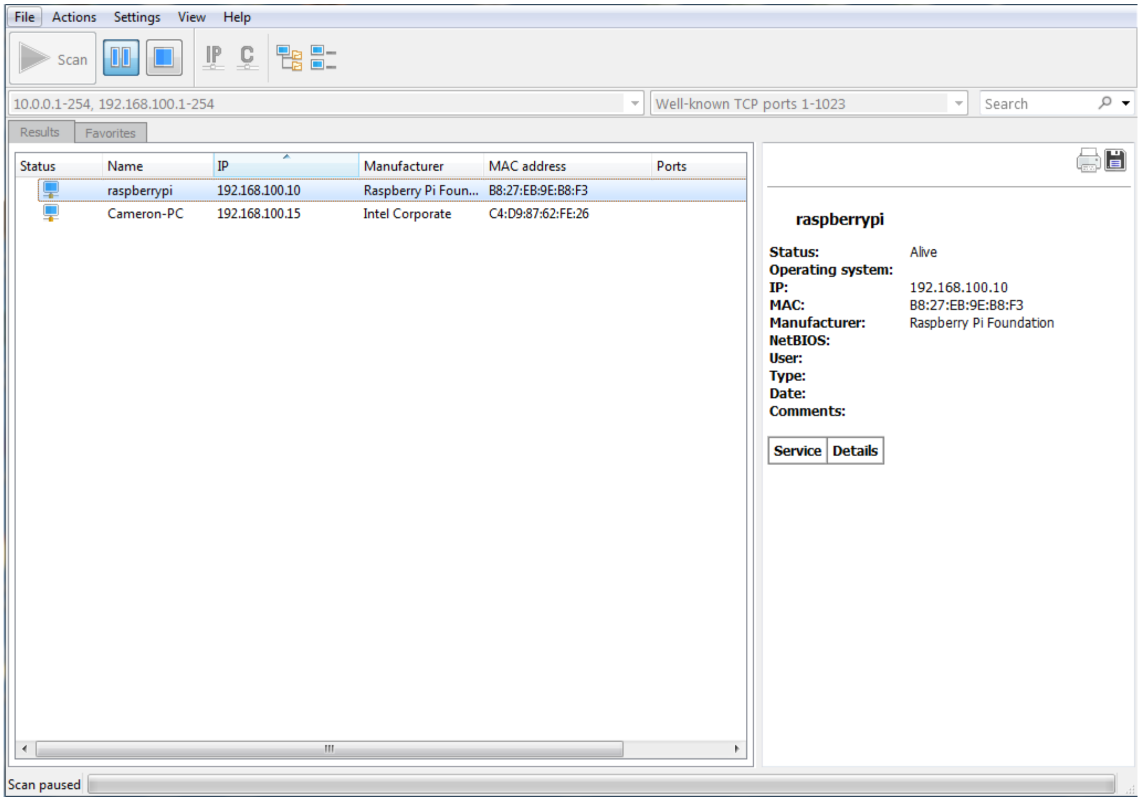
Finding the Raspberry Pi

You will need to find the IP address of your raspberry Pi. There are numerous ways to do so, including the following.

First, installing a free port scanner. This saves time and means you don't have to plug in your Raspberry Pi to a screen and keyboard. So visit the Advanced Port Scanner website. https://www.advanced-port-scanner.com/

Now follow their instructions and install Advanced Port Scanner. The Download button is on their main page. Note this software is free.

Once installed power on the Raspberry Pi.Raspian light will boot faster than normal Raspian as there is no GUI to load. Then use Advanced Port Scanner to look for your Raspberry Pi.



10.0.0.1-254, 192.168.100.1-254

Well-known TCP ports 1-1023

Status	Name	IP	Manufacturer	MAC address	Ports
	raspberrypi	192.168.100.10	Raspberry Pi Foun...	B8:27:EB:9E:B8:F3	
	Cameron-PC	192.168.100.15	Intel Corporate	C4:D9:87:62:FE:26	

raspberrypi

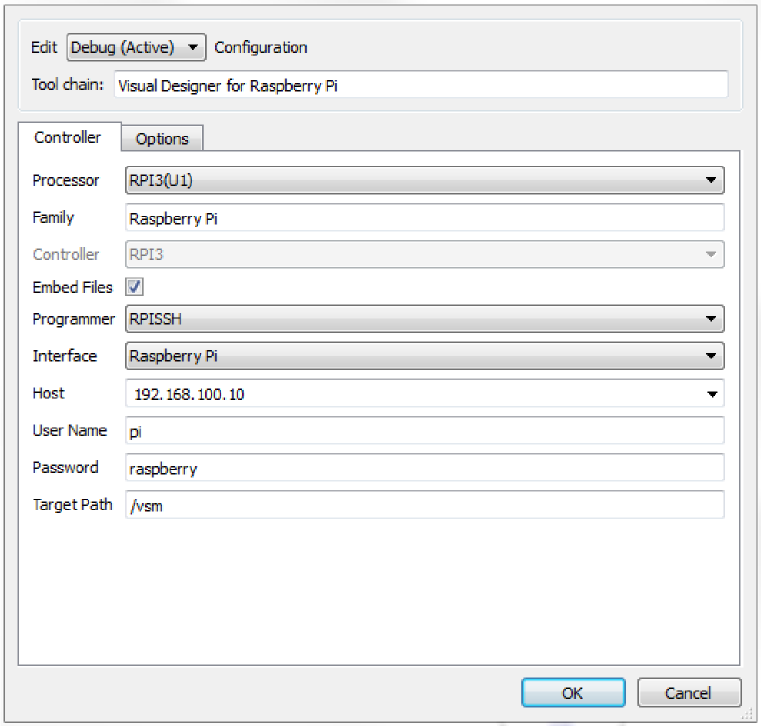
Status:	Alive
Operating system:	
IP:	192.168.100.10
MAC:	B8:27:EB:9E:B8:F3
Manufacturer:	Raspberry Pi Foundation
NetBIOS:	
User:	
Type:	
Date:	
Comments:	

You will most likely have lots of devices on your network. You will be looking for a device with name raspberrypi and manufacturer Raspberry Pi Foundation. Find the IP which in this case is 192.168.100.10. Do not turn off the raspberry Pi.

Configuration of the Raspberry Pi

Method 1: Inside Proteus

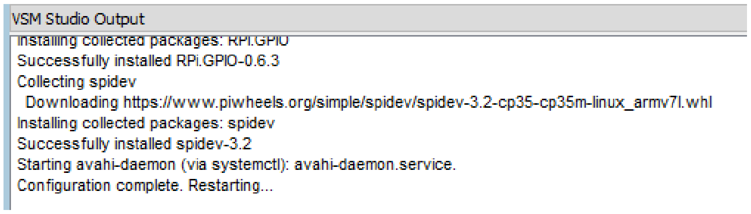
Create a blank Raspberry Pi Project in proteus and open project settings. Change Programmer from None to RPISSH and enter the IP address into the Host section.


Edit Debug (Active) Configuration
Tool chain: Visual Designer for Raspberry Pi

Controller | Options

Processor: RPI3(U1)
Family: Raspberry Pi
Controller: RPI3
Embed Files: ☑
Programmer: RPISSH
Interface: Raspberry Pi
Host: 192.168.100.10
User Name: pi
Password: raspberry
Target Path: /vsm

Now Upload the blank project to your Pi. A dialogue box will pop up asking for confirmation to complete the Raspberry Pi configuration, press OK to continue (Picture may be required). This section will take a while as it downloads necessary libraries and sets it's self up Once complete, the Raspberry Pi will reboot and connection will be lost giving Error Code: -1

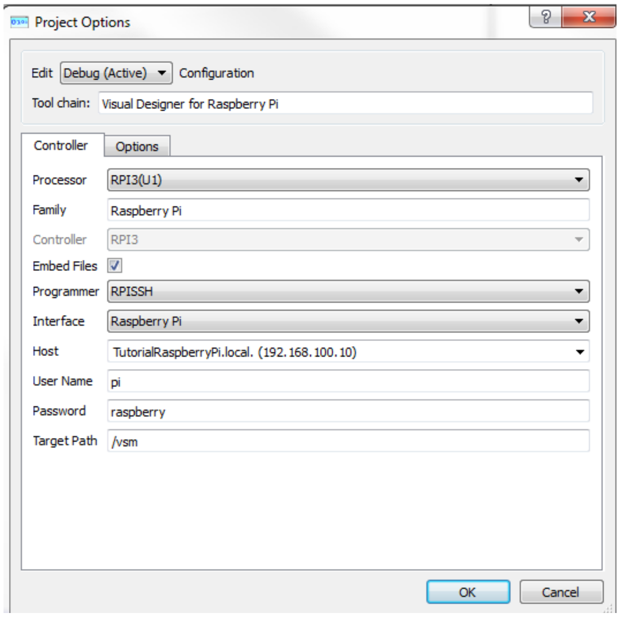

VSM Studio Output

```
Installing collected packages: RPi.GPIO
Successfully installed RPi.GPIO-0.6.3
Collecting spidev
  Downloading https://www.piwheels.org/simple/spidev/spidev-3.2-cp35-cp35m-linux_armv7l.whl
Installing collected packages: spidev
Successfully installed spidev-3.2
Starting avahi-daemon (via systemctl): avahi-daemon.service.
Configuration complete. Restarting...
```

Wait a moment for the Raspberry Pi to reboot and the process is complete.

Your raspberry pi will now appear in Proteus every time you boot it up and advanced port scanner is no longer needed.

Now if you go into program settings the raspberry pi will be there with whatever name you provided in the configure.sh file.


Project Options

Edit Debug (Active) Configuration
Tool chain: Visual Designer for Raspberry Pi

Controller | Options

Processor: RPI3(U1)
Family: Raspberry Pi
Controller: RPI3
Embed Files: ✓
Programmer: RPISSH
Interface: Raspberry Pi
Host: TutorialRaspberryPi.local. (192.168.100.10)
User Name: pi
Password: raspberry
Target Path: /vsm

Method 2: PuTTY

You should be able to work with PuTTY to talk to the Raspberry Pi without plugging in a keyboard, mouse and screen. If you are using Raspbian and are plugging in a keyboard and mouse you may not need this step.

This link will take you to PuTTYs main site and will talk you through the download steps.

https://www.raspberrypi.org/documentation/remote-access/ssh/windows.md

Connect PuTTY to the Raspberry Pi and write:

```
sudo /boot/configure.sh
```

This may take a while to install everything but once it has finished you will need to reboot the Raspberry Pi.

Method 3 : Keyboard and Mouse

If you are instead using keyboard and screen open the command line which is called Terminal in Raspbian. Now write:

```
sudo /boot/configure.sh
```

This may take a while to install everything but once it has finished you will need to reboot the Raspberry Pi.

ARDUINO TUTORIALS

Introduction

It's remarkably simple to design and create electronic projects with Visual Designer. These short tutorials each cover a separate project and include various different techniques and methods in the project design.

Arduino Tutorials

Tutorial 1 : Flashing an LED (Grove).

Covers project creation, adding peripherals. basic flowchart programming and Proteus simulation.

Tutorial 2 :Nightlight (Grove)

Includes grove module configuration, decision making flowcharts, interactive simulation and basic debugging.

Tutorial 3: Data Storage (AdaFruit Shield)

Includes resource handling, adding shields, taking measurements and programming physical hardware.

Tutorial 4 : Motor Control (Adafruit Shield)

Includes programming with loops, adding and calling sub-routines and using the clipboard..

Tutorial 5: LED (Schematic Design)

Includes schematic entry for the electronics, using interrupts and CPU methods in Visual Designer.

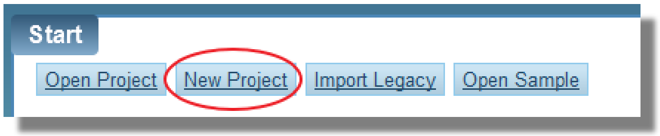
Tutorial 1 : Flashing LED's

Introduction

This tutorial starts when you open Proteus and finishes with the flashing of an LED in a Proteus simulation. The goal here is to cover the basics with the most simple of projects.

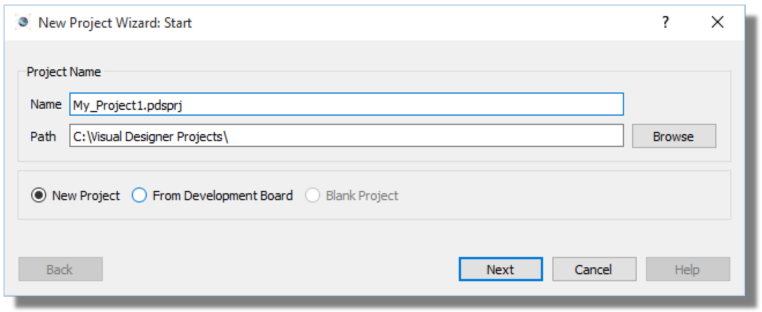
New Project Wizard

Visual Designer is integrated into the Proteus Design Suite and so we start our Visual Designer Projects from the Proteus Home Screen. The easiest and best way to create a Visual Designer Project is, unsurprisingly, through the New Project Wizard.



New project from the Home page of Proteus

The first page allows you to select the name and destination of your project.


New Project Wizard: Start

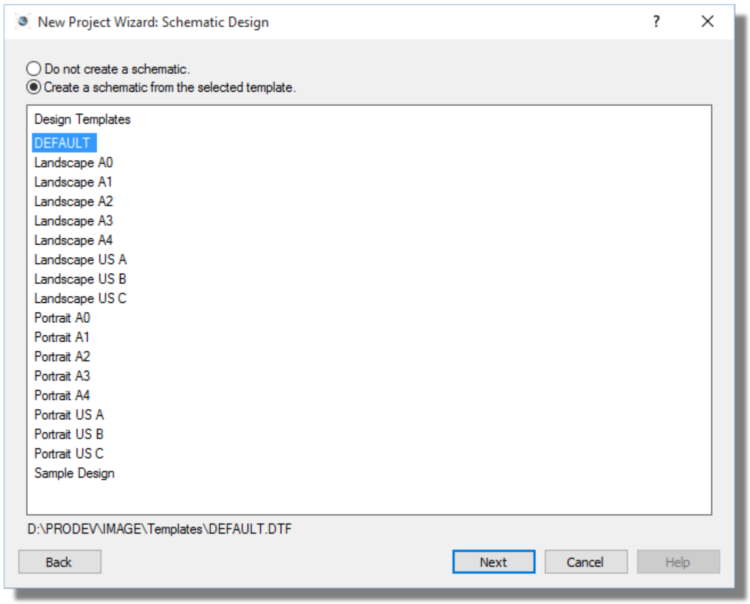
Project Name

Name: My_Project1.pdsprj

Path: C:\Visual Designer Projects\

[✓ New Project] From Development Board Blank Project

The second screen of the wizard allows us to select which size of schematic sheet we want. The default is fine for all but the largest of projects and is certainly OK for the flashing of an LED.

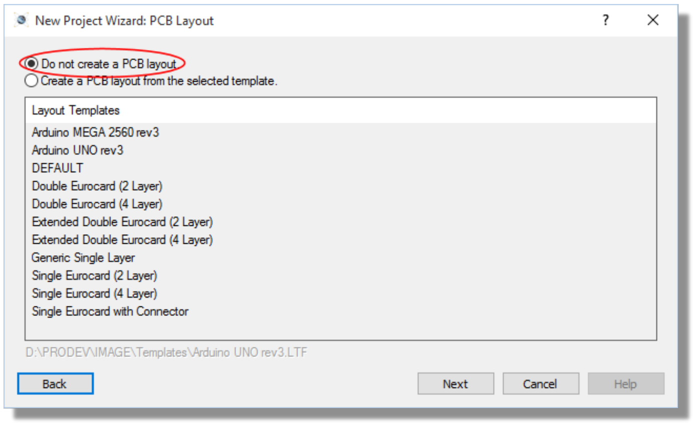

New Project Wizard: Schematic Design

Do not create a schematic. [✓ Create a schematic from the selected template.]

Design Templates
- DEFAULT
- Landscape A0
- Landscape A1
- Landscape A2
- Landscape A3
- Landscape A4
- Landscape US A
- Landscape US B
- Landscape US C
- Portrait A0
- Portrait A1
- Portrait A2
- Portrait A3
- Portrait A4
- Portrait US A
- Portrait US B
- Portrait US C
- Sample Design

D:\PRODEV\IMAGE\Templates\DEFAULT.DTF

If your license includes Proteus PCB Design the next screen will offer you an option to create an accompanying PCB with the project. It's not necessary for our purposes and beyond the scope of this document so we'll leave the option disabled.


New Project Wizard: PCB Layout

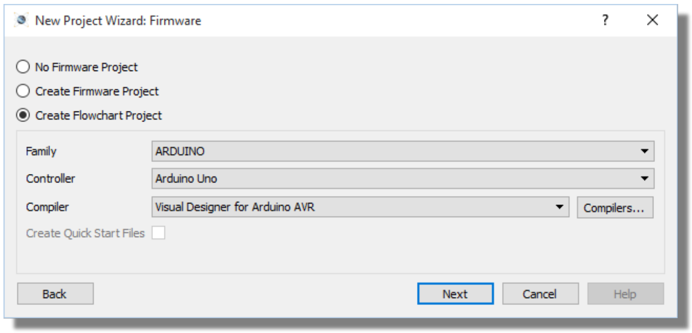
[✓ Do not create a PCB layout] Create a PCB layout from the selected template.

Layout Templates
- Arduino MEGA 2560 rev3
- Arduino UNO rev3
- DEFAULT
- Double Eurocard (2 Layer)
- Double Eurocard (4 Layer)
- Extended Double Eurocard (2 Layer)
- Extended Double Eurocard (4 Layer)
- Generic Single Layer
- Single Eurocard (2 Layer)
- Single Eurocard (4 Layer)
- Single Eurocard with Connector

D:\PRODEV\IMAGE\Templates\Arduino UNO rev3.LTF

You may not see this screen at all if your license key doesn't include PCB Design.

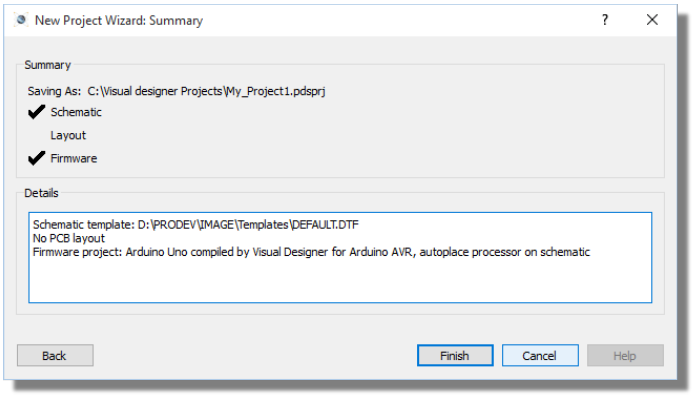
Next up is the firmware screen and it is here that we really define our Visual Designer project. First, we change the radio buttons at the top to the option for Flowchart Project and then we select the Arduino family and either the Arduino Uno or the Arduino Mega from the controller combo box.



Family: ARDUINO
Controller: Arduino Uno
Compiler: Visual Designer for Arduino AVR
Create Quick Start Files

The firmware selection refers to a Proteus VSM simulation project rather than a Visual Designer flowchart project.

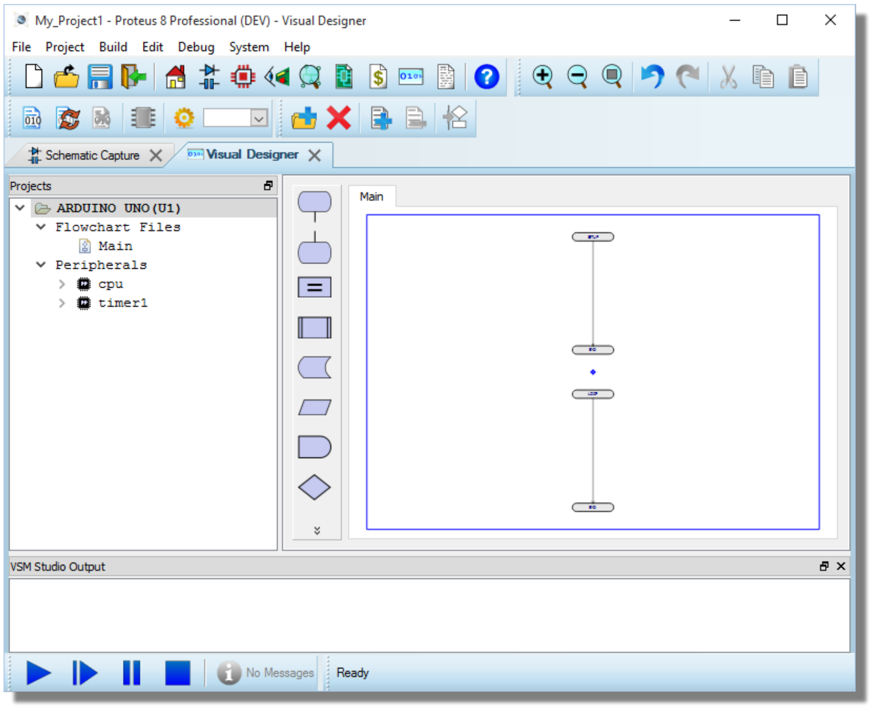
Finally, you will be presented with a summary page of your configuration.



Summary
Saving As: C:\Visual designer Projects\My_Project1.pdsprj
✓ Schematic
Layout
✓ Firmware

Details
Schematic template: D:\PRODEV\IMAGE\Templates\DEFAULT.DTF
No PCB layout
Firmware project: Arduino Uno compiled by Visual Designer for Arduino AVR, autoplace processor on schematic

After you exit the dialogue form the project will be created and you will see both a skeleton schematic with the Arduino processor placed and a skeleton chart project with the familiar Setup and Loop routines on the Editing Window.



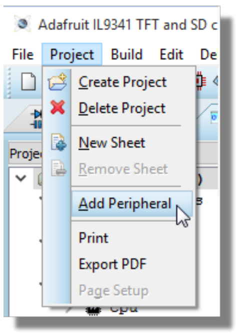
The standard Arduino Sketch includes two functions, namely Setup and Loop. The Setup function is intended for one time initialization and the Loop function provides a place for the main program loop. Visual Designer for Arduino uses the same paradigm and you **must have at least one of these two constructs in your flowchart** for it to compile. If you have deleted them, just add an event block and name it appropriately.

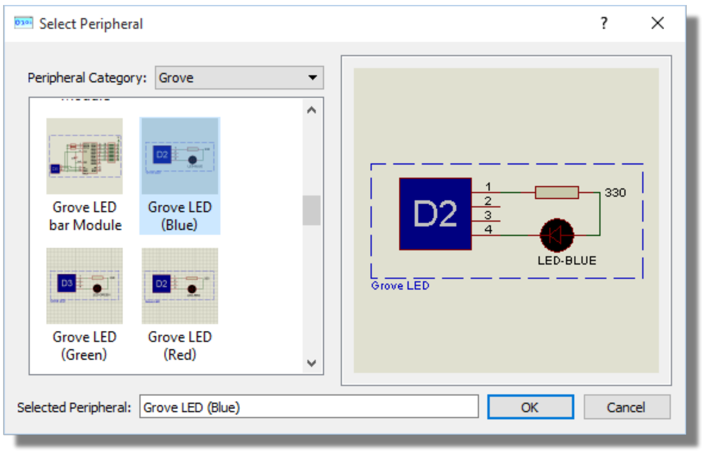
Now we are ready to start designing our project.

Adding a Peripheral

The first thing to do is to add the peripheral(s) for the project. These can be Arduino shields or just little Grove sensors. The Grove system consists of an Arduino shield with lots of 4-pin headers into which any number of sensors, buttons and LED's can be plugged. This makes it very flexible and ideal for experimenting.

We'll be adding a Grove LED. In Visual Designer, we right click on the Project Tree and select the Add Peripheral command. Next, switch to Grove and select one of the LED's.

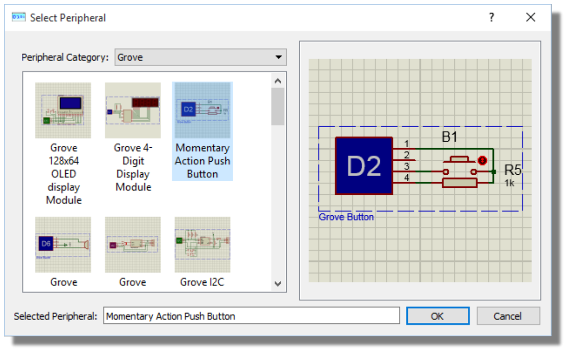




Peripheral Category: Grove

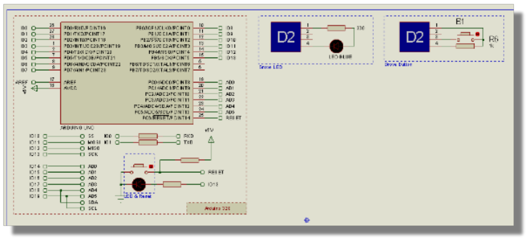
Selected Peripheral: Grove LED (Blue)

Repeat the process to add a Grove button. If you then switch to the schematic tab you will see that the 'virtual hardware' has been autoplaced for you.

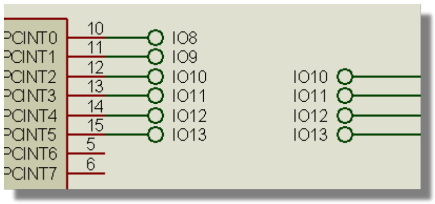

Select Peripheral
Grove 128x64 OLED display Module
Grove 4-Digit Display Module
Momentary Action Push Button
Grove
Grove
Grove I2C
D2
B1
R5
1k
Grove Button
OK
Cancel

Peripheral Category: Grove

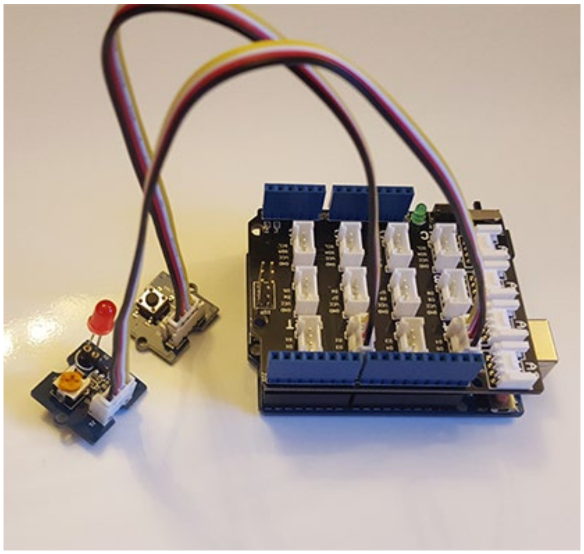
Selected Peripheral: Momentary Action Push Button



To avoid clutter on the schematic, connections are made on the schematic by giving terminals the same name. Any terminals with the same name can be thought of having an invisible wire between them

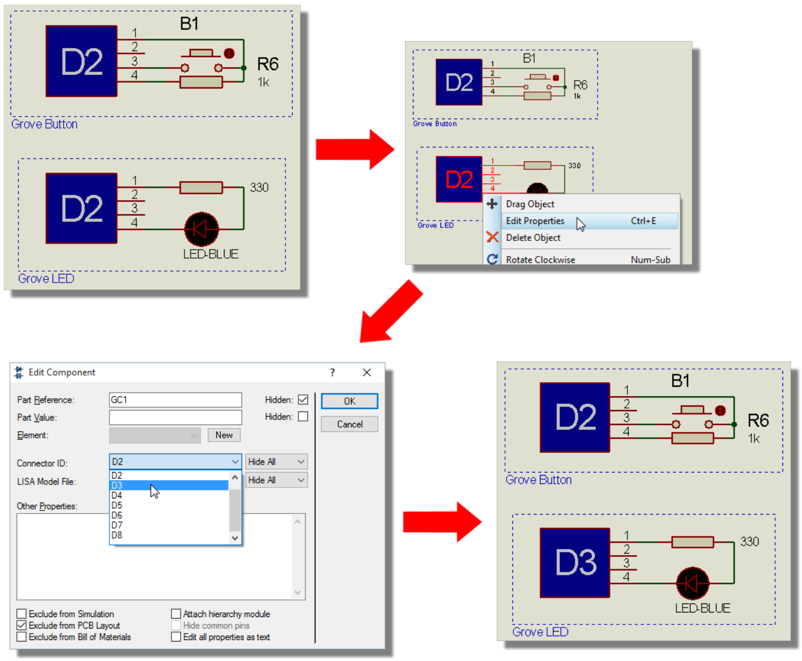


In real hardware, this equates to connecting up the Grove button and Grove LED to the shield and attaching the shield to the Arduino Uno.



Grove LED and Grove Button connected onto the Arduino UNO.

Note that you must plug the peripherals into the same connectors on the Grove shield as they are designated on the Proteus schematic. You must also ensure that you don't have two peripherals with the same connector ID on the schematic; if you do, you need to switch to the schematic and change the ConnectorID on one of the peripherals.



Edit Component

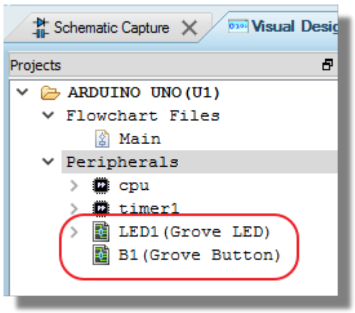
Part Reference:	GC1	Hidden: ☑	OK
Part Value:		Hidden: ☐	Cancel
Element:		New	
Connector ID:	D2	Hide All	
LISA Model File:		Hide All	
Other Properties:			

D2
D3
D4
D5
D6
D7
D8

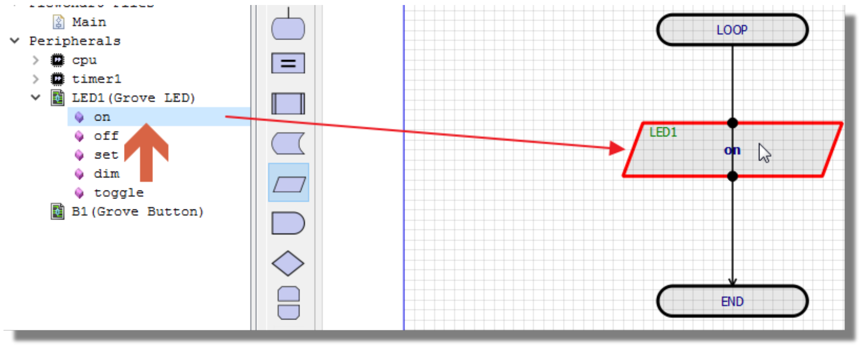
☐ Exclude from Simulation
☑ Exclude from PCB Layout
☐ Exclude from Bill of Materials
☐ Attach hierarchy module
☐ Hide common pins
☐ Edit all properties as text

Drag and Drop Flowchart Design

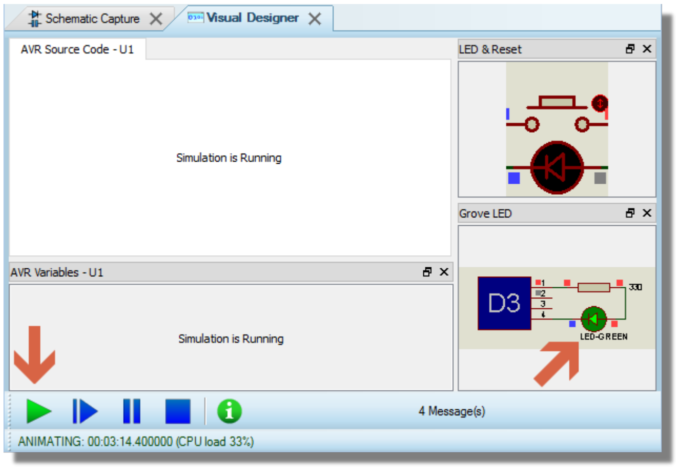
Back in Visual Designer, you'll notice that the Project Tree now has two entries under the peripherals section.



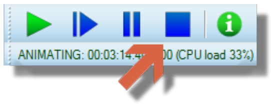
The beauty of Visual Designer is that you can expand the peripherals to find a list of methods. These methods are the primary way in which we interact with the hardware. For example, drag and drop the 'On' method of the LED into the loop routine.



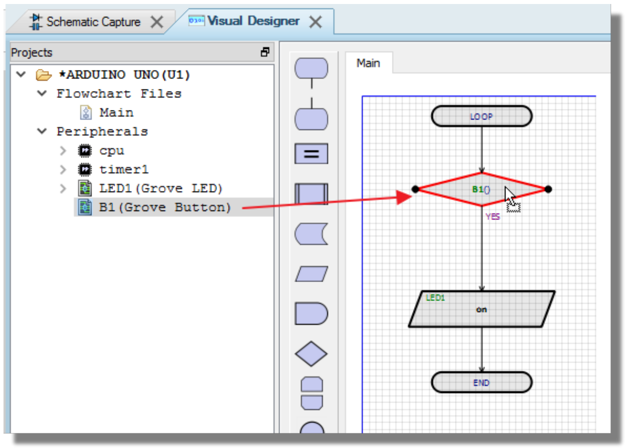
To test this simply press the play button. The program will compile, the simulator will start and the LED will turn on. You can see this either via the Active popup at the right of Visual Designer or by switching to the schematic tab.



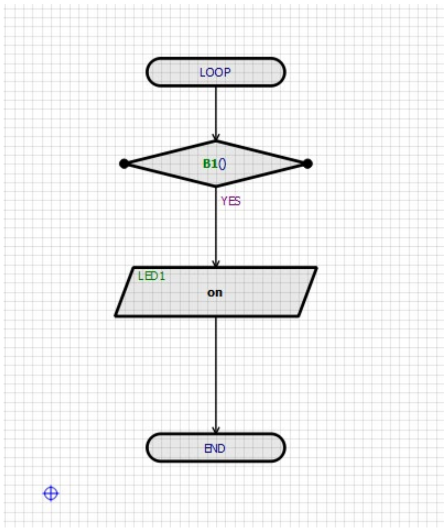
Press stop to exit the simulation and we'll extend our program to switch the LED on and off with a button press. You'll notice that the button has no methods in the project tree.



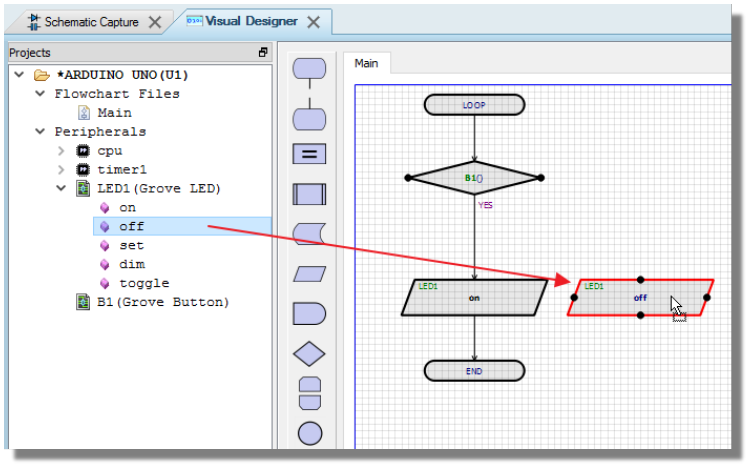
The only sensible information we need from the button is to know whether it has been pressed and for that we have what we call a 'sensor function'. A sensor function returns either TRUE or FALSE depending on whether the basic peripheral function is true or not. For example, the sensor function for a button will return TRUE when the button is pressed and FALSE if it isn't. Similarly, the sensor function for an LED will return TRUE when the LED is on and FALSE when the LED is off. Place the sensor function for the button by dragging and dropping from the button itself onto the loop function of the flowchart.



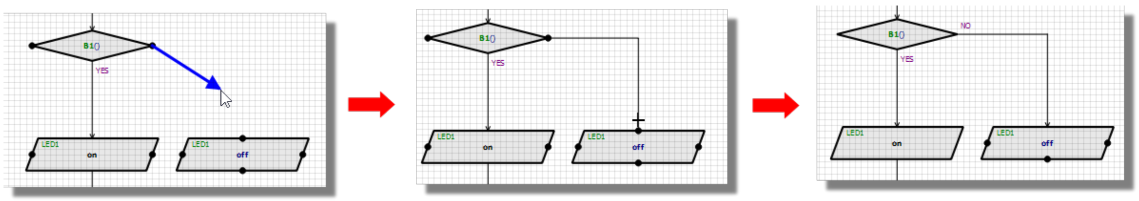
You will notice that it appears as a decision block. Sensor functions always return TRUE or FALSE and a decision block allows us to split our code into two conditional paths. Your loop routine should currently look something like the following.



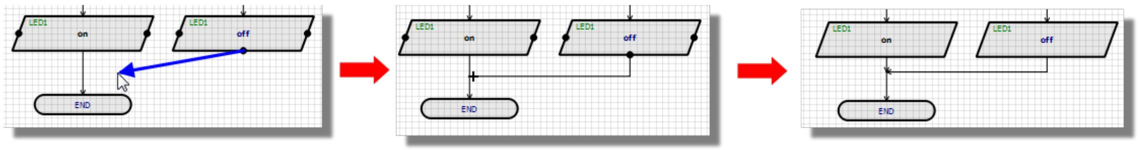
We are constantly testing the button to see if it is pressed and we are turning on the LED if it is. However, we still need to tell the program what to do if the button is not pressed. We want the LED to turn off so lets drag and drop that method into space alongside the ON operation.



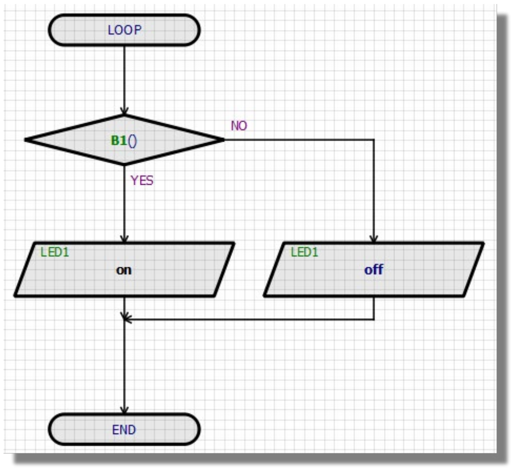
Now, we need to wire from the decision block to the top of the OFF command. To place a flowline click on an output node (the decision block), move the mouse to an available input node (top of the OFF routine) and click the mouse again.



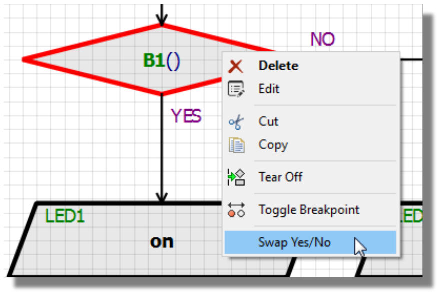
Similarly we need to connect the flow from the bottom of the OFF command back into the main loop.



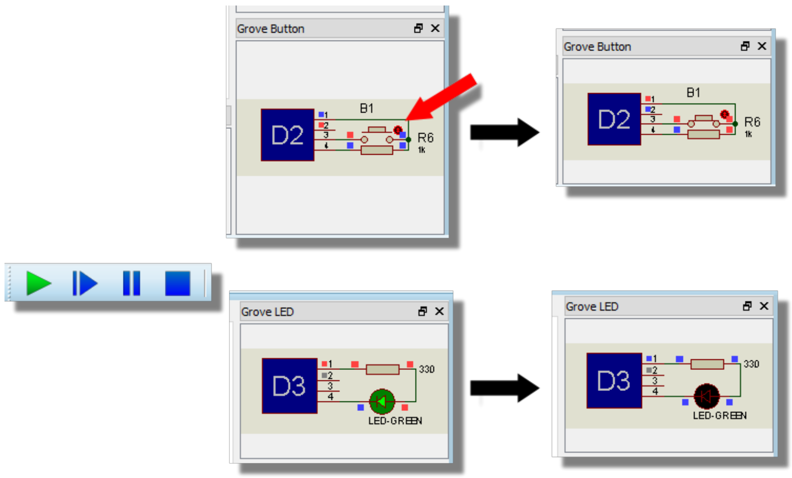
Your program should now look like the following



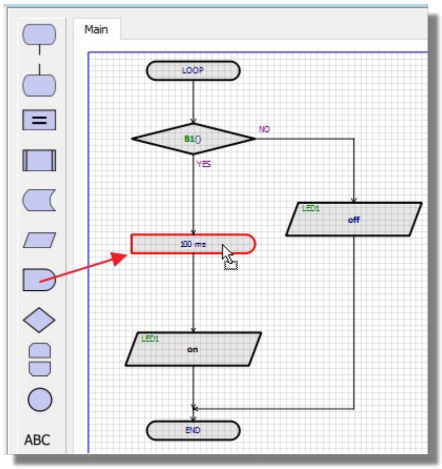
Note that the decision block has two labels indicating which code path is followed if the decision resolves to TRUE (YES) and which code path is followed if the decision resolves to FALSE (NO). If these are the wrong way around you can switch them by right clicking on the decision block and selecting the swap command from the context menu.



Having made this change, our program now turns the LED off when the button is down and on when the button is released. To compile and test, first press the play button and then use the mouse to press the button down.



You can experiment further here with changing the decision or swapping in the toggle method. You might also find it useful to add a delay.

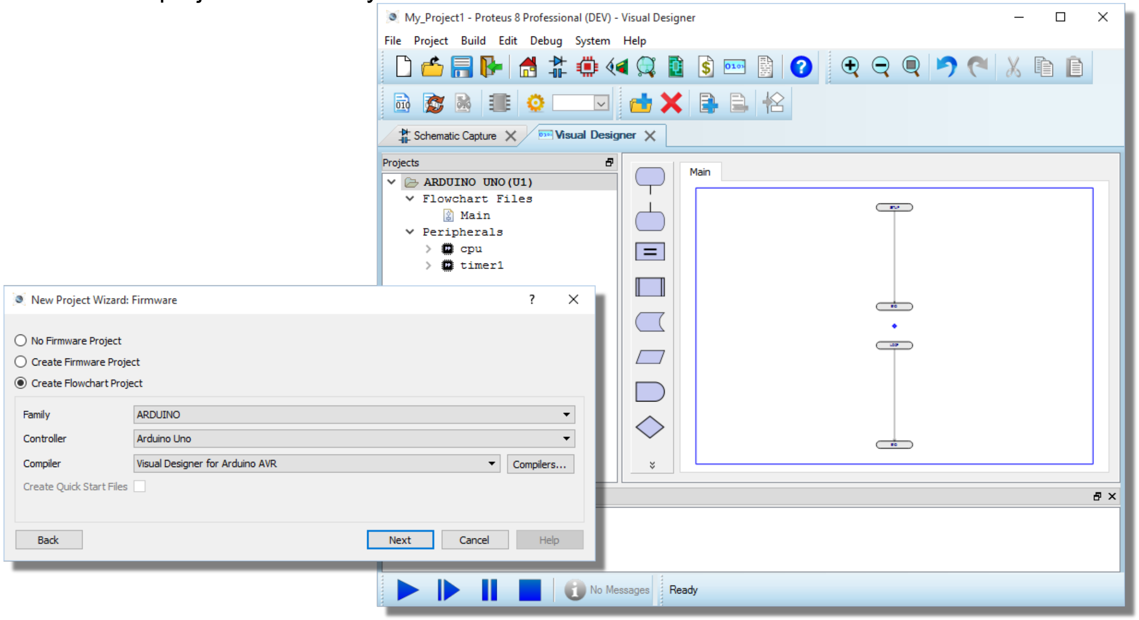


For more information, read Tutorial 2.

TUTORIAL 2: LUX SENSOR

Introduction

In this tutorial we are going to use a couple of Grove sensors and an LED to design a mini nightlight. Project setup is covered in the first tutorial so we'll assume that you have a blank Arduino Uno project in front of you.



New Project Wizard: Firmware

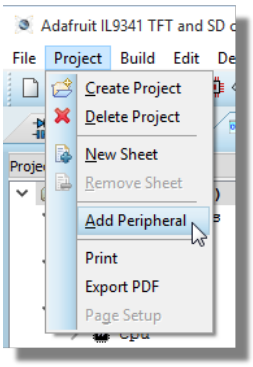
( ) No Firmware Project
( ) Create Firmware Project
(●) Create Flowchart Project

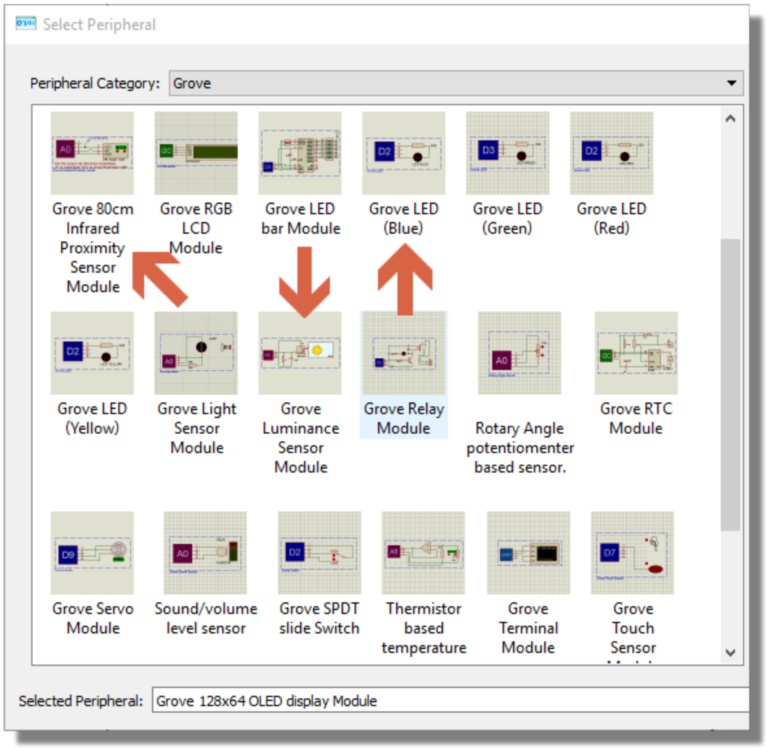
Family	ARDUINO
Controller	Arduino Uno
Compiler	Visual Designer for Arduino AVR
Create Quick Start Files	☐

This tutorial is also covered by a short movie on the Labcenter Youtube Channel

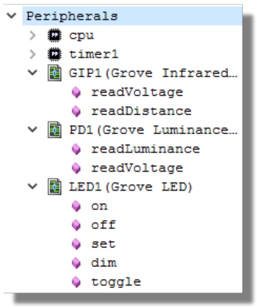
Adding Hardware

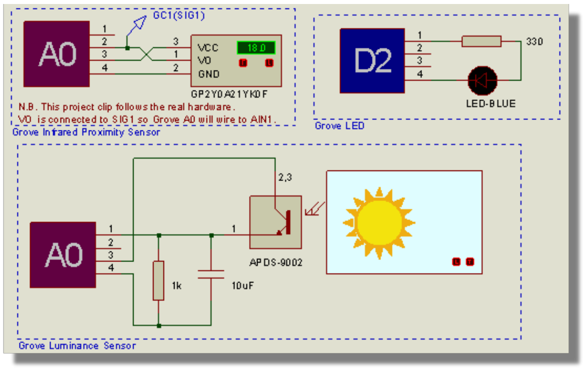
We'll use the Grove infra-red proximity sensor and the Grove luminance sensor along with a Grove LED. All of these can be picked via the Add Peripheral command on the Project Menu.




Select Peripheral
Peripheral Category: Grove
Grove 80cm Infrared Proximity Sensor Module
Grove RGB LCD Module
Grove LED bar Module
Grove LED (Blue)
Grove LED (Green)
Grove LED (Red)
Grove LED (Yellow)
Grove Light Sensor Module
Grove Luminance Sensor Module
Grove Relay Module
Rotary Angle potentiomenter based sensor.
Grove RTC Module
Grove Servo Module
Sound/volume level sensor
Grove SPDT slide Switch
Thermistor based temperature
Grove Terminal Module
Grove Touch Sensor
Selected Peripheral: Grove 128x64 OLED display Module

After they have been added to the project, you will see the methods for controlling the peripherals in the Project Tree and the 'virtual hardware' for the peripherals placed for you on the schematic.

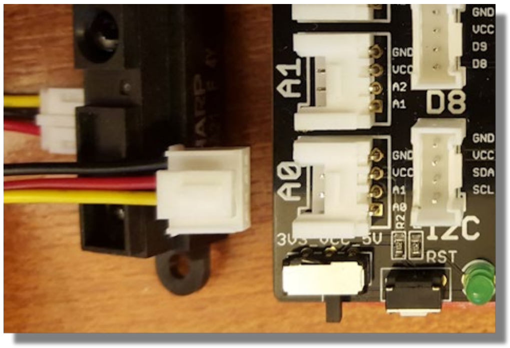


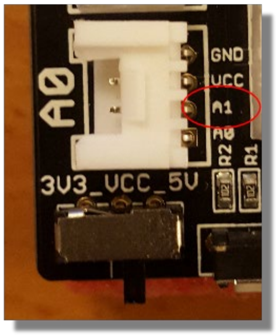

GC1(SIG1)
A0
VCC
VO
GND
GP2Y0A21YK0F
N.B. This project clip follows the real hardware.
VO is connected to SIG1 so Grove A0 will wire to AIN1.
Grove Infrared Proximity Sensor
D2
330
LED-BLUE
Grove LED
2,3
APDS-9002
1k
10uF
Grove Luminance Sensor

⚠ Note that the connectors are both labeled A0 and will need to be manually changed to be A0 & A2.

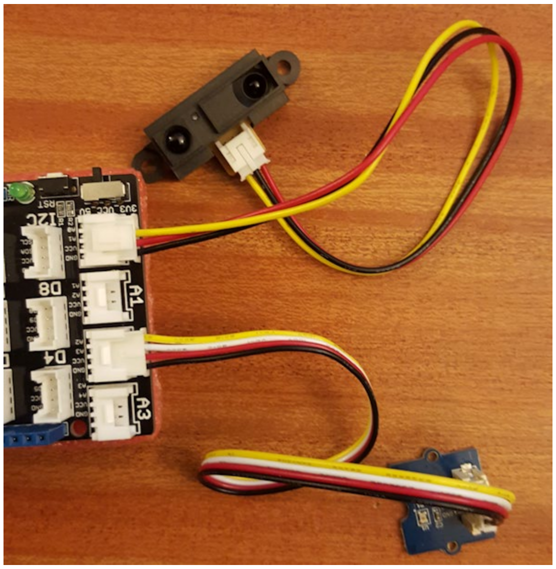
It is vitally important to configure the Grove connectors properly and this is often not as simple as it first appears. Let's take an example:

The real proximity sensor requires only three wires with the fourth left NC. Note therefore that, when plugged into socket A0 the sensor is connecting to analog pin A1.





This is fine, **<u>except</u>** if you then plug the luminance sensor into socket A1. The luminance sensor will use analog pin A1 on socket A1 and therefore clash. You should plug this sensor into socket A2.



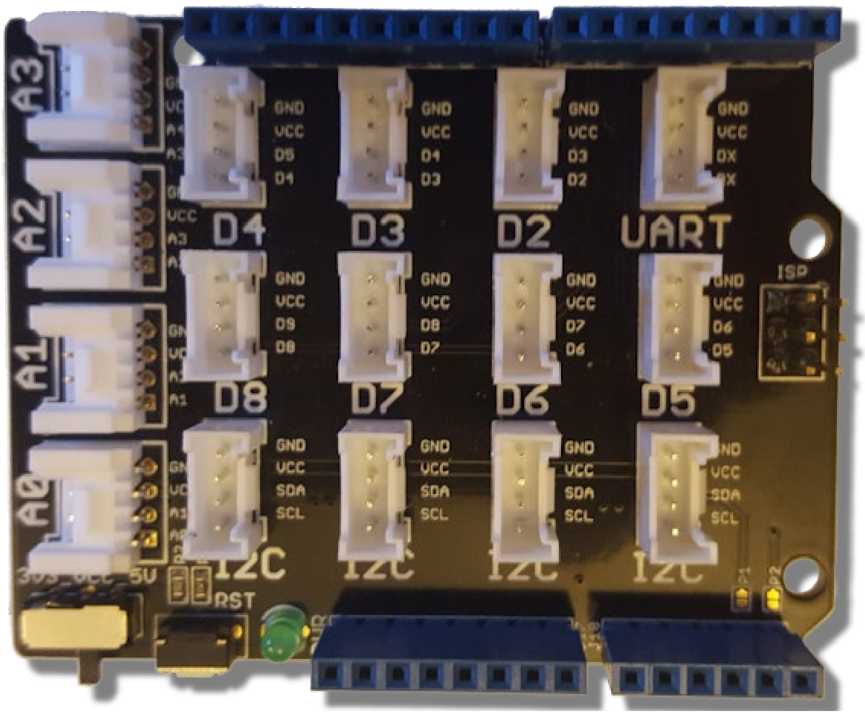
<u>Golden Rule</u>

<u>Look closely at the Grove board to see the connections and double check against the sensors for conflicts.</u>

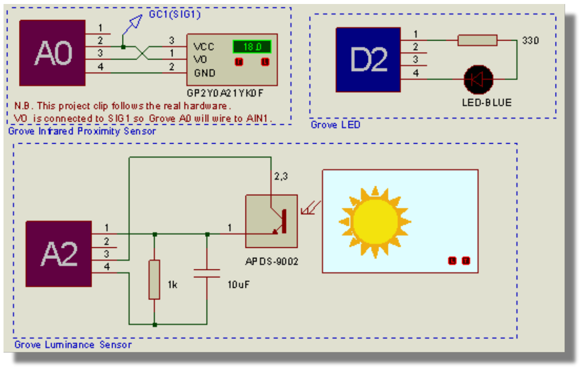
The <u>sockets</u> include connections through to two Arduino IO pins and there is a pin overlap. For example:

- Socket A0 can use pins A0 and A1
- Socket A1 can use pins A1 and A2
- Socket A2 can use pins A2 and A3.
- etc.

Digital sockets are exactly the same.



Our configuration for this project is therefore proximity sensor in socket A0 and luminance sensor in socket A2. The LED can be in any digital socket we like.

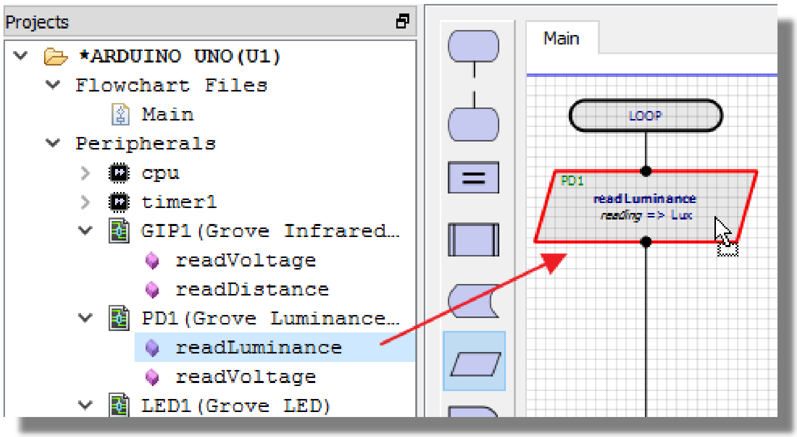

A0
GC1(SIG1)
VCC
VO
GND
GP2Y0A21YK0F
N.B. This project clip follows the real hardware.
VO is connected to SIG1 so Grove A0 will wire to AIN1.
Grove Infrared Proximity Sensor
D2
330
LED-BLUE
Grove LED
A2
2,3
APDS-9002
1k
10uF
Grove Luminance Sensor

Designing the Program

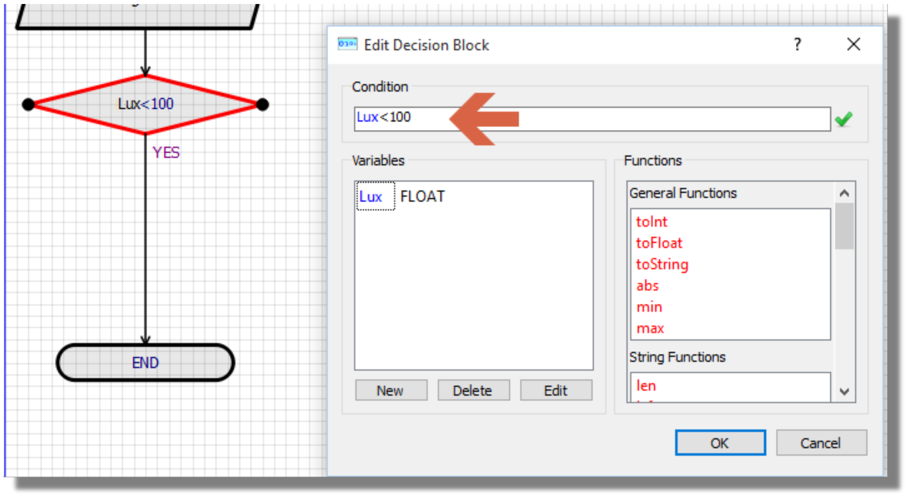
In our nightlight mock up we are looking for our LED to light when the following are true:

- It's is dark.
- Someone is nearby.

These are both decisions on the flowchart. Start by drag and drop the readLuminance() method onto the loop routine of the chart.



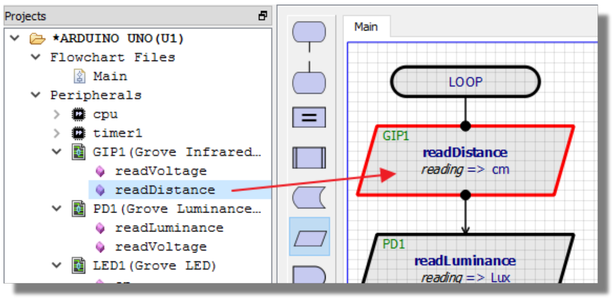
This sensor returns a lux value between 0 and 1000, with 0 being pitch black and 1000 max light. Let's then use our result in a decision block and set our initial test value to around 100.



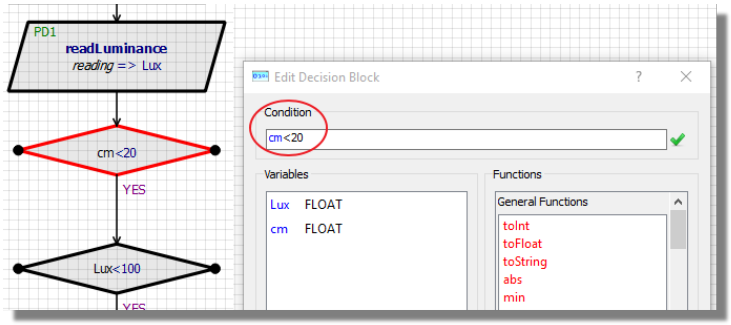
The datasheet for the **APDS-9002** photo sensor used in this Grove module provides data on lux values.

See Also: https://www.seeedstudio.com/Grove-Luminance-Sensor-p-1941.html

For the proximity sensor, we want to read a distance so drag and drop into the top of the loop.

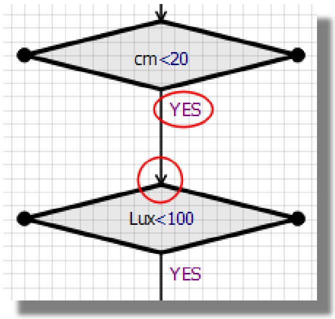


The proximity sensor can detect up to 80cm but we'll set our initial test to 20cm.

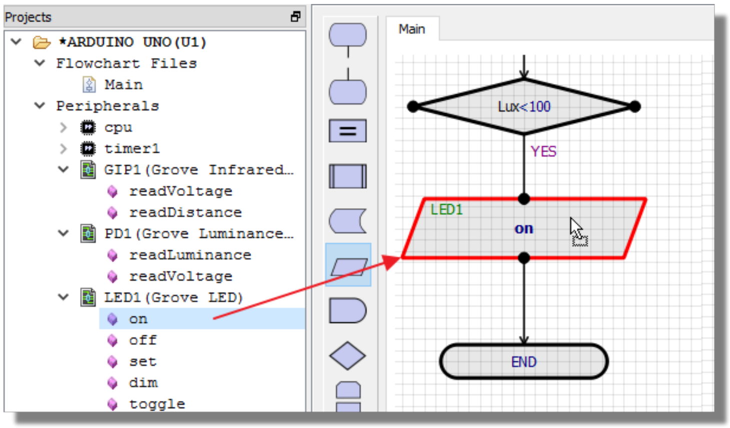


See Also: http://wiki.seeedstudio.com/Grove-80cm_Infrared_Proximity_Sensor/

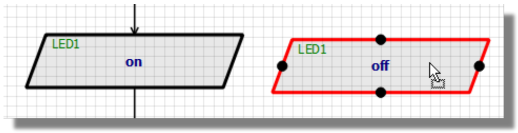
Since, we need both conditions to be true for the LED to turn on, make sure that the second decision is on the YES branch of the first decision.



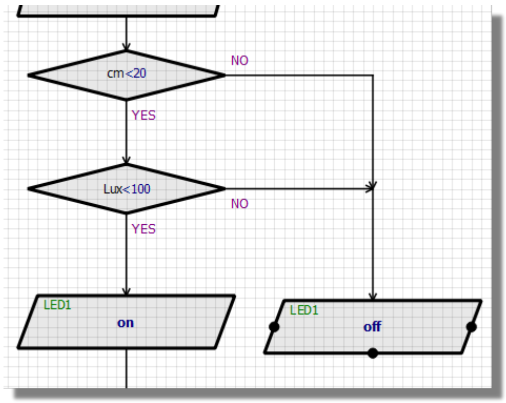
Next, drag and drop the led on method at the bottom of the YES branch.



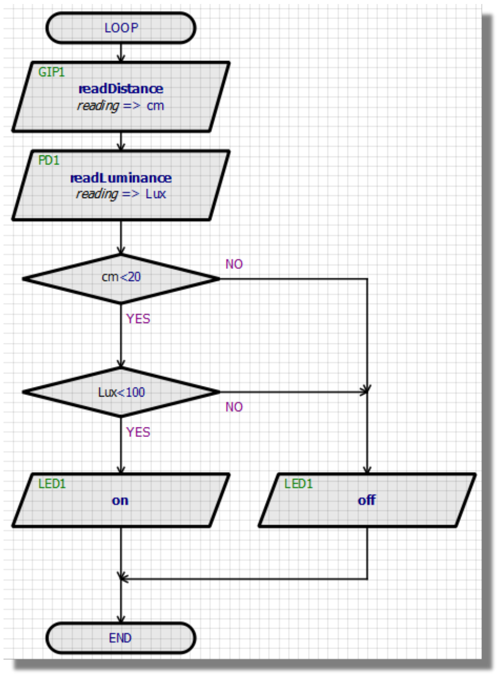
Finally, place the LED off method to the right of the main loop beside the LED on method.



Since we want the LED to turn off if either condition is false we can wire both NO branches from the decisions to this method.

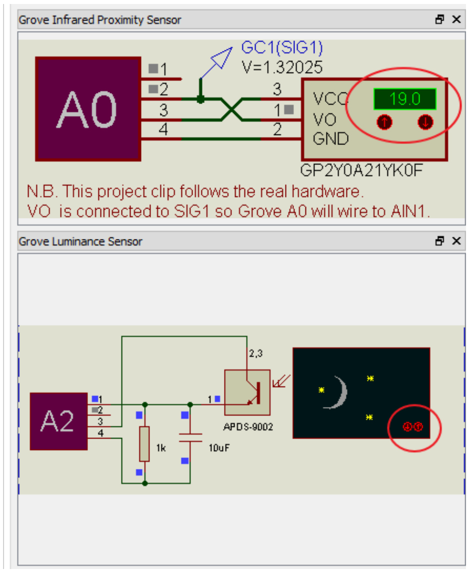


Finally, connect the bottom of the NO program branch back into the main program loop.



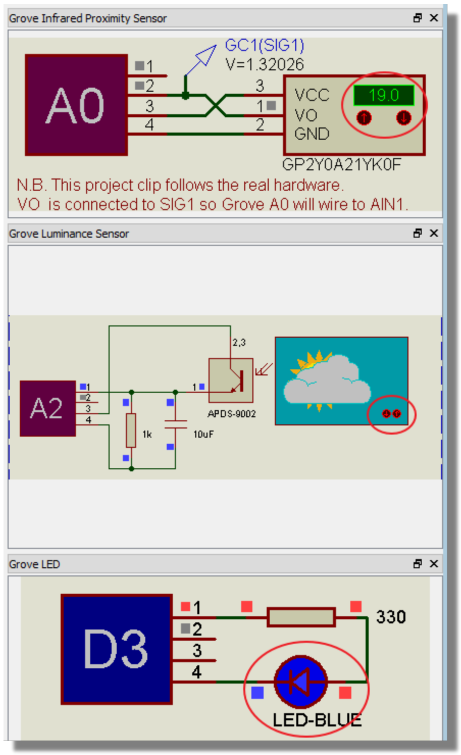
Simulating and Testing

To test our program, all we need to do is press the play button and then adjust the distance display on the proximity sensor and the lux display on the luminance sensor.



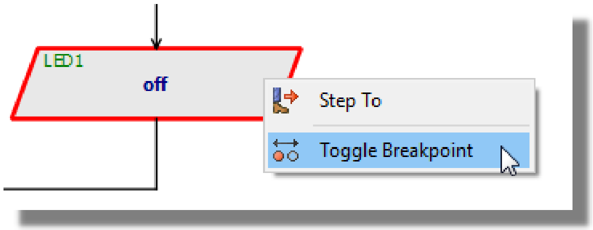
N.B. This project clip follows the real hardware.
VO is connected to SIG1 so Grove A0 will wire to AIN1.

The LED should turn on when the distance value is less than 20 and it's a little cloudy.

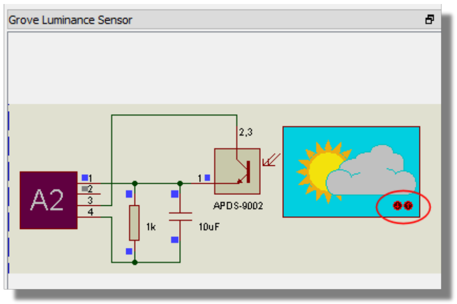


N.B. This project clip follows the real hardware.
VO is connected to SIG1 so Grove A0 will wire to AIN1.

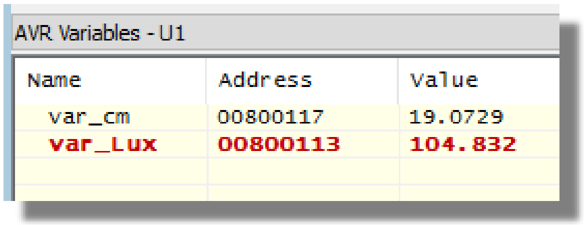
Pause the simulation and set a breakpoint on the LED off method.



Press play again to run the simulation and change either the distance or the light until the breakpoint triggers



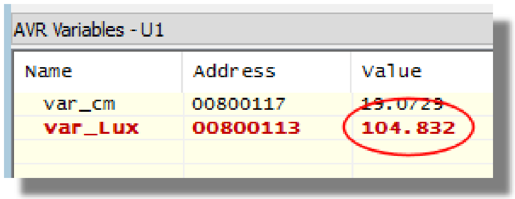
You'll notice that both our variables are available in the variables display.



AVR Variables - U1

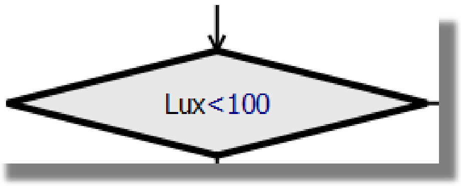
Name	Address	Value
var_cm	00800117	19.0729
var_Lux	00800113	104.832

Comparing these values to the tested decisions will tell us which condition has failed.

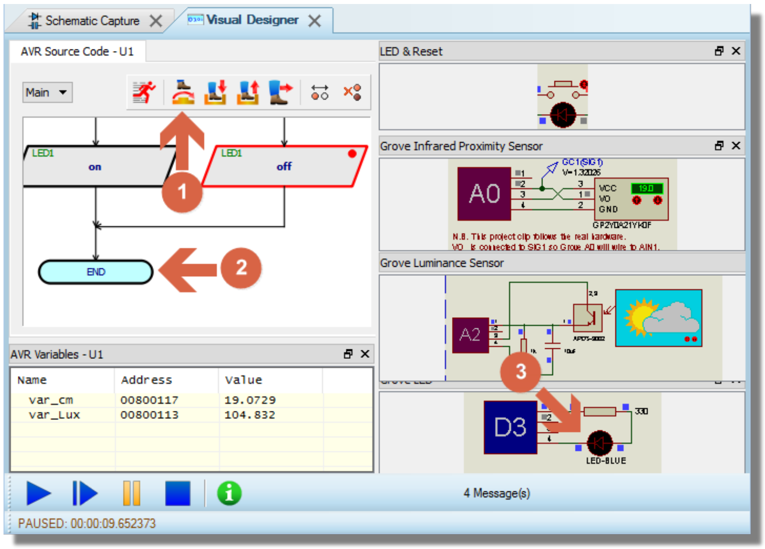


AVR Variables - U1

Name	Address	Value
var_cm	00800117	19.0729
var_Lux	00800113	104.832



As you single step via the icons at the top you'll notice the LED turn off.



AVR Variables - U1

Name	Address	Value
var_cm	00800117	19.0729
var_Lux	00800113	104.832

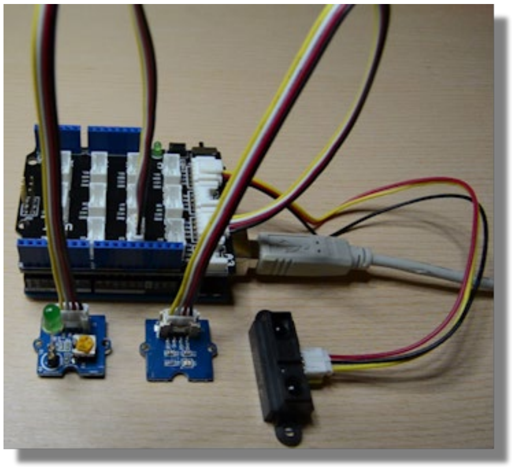
PAUSED: 00:00:09.652373

1, Single step once. 2, Notice the END block is highlighted. 3. LED has turned off

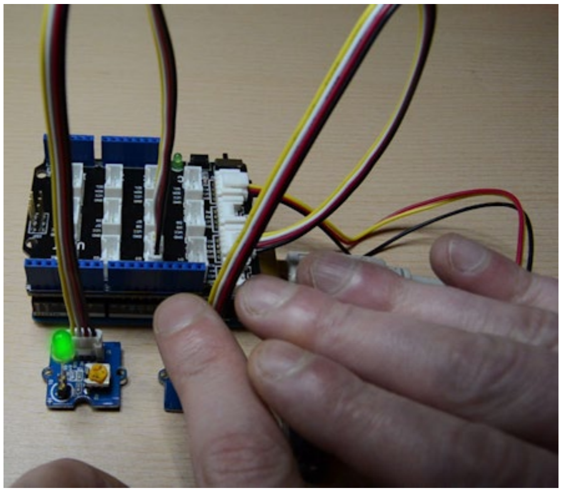
Programming

Since everything seems to working in good order, the remaining task is to plug in the Arduino and program the real hardware. The process here is:

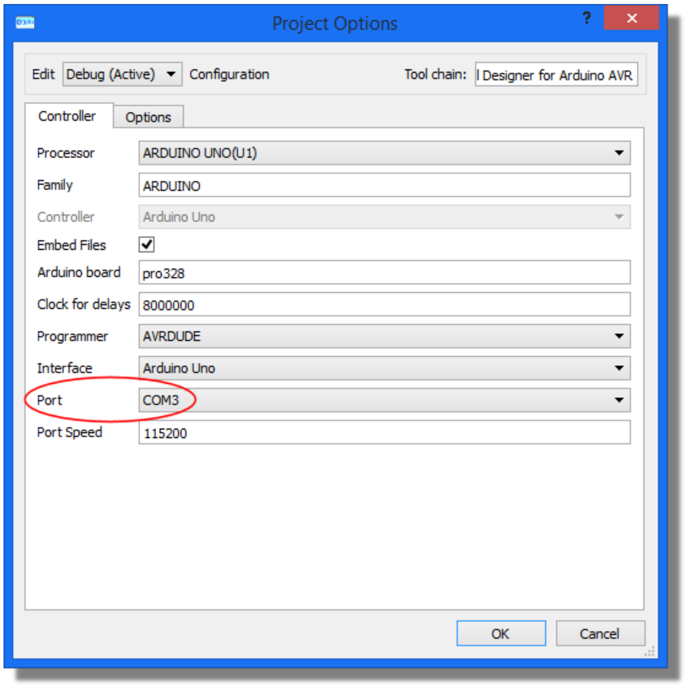
1) Plug the Grove sensors into the Grove baseboard in the same connectors as on the schematic. See also the golden rule discussed earlier.
2) Connect the baseboard to the Arduino and connect the Arduino to the PC. Until you have something like so:



3) Click the program upload button in Visual Designer.
4) Test in hardware.



If the programming fails, click on the project settings icon (next to the icon shown above) to make sure the COM port and settings are correct.



Debug (Active)
Tool chain: Designer for Arduino AVR

Processor	ARDUINO UNO(U1)
Family	ARDUINO
Controller	Arduino Uno
Embed Files	✓
Arduino board	pro328
Clock for delays	8000000
Programmer	AVRDUDE
Interface	Arduino Uno
Port	COM3
Port Speed	115200

TUTORIAL 3: DATA STORAGE

Introduction

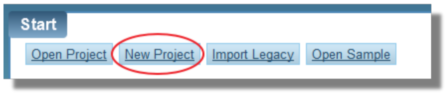
This tutorial introduces data resources and the storage model. We'll be creating a program to display a bitmap on an SD card onto the TFT display, although the same principles would hold true for audio (.WAV) files on the Wave Shield.

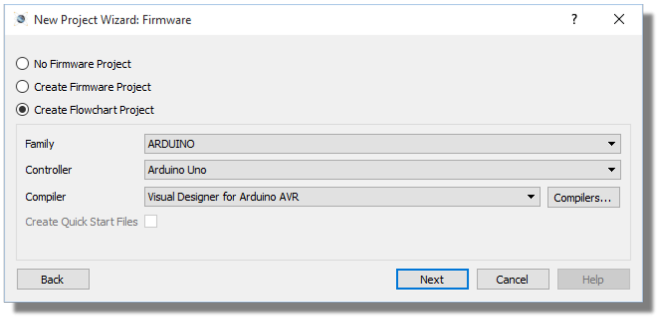
It is assumed as this stage that you have worked through the other tutorials and are familiar with general placement and connecting of flowblocks.

You can watch a short movie of this tutorial on the Labcenter Youtube Channel.

Setup

We'll set up this project in the usual way via the new project wizard and configure it as an Arduino Uno flowchart project.

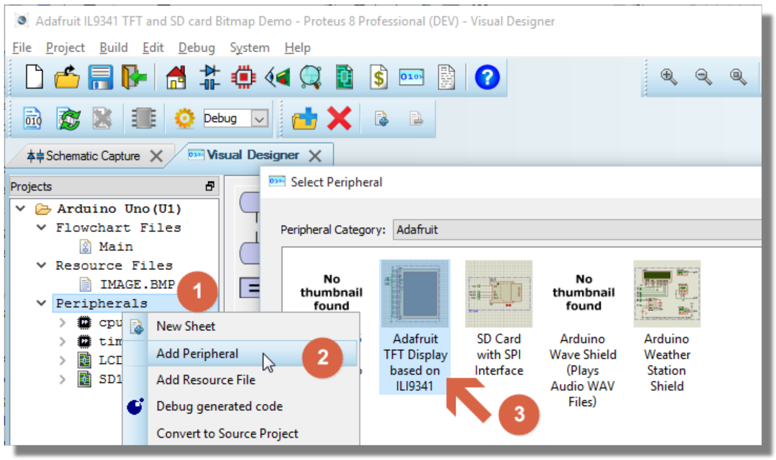




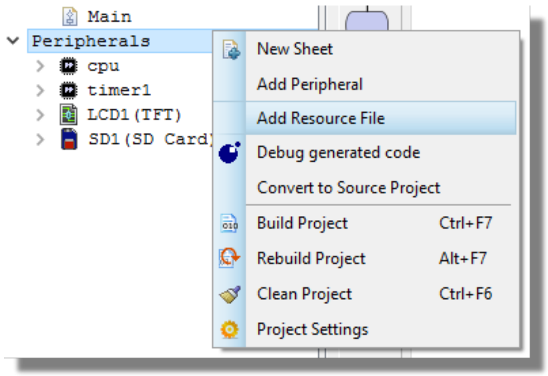
No Firmware Project
Create Firmware Project
(●) Create Flowchart Project

Family: ARDUINO
Controller: Arduino Uno
Compiler: Visual Designer for Arduino AVR
Create Quick Start Files ☐

Next, we'll want to add our peripheral shield. We right click on the Project Tree and add the Adafruit TFT shield. Note that this is not a module for the Grove Shield but a completely different shield which contains both an SD card and a TFT display.


Adafruit IL9341 TFT and SD card Bitmap Demo - Proteus 8 Professional (DEV) - Visual Designer
File
Project
Build
Edit
Debug
System
Help
Debug
Schematic Capture
Visual Designer
Projects
Arduino Uno (U1)
Flowchart Files
Main
Resource Files
IMAGE.BMP
Peripherals
cpu
tim
LCD
SD1
1
New Sheet
Add Peripheral
Add Resource File
Debug generated code
Convert to Source Project
2
Select Peripheral
Peripheral Category: Adafruit
No thumbnail found
Adafruit TFT Display based on ILI9341
SD Card with SPI Interface
No thumbnail found
Arduino Wave Shield (Plays Audio WAV Files)
Arduino Weather Station Shield
3

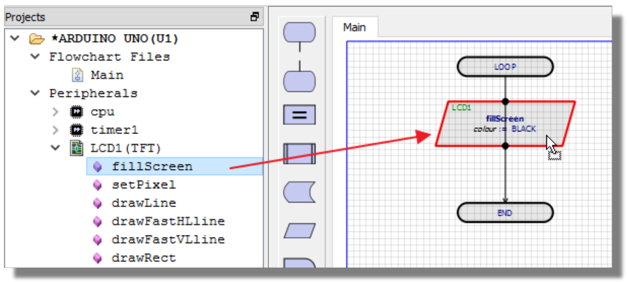
The last thing we need to do is add the resource. In our case this is the picture that we want to store in the SD card and display on the TFT display. We do this via the Add Resource command on the right click context menu.

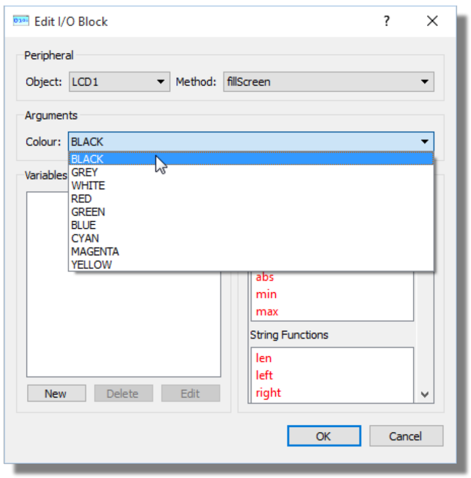


⚠ Be careful with the naming of resource files. The Arduino SD stack by default supports only 8.3 filenames in order to support FAT16 cards. This means that the namestem must be 8 characters or less and the extension must be 3 characters or less. For example, a picture called ProteusLogo.bmp will not load whereas renaming it to logo.bmp or image.bmp will work fine.

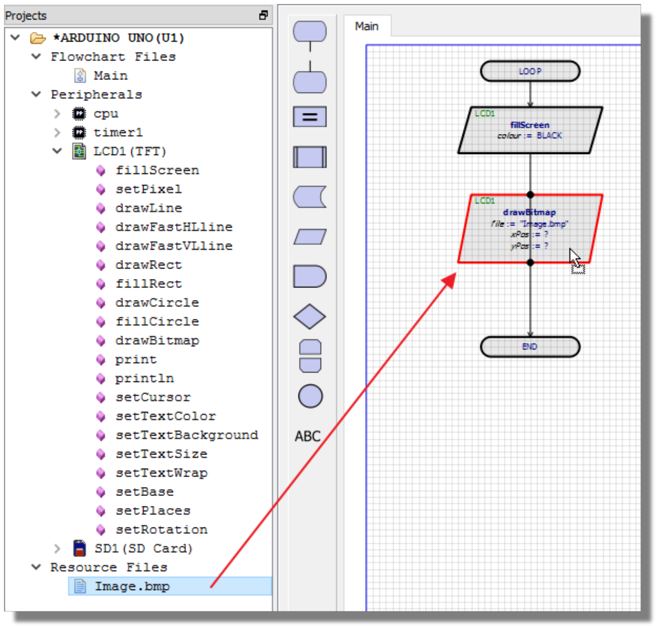
Designing the Program

Now that we have all the pieces we need, the remaining job is to create the program. This could hardly be easier in Visual Designer. First, drag the fillSreen() method onto the top of the loop and set the fill colour to black.



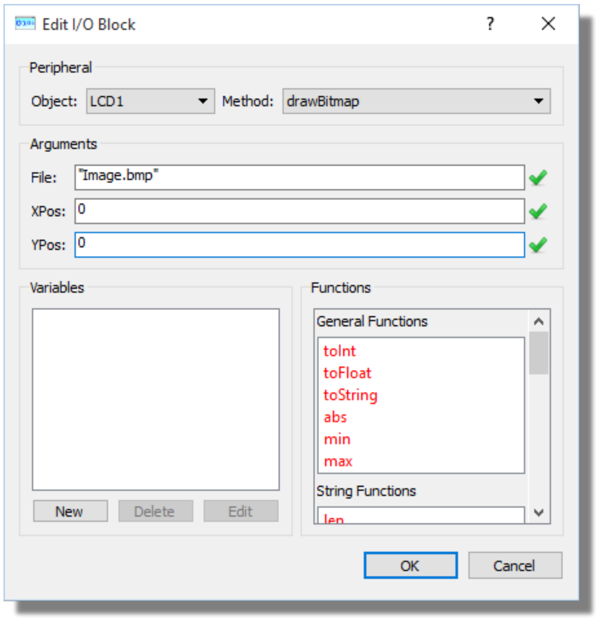


Next drag and drop the bitmap resource into the loop routine. Visual Designer knows that the only sensible target for the resource is the TFT display and it knows the method for rendering the resource is drawBitmap() and so it will automatically configure the flowblock for you.



Projects
- *ARDUINO UNO(U1)
  - Flowchart Files
    - Main
  - Peripherals
    - cpu
    - timer1
    - LCD1(TFT)
      - fillScreen
      - setPixel
      - drawLine
      - drawFastHLline
      - drawFastVLline
      - drawRect
      - fillRect
      - drawCircle
      - fillCircle
      - drawBitmap
      - print
      - println
      - setCursor
      - setTextColor
      - setTextBackground
      - setTextSize
      - setTextWrap
      - setBase
      - setPlaces
      - setRotation
    - SD1(SD Card)
  - Resource Files
    - Image.bmp

Now you need to edit the Image and set the xPos and yPos to be 0



Edit I/O Block

Peripheral

Object: LCD1 Method: drawBitmap

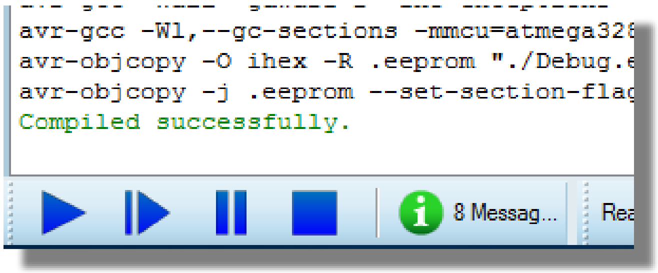
Arguments

File: "Image.bmp"

XPos: 0

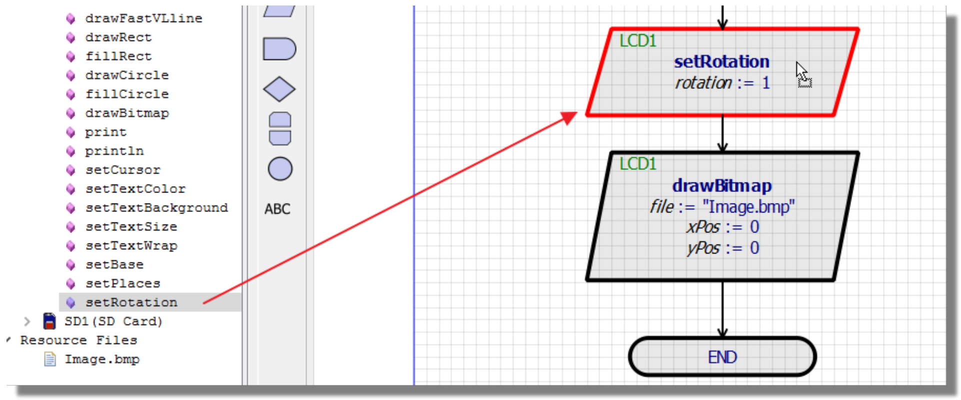
YPos: 0

Press the play button to compile and run the simulation.



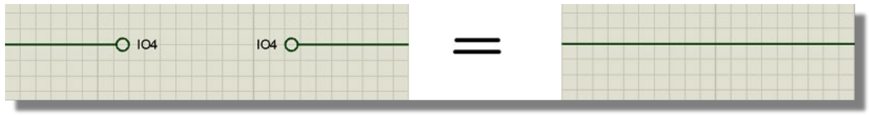
```
avr-gcc -Wl,--gc-sections -mmcu=atmega32
avr-objcopy -O ihex -R .eeprom "./Debug.
avr-objcopy -j .eeprom --set-section-fla
Compiled successfully.
```

If necessary, you can configure the rotation in the setup routine via the same drag and drop method.



Taking Measurements

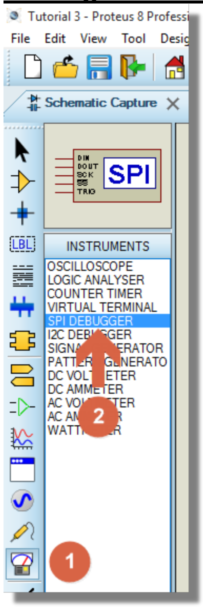
One of the nice things about Visual Designer is that it contains the Proteus schematic. This means that, if you want to, you can look a little deeper at what is happening in your design. For example, in our example the SD card is connected to the processor on the SPI bus and so we can place and wire a protocol analyzer to examine the SPI packets. In Proteus, all signals and waveforms travel through the wires, and two terminals with the same name have an 'invisible wire' between them.



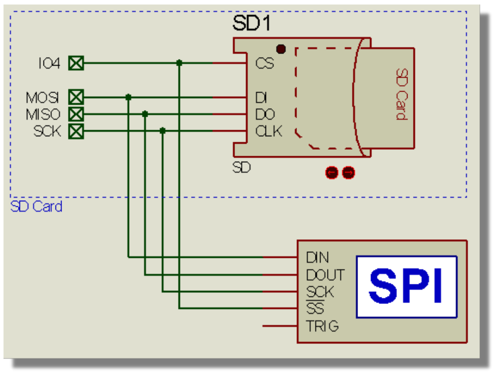
Two terminals with the same name are connected together as though they were wired.

Given this, the easiest way to connect the SPI terminal is to place one and wire it to terminals with the same names as the other SPI lines. The alternative is to connect up to existing wires.

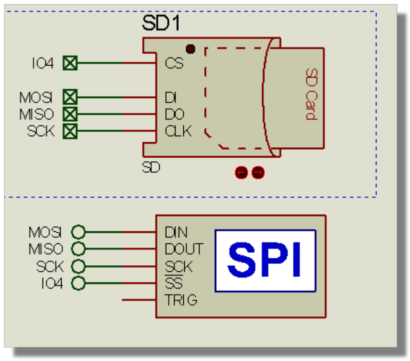
Placing a SPI Analyser:



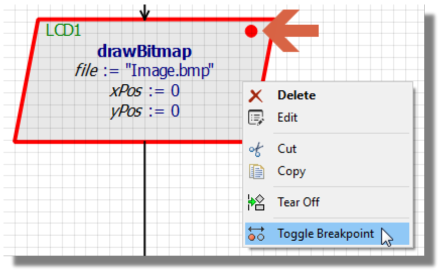
Wiring SPI Analyser:



OR



Next, set a breakpoint in the drawBitmap() command as this is where the majority of the SPI transmissions happen.



Now, when we run the simulation we'll see some initialisation chatter on the SPI bus until we hit our breakpoint. When we single step, we will see an awful lot of data being moved from the SD card corresponding to the bitmap being read by the processor. You can drill all the way down to bit level on any packet if need be.

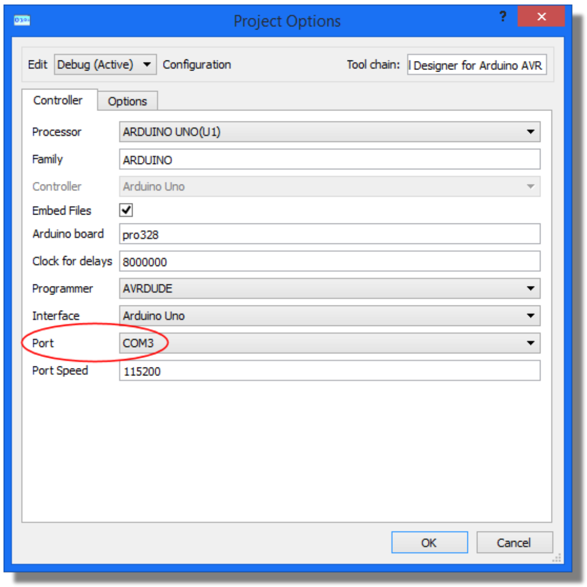
The same basic process will work with any instrument (Oscilloscope, Logic Analyser, I2C Analyser etc.).

Attaching instrumentation is essential for analysis but will slow down simulation. It's worth deleting instruments when a debug session is complete.

Programming Hardware

When we are finished development we can easily program the physical hardware. First, connect the Adafruit TFT shield to the Arduino Uno and make sure a compatible SD card is inserted in the shield. Then, connect to the PC and then simply press the program button.

If the upload doesn't start, use the settings dialogue to specify the COM port that you have connected to and try again.


Project Options

Edit: Debug (Active) Configuration Tool chain: Designer for Arduino AVR

Controller | Options

Processor	ARDUINO UNO(U1)
Family	ARDUINO
Controller	Arduino Uno
Embed Files	✓
Arduino board	pro328
Clock for delays	8000000
Programmer	AVRDUDE
Interface	Arduino Uno
Port	COM3
Port Speed	115200

You should see an upload complete message in the simulation log and your TFT should be displaying the picture.

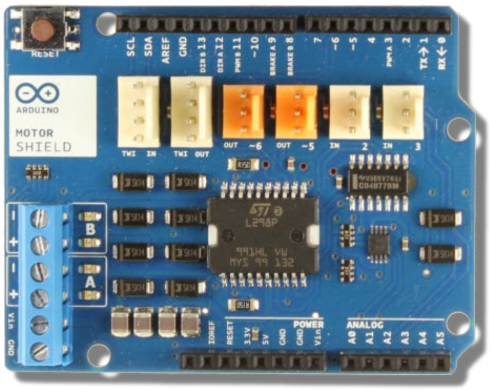
This programming process is twofold. First, the resource is written to the SD card and then the firmware program is sent to the AVR microcontroller.

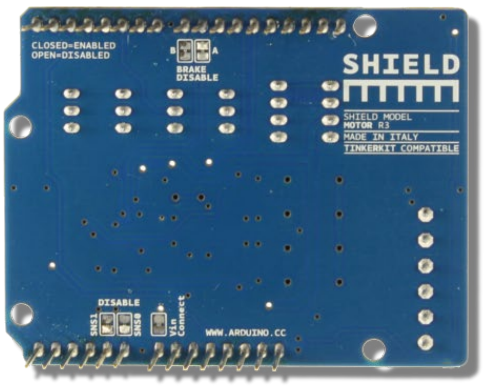
They physical SD Card must be FAT16 or FAT32 and big enough to hold the resource file that will be programmed onto it.

TUTORIAL 4 : MOTOR CONTROL

Introduction

This tutorial shows how you can easily control both DC motors and Stepper motors with Visual Designer. We'll do this using the Arduino Motor Shield V1 R3.


Front View

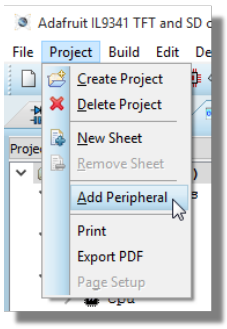

Rear View

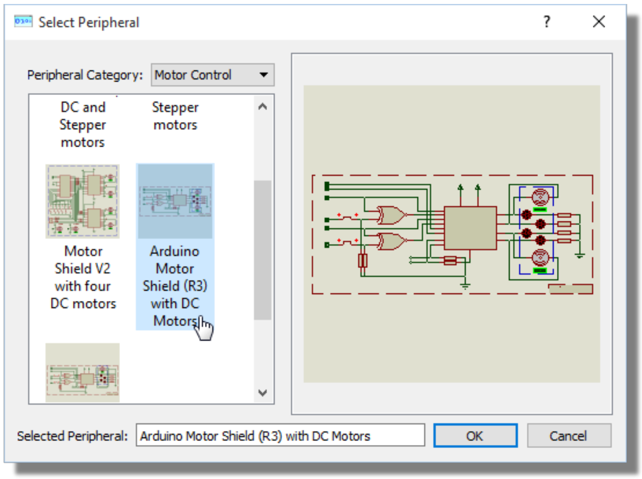
See Also: https://www.arduino.cc/en/Main/ArduinoMotorShieldR3

You can experiment with Servo motors using the Grove Servo peripheral and you can also use BLDC motors by drawing on the schematic.

DC Motor Control

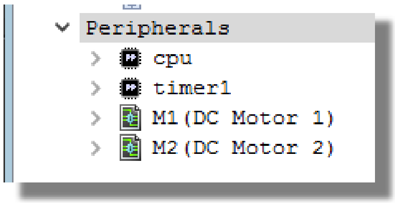
Start by creating a new Visual Designer Project and then Add the Arduino Shield with DC Motors in the normal way.



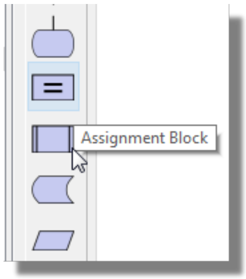

Select Peripheral
Peripheral Category:
Motor Control
DC and Stepper motors
Stepper motors
Motor Shield V2 with four DC motors
Arduino Motor Shield (R3) with DC Motors
Selected Peripheral:
Arduino Motor Shield (R3) with DC Motors
OK
Cancel

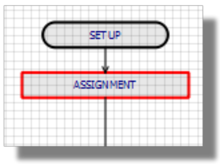
Note that under the Motor Control heading there are actually two shields. These are identical except that the first has the board configured and populated with two DC motors and the second is configured and populated with one stepper motor. In the real hardware of course you would need to configure and populate the shield manually according to which project you were programming the Arduino with.

You should notice that in the Project Tree you now have two motors with associated methods.

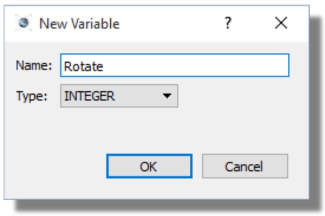


Let's write a small program to drive the motors in opposite directions. The first thing we want to do is initialise a variable for the speed of the motors so drag and drop an assignment block into the Setup routine (we only need to do this once).

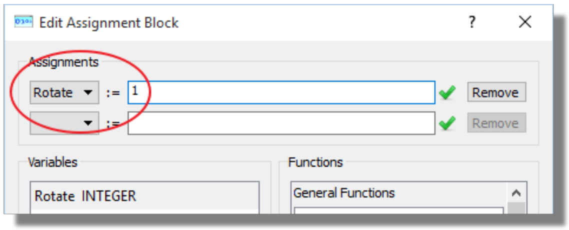




Edit the block, create a new variable and assign a value.



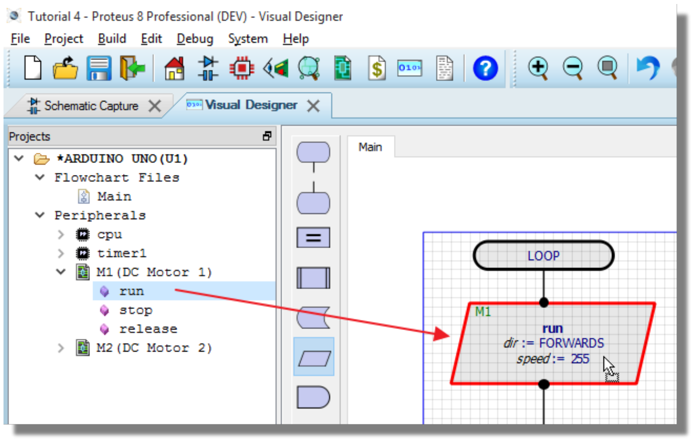
Name: Rotate
Type: INTEGER



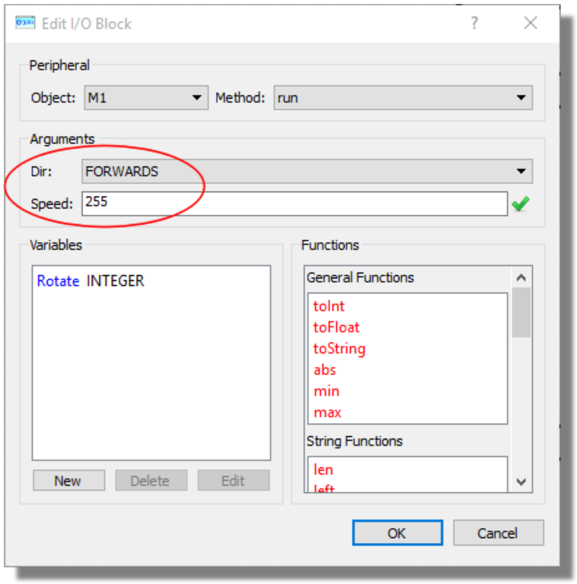
Rotate := 1

Rotate INTEGER

Now, let's drive the first motor forwards. Drag and drop the run methods into the loop routine.

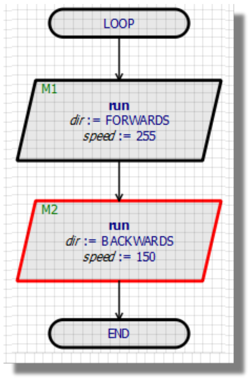


Edit the block, set the direction forwards and assign the speed to our variable.

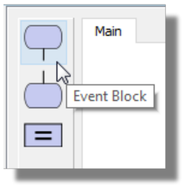


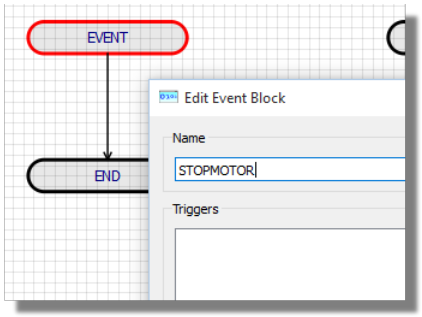
Object: M1
Method: run
Dir: FORWARDS
Speed: 255
Rotate INTEGER

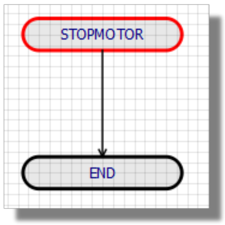
Repeat the process for the second motor to send it spinning backwards.



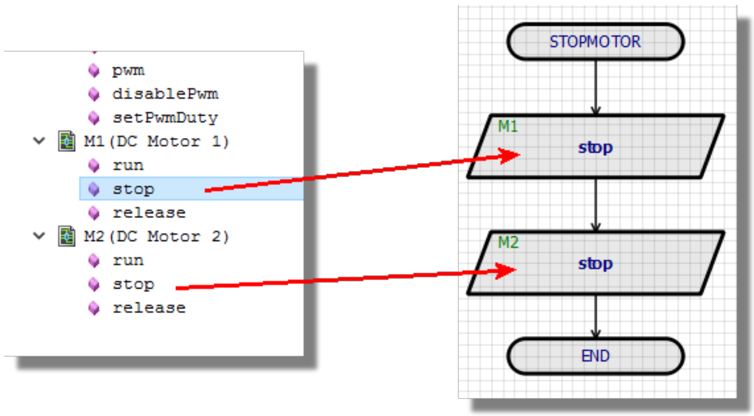
Next, we'll add a sub-routine to stop the motors. Drag and drop an Event block into an empty place, edit the Event block and give it a sensible name.



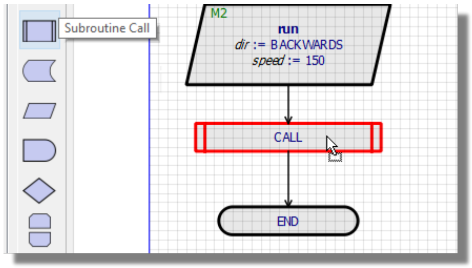


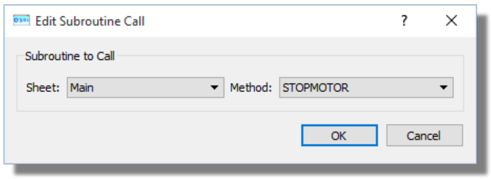


Now, drag and drop the stop routine for both motors into the sub-routine we've just added.



Finally, drag and drop a subroutine call block into the main loop and edit it to call our routine.



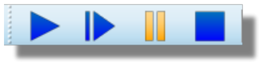

Edit Subroutine Call

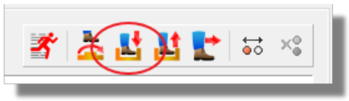
Subroutine to Call

Sheet: Main Method: STOPMOTOR

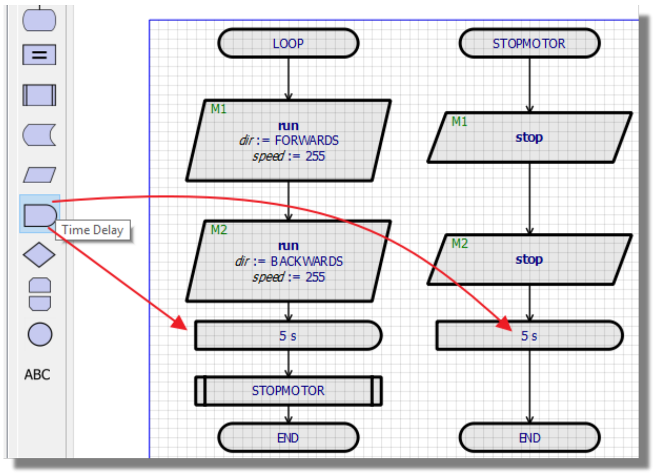
OK Cancel

To see this in action press the pause button and then single step the code using the Step-Into icon at the top of the source window. You'll notice that the code executes through the sub-routine and then returns to the main loop.

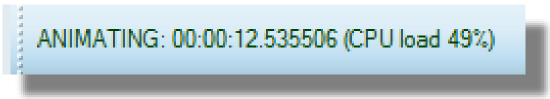




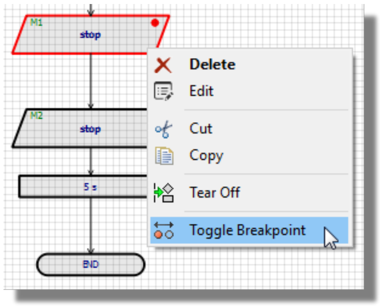
We have most of the pieces of our program in place but we are starting and stopping the motors very quickly which isn't going to help them build momentum. Much better to add a delay in the drive phase and a similar time delay in the stop phase. Again, this is drag and drop from the flowblock menu onto the chart routines.



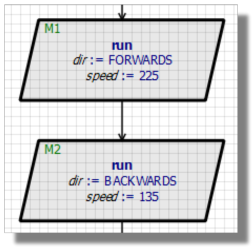
If you run the simulation now you can monitor the drive and spin down loops via the time display on the status bar

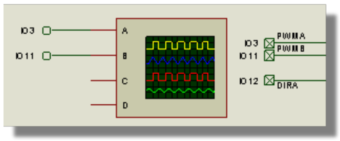

ANIMATING: 00:00:12.535506 (CPU load 49%)

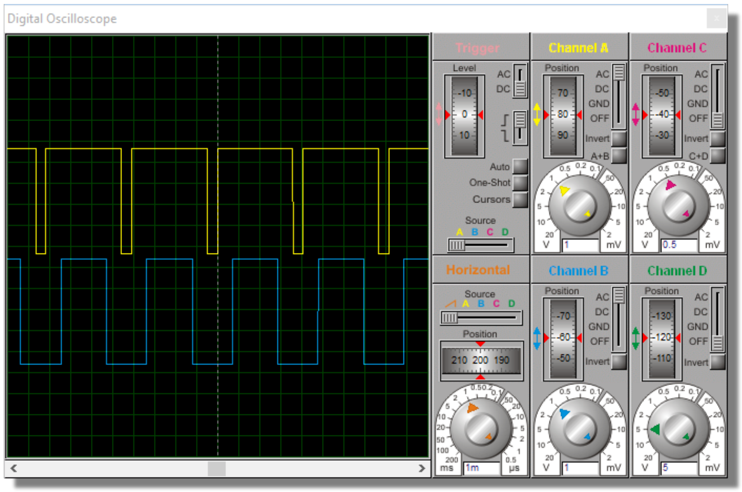
You could also pause the simulation, set a breakpoint and step the code. Unlike the real hardware the motors won't lose momentum when you are at a breakpoint - a real advantage to debugging in software.



For more advanced analysis you could place and wire an oscilloscope on the schematic and watch the PWM waveforms drive the motor.



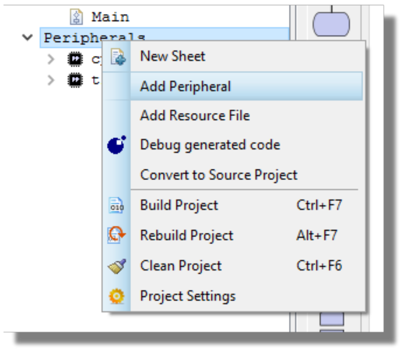


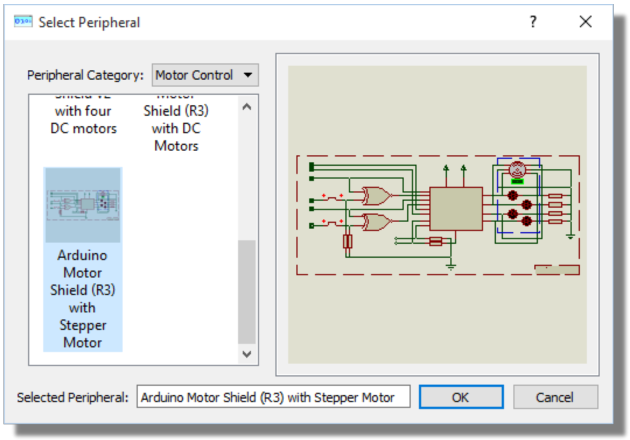


As always, you can program the physical board at any time via the upload button.

Stepper Motor Control

For the stepper motor we are using the same hardware shield (Arduino Motor Shield V1 R3) but configured for a single unipolar stepper motor. Start a new Visual Designer project and bring in the virtual shield via the Add Peripheral command.

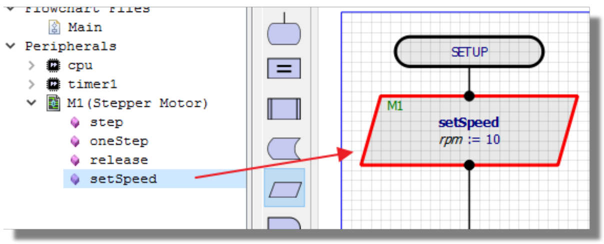


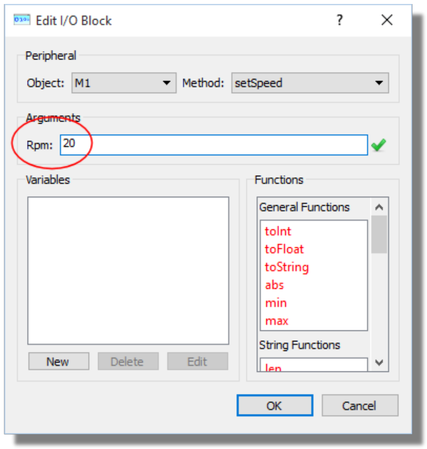

Select Peripheral
with four DC motors
Shield (R3) with DC Motors
Arduino Motor Shield (R3) with Stepper Motor
OK
Cancel

Peripheral Category: Motor Control

Selected Peripheral: Arduino Motor Shield (R3) with Stepper Motor

We'll design a small program to step the motor in one direction and then reverse direction and step backwards. Our first job is to define the speed which we can do by dragging the setSpeed() method into the Setup routine.

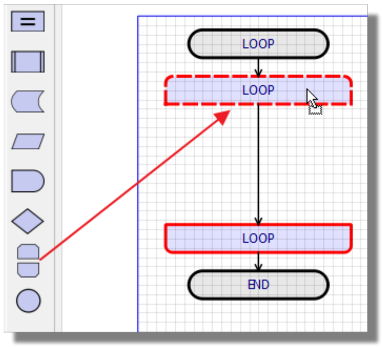


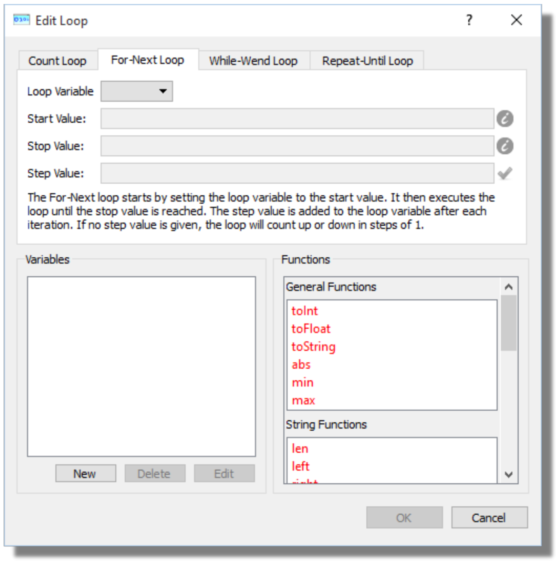


Object: M1 Method: setSpeed

Rpm: 20

In order to step a set number of times we want to use a loop construct. Drag and drop one onto the **Main (Loop)** routine and edit the block.





Loop Variable

Start Value:

Stop Value:

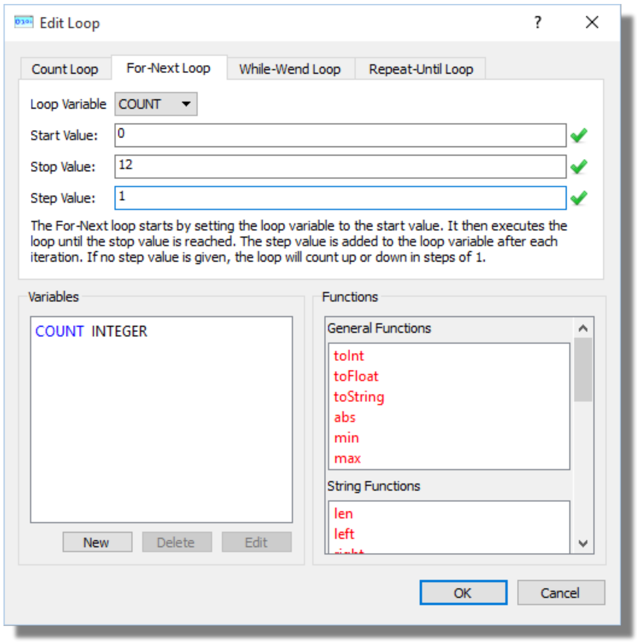
Step Value:

The For-Next loop starts by setting the loop variable to the start value. It then executes the loop until the stop value is reached. The step value is added to the loop variable after each iteration. If no step value is given, the loop will count up or down in steps of 1.

General Functions: toInt, toFloat, toString, abs, min, max

String Functions: len, left

We'll use a For-Next loop here so add a variable for count, initialise to zero and let's go round the loop 12 times.



Loop Variable: COUNT

Start Value: 0

Stop Value: 12

Step Value: 1

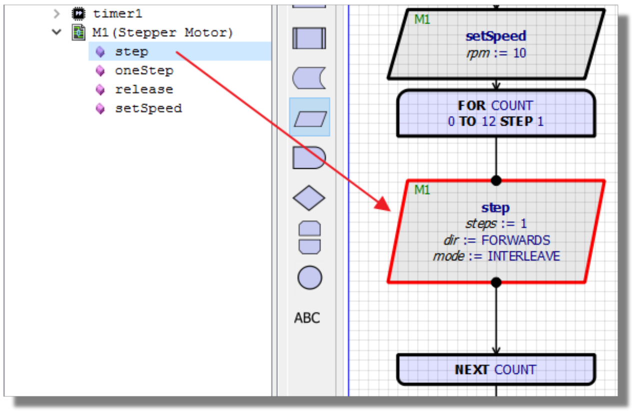
The For-Next loop starts by setting the loop variable to the start value. It then executes the loop until the stop value is reached. The step value is added to the loop variable after each iteration. If no step value is given, the loop will count up or down in steps of 1.

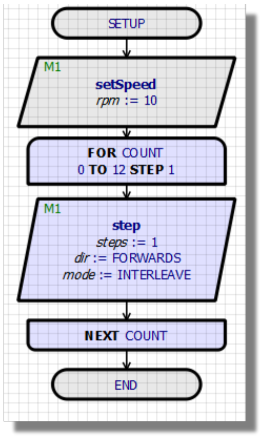
COUNT INTEGER

General Functions: toInt, toFloat, toString, abs, min, max

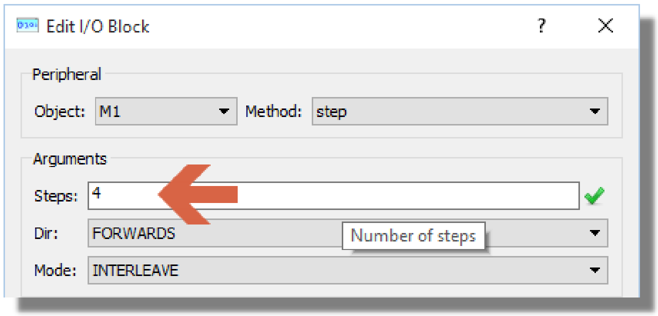
String Functions: len, left

Now, drag the step() method of the motor inside our loop - you'll notice the colour change to indicate it's part of the loop construct.





Edit the flowblock and make the number of steps 4 so that we get sensible movement, set the direction to forwards and the mode to interleave.


Edit I/O Block

Peripheral

Object: M1 Method: step

Arguments

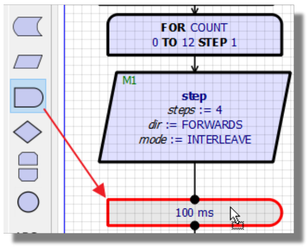
Steps: 4

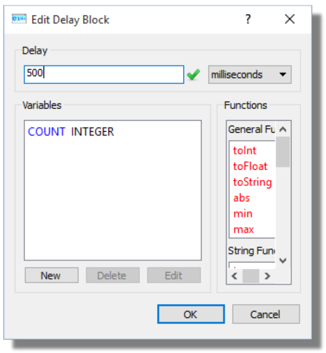
Dir: FORWARDS

Mode: INTERLEAVE

Single Mode means single-coil activation, Double mode means 2 coils are activated at once (for higher torque) and "interleave" mode means that it alternates between single and double to get twice the resolution (but of course its half the speed). Lots more info on this in various internet resources.

Add a modest delay after the step routine to allow the drive phase to complete before we iterate the loop.




Edit Delay Block

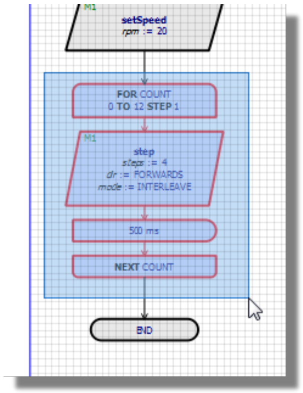
Delay

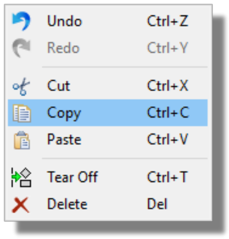
500 milliseconds

Variables

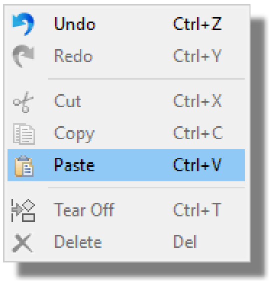
COUNT INTEGER

That's the forwards motion in place. Driving the motor in the opposite direction is identical except that we change the direction in the step routine. You can place a second loop and drag methods in or we can just use the trusty copy and paste. Drag a box with the **right** mouse around the loop and select copy from the resulting context menu.

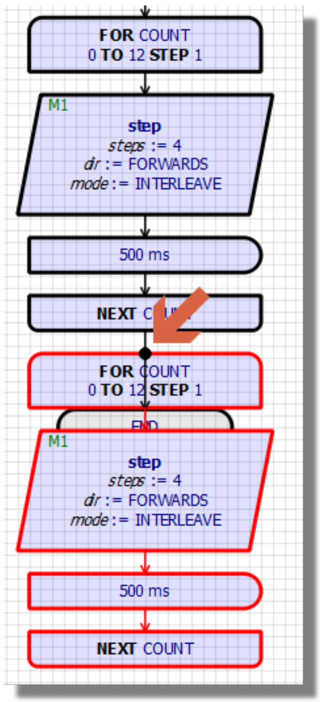




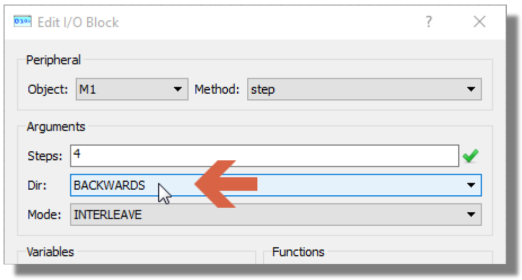
Next, right click (or CTRL+V) to paste a second set of blocks.



Drag them into place underneath the first loop, releasing the mouse when the top node appears with a dot.



Finally, edit the step method and change the direction.


Edit I/O Block

Peripheral

Object: M1 Method: step

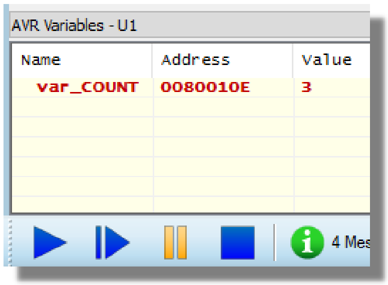
Arguments

Steps: 4

Dir: BACKWARDS

Mode: INTERLEAVE

That's it. Press play and watch the motor step forwards and backwards or pause, set a breakpoint and single-step your program. Note that the counter variable is displayed in the variables window which will tell you instantly how many more iterations of your loop remain.


AVR Variables - U1

Name	Address	Value
var_COUNT	0080010E	3

TUTORIAL 5: SCHEMATIC BREADBOARD

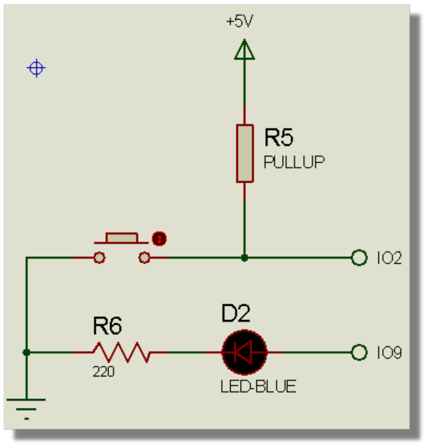
Introduction

So far in all of the tutorials we have used pre-existing Arduino shields or Grove modules for the hardware. There is however nothing to stop us designing our own hardware directly on the schematic. We will cover the basics in this tutorial with a button and an LED. Note however that there are literally thousands of parts that can be simulated in the Proteus libraries. Start by creating a new project with a schematic, no PCB and with Firmware for Arduino Uno using the Visual Designer Compiler for Arduino.

This tutorial is also covered by a short movie on the Labcenter Youtube channel.

Drawing on the Schematic

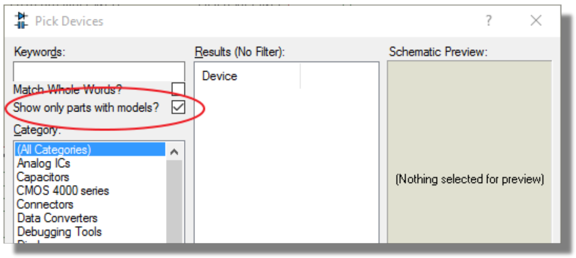
There is a good deal of documentation on schematic drawing in the main Proteus help files so we'll cover only in brief here. Our task is to draw the following circuitry on the schematic;



The three main steps are pick, place and wire.

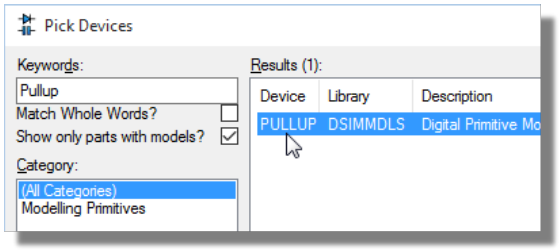
Picking

To pick from the library select the 'P' button above the parts bin on the schematic. This presents us with the library browser. We are only interested in those parts that we can simulate so start by checking the box to filter the results.

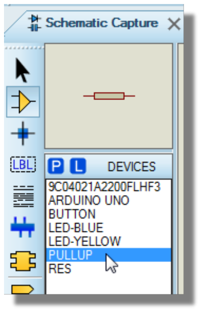


Show only parts with models? ✓

We will need an LED, a 220 Ohm resistor, a pullup resistor and a button, all of which we can find by typing keywords at the top left of the dialogue form.



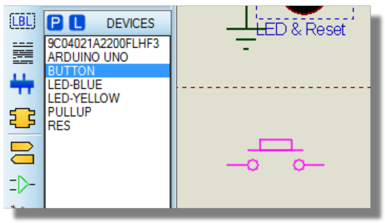
To pick a part into the parts bin, simply double click on the result.

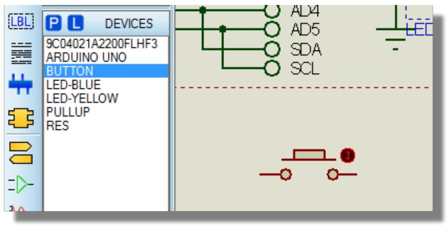


You may be asked if you want to replace the existing part. This will happen when you pick a part already in the parts bin; for example, there is already an LED in the parts bin because it's used in the Arduino Uno Shield.

Placing

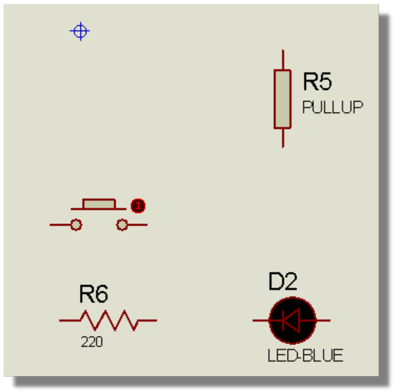
To place a part, select it in the parts bin, left click once on the schematic, then drag into place and left click again to drop.



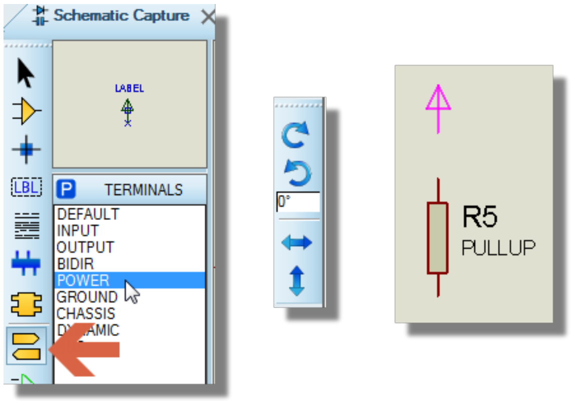


If you need to rotate, you can do this with the plus and minus keys on the keyboard during placement.

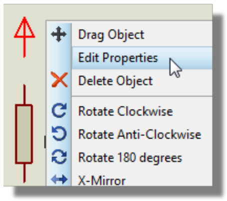
Place all of the parts approximately in position - something like the following:

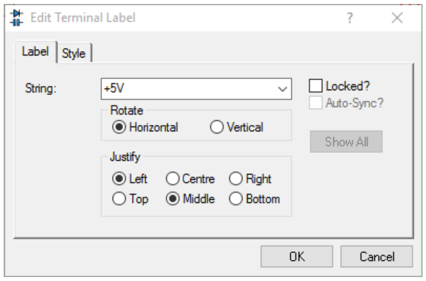


We can apply power and ground by selecting terminal mode and placing /rotating in the same way as we did the parts.



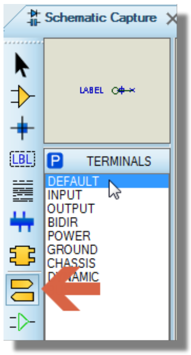
Edit the power terminal and specify the correct voltage.

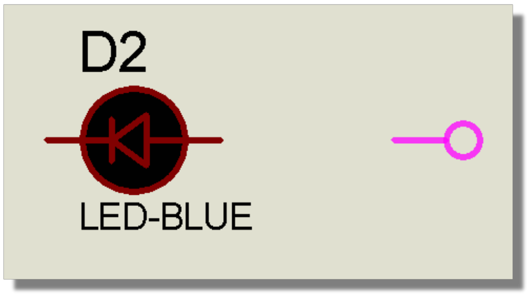




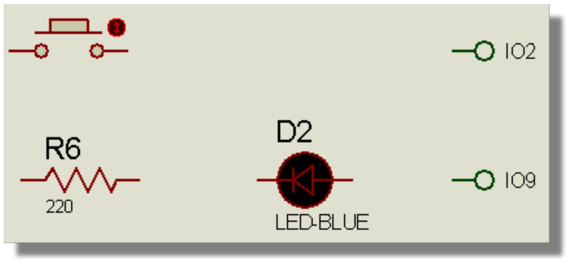
String: +5V
Locked? ☐
Auto-Sync? ☐
Rotate: [✓ Horizontal] Vertical
Justify: [✓ Left] Centre Right
Top [✓ Middle] Bottom

We'll also need a terminal to connect our button and our LED up to the correct pin on the Arduino. We can use default terminals for this.

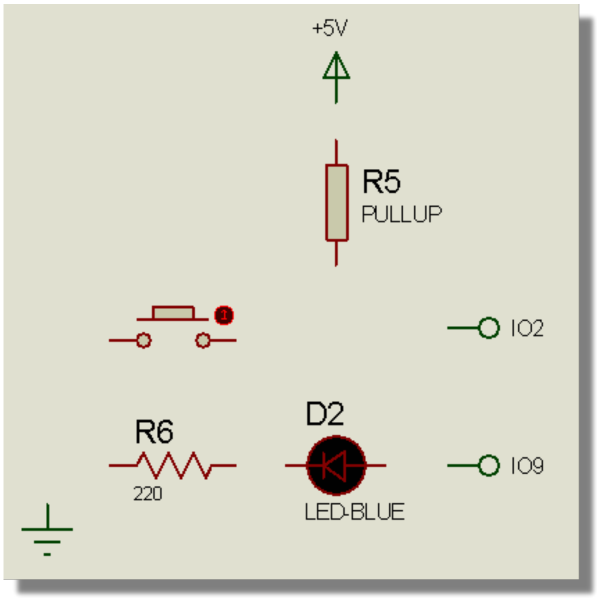




It is here that the hardware and the firmware designs start to merge. The best way to wait for a button press in firmware is via a pin change interrupt. This means however that we need to connect the button to a pin on the Arduino for which a pin change interrupt is available. On the Uno we can use IO2 for this and can connect the LED to any available IO pin.

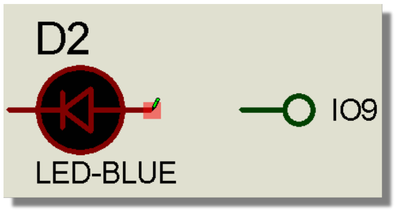


You want it to look like the one below (without the wires at this stage).

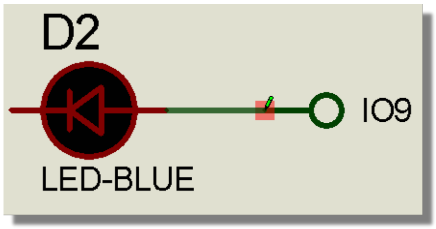


Wiring

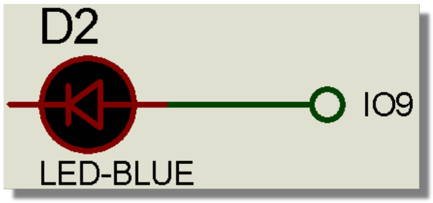
Wiring on the schematic is point to point. First, hover the mouse over a pin until the cursor turns green and then left click



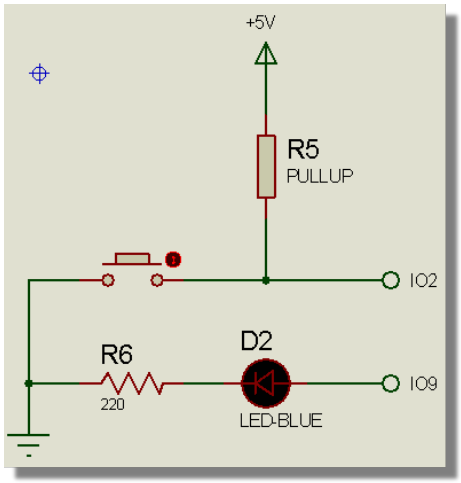
Next, move the mouse to the destination pin/wire until the cursor turns green again.



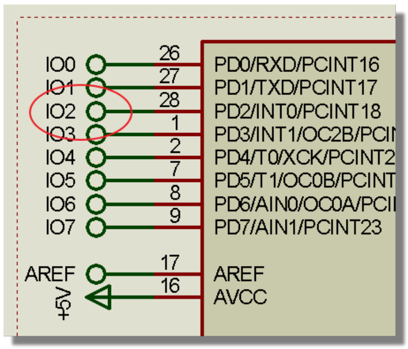
Left click the mouse again to complete the wire.



After wiring is complete your schematic should look like the following.

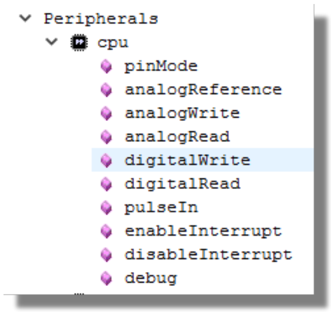


Remember the two terminals with the same name on the schematic have an 'invisible wire' between them so by wiring the button to a terminal called IO2 we have actually connected it to the IO2 pin on the Arduino chip.

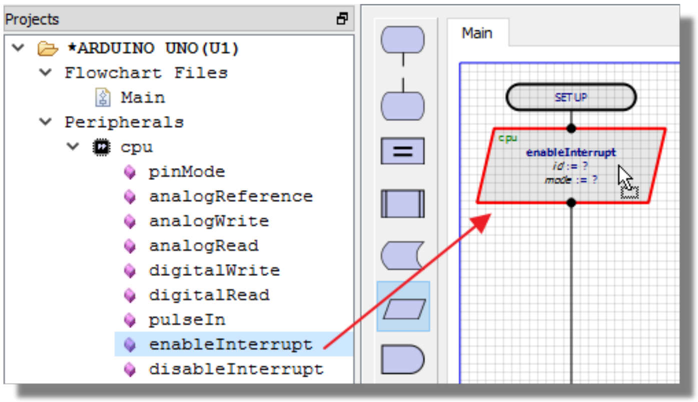


Designing our Program

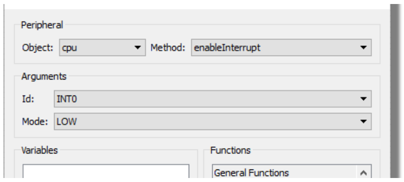
Everything we have done so far is hardware design. The program design itself takes place on the Visual Designer tab. Unlike our other tutorials we don't have any external peripheral methods in the project tree so we will have to drive the electronics directly using the CPU methods.



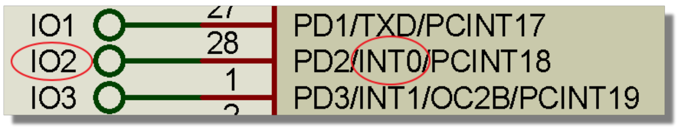
We'll start by dragging and dropping the pin change interrupt routine in to the Setup routine.



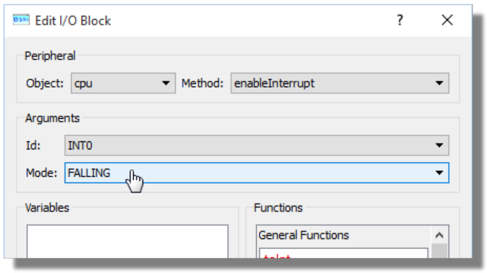
Now, there are two available interrupt pins on the Uno but we need to select INT0 because we wired the button onto IO2.



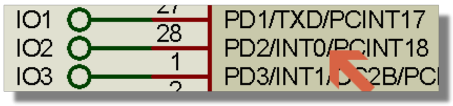
Peripheral
Object: cpu Method: enableInterrupt
Arguments
Id: INT0
Mode: LOW



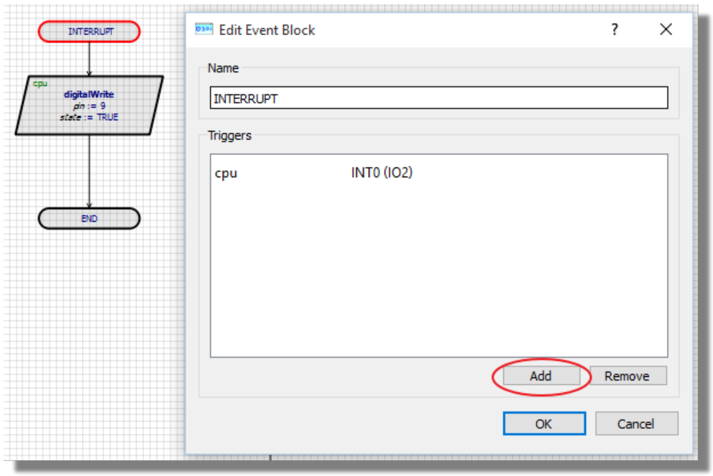
Similarly, we need to set the interrupt to trip on a falling edge because we wired the button to pull low when pressed.



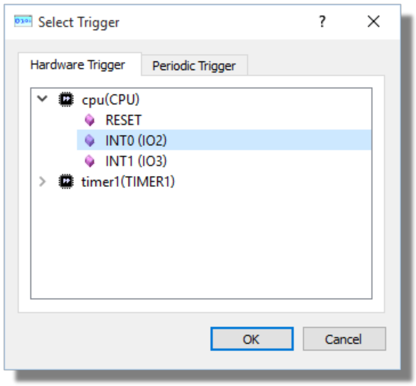
Edit I/O Block
Peripheral
Object: cpu Method: enableInterrupt
Arguments
Id: INT0
Mode: FALLING



Now we need to handle the interrupt event. Drag and drop an event block onto the Flowchart Editor and edit. Give a sensible name and then specify the trigger as INT0.

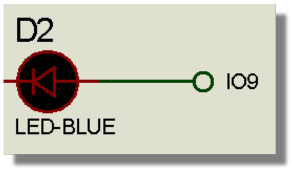


Edit Event Block
Name
INTERRUPT
Triggers
cpu INT0 (IO2)

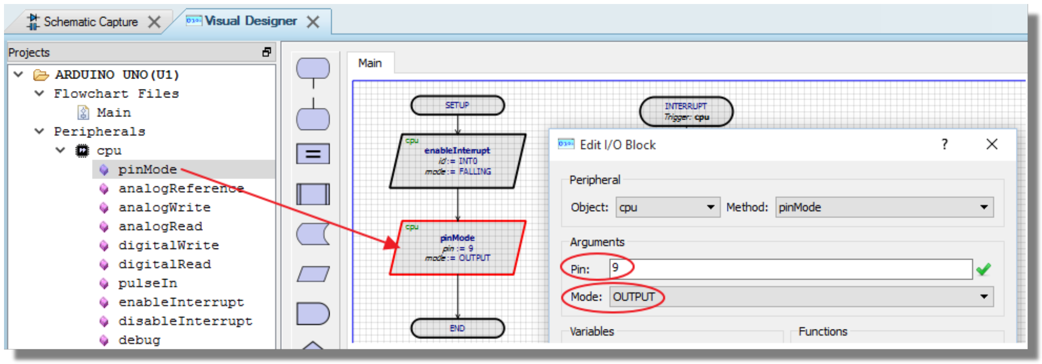


Select Trigger

We are going to turn on an LED when the button was pressed so we need to write out a logic high on one of the IO lines. Specifically, it needs to be IO9 because that's where we wired the LED on the schematic.

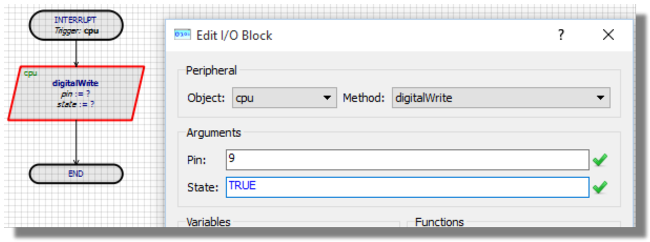


Start by setting the pin to be an output pin. We can do this directly in the Setup routine by dragging a pinMode routine on to the Flowline.



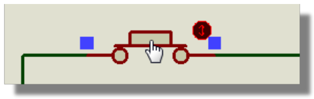
Object: cpu
Method: pinMode
Pin: 9
Mode: OUTPUT

Then drag and drop the digitalWrite method into the interrupt handler, name the pin and set the state to true.

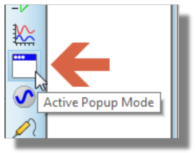


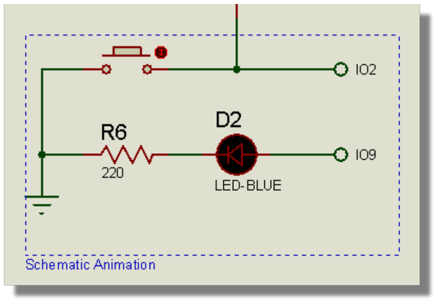
Object: cpu
Method: digitalWrite
Pin: 9
State: TRUE

That's it. Press play to simulate, switch to the schematic tab and press the button to test.



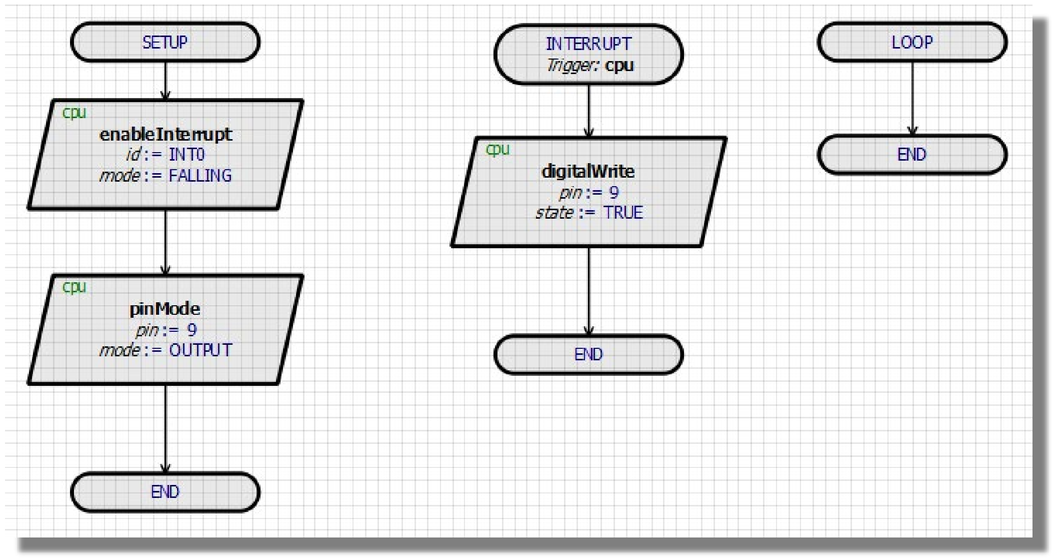
To bring an area of the schematic into the Visual Designer we need to specify it as an Active Popup. To do this, stop the simulation, select Active Popup mode and drag a box with the left mouse around the components of interest.





This time, when you press play you can test in Visual Designer via the Active Popups. It's a matter of preference in this case but, when debugging, it's often very useful to be able to see the circuit at the same time.

You can easily add a few blocks to change the program into a toggle or continue with developing the hardware on the schematic.
Your Flowchart project should finally look something like the following:



⚠ Interrupts are really useful but be aware that in the Arduino environment you are limited in what you can do. For example, attempting to write to an LCD inside an interrupt routine will simply fail because the Arduino stack for writing to the LCD itself uses interrupts. Caution is required.

RASPBERRY PI TUTORIALS

Introduction

It's remarkably simple to design and create electronic projects with Visual Designer. These short tutorials each cover a separate project and include various different techniques and methods in the project design.

Raspberry Pi Tutorials

Tutorial 1 : Flashing an LED (raw Pi).

Covers project creation, adding peripherals. basic flowchart programming and Proteus simulation.

Tutorial 2 : Nightlight (w/ Grove).

Includes grove module configuration, decision making flowcharts, interactive simulation and basic debugging.

Tutorial 3 : Motor Control (w/ rPi Hat)

Includes programming with loops, adding and calling sub-routines and using the clipboard..

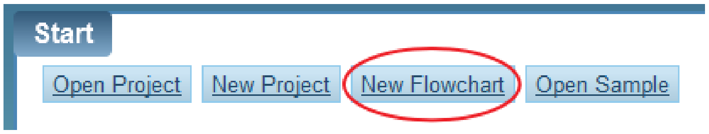
Tutorial 1 : Flashing LED

Introduction

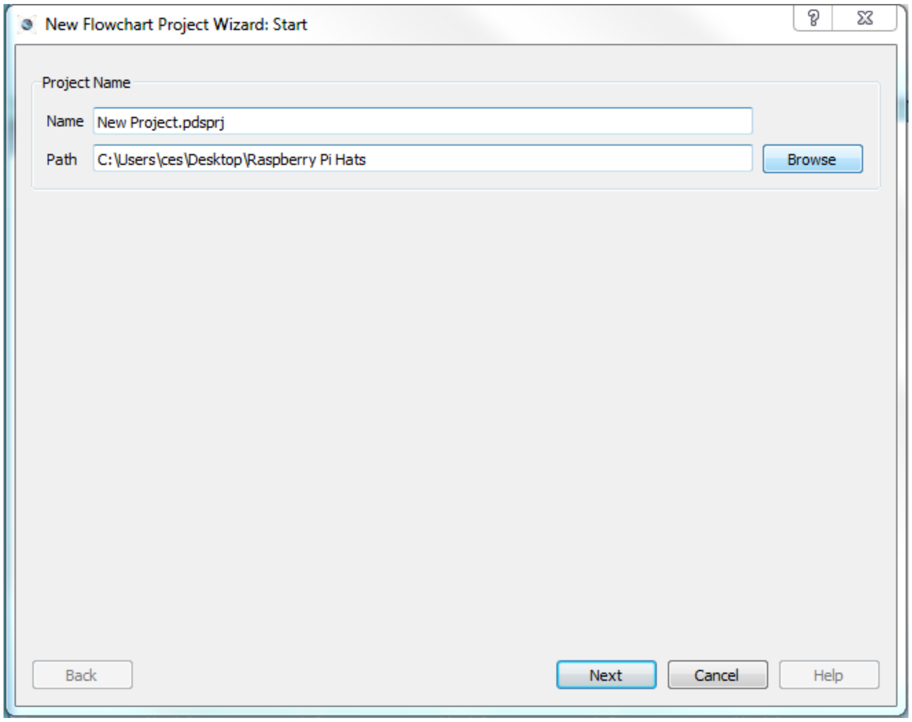
This tutorial covers the basics of project creation, working with flowcharts, raspberry Pi simulation and programming the real hardware. The actual goal of flashing the LED is kept as simple as possible so that we can cover the end-to-end process. Working with more complex peripherals and hats is covered in subsequent tutorials.

New Project Wizard

Visual Designer is integrated into the Proteus Design Suite and so we start our Visual Designer Projects from the Proteus Home Screen. The easiest and best way to create a Visual Designer Project is, unsurprisingly, through the New Flowchart Wizard.



The first page allows you to select the name and destination of your project.

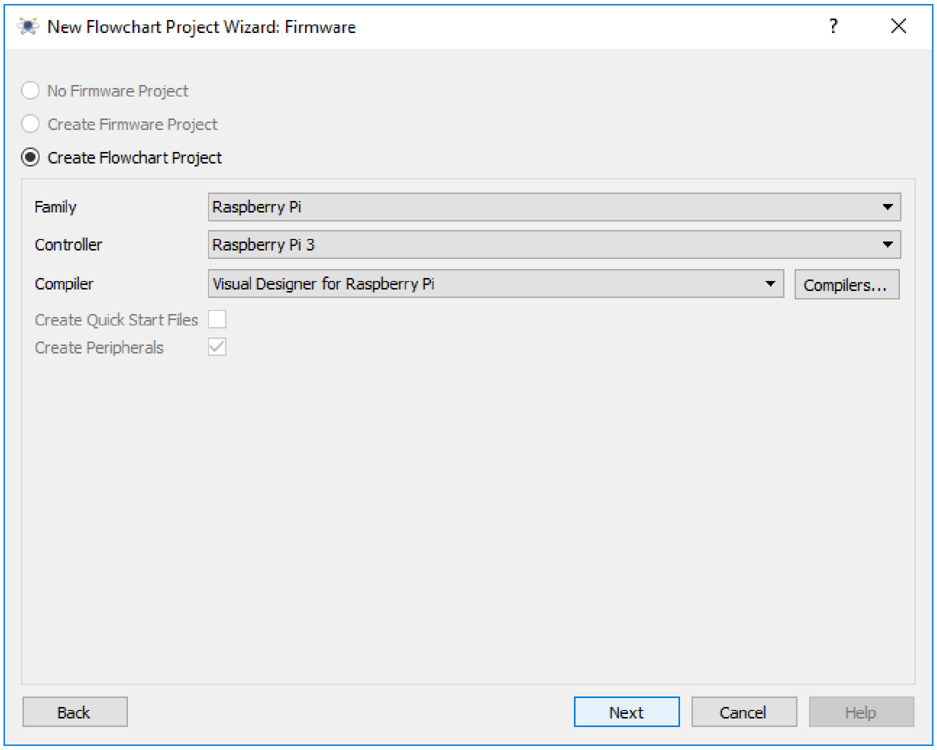

New Flowchart Project Wizard: Start

Project Name

Name: New Project.pdsprj

Path: C:\Users\ces\Desktop\Raspberry Pi Hats

Next up is the firmware screen and it is here that we really define our Visual Designer project. First, we change the radio buttons at the top to the option for Flowchart Project and then we select the Raspberry Pi family and either the RPi3 or the RPi3 + GrovePi Hat from the controller combo box.

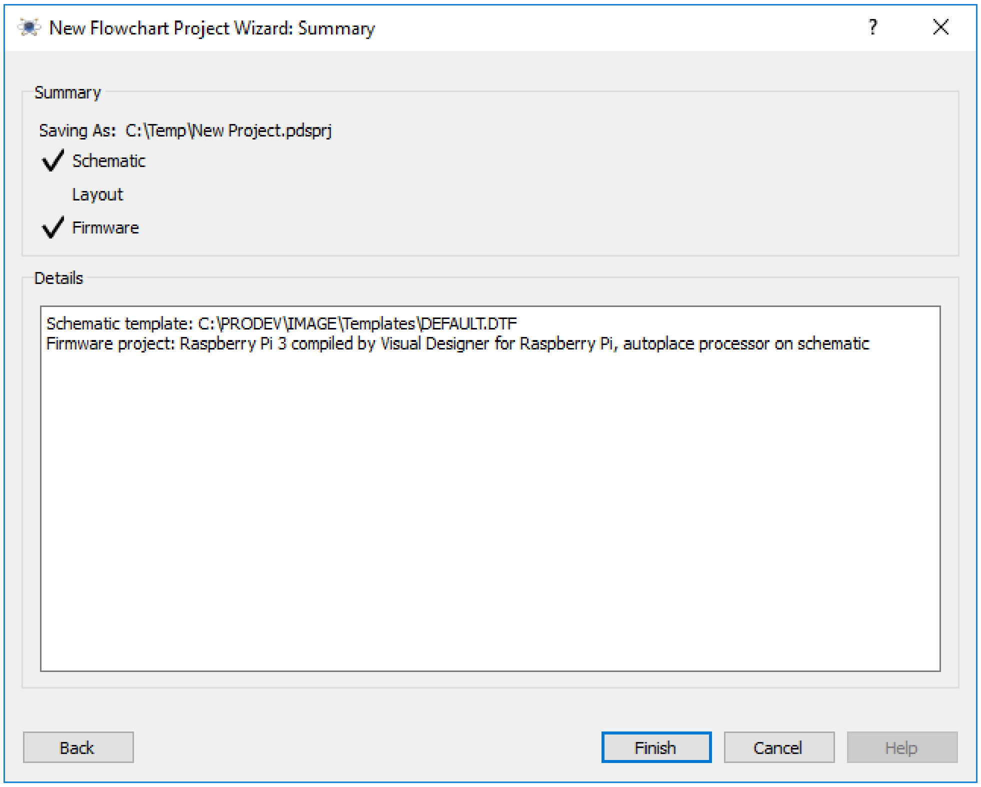


New Flowchart Project Wizard: Firmware

( ) No Firmware Project
( ) Create Firmware Project
(●) Create Flowchart Project

Family	Raspberry Pi
Controller	Raspberry Pi 3
Compiler	Visual Designer for Raspberry Pi
Create Quick Start Files	☐
Create Peripherals	✓

Finally, you will be presented with a summary page of your configuration.



Saving As: C:\Temp\New Project.pdsprj

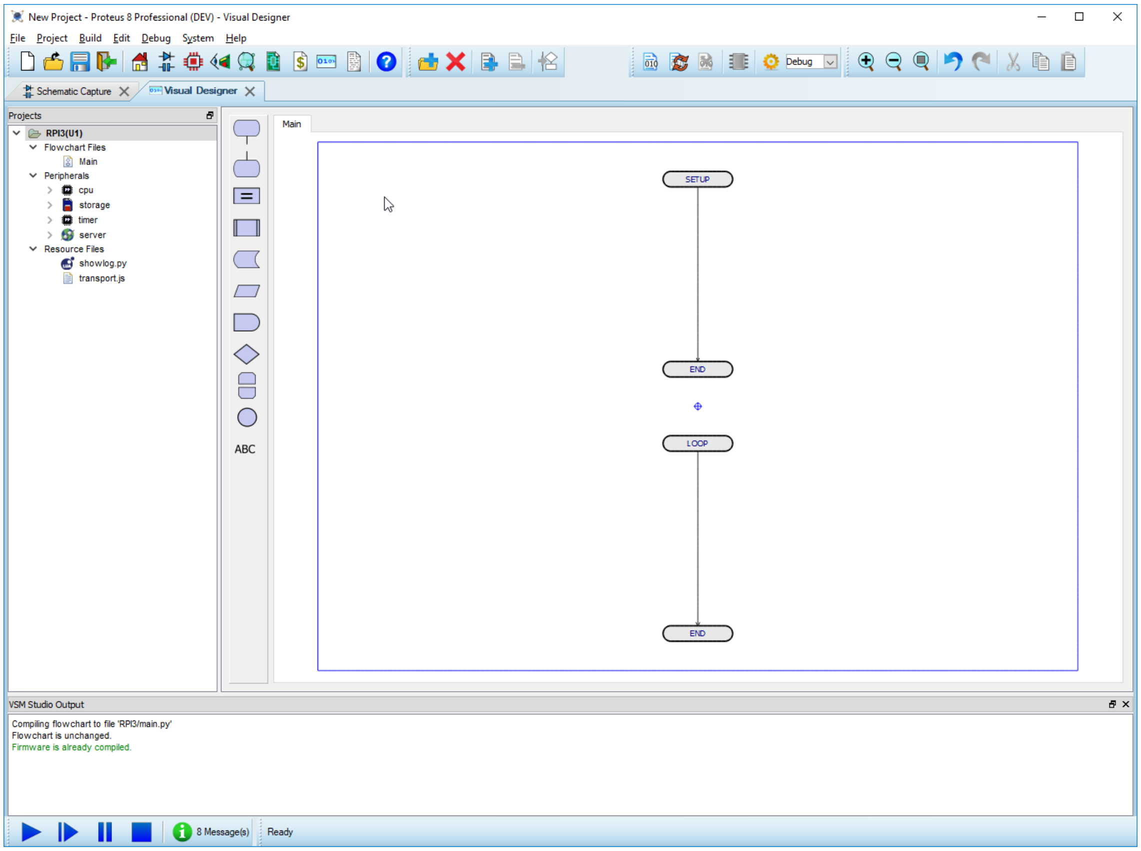
✓ Schematic

Layout

✓ Firmware

Schematic template: C:\PRODEV\IMAGE\Templates\DEFAULT.DTF
Firmware project: Raspberry Pi 3 compiled by Visual Designer for Raspberry Pi, autoplace processor on schematic

After you exit the dialogue form the project will be created and you will see both a skeleton schematic with the Raspberry Pi processor placed and a skeleton chart project with the familiar Setup and Loop routines on the Editing Window.



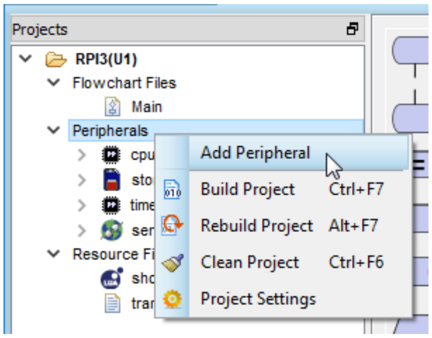
```
Compiling flowchart to file 'RPI3/main.py'
Flowchart is unchanged.
Firmware is already compiled.
```

Now we are ready to start designing our project.

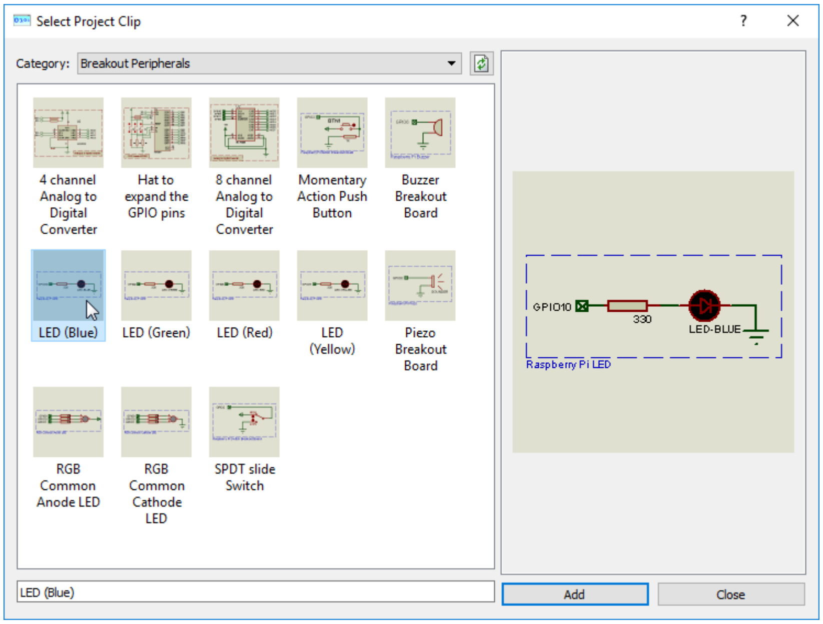
Adding a Peripheral

The first thing to do is to add the peripheral(s) for the project. These can be Raspberry Pi Hats or just little breakout peripherals. The Raspberry Pi does not need a Hat to operate and you can add peripherals which talk to the GPIO Pins Directly.

We'll be adding a Breakout LED. In Visual Designer, we right click on the Project Tree and select the Add Peripheral command. Next, switch to Breakout Peripherals and select one of the LED's.

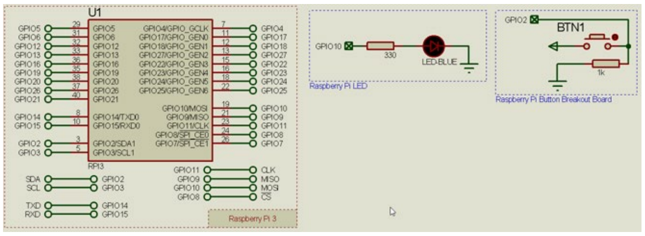


Add the Peripheral from the context menu command in VSM Studio



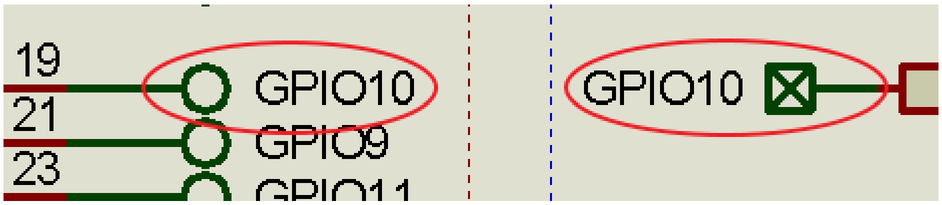
Double click on the LED to add it to the project.

Repeat the process to add a Breakout button. If you then switch to the schematic tab you will see that the 'virtual hardware' has been autoplaced for you.



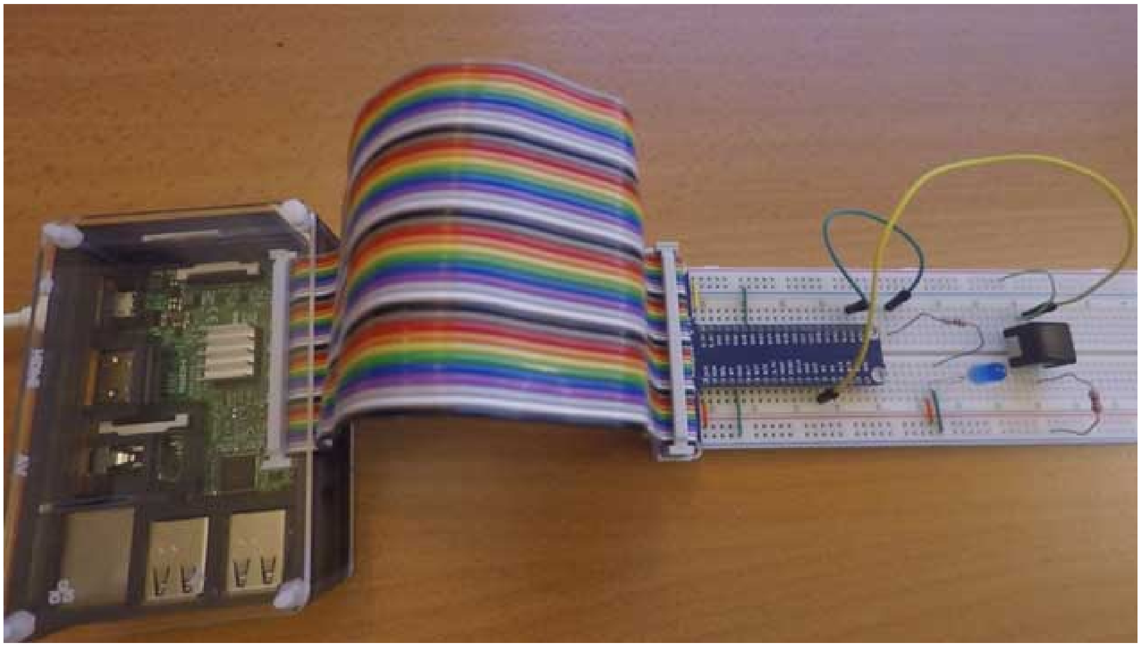
Schematic view with both LED and push button auto-placed.

To avoid clutter on the schematic, connections are made on the schematic by giving terminals the same name. Any terminals with the same name can be thought of having an invisible wire between them.

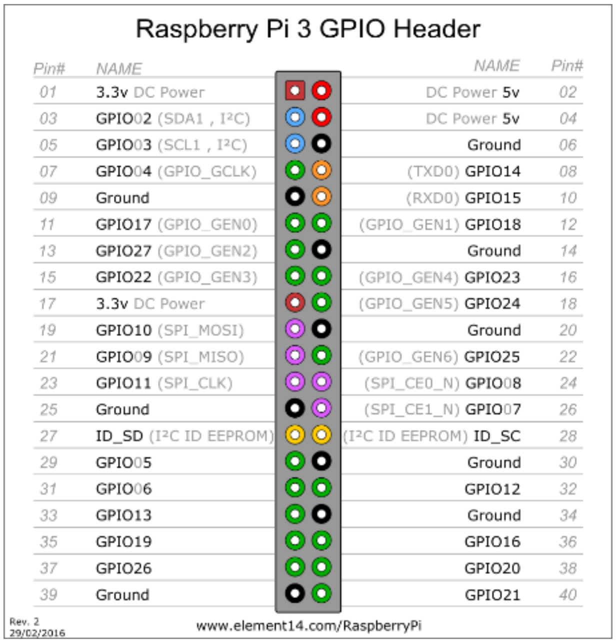


Terminals with the same name are considered connected together.

In real hardware, this equates to connecting the Breakout button and Breakout LED to the GPIO Pins on the Raspberry Pi



Note that you must attach the peripherals onto the same GPIO pins on the Raspberry Pi as they are designated on the Proteus schematic. You may need the pinout to see which GPIO pins go where.



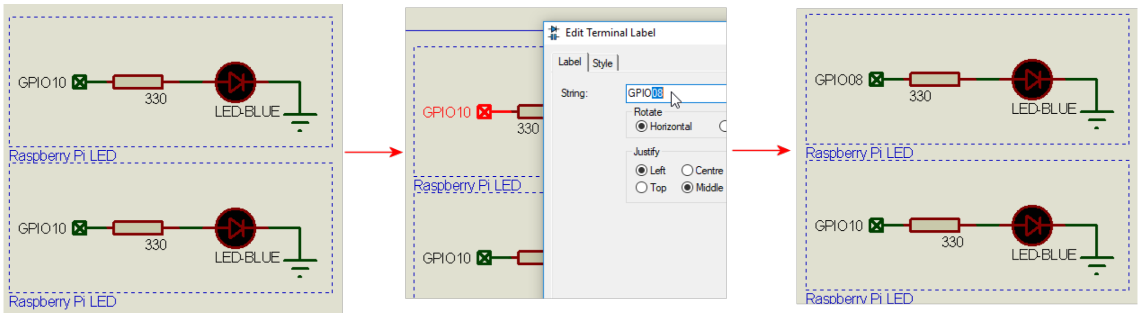
Raspberry Pi 3 GPIO Header

Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I²C)	DC Power 5v	04
05	GPIO03 (SCL1 , I²C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)	(I²C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

Rev. 2
29/02/2016
www.element14.com/RaspberryPi

GPIO pins on the Raspberry Pi

You must also ensure that you don't have two peripherals with the same terminal label on the schematic; if you do, you need to switch to the schematic and edit the terminal labels on one of the peripherals. Some pins may need unlocking.

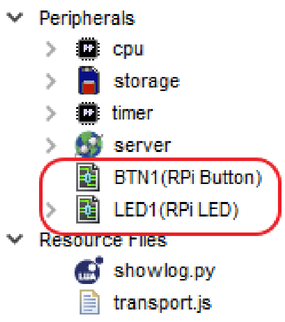


Edit Terminal Label
Label | Style
String: GPIO08
Rotate: (●) Horizontal
Justify: (●) Left ( ) Centre ( ) Top (●) Middle

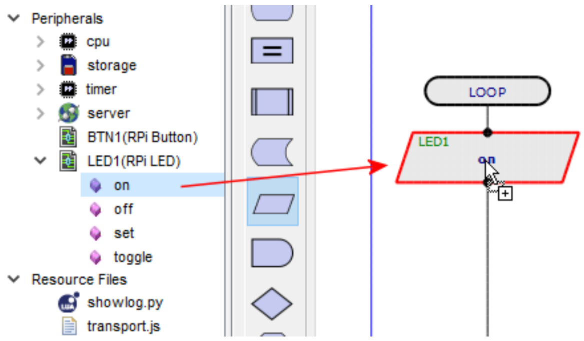
If you have two terminals with the same name they will be connected ! Need to edit one and re-assign to another GPIO line.

Flowchart Design

Back in Visual Designer, you'll notice that the Project Tree now has two entries under the peripherals section.

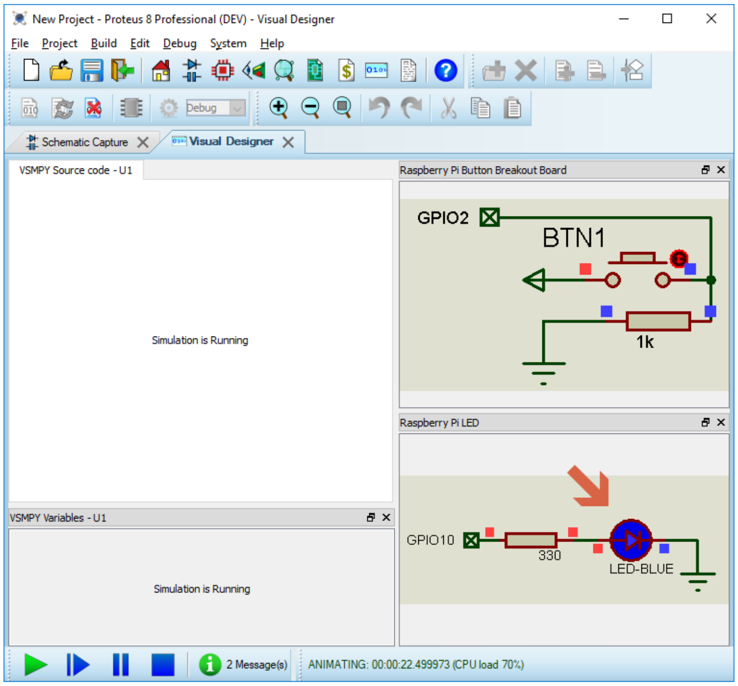


The beauty of Visual Designer is that you can expand the peripherals to find a list of methods. These methods are the primary way in which we interact with the hardware. For example, drag and drop the 'On' method of the LED into the loop routine



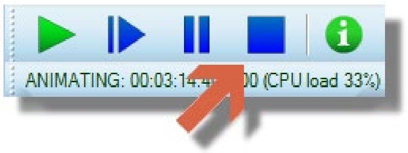
Drag and Drop methods onto the program.

To test this simply press the play button. The program will compile, the simulator will start and the LED will turn on. You can see this either via the Active popup at the right of Visual Designer or by switching to the schematic tab.



Run the simulation to test results.

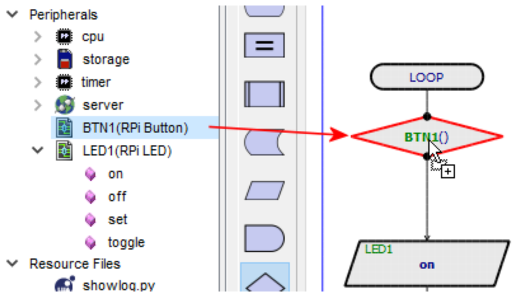
Press stop to exit the simulation and we'll extend our program to switch the LED on and off with a button press. You'll notice that the button has no methods in the project tree



Stop the simulation when finished.

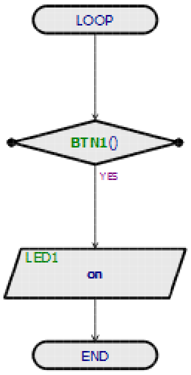
The only sensible information we need from the button is to know whether it has been pressed and for that we have what we call a 'sensor function'. A sensor function returns either TRUE or FALSE depending on whether the basic peripheral function is true or not. For example, the sensor function for a button will return TRUE when the button is pressed and FALSE if it isn't.

Similarly, the sensor function for an LED will return TRUE when the LED is on and FALSE when the LED is off. Place the sensor function for the button by dragging and dropping from the button itself onto the loop function of the flowchart

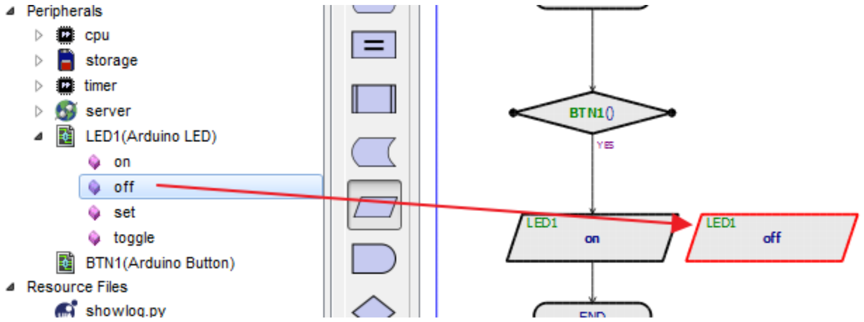


Drag the sensor function onto the flowchart

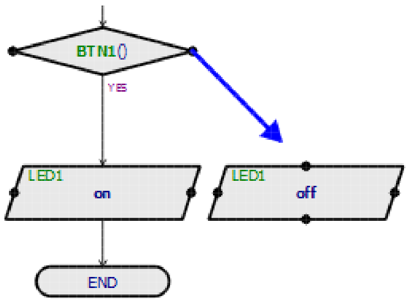
You will notice that it appears as a decision block. Sensor functions always return TRUE or FALSE and a decision block allows us to split our code into two conditional paths. Your loop routine should currently look something like the following.



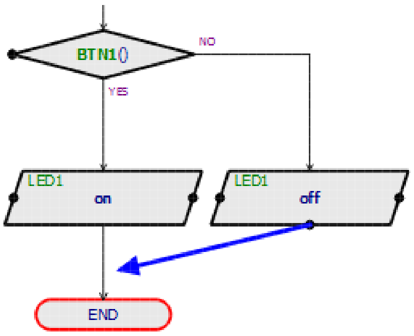
We are constantly testing the button to see if it is pressed and we are turning on the LED if it is. However, we still need to tell the program what to do if the button is not pressed. We want the LED to turn off so lets drag and drop that method into space alongside the ON operation.



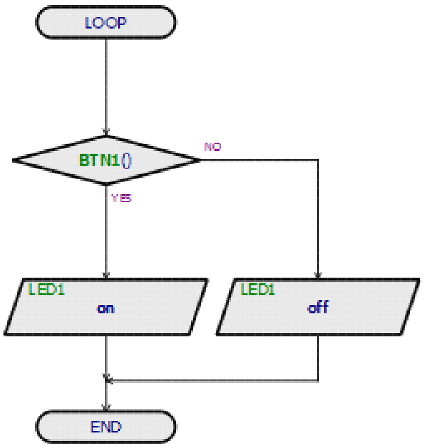
Now, we need to wire from the decision block to the top of the OFF command. To place a flowline click on an output node (the decision block), move the mouse to an available input node (top of the OFF routine) and click the mouse again.



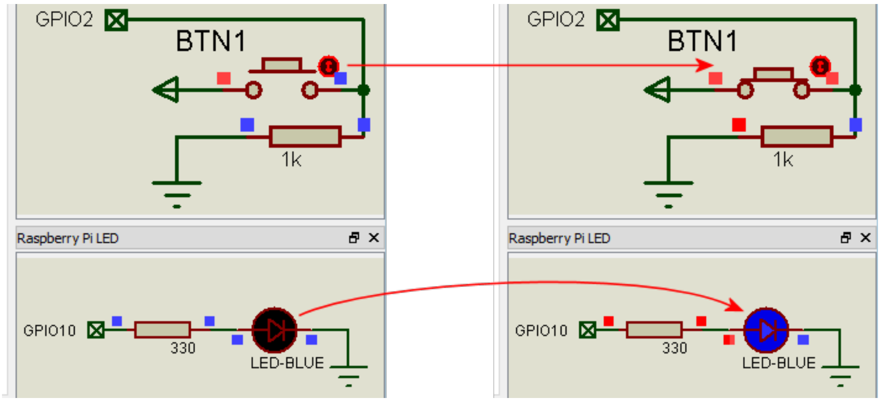
Similarly we need to connect the flow from the bottom of the OFF command back into the main loop.



Your program should now look like the following.

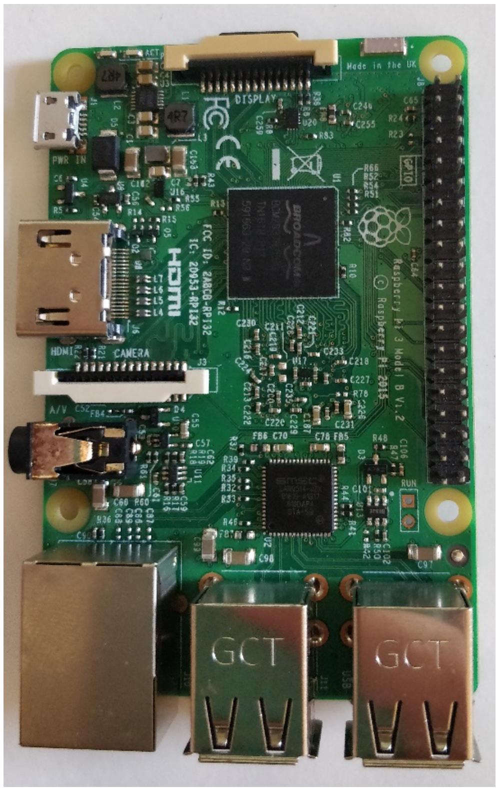


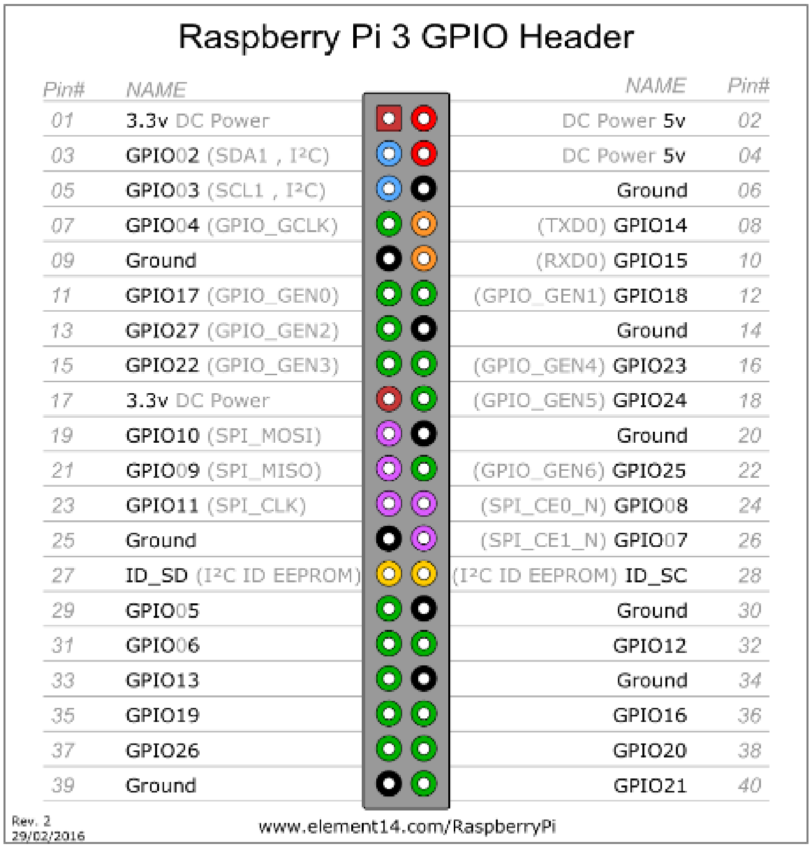
Having made this change, our program now turns the LED off when the button is down and on when the button is released. To compile and test, first press the play button and then use the mouse to press the button down..



Programming the Hardware

This section assumes that you have completed the initial configuration of the Raspberry Pi.





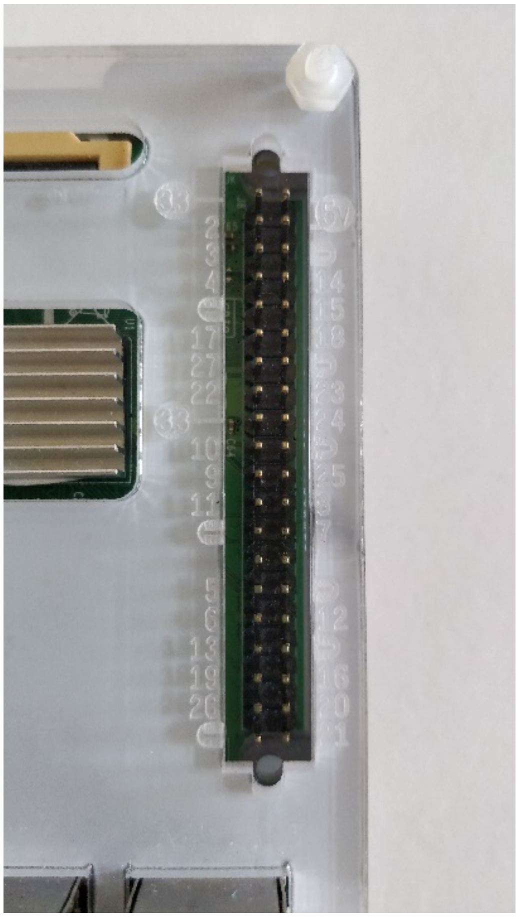
Raspberry Pi 3 GPIO Header

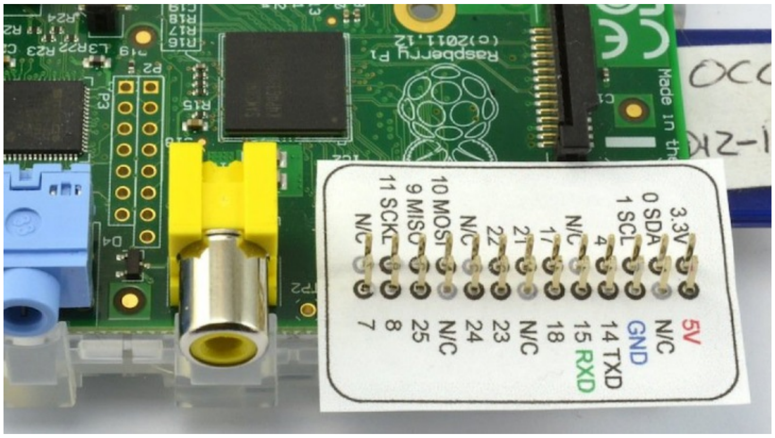
Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I²C)	DC Power 5v	04
05	GPIO03 (SCL1 , I²C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)	(I²C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

Rev. 2
29/02/2016

www.element14.com/RaspberryPi

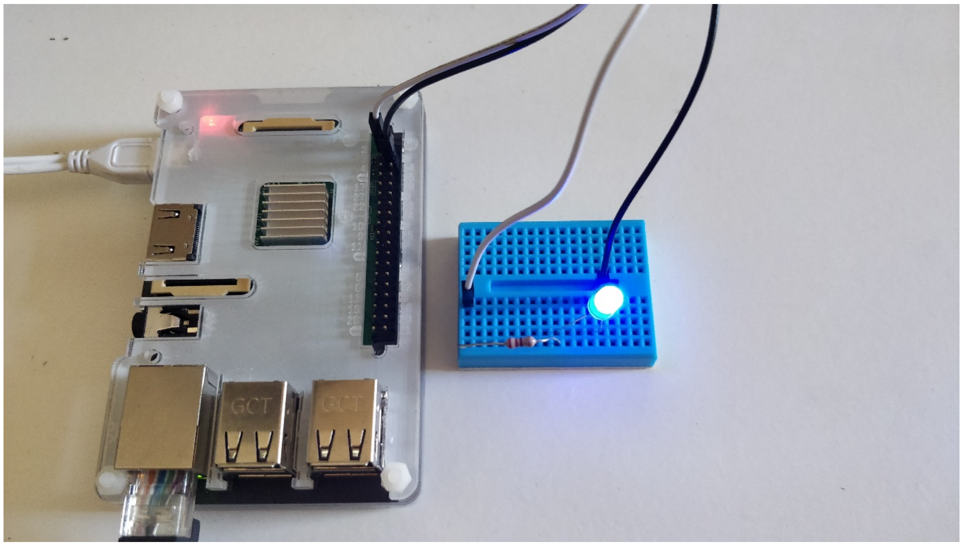
Here on the Left is a Raspberry Pi 3, and on the Right is the pinout for the GPIO Pins. The very first thing you need to understand is the orientation. Both these images are oriented in the same way. The top 2 pins are both power (Left pin is 3.3V and Right pin is 5V). You can find this pinout on the raspberry pi website. Some of the Raspberry Pi I cases have the pinout engraved into it making it very easy to find each pin. Failing that, you can use a template or make one with a little bit of card to make life easier.



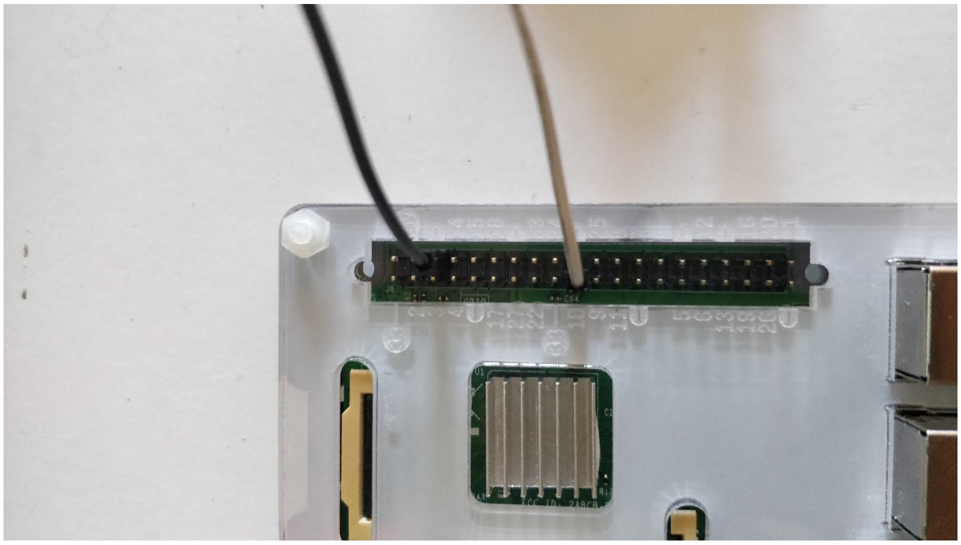


Engraved casing or simple template to identify lines on the header block.

Now in our program we connected the LED to GPIO 10. Not to be confused with Pin 10 which is GPIO 15. GPIO and Pin numbers are very different so make sure you connect the correct one. Just like the schematic we've included a protection resistor to prevent he LED from getting damaged by the current from the Raspberry Pi.



Here you can see we've used Male to Female wires connecting the Raspberry Pi to the Breadboard.

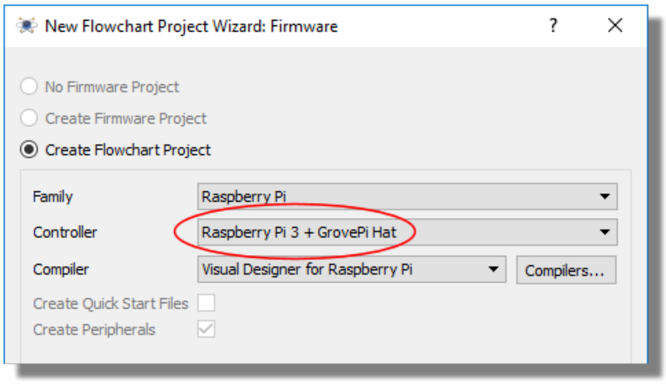


Assuming all is correct your led should flash just like the project.

Tutorial 2 : Night Light (w/ Grove)

Introduction

In this tutorial we are going to use a couple of Grove sensors and an LED to design a mini nightlight. Project setup is covered in the first tutorial however this time we will want to change the controller to the RPI3 + GrovePi Hat.



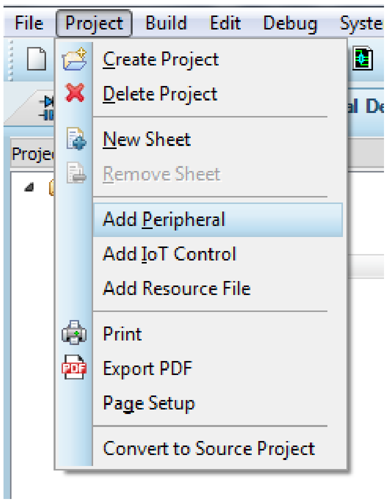
New Flowchart Project Wizard: Firmware

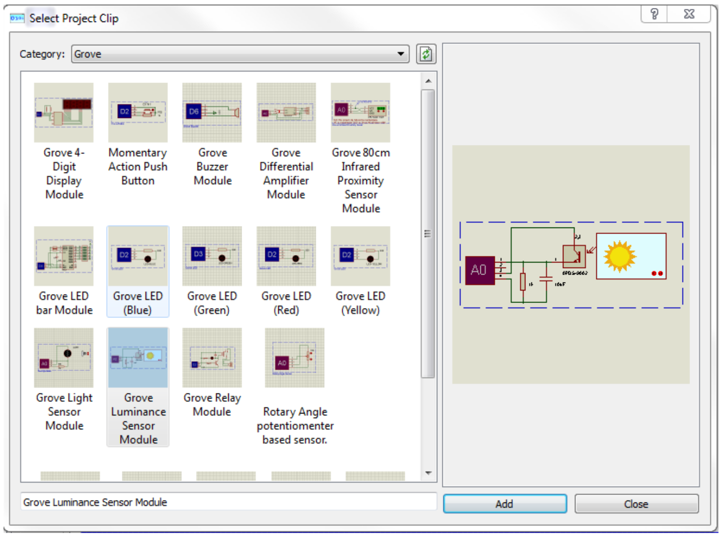
No Firmware Project
Create Firmware Project
Create Flowchart Project [✓]

Family: Raspberry Pi
Controller: Raspberry Pi 3 + GrovePi Hat
Compiler: Visual Designer for Raspberry Pi
Create Quick Start Files
Create Peripherals ✓

Adding Hardware

We'll use the Grove infra-red proximity sensor and the Grove luminance sensor along with a Grove LED. All of these can be picked via the Add Peripheral command on the Project Menu.





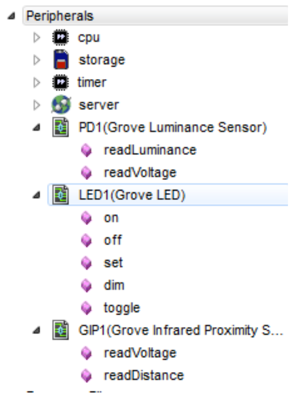
Select Project Clip

Category: Grove

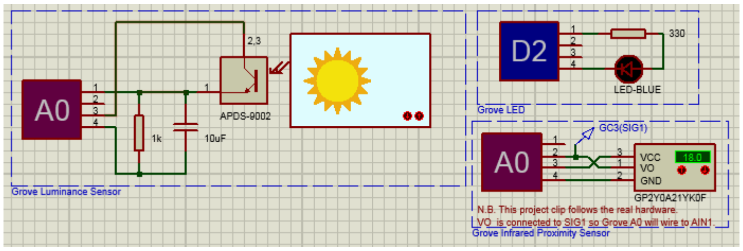
Grove 4-Digit Display Module
Momentary Action Push Button
Grove Buzzer Module
Grove Differential Amplifier Module
Grove 80cm Infrared Proximity Sensor Module
Grove LED bar Module
Grove LED (Blue)
Grove LED (Green)
Grove LED (Red)
Grove LED (Yellow)
Grove Light Sensor Module
Grove Luminance Sensor Module
Grove Relay Module
Rotary Angle potentiometer based sensor.

Grove Luminance Sensor Module

After they have been added to the project, you will see the methods for controlling the peripherals in the Project Tree and the 'virtual hardware' for the peripherals placed for you on the schematic.



- Peripherals
  - cpu
  - storage
  - timer
  - server
  - PD1(Grove Luminance Sensor)
    - readLuminance
    - readVoltage
  - LED1(Grove LED)
    - on
    - off
    - set
    - dim
    - toggle
  - GIP1(Grove Infrared Proximity S...
    - readVoltage
    - readDistance



N.B. This project clip follows the real hardware.
VO is connected to SIG1 so Grove A0 will wire to AIN1.

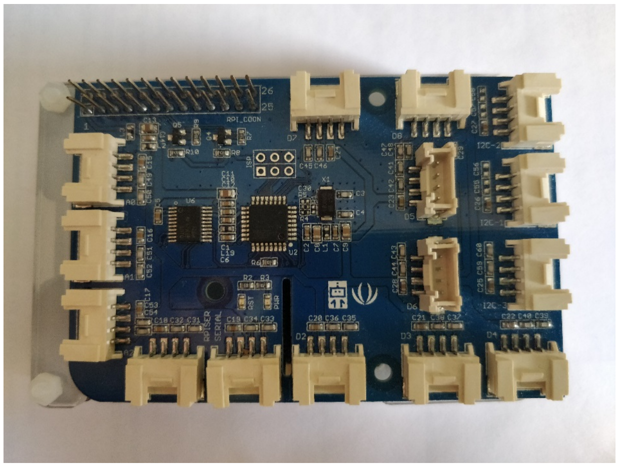
It is vitally important to configure the Grove connectors properly and this is often not as simple as it first appears. Let's take an example:

The real proximity sensor requires only three wires with the fourth left NC. Note therefore that, when plugged into socket A0 the sensor is connecting to analogue pin A1.

This is fine, **except** if you then plug the luminance sensor into socket A1. The luminance sensor will use analogue pin A1 on socket A1 and therefore clash. You should plug this sensor into socket A2.

Golden Rule

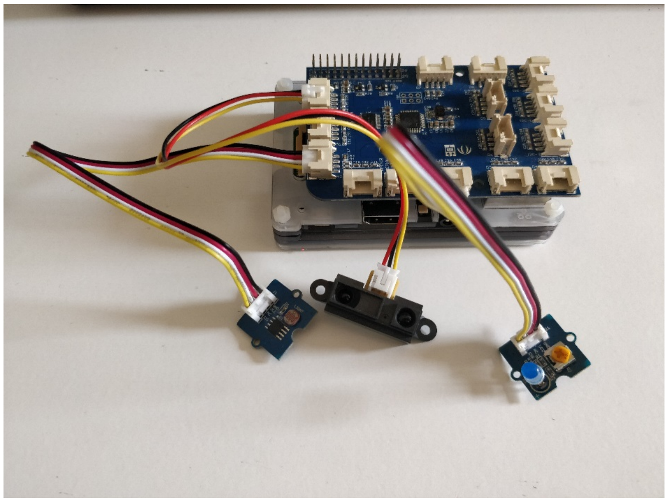
Look closely at the Grove board to see the connections and double check against the sensors for conflicts. Unlike the Arduino the conflicts are not shown on the board and need to be checked separately.



The sockets include connections through to two Raspberry Pi IO pins and there is a pin overlap. For example:

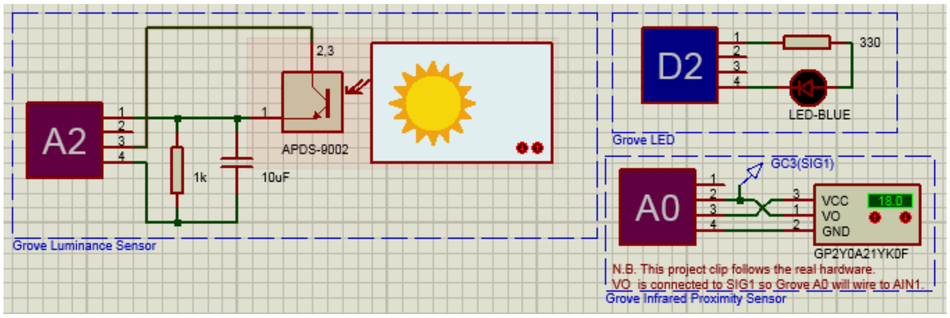
- Socket A0 can use pins A0 and A1
- Socket A1 can use pins A1 and A2
- Socket A2 can use pins A2 and A3.
- etc

Digital sockets are exactly the same.



Grove Peripherals connected.

Our configuration for this project is therefore proximity sensor in socket A0 and luminance sensor in socket A2. The LED can be in any digital socket we like.

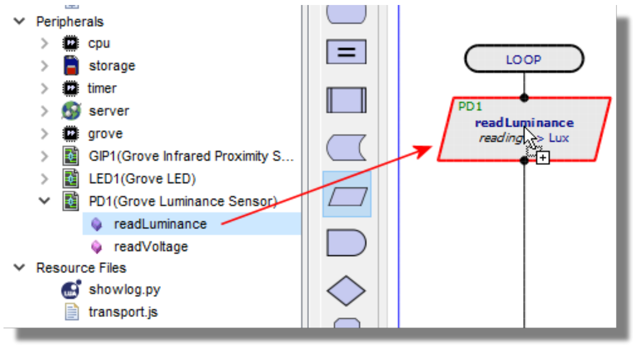

A2
1
2
3
4
2,3
1
APDS-9002
1k
10uF
Grove Luminance Sensor
D2
330
LED-BLUE
Grove LED
GC3(SIG1)
A0
VCC
VO
GND
18.0
GP2Y0A21YK0F
N.B. This project clip follows the real hardware.
VO is connected to SIG1 so Grove A0 will wire to AIN1.
Grove Infrared Proximity Sensor

Designing the Program

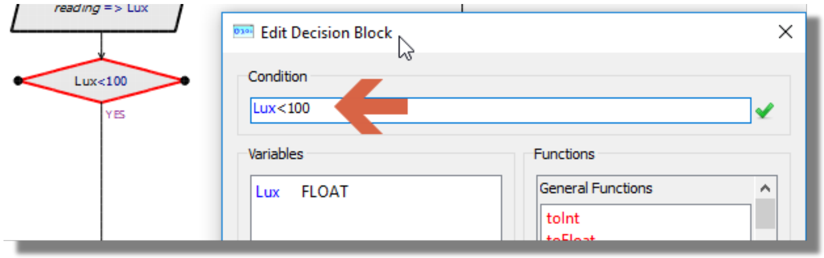
In our Nightlight mock up we are looking for our LED to light when the following are true:

- It's is dark.
- Someone is nearby.

These are both decisions on the flowchart. Start by drag and drop the readLuminance() method onto the loop routine of the chart.



This sensor returns a lux value between 0 and 1000, with 0 being pitch black and 1000 max light. Let's then use our result in a decision block and set our initial test value to around 100.



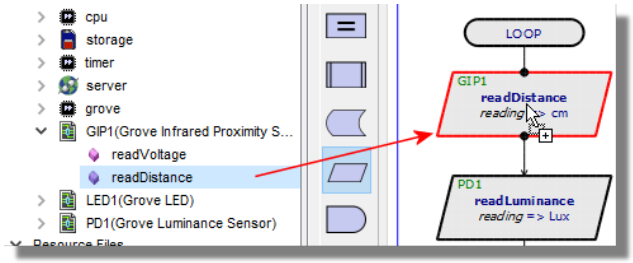
Lux<100

Lux FLOAT

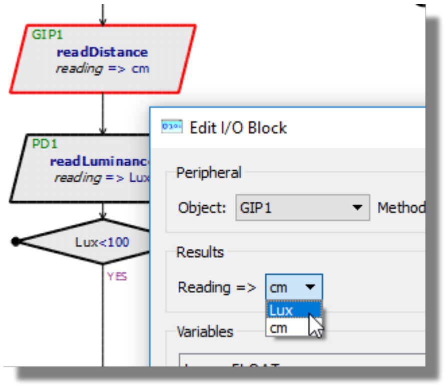
The datasheet for the APDS-9002 photo sensor used in this Grove module provides data on lux values.

See Also: http://wiki.seeedstudio.com/Grove-Luminance_Sensor/

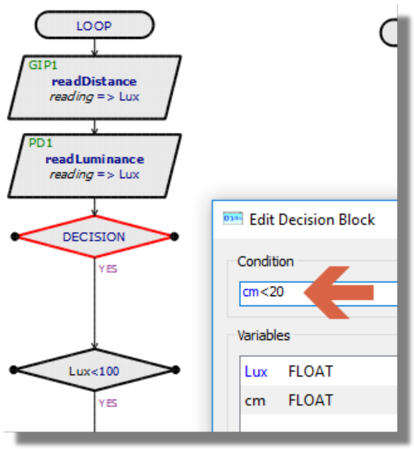
For the proximity sensor, we want to read a distance so drag and drop into the top of the loop.



Set the Variable that it's reading to be LUX

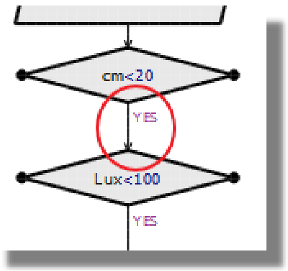


The proximity sensor can detect up to 80cm but we'll set our initial test to 20cm. Add another Decision block above the LUX Decision block, and set to be cm<20:

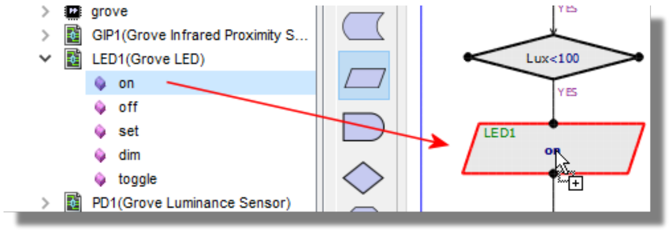


See Also: http://wiki.seeedstudio.com/Grove-80cm_Infrared_Proximity_Sensor/

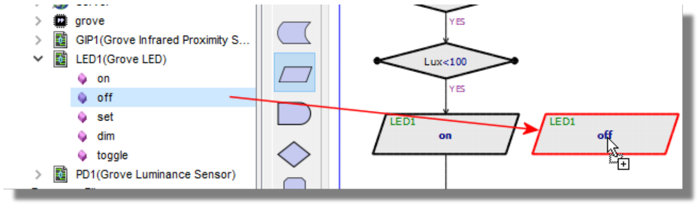
Since, we need both conditions to be true for the LED to turn on, make sure that the second decision is on the YES branch of the first decision.



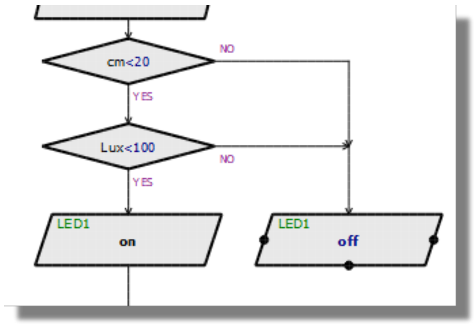
Next, drag and drop the led on method at the bottom of the YES branch.


grove
GIP1(Grove Infrared Proximity S...
LED1(Grove LED)
on
off
set
dim
toggle
PD1(Grove Luminance Sensor)
YES
Lux<100
YES
LED1
on

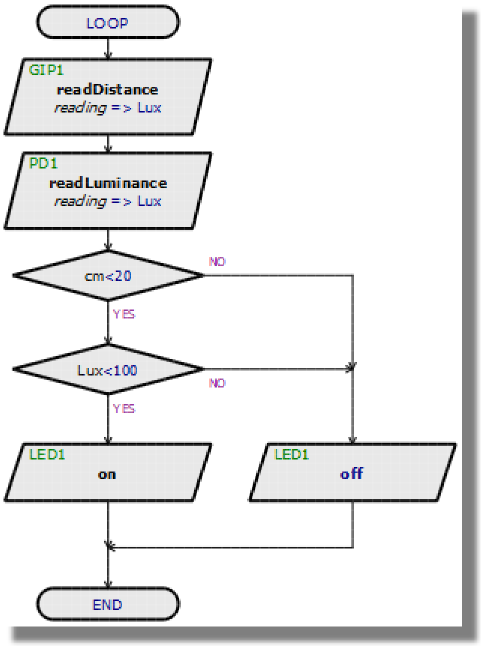
Finally, place the LED off method to the right of the main loop beside the LED on method.



Since we want the LED to turn off if either condition is false we can wire both NO branches from the decisions to this method.

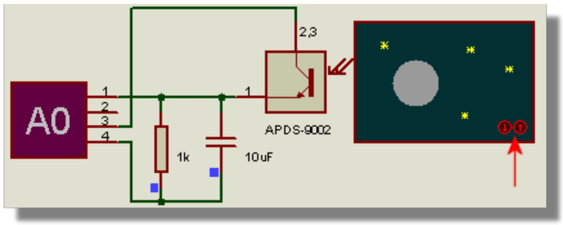


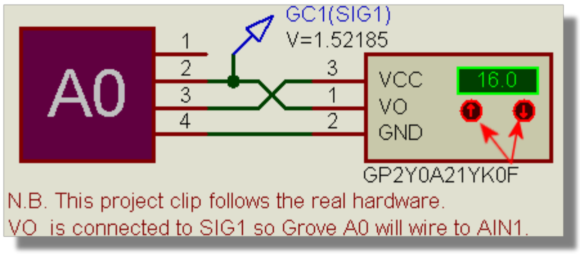
Finally, connect the bottom of the NO program branch back into the main program loop. Your final program should look like the following:



Simulation and Testing

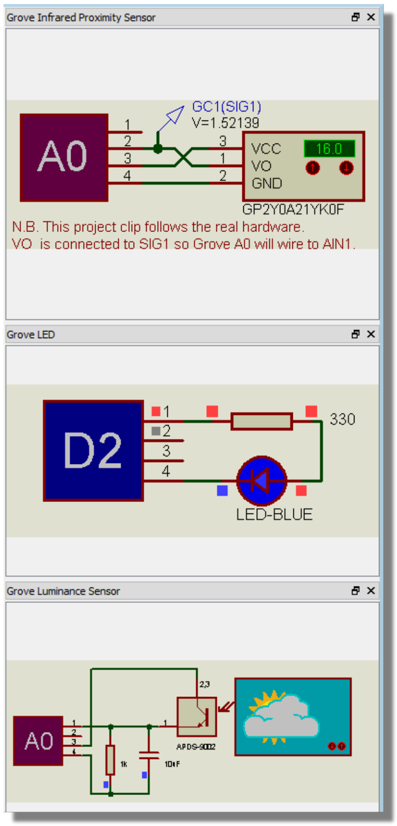
To test our program, all we need to do is press the play button and then adjust the distance display on the proximity sensor and the lux display on the luminance sensor.





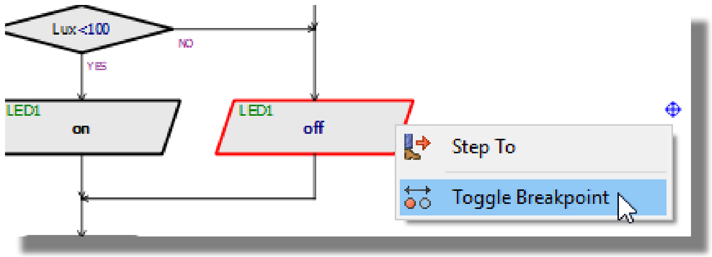
N.B. This project clip follows the real hardware.
VO is connected to SIG1 so Grove A0 will wire to AIN1.

The LED should turn on when the distance value is less than 20 and it's a little cloudy.

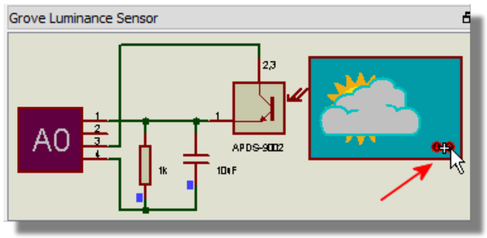


N.B. This project clip follows the real hardware.
VO is connected to SIG1 so Grove A0 will wire to AIN1.

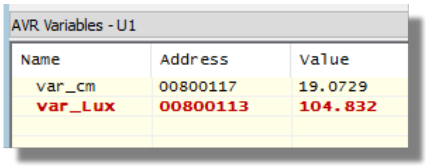
Pause the simulation and set a breakpoint on the LED off method.



Press play again to run the simulation and change either the distance or the light until the breakpoint triggers

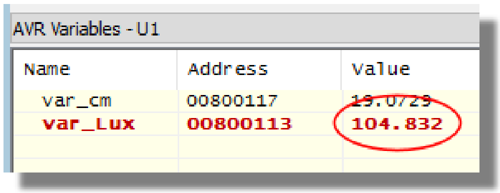


You'll notice that both our variables are available in the variables display.

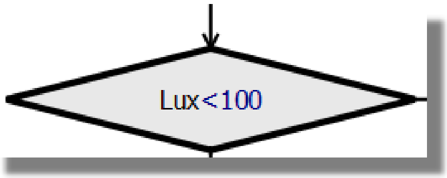

AVR Variables - U1

Name	Address	Value
var_cm	00800117	19.0729
var_Lux	00800113	104.832

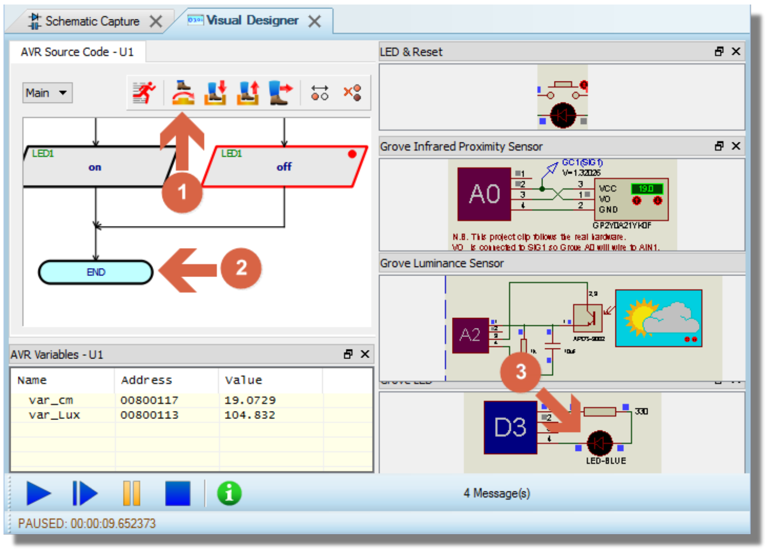
Comparing these values to the tested decisions will tell us which condition has failed.


AVR Variables - U1

Name	Address	Value
var_cm	00800117	19.0729
var_Lux	00800113	104.832



As you single step via the icons at the top you'll notice the LED turn off.



AVR Variables - U1

Name	Address	Value
var_cm	00800117	19.0729
var_Lux	00800113	104.832

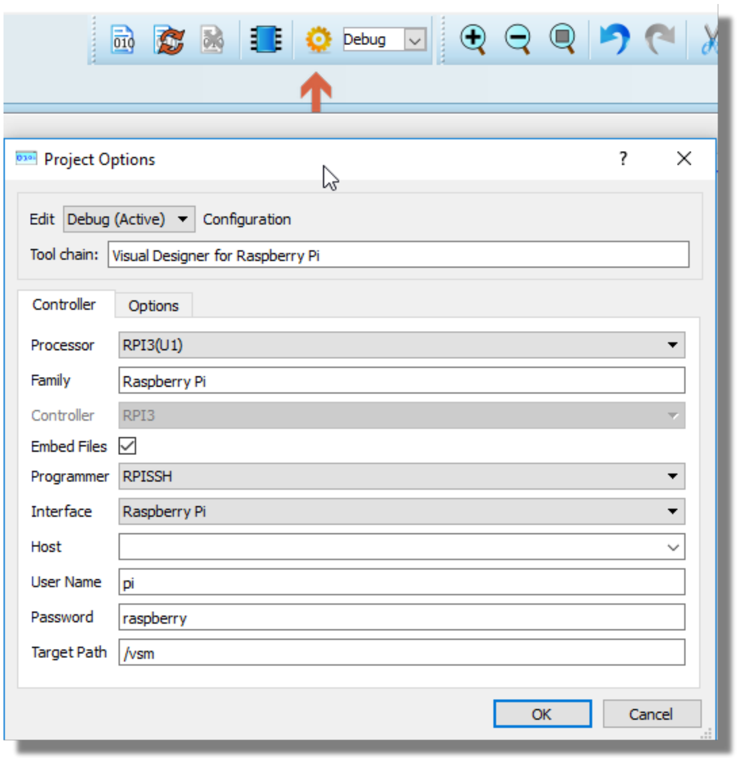
Programming

⚠ Before programming the Raspberry Pi you need to first perform the one time. See ***Programming the Raspberry Pi***

Since everything seems to working in good order, the remaining task is to plug in the Raspberry Pi and program the real hardware. The process here is:

1) Plug the Grove sensors into the Grove baseboard in the same connectors as on the schematic. See also the golden rule discussed earlier.

2) Connect the Grove hat to the Raspberry Pi and connect the Raspberry Pi to the same WIFI as the computer. Now within Proteus within the project options change the Programmer from NONE to RPISSH and the raspberry pi should appear.



Edit Debug (Active) Configuration
Tool chain: Visual Designer for Raspberry Pi

Processor: RPI3(U1)
Family: Raspberry Pi
Controller: RPI3
Embed Files: ✓
Programmer: RPISSH
Interface: Raspberry Pi
Host:
User Name: pi
Password: raspberry
Target Path: /vsm

3) Click the program upload button in Visual Designer.

4) The program will now run on the Hardware.

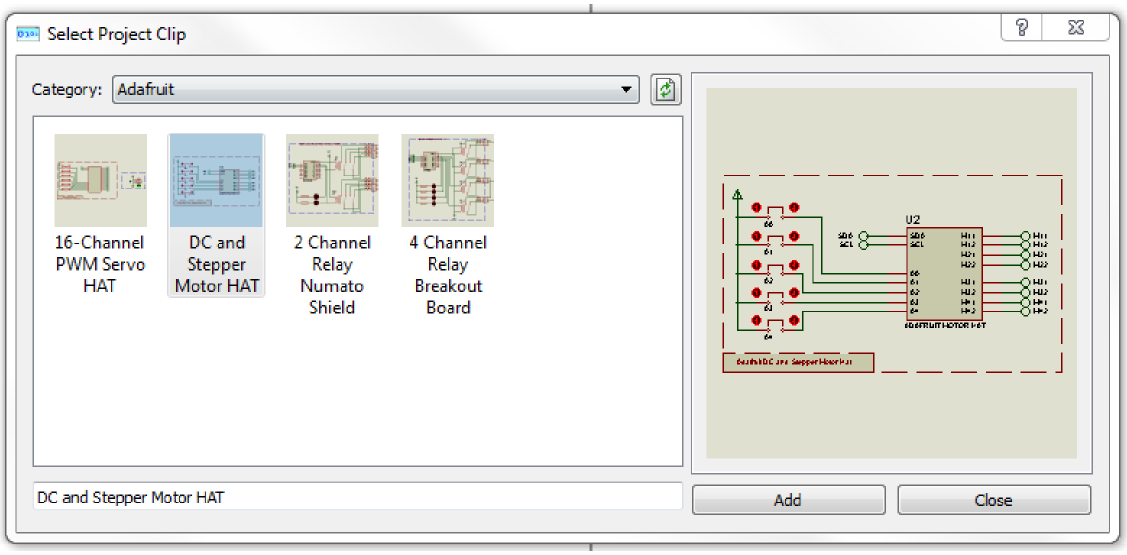
Tutorial 3 : Motor Control

Introduction

This tutorial shows how you can easily control both DC motors and Stepper motors with Visual Designer. We'll do this using the Adafruit DC and Stepper motor hat for Raspberry Pi.

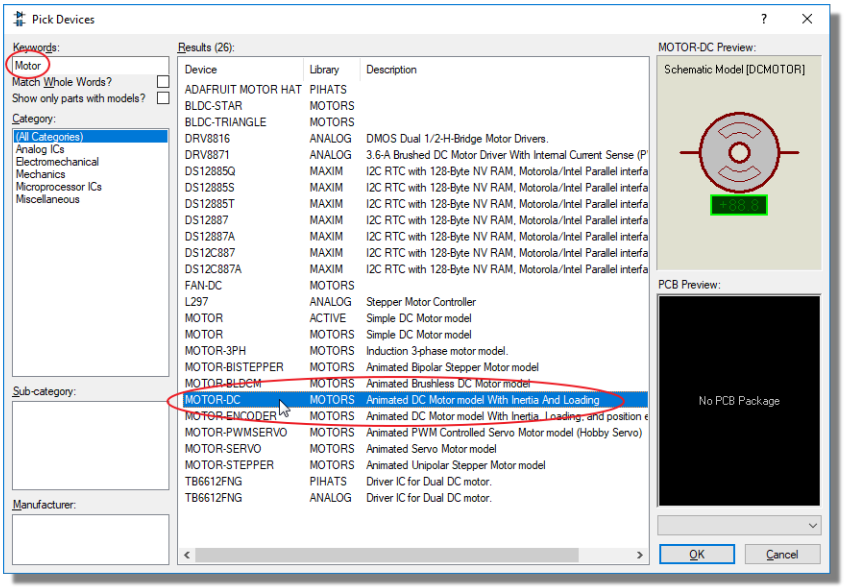
Setup

Start by creating a new Visual Designer Project and then Add the Raspberry Pi Hat with DC and Stepper Motors from the peripheral menu.



Picking the Adafruit Hat from the Add Peripherals command in Visual Designer

Unlike the Arduino Hat the Raspberry Pi Hat lets you connect the motors yourself. This means you need to first find the motors via the schematic pick devices form (MOTOR-DC & MOTOR-BISTEPPER) and double click to insert them into the project.



Pick Devices

Keywords: Motor

Match Whole Words?

Show only parts with models?

Category:

(All Categories)
Analog ICs
Electromechanical
Mechanics
Microprocessor ICs
Miscellaneous

Sub-category:

Manufacturer:

Results (26):

Device	Library	Description
ADAFRUIT MOTOR HAT	PIHATS	
BLDC-STAR	MOTORS	
BLDC-TRIANGLE	MOTORS	
DRV8816	ANALOG	DMOS Dual 1/2-H-Bridge Motor Drivers.
DRV8871	ANALOG	3.6-A Brushed DC Motor Driver With Internal Current Sense (P
DS12885Q	MAXIM	I2C RTC with 128-Byte NV RAM, Motorola/Intel Parallel interfa
DS12885S	MAXIM	I2C RTC with 128-Byte NV RAM, Motorola/Intel Parallel interfa
DS12885T	MAXIM	I2C RTC with 128-Byte NV RAM, Motorola/Intel Parallel interfa
DS12887	MAXIM	I2C RTC with 128-Byte NV RAM, Motorola/Intel Parallel interfa
DS12887A	MAXIM	I2C RTC with 128-Byte NV RAM, Motorola/Intel Parallel interfa
DS12C887	MAXIM	I2C RTC with 128-Byte NV RAM, Motorola/Intel Parallel interfa
DS12C887A	MAXIM	I2C RTC with 128-Byte NV RAM, Motorola/Intel Parallel interfa
FAN-DC	MOTORS	
L297	ANALOG	Stepper Motor Controller
MOTOR	ACTIVE	Simple DC Motor model
MOTOR	MOTORS	Simple DC Motor model
MOTOR-3PH	MOTORS	Induction 3-phase motor model.
MOTOR-BISTEPPER	MOTORS	Animated Bipolar Stepper Motor model
MOTOR-BLDCM	MOTORS	Animated Brushless DC Motor model
MOTOR-DC	MOTORS	Animated DC Motor model With Inertia And Loading
MOTOR-ENCODER	MOTORS	Animated DC Motor model With Inertia, Loading, and position e
MOTOR-PWMSERVO	MOTORS	Animated PWM Controlled Servo Motor model (Hobby Servo)
MOTOR-SERVO	MOTORS	Animated Servo Motor model
MOTOR-STEPPER	MOTORS	Animated Unipolar Stepper Motor model
TB6612FNG	PIHATS	Driver IC for Dual DC motor.
TB6612FNG	ANALOG	Driver IC for Dual DC motor.

MOTOR-DC Preview:

Schematic Model [DCMOTOR]

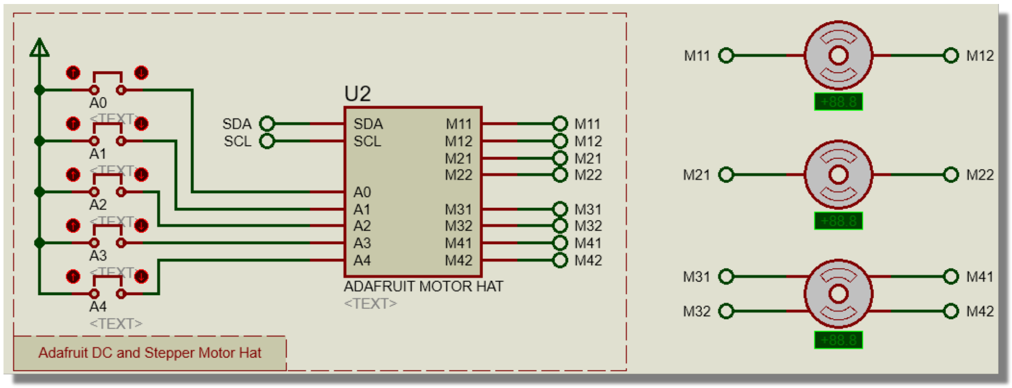
PCB Preview:

No PCB Package

OK Cancel

Using the Pick Devices Dialogue on the Schematic to add the motors to your design.

Then connect the part to the hat using the terminals. The usual rules apply in that terminals with the same name are deemed connected together

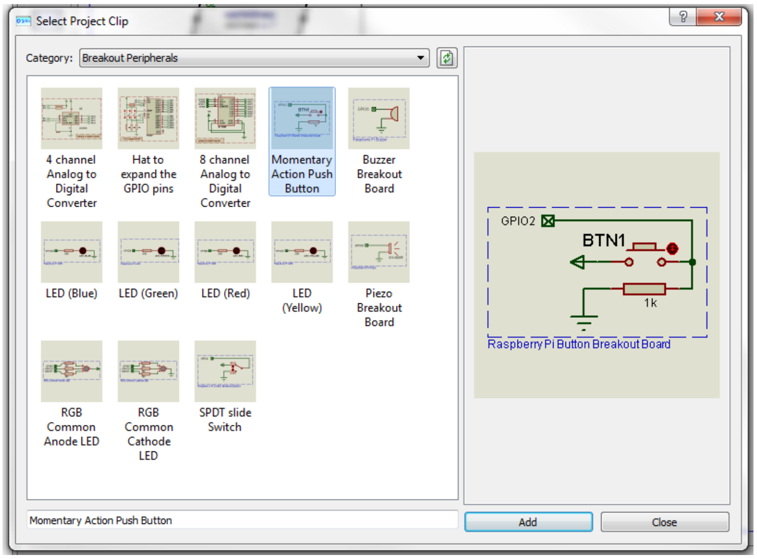


Here you can see we have connected one stepper motor and two DC motors with connections made by terminals.

Programming

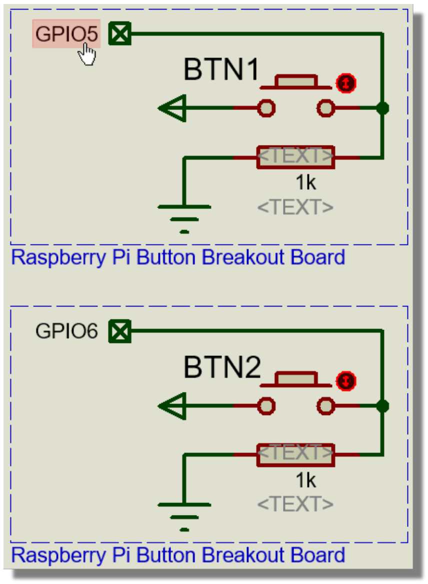
For this project we will have 2 buttons, one will start a dc motor and the other will stop that motor. After we have this project working we will then simulate it and work with the debugging tools.

Firstly, you will need 3 components, 2 buttons and a motor. Add the two buttons using the add peripheral method.


Select Project Clip
Category: Breakout Peripherals
4 channel Analog to Digital Converter
Hat to expand the GPIO pins
8 channel Analog to Digital Converter
Momentary Action Push Button
Buzzer Breakout Board
LED (Blue)
LED (Green)
LED (Red)
LED (Yellow)
Piezo Breakout Board
RGB Common Anode LED
RGB Common Cathode LED
SPDT slide Switch
GPIO2
BTN1
1k
Raspberry Pi Button Breakout Board
Momentary Action Push Button
Add
Close

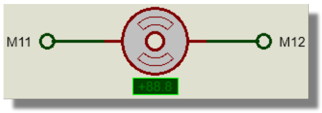
Add Button Peripheral.

Make sure you change the GPIO pins so that they are not both using the same pin


GPIO5
BTN1
<TEXT>
1k
<TEXT>
Raspberry Pi Button Breakout Board
GPIO6
BTN2
<TEXT>
1k
<TEXT>
Raspberry Pi Button Breakout Board

Make sure the buttons are assigned to unique GPIO Lines

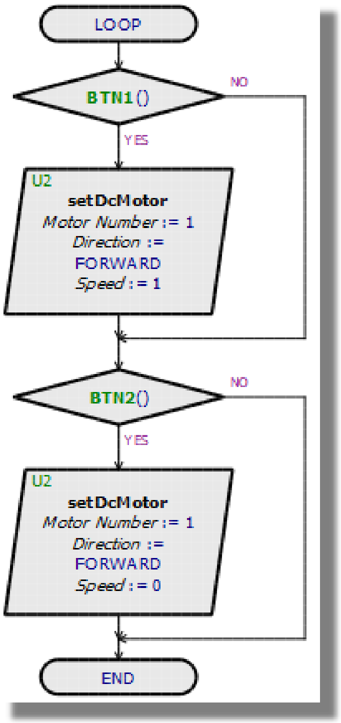
You should already have at least one DC motor. If not then make pick one from the libraries and make sure it is connected to motor 1 (M11,M12) of the hat. It should look something like this:



Correctly connected DC Motor on Motor 1 of the Hat.

Note that if you want the motor to appear in Visual Designer during simulation you will need to draw an Active Popup box around the motor.

Now you can drag the code blocks on. We want to change the speed of motor 1 each time a button is pressed. Button 1 will start the motor and button 2 will stop the motor. You won't need any code in setup. Your finished program should look like the following



Motor Control flowchart. Uses the sensor functions for the buttons to make decisions.

Testing and Debugging

Now that the project is ready to go we can test it out in simulation. The remainder of this tutorial is essentially a detailed look at simulation and stepping commands to make sure that you understand how to control a running simulation and how you might debug problems with your code.

The simulation is controlled from the animation control panel at the bottom left of the Proteus application.

Here you will find the play, step, pause and stop buttons. The play button will start the project or continue if the project is paused.

The step button will allow you to slowly step through the project pausing after every block of code.

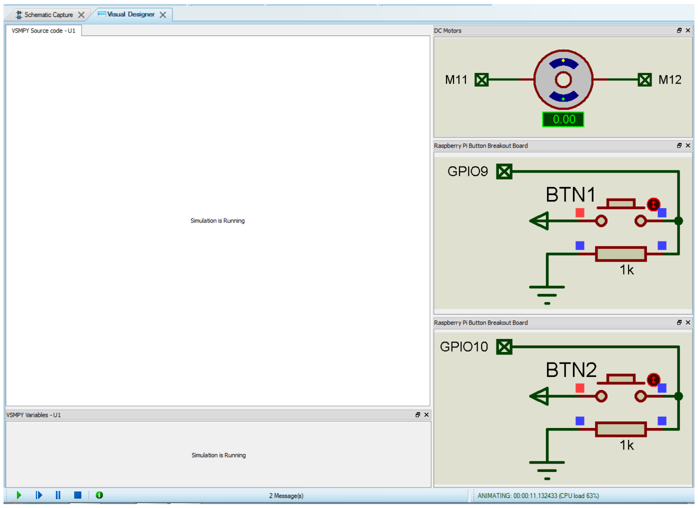
The pause button will pause time allowing you to look at variables, voltages and other elements.

The pause button will not stop the project and you can restart it with the play button.

The stop button will stop the whole project and allow you to edit and modify it.

Basic Single Stepping

We will start by simply pressing run, the program will start, and you will enter simulation mode. Press button one to start the motor going. Let the motor gain some speed then press button 2 to stop power to the motor. The motor will now slow down. Once you have tested this and confirmed that everything is working properly you can stop the program using the stop button.



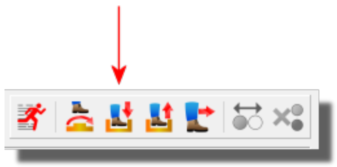
Simulation is Running

Simulation is Running

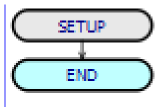
2 Message(s)

ANIMATING: 00:00:11.132433 (CPU load 63%)

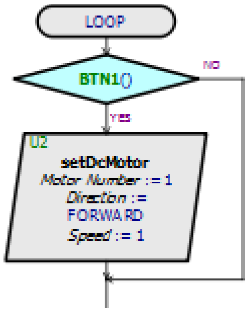
Next, press the step button once to start then pause the program. At this stage the program code is currently sat in the setup block. Next, pres the Step Into button to get to the end of the Setup block:



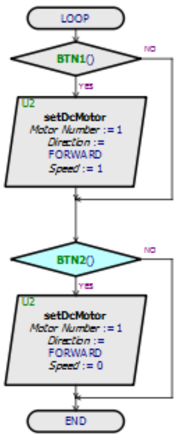
You will now see the End of setup light up. This shows that the setup function has ended.



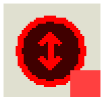
Pressing the Step Into button again will advance the program to the first decision block of the loop routine.

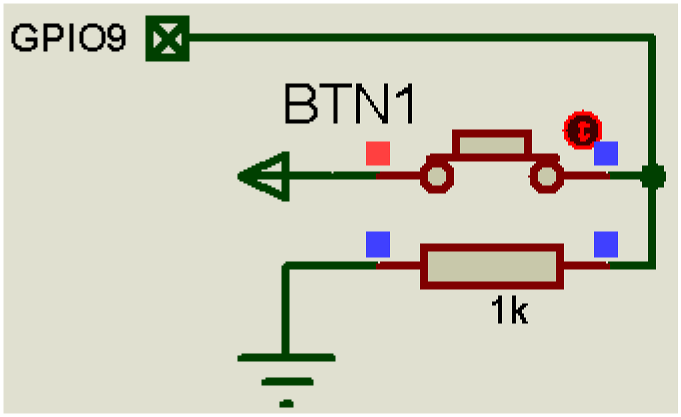


If button 1 is pressed the code will follow the 'Yes' line, if its not pressed it will follow the 'No' line. In our case the button is not pressed so when we keep stepping the code the program will move through to the next decision block. We'll then continue running around this loop routine until a button is pressed.



Finally, note that you can lock a button in place by pressing the red vertical arrows in the side of it. This way it will be held down even when you take your mouse away

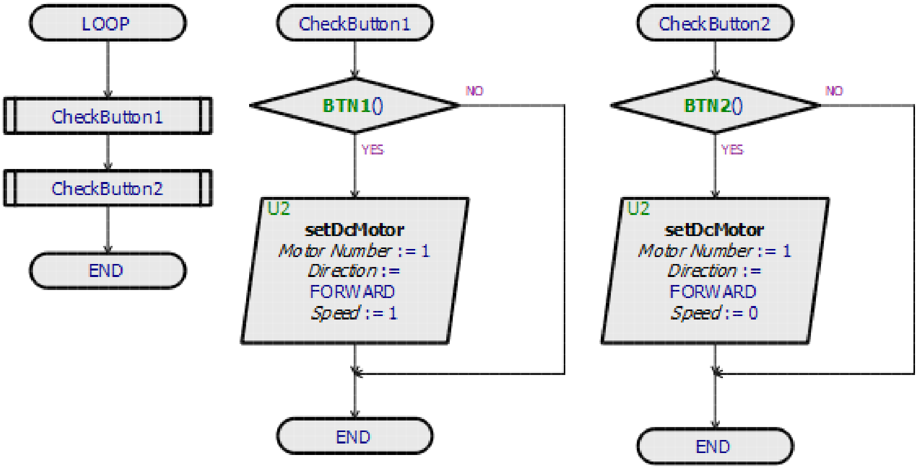




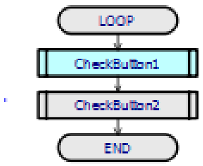
Stepping and Routines

Let's look now at how the step into and step out debugging commands work when we have sub-routines in our program. These basically navigate us in and out of the program stack.

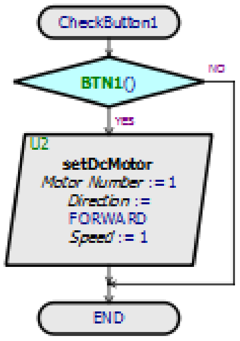
To help explain, we have modified our program to include a couple of sub-routine calls.



In this new program you can step through however it will only highlight the event call and not enter the event block. It will jump between CheckButton1, CheckButton2 and End without entering the two events. If you do want to enter an sub-routine wait until the event is highlighted.



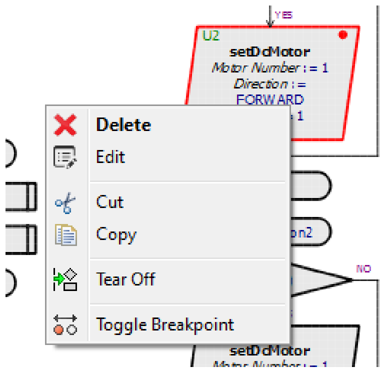
Then press step into to enter the routine and highlight the first code block in the routine.



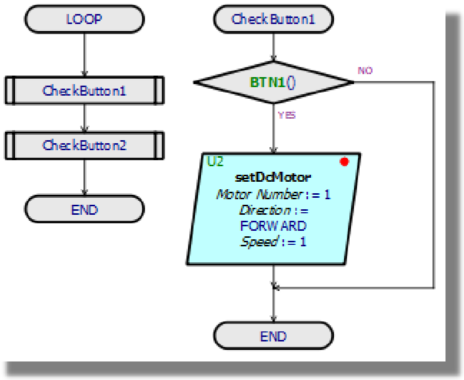
We could continue to step through the routine with our normal step commands or use the step out button to exit the routine and pause at the next code block following the return from the routine.

Breakpoints

Breakpoints allow you to stop the code when it reaches a specific block of code. To toggle breakpoints, enter Visual designer, right click on the block you wish to place the breakpoint and toggle breakpoint.



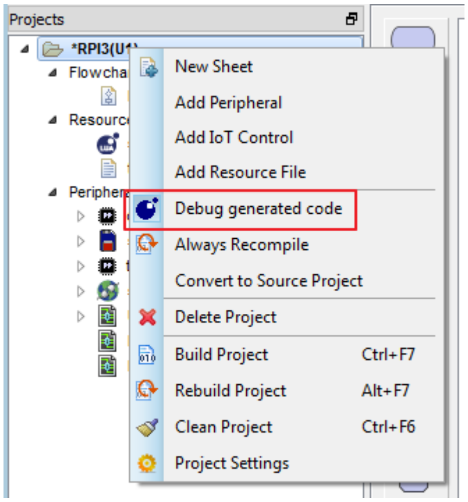
Now run the program, assuming button 1 is up nothing will happen. The code will run normally until you press button 1. Once button 1 has been pressed the code will pause on that block.



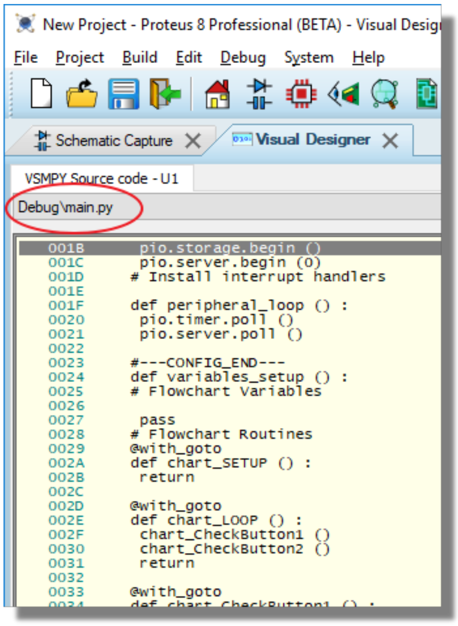
From here you can step through the code to debug further.

Debugging Generated Code

If you want to look deeper into what is happening in the code you can look directly at the code and not the blocks. Right click on the project menu and select debug generated code.



Now when you run the project the flowchart won't be present. Instead there is a blank screen. By pressing the step button once it will load up the python script. Now every time you step it will step line by line opposed to block by block.

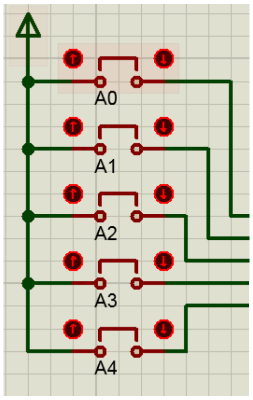


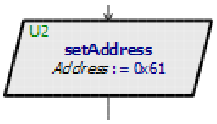
```
  pio.storage.begin ()
  pio.server.begin (0)
 # Install interrupt handlers

 def peripheral_loop () :
  pio.timer.poll ()
  pio.server.poll ()

 #---CONFIG_END---
 def variables_setup () :
 # Flowchart Variables

  pass
 # Flowchart Routines
 @with_goto
 def chart_SETUP () :
  return

 @with_goto
 def chart_LOOP () :
  chart_CheckButton1 ()
  chart_CheckButton2 ()
  return

 @with_goto
 def chart_CheckButton1 () :
```

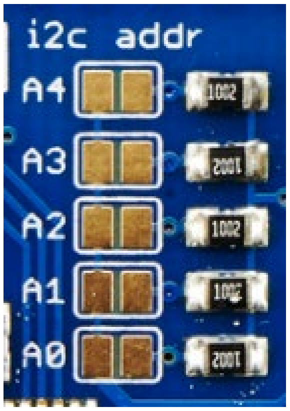
You should now be able to navigate and follow the code using the same debugging and stepping tools we discussed earlier. To navigate through different routines, use the dropdown box above the code.

Note about Addresses

Some hats are stackable such as this motor hat. When using multiple hats together you may need to change the I2C addresses. You can do this by using the jumpers. The Hat will have a base address, for the motor hat the base address is 0x60. The Jumpers work in binary so by connecting A0 the address will go to 0X61, and connecting A1 the address will go to 0x62 and so on. The motor hat has a possibility of 32 different addresses including address 0x60.







TURTLE SIMULTAION

INTRODUCTION

Overview

Robot turtles are extremely popular and fun to work with and are used extensively in both education and hobbyist circles. While the possibilities are endless, problem solving typically falls into three distinct categories:

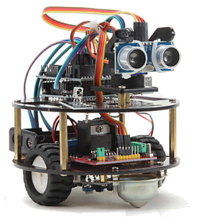
1) Line Following.
2) Obstacle Avoidance.
3) Maze Escape.

Proteus combines an electrical model of the turtle (motors, sensors, etc,) together with the Arduino baseboard and then presents a simple but flexible virtual world in which the turtle motion can be simulated. Control of the turtle in Visual Designer is vastly simplified by the use of high level methods in a flowchart program, while greater flexibility and complexity can be managed by writing Arduino C code directly. Regardless of the coding choice the simulation, test and debug takes place entirely inside the Proteus software before deploying to the physical turtle.

The following robot turtles are currently supported:

Funduino

Web Link



The Funduino turtle is a simple robot using an Arduino Uno baseboard alongside the motor driver board and 3 line hunter sensors. The ultrasonic range finder sits on top along with a stepper motor which allows you to swivel the sensor head. All of this is modelled in Proteus meaning that you can write your flowchart or firmware program and then test it in software.

Since the Funduino is the cheapest turtle we could find (around $50 at the time of writing) it's not surprising to find some limitations. In particular, there are no position encoders on the motor wheels and the line hunter sensors are digital.

Zumo

Web Link

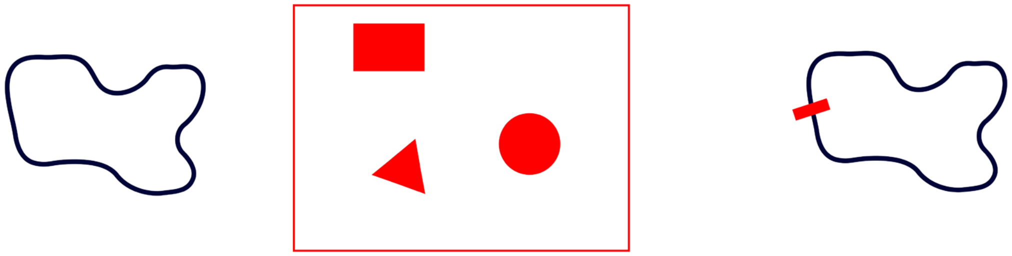
The Zumo robot for Arduino is an Arduino-controllable tracked robot platform. It includes two micro metal gearmotors coupled to a pair of silicone tracks, a stainless steel bulldozer-style blade, an array of six infrared reflectance sensors for line following or edge detection, a buzzer for simple sounds and music, a 3-axis accelerometer, magnetometer, and gyro for detecting impacts and tracking orientation.

The zumo is a more advanced turtle than the Funduino and can perform far better at line following and maze escape challenges but it does not have ultrasonic range finder and therefore is not as well suited to obstacle avoidance challenges.

Virtual Environment / Obstacle Map

In order to test a written program the robot turtle needs an environment in which to operate during a simulation run. While there are many sophisticated and complex physics engines on the market, the goal of our simulation is to check and debug the firmware program and so we have kept things very simple. Our virtual world is a picture drawn in MS Paint or similar in which the following simple rules apply:

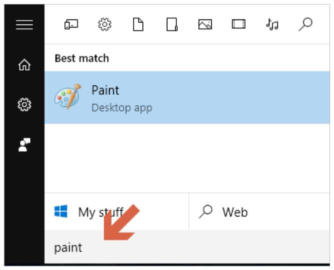
- 1 pixel is 1mm.
- Black is to be followed.
- Red is to be avoided.
- Green is a breakpoint and will pause your program !

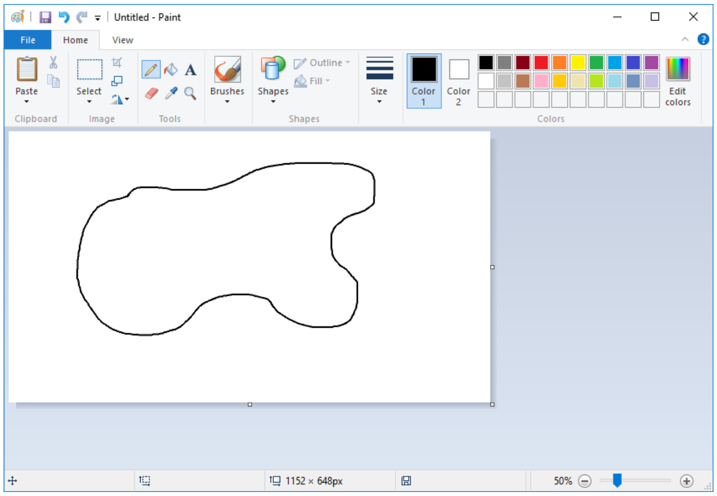


From Left to Right : Line Following example, Obstacle Avoidance example and Line Following with Obstacle example

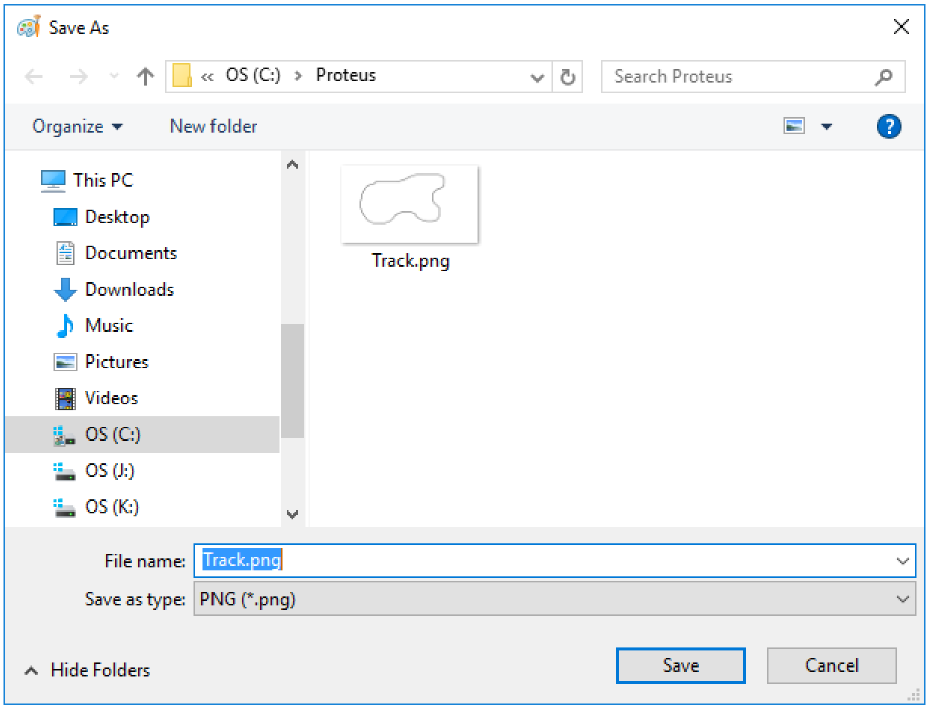
To create and apply an environment for simulation

1) Draw the required route / obstacles in the graphics package of your choice. Remember that the scale is 1 pixel to 1 mm so drawing a 5 pixel wide line will equate to 5mm in the real world. Width of the line being followed can affect the algorithm (e,g if all the sensors are never over the line) so it's important to give this some thought.





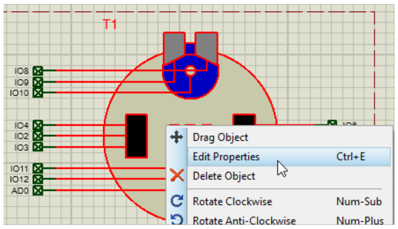
2) Save the graphic as a PNG file into the same directory as your Proteus project.

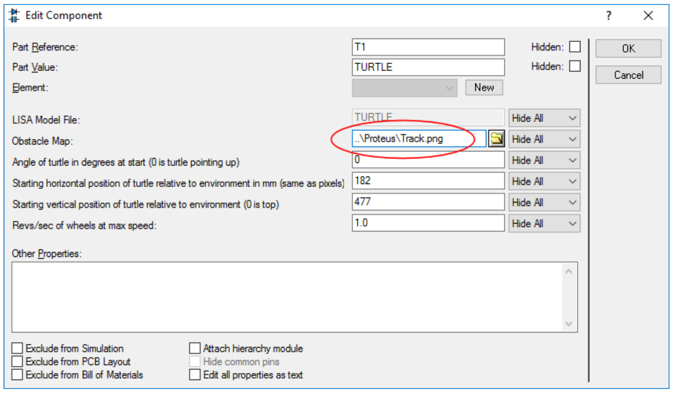


File name: Track.png

Save as type: PNG (*.png)

3) Edit the turtle component on your schematic and specify the graphic you have just saved as the obstacle map.





Edit Component

Part Reference:	T1	Hidden: ☐
Part Value:	TURTLE	Hidden: ☐
Element:		New
LISA Model File:	TURTLE	Hide All
Obstacle Map:	..\Proteus\Track.png	Hide All
Angle of turtle in degrees at start (0 is turtle pointing up)	0	Hide All
Starting horizontal position of turtle relative to environment in mm (same as pixels)	182	Hide All
Starting vertical position of turtle relative to environment (0 is top)	477	Hide All
Revs/sec of wheels at max speed:	1.0	Hide All

Other Properties:

☐ Exclude from Simulation
☐ Exclude from PCB Layout
☐ Exclude from Bill of Materials
☐ Attach hierarchy module
☐ Hide common pins
☐ Edit all properties as text

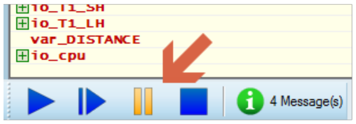
OK
Cancel

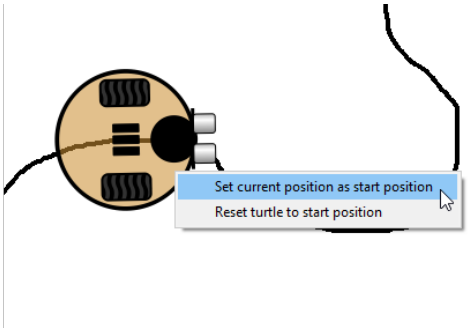
4) Run the simulation.

Positioning the turtle in the virtual world

- The turtle can be picked up and placed somewhere else on the map with the left mouse.
- You can rotate the turtle by holding the CTRL key and using the left mouse.

You can specify an initial position by first pausing the simulation, then positioning as required and finally, right clicking and specifying the current position as the new start position.




Set current position as start position
Reset turtle to start position

TUTORIAL - OBSTACLE AVOIDANCE

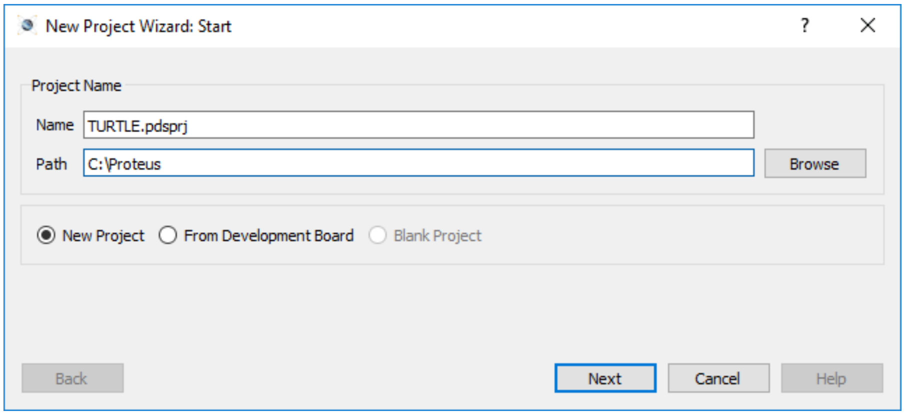
Introduction

This tutorial shows how to set up a Visual Designer project with the Funduino turtle, program it to avoid obstacles and then test and deploy to the real hardware.

While we are using the Funduino turtle for this tutorial we could easily use the Zumo turtle and change the challenge to something like a line follower. The principles are the same for both turtles.

Project Setup

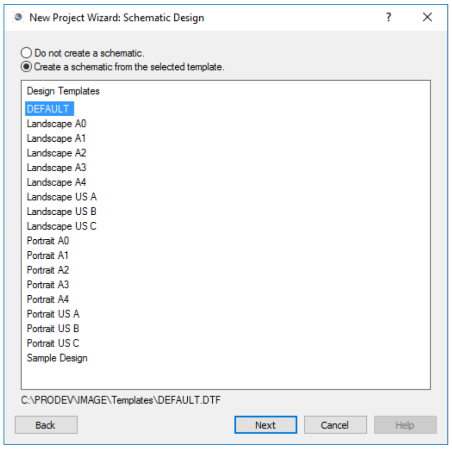
The first stage is to create the new project and add the Virtual Turtle.
1) Open the New Project Wizard from the Proteus home page and specify the project name and destination path as required.


New Project Wizard: Start

Project Name
Name: TURTLE.pdsprj
Path: C:\Proteus

(● New Project) ○ From Development Board ○ Blank Project

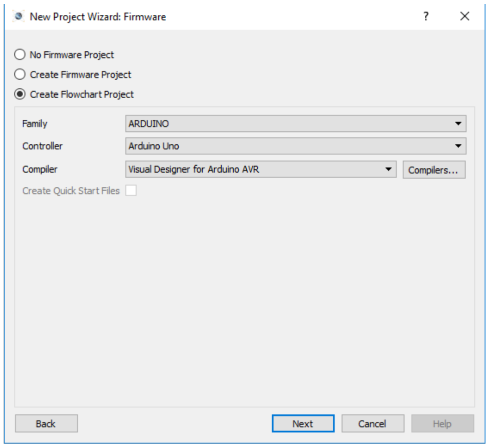
2) Create the schematic from the default template and then create a flowchart project with the Arduino Uno.


New Project Wizard: Schematic Design

○ Do not create a schematic.
● Create a schematic from the selected template.

Design Templates
DEFAULT
Landscape A0
Landscape A1
Landscape A2
Landscape A3
Landscape A4
Landscape US A
Landscape US B
Landscape US C
Portrait A0
Portrait A1
Portrait A2
Portrait A3
Portrait A4
Portrait US A
Portrait US B
Portrait US C
Sample Design

C:\PRODEV\IMAGE\Templates\DEFAULT.DTF

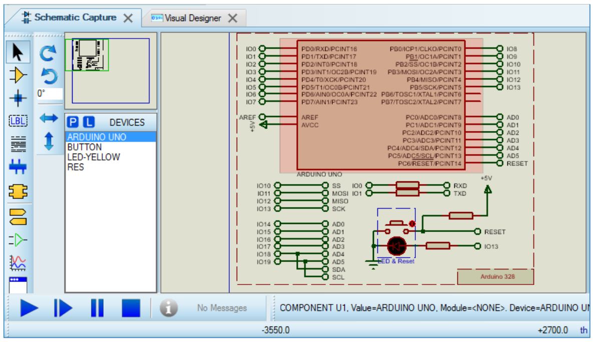

New Project Wizard: Firmware

○ No Firmware Project
○ Create Firmware Project
● Create Flowchart Project

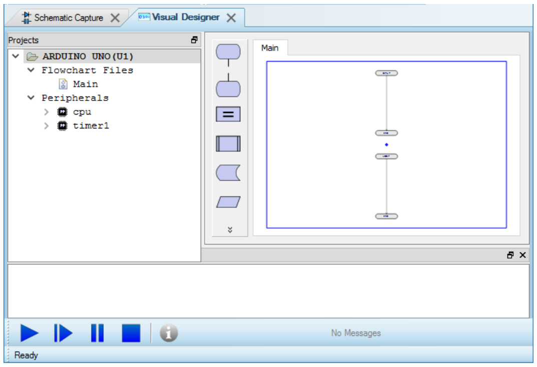
Family: ARDUINO
Controller: Arduino Uno
Compiler: Visual Designer for Arduino AVR
Create Quick Start Files ☐

If your license key includes Proteus PCB Design you will need to choose not to create a PCB on that page of the wizard. If your license does not include PCB design you will not see the PCB design page of the wizard.

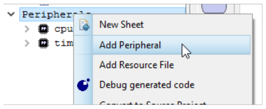
3) Continue and exit the wizard to create your project. You should see on the Visual Designer tab that you have a skeleton flowchart with the familiar Arduino setup and loop routines while on the schematic you'll find the Arduino Uno pre-placed.

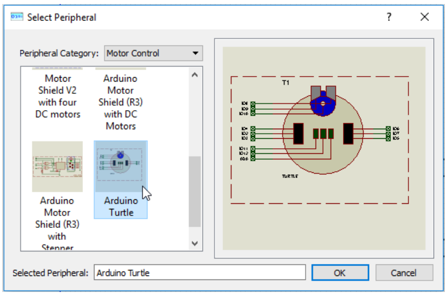


COMPONENT U1, Value=ARDUINO UNO, Module=<NONE>, Device=ARDUINO U



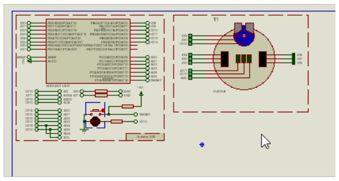
4) We now need to add the Virtual turtle to the project. We do this from the Add peripheral command in Visual Designer.

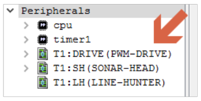



Select Peripheral
Motor Shield V2 with four DC motors
Arduino Motor Shield (R3) with DC Motors
Arduino Motor Shield (R3) with Stepper
Arduino Turtle
OK
Cancel

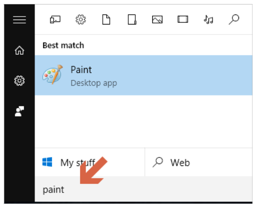
Peripheral Category: Motor Control

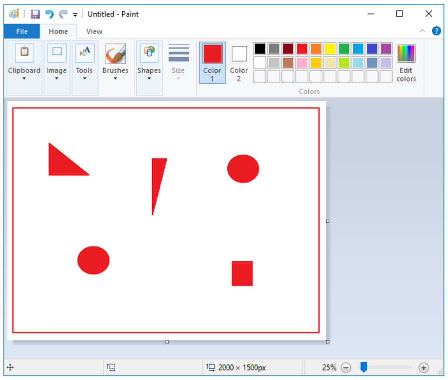
Selected Peripheral: Arduino Turtle

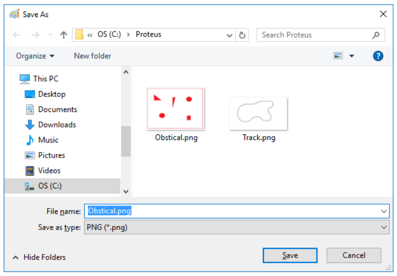




5) The next thing we need to do is create an obstacle map or virtual world for our turtle to simulate in. To do this we open MS Paint (click on START button and type in 'paint' in WIN8 or higher) or similar and draw some obstacles. The two vital points to remember are that anything red is considered an obstacle and that one pixel equals one millimeter. If we set our canvas to 2000 pixels wide by 1500 pixels that corresponds to around 2 meters by 1 1/2 meters in the real world which is plenty for an obstacle avoidance playground. After that, simply place a few red shapes inside a 'play area' to act as the obstacles and then save our as a PNG file.



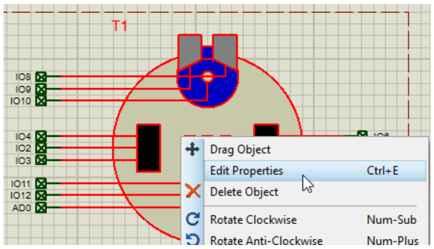


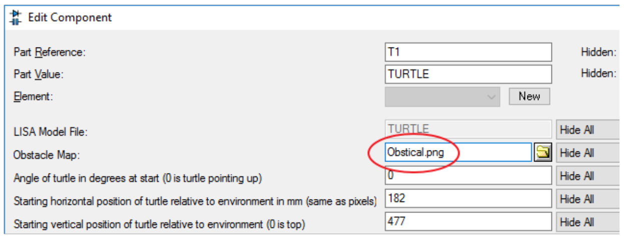


File name: Obstical.png
Save as type: PNG (*.png)

If we were setting up a line-following environment we would use black lines for the course to be followed and then shrink the canvas size around the map (the world doesn't need to be much bigger than the race-course).

6) The final task is to tell Proteus which obstacle map to simulate the turtle inside. We do this by editing the Virtual Turtle on the schematic and specifying the location of the obstacle map in the resulting dialogue form.





Edit Component

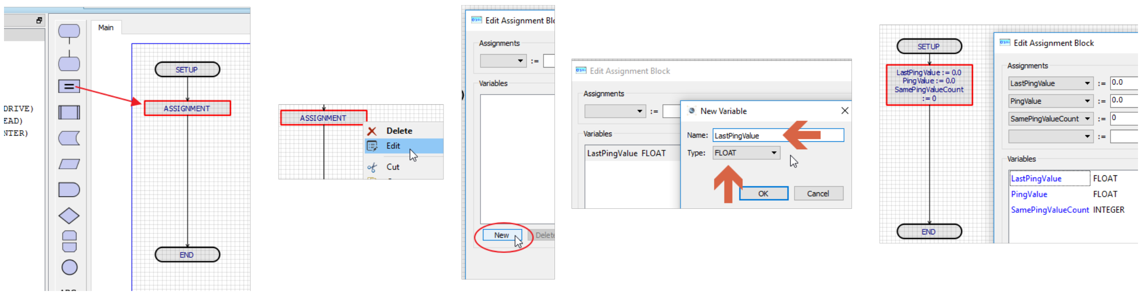
Part Reference:	T1	Hidden:
Part Value:	TURTLE	Hidden:
Element:		New
LISA Model File:	TURTLE	Hide All
Obstacle Map:	Obstical.png	Hide All
Angle of turtle in degrees at start (0 is turtle pointing up)	0	Hide All
Starting horizontal position of turtle relative to environment in mm (same as pixels)	182	Hide All
Starting vertical position of turtle relative to environment (0 is top)	477	Hide All

Firmware Design

When we added the Funduino turtle to our project above we received a number of high level methods that we can use on the flowchart to control the turtle. These are discussed individually in the Visual Designer methods topic which may prove a useful reference as we work through the program. We'll also assume here that the user is familiar with the basics of flowchart design in Visual Designer - these are covered in depth in other tutorials.

You'll find the completed program in the Obstacle Avoidance sample design so if you are comfortable with the firmware design you can load the completed version (File Menu -> Open Sample, type 'obstacle avoidance', select and open) and continue to the simulation and debug section of the tutorial.

Our basic algorithm will be to continuously send out sonar bursts (ping method), to compare against previous values (test if we are stuck) and then check distance to obstacle if not stuck (test if we need to turn now). The first step is to create some working variables and initialize them in the setup routine.



Name: LastPingValue
Type: FLOAT

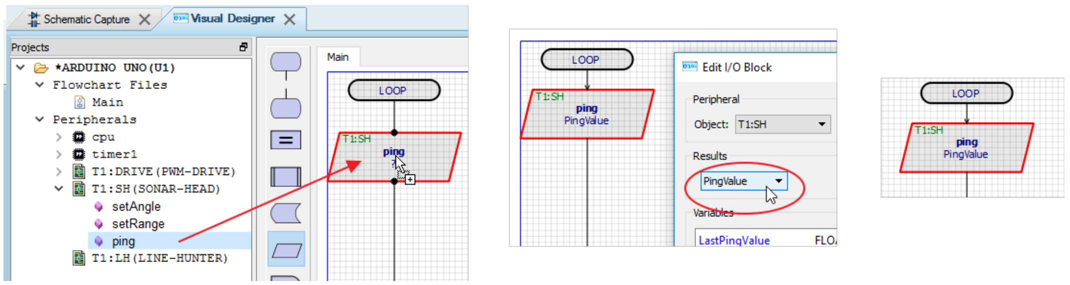
LastPingValue := 0.0
PingValue := 0.0
SamePingValueCount := 0

LastPingValue	:=	0.0
PingValue	:=	0.0
SamePingValueCount	:=	0

LastPingValue	FLOAT
PingValue	FLOAT
SamePingValueCount	INTEGER

Add 3 variables: LastPingValue (Float), PingValue (Float) & SamePingValueCount (Integer).

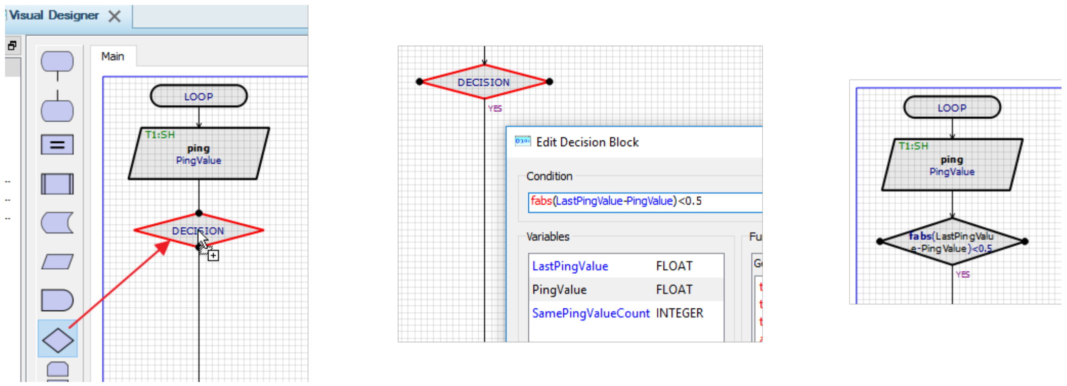
At the top of the loop routine we then want to set out ping method by dragging and dropping from the project tree and assigning the result to our pingValue variable.



```
*ARDUINO UNO(U1)
  Flowchart Files
    Main
  Peripherals
    cpu
    timer1
    T1:DRIVE(PWM-DRIVE)
    T1:SH(SONAR-HEAD)
      setAngle
      setRange
      ping
    T1:LH(LINE-HUNTER)
```

Object: T1:SH
Results: PingValue

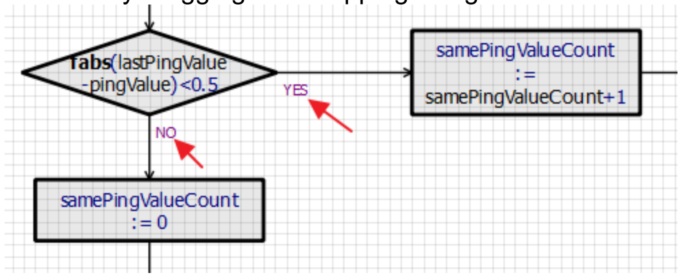
The result of this is the basis of our decision making. Our first test is to see whether the value is approximately the same as the last one. If it is we'll consider ourselves to be the same distance from the obstacle. What we need then is to place a decision block, subtract the current pingvalue from the previous ping value and we'll consider a difference of less than 0.5 to mean that our distance from the obstacle is unchanged. Note that we use the fabs() math function to return an absolute floating point value.



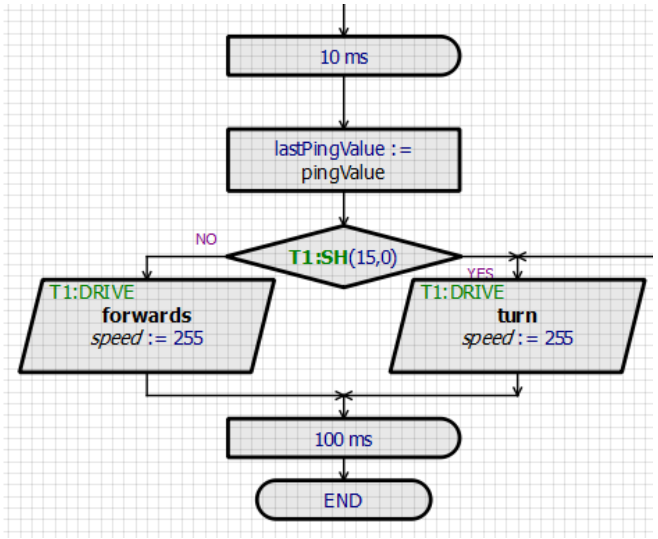
Condition: fabs(LastPingValue-PingValue)<0.5

LastPingValue	FLOAT
PingValue	FLOAT
SamePingValueCount	INTEGER

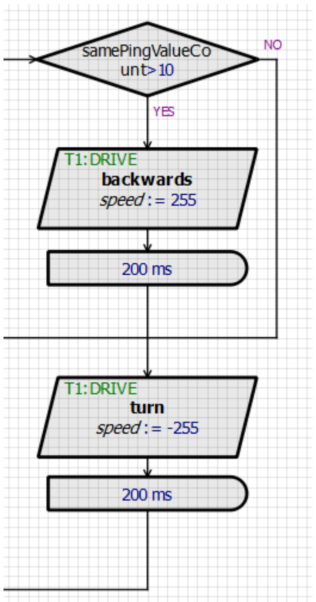
This branches our program into two forks, one where we are considered the same distance from the obstacle and the other where we are not. As in the image below, we do this by incrementing our samevaluecount variant in one direction (YES), or resetting it in the other direction (NO). This is done by dragging and dropping assignment blocks.


fabs(lastPingValue -pingValue)<0.5
YES
NO
samePingValueCount := samePingValueCount+1
samePingValueCount := 0

In the case where we are not stuck our job is fairly simple. Without going into detail we test whether we are closer than 15cm to the obstacle and we turn right if we are or keep going forwards if we are not. A delay is used at the end to allow a little driving time before the next iteration of the loop.


10 ms
lastPingValue := pingValue
T1:SH(15,0)
NO
YES
T1:DRIVE forwards speed := 255
T1:DRIVE turn speed := 255
100 ms
END

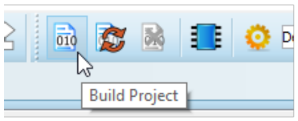
In the case where we may be stuck we need to test whether we have registered the same distance from an obstacle many times and, if so, we will reverse for a little bit and then turn right for a little bit before rejoining the main loop at the bottom. This is shown below.

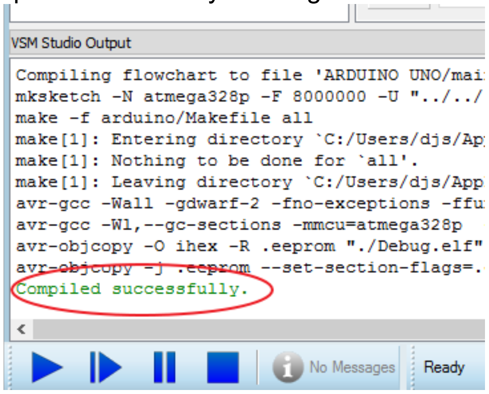


As said earlier, the completed code can be found in the sample in the 'Avoid Obstacle' sample design in the Visual Designer section.

Simulation and Debug

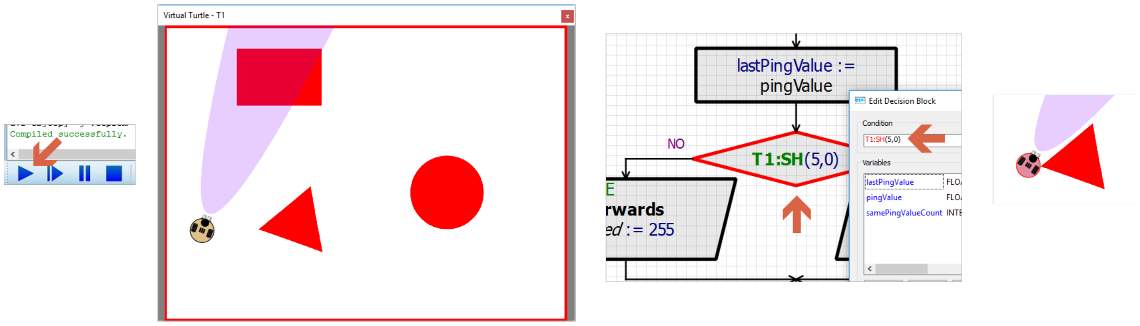
Having written our program we can compile via the Build Project command or the icon at the top of Visual Designer. Progress will be shown on the output window at the bottom of the editing window and we should eventually see a compiled successfully message.



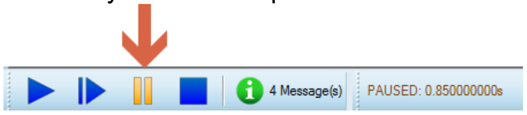

VSM Studio Output

```
Compiling flowchart to file 'ARDUINO UNO/mai
mksketch -N atmega328p -F 8000000 -U "../../
make -f arduino/Makefile all
make[1]: Entering directory `C:/Users/djs/Ap
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `C:/Users/djs/App
avr-gcc -Wall -gdwarf-2 -fno-exceptions -ffu
avr-gcc -Wl,--gc-sections -mmcu=atmega328p
avr-objcopy -O ihex -R .eeprom "./Debug.elf"
avr-objcopy -j .eeprom --set-section-flags=.
Compiled successfully.
```

Pressing the play button on the animation control panel will then start the simulation. You will see a popup window hosting the obstacle map that we created in the setup and a little turtle driving around inside it. The purple cone shows the range and scope of the sonar and the turtle behaviour is dictated by the program we have just written. For example, if we stop the simulation and change the distance test from 15cm to 5cm and then re-run the simulation you will see that the turtle is getting very close to obstacles before turning and will actually crash when approaching at certain angles.



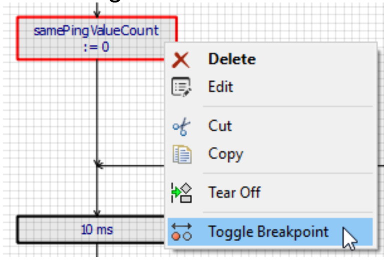
We can pause the simulation at any time via the pause button on the animation control panel



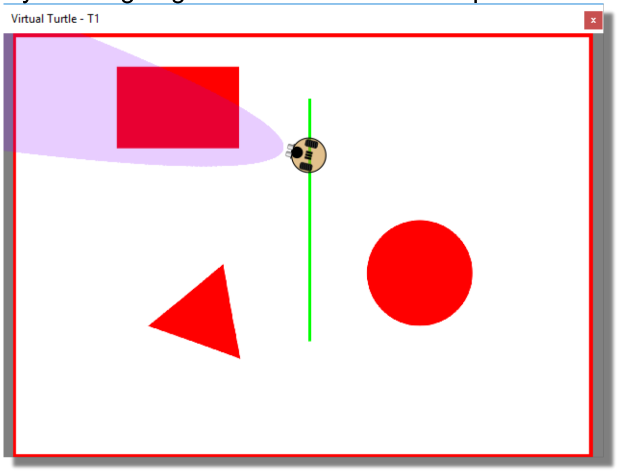
Setting Breakpoints

If we need to we can set breakpoints in several different ways:

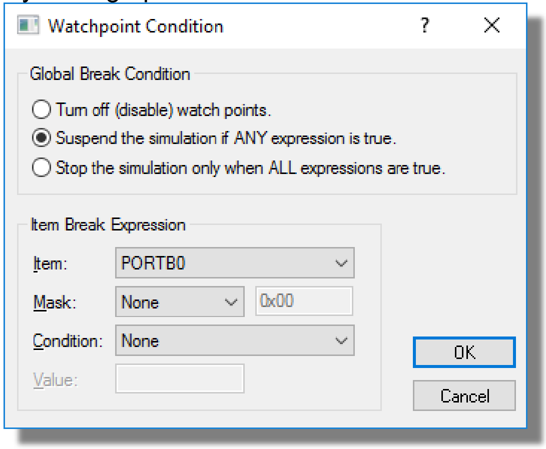
1) Via the right click context menu on one of the flowchart blocks.



2) By drawing in green on the obstacle map.



3) By setting up a condition on a watch window item.



Watchpoint Condition

Global Break Condition

- Turn off (disable) watch points.
- [✓ Suspend the simulation if ANY expression is true.]
- Stop the simulation only when ALL expressions are true.

Item Break Expression

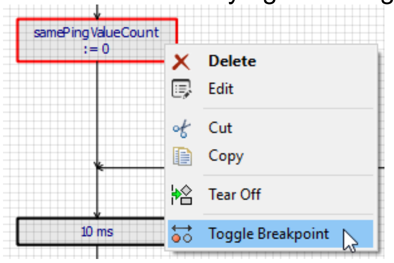
Item:	PORTB0
Mask:	None 0x00
Condition:	None
Value:	

Regardless of how the breakpoint is triggered, the reason that we set it is so that we can investigate the behaviour of the turtle more closely at that point. With Proteus, we are in the unique position that we can control time because the turtle is simulating under our control. This means for example that we can step through the code and the turtle will behave exactly as it would have were there no debugging in place (motors won't lose momentum etc.).

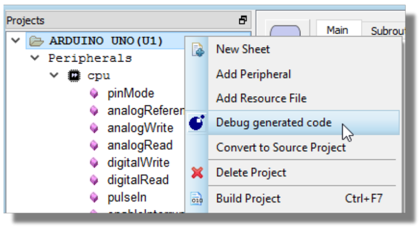
Single Stepping

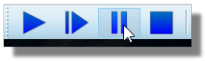
We can step through the code for our turtle program at multiple levels:

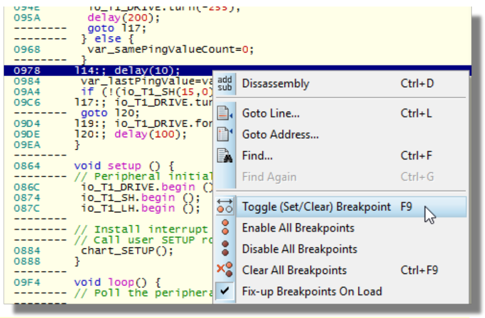
1) At flowchart level by right clicking and setting a breakpoint on the flowchart block.



2) At source code level by first stopping the simulation and then selection the debug generated code command from the project tree.



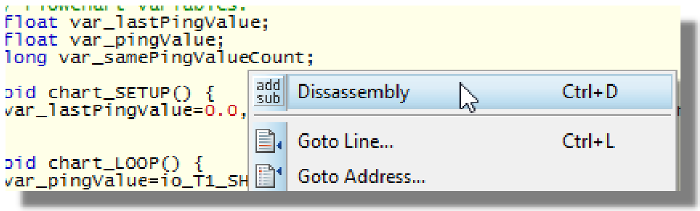




```
095A         delay(200);
--------     goto l17;
--------    } else {
0968         var_samePingValueCount=0;
--------    }
0978       l14:; delay(10);
0984        var_lastPingValue=va
09A4        if (!(io_T1_SH(15,0)
09C6       l17:; io_T1_DRIVE.tur
--------    goto l20;
09D4       l19:; io_T1_DRIVE.for
09DE       l20:; delay(100);
09EA       }
--------
0864       void setup () {
--------   // Peripheral initial
086C        io_T1_DRIVE.begin ()
0874        io_T1_SH.begin ();
087C        io_T1_LH.begin ();
--------
--------   // Install interrupt
--------   // Call user SETUP ro
0884        chart_SETUP();
0888       }
--------
09F4       void loop() {
--------   // Poll the periphera
```

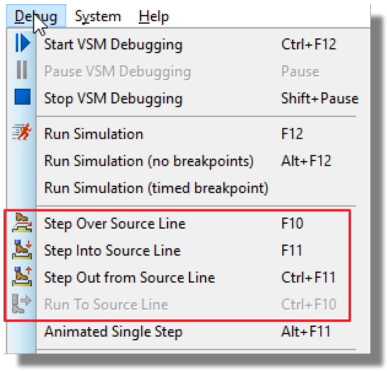
⚠ Make sure you use the Debug Generated Code command and not the Convert to Source Project command here. This will switch your project from flowchart to source code but it is **not** a reversible operation.

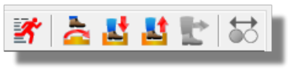
3) At machine code level by selecting the disassembly command from the context menu.



```
float var_lastPingValue;
float var_pingValue;
long var_samePingValueCount;

oid chart_SETUP() {
var_lastPingValue=0.0,

oid chart_LOOP() {
var_pingValue=io_T1_SH
```

In either case the single step commands can be found on the debug menu or via the icon strip at the right hand side of the main window.

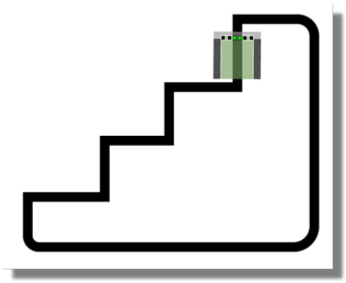

Debug
System
Help
Start VSM Debugging Ctrl+F12
Pause VSM Debugging Pause
Stop VSM Debugging Shift+Pause
Run Simulation F12
Run Simulation (no breakpoints) Alt+F12
Run Simulation (timed breakpoint)
Step Over Source Line F10
Step Into Source Line F11
Step Out from Source Line Ctrl+F11
Run To Source Line Ctrl+F10
Animated Single Step Alt+F11

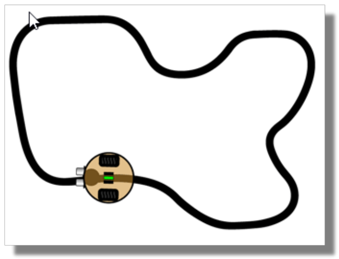


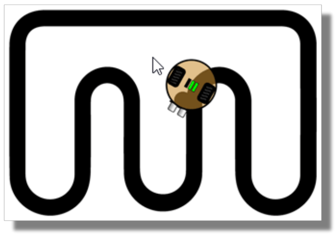
You can also use the shortcut keys F10 for step over and F11 for step into.

Changing the Virtual Environment

One of the neatest things about virtual turtle simulation is just how easy and fast it is to test your program against challenges of different sizes and complexity. All you have to do is change the picture and then enter the name in the properties of the component. Here's a few line following examples

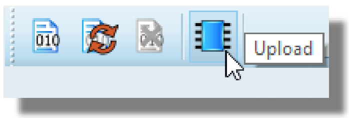






Programming the Physical Turtle

Proteus includes built in support for the common AVRDUDE programmer so that you can program the real hardware directly from the Proteus software.



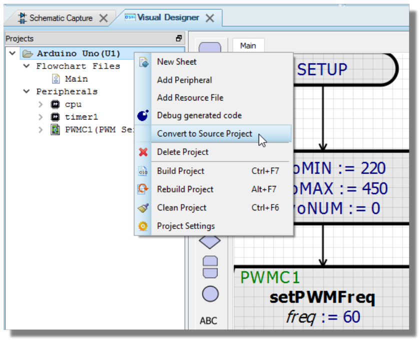
This process is discussed in full in the Visual Designer help documentation linked below.

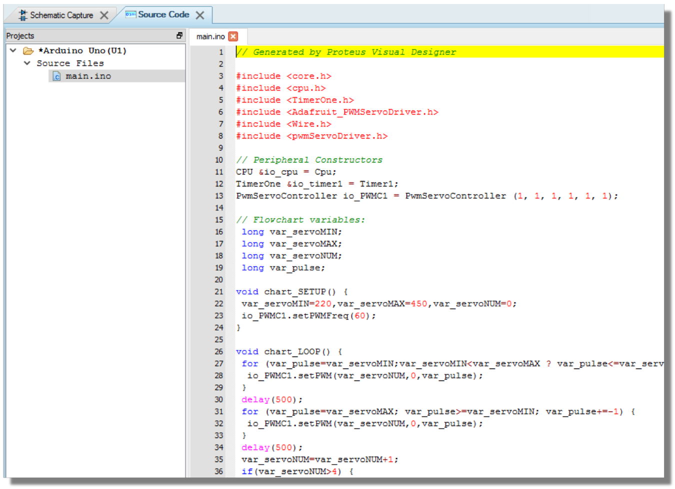
VISUAL DESIGNER COMMAND REFERENCE

Introduction

When working in Visual Designer, a lot of the complexity of driving the turtle is provided by several high level methods that abstracts much of the electronics complexity, leaving the user to focus on the control algorithms. Since each turtle is made up of different pieces the driver methods in Visual Designer for each are also different. These are discussed below.

Advanced users can simply convert their project to a source code project via the context menu command on the project tree and then write their programs in Arduino C.





```
// Generated by Proteus Visual Designer

#include <core.h>
#include <cpu.h>
#include <TimerOne.h>
#include <Adafruit_PWMServoDriver.h>
#include <Wire.h>
#include <pwmServoDriver.h>

// Peripheral Constructors
CPU &io_cpu = Cpu;
TimerOne &io_timer1 = Timer1;
PwmServoController io_PWMC1 = PwmServoController (1, 1, 1, 1, 1, 1);

// Flowchart variables:
 long var_servoMIN;
 long var_servoMAX;
 long var_servoNUM;
 long var_pulse;

void chart_SETUP() {
 var_servoMIN=220,var_servoMAX=450,var_servoNUM=0;
 io_PWMC1.setPWMFreq(60);
}

void chart_LOOP() {
 for (var_pulse=var_servoMIN;var_servoMIN<var_servoMAX ? var_pulse<=var_serv
  io_PWMC1.setPWM(var_servoNUM,0,var_pulse);
 }
 delay(500);
 for (var_pulse=var_servoMAX; var_pulse>=var_servoMIN; var_pulse+=-1) {
  io_PWMC1.setPWM(var_servoNUM,0,var_pulse);
 }
 delay(500);
 var_servoNUM=var_servoNUM+1;
 if(var_servoNUM>4) {
```

⚠ Once a program is switched to be a source code project from a flowchart project you cannot switch back.

This may have some advantages as users can access the on board peripherals (e.g. gyro, accelerometer) at the register level but will undoubtedly be more challenging. At the end of the day, the Proteus simulation is of the real hardware however which means that users can develop their programs in whatever way they want. In particular, it means that 3rd party libraries and source code examples on the internet can be added, used and tested in a Proteus code project.

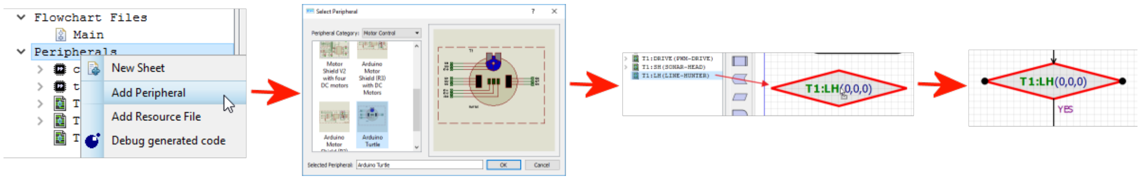
Funduino Turtle

Control of the turtle falls into three categories :

- Using the line hunter sensors to detect position relative to a line..
- Using the sonar head to detect obstacles.
- Driving the two DC motors attached to the wheels.

Line Hunter Control

There are three sensors underneath the Funduino turtle. These provide a digital response to the circuit based on whether they detect the line or not. In Visual Designer we query these sensors with a single decision block called the sensor function. You use the sensor function by dragging and dropping directly from the peripheral in the project tree as shown below:

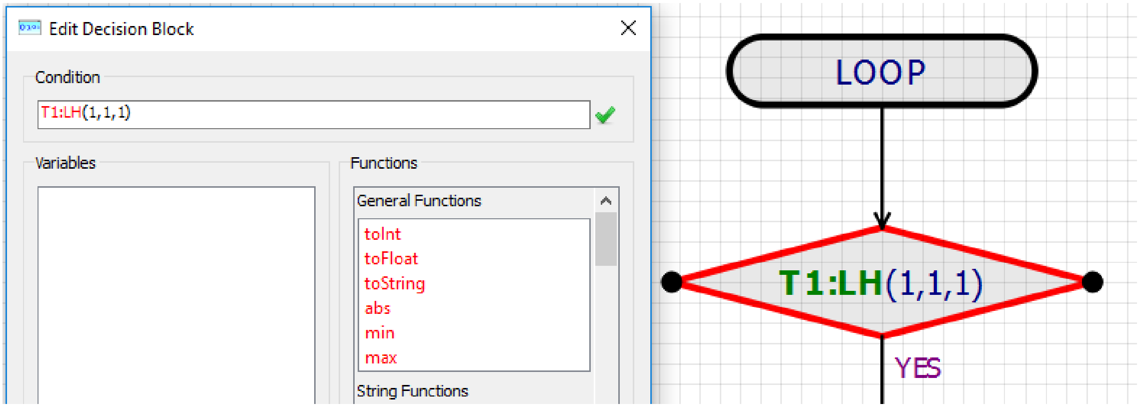


We then need to edit the decision to test the sensors based on a set of conditions we wish to be true. There are three parameters corresponding to the left, center and right sensors and for each one we can test as follows:

```
1  : Must be TRUE.

0  : Must be FALSE.

-1 : Don't care.
```

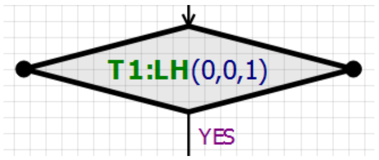
Examples

If we want to test that all 3 sensors are over the line we would set all three parameters to 1.



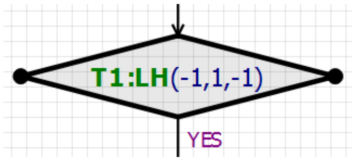
All three sensors can see the line

When this decision is true you know that the turtle is fully over the line and you may choose to attempt some damping or correction if the turtle is wobbling.
If we want to check when we need to make a sharp turn right then we are looking at the case when only the right hand sensor is picking up the line.



Only the right hand sensor sees the line

If we want to simply drive forwards then testing that the middle sensor is over the line is likely sufficient, provided we do so after other relevant conditions have been tested.

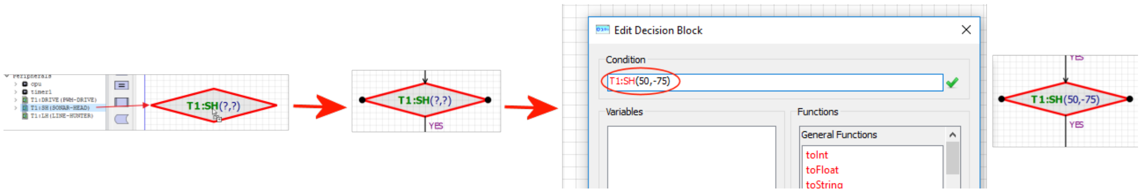


Sonar Head Control

The ultrasonic sonar sends out short bursts (pings) to detect obstacles and is mounted on a rotatable unit. This allows users to first position the sonar head and then check for obstacles inside a certain range. There are three methods and a sensor function for the sonar head.

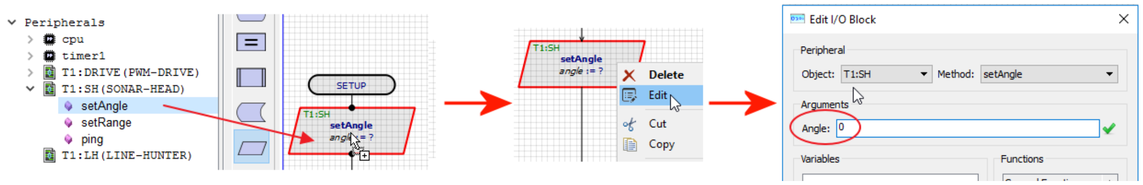
Sensor (Sonar) Function

The sensor function is a decision block that allows you to query the peripheral and it is used by dragging and dropping from the peripheral itself in the Project Tree. In the case of the Sonar Head the decision block takes two arguments for distance and head angle. It returns true if an object is detected within the specified distance at the given head angle. The example below would return TRUE if an obstacle was within 50cm on the left of the turtle.



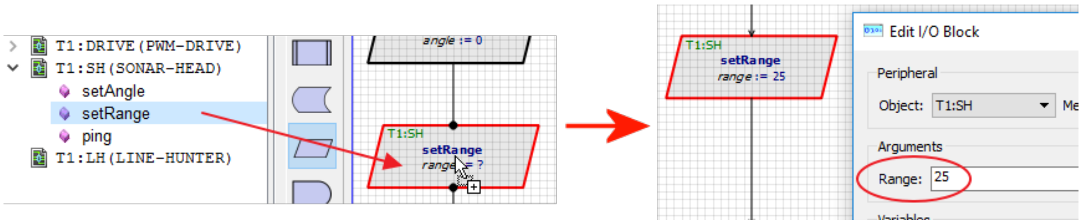
setAngle() Method

The setAngle() method allows you to position the sonar head. For most applications this will be added to the setup routine with a value of 0 (start by looking straight ahead) and then if required through program execution.



setRange() Method

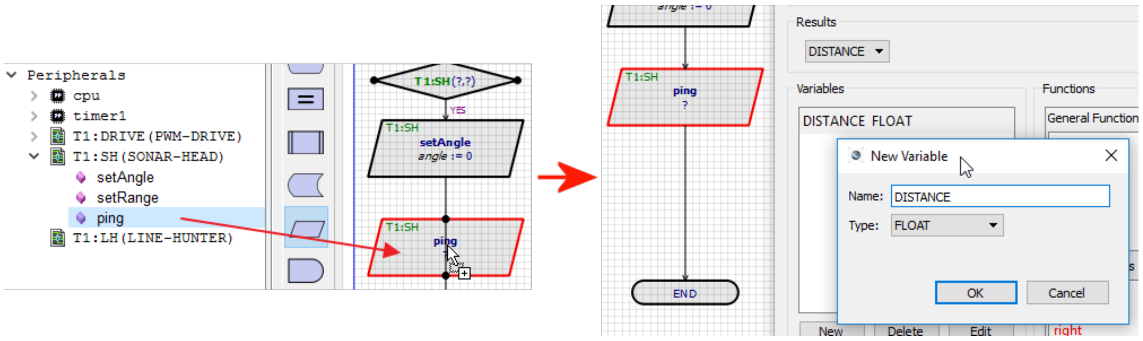
The setRange() method allows you to specify the maximum range in which you want to detect obstacles. This is significant as it determines the time-frame in which the firmware will consider the reply to a ping to count as a reflection from an obstacle. You would normally set this (as small as possible) inside the setup routine and then adjust as required in the program.



ping() Method

The ping() method fires a sonar burst, converts time to response into distance in cm and returns either that value or -1 if nothing is detected. After placement, the user must edit the method call and assign the return value to a variable.

⚠ This variable must be a FLOAT variable otherwise it will not appear in the 'Results' drop down list.

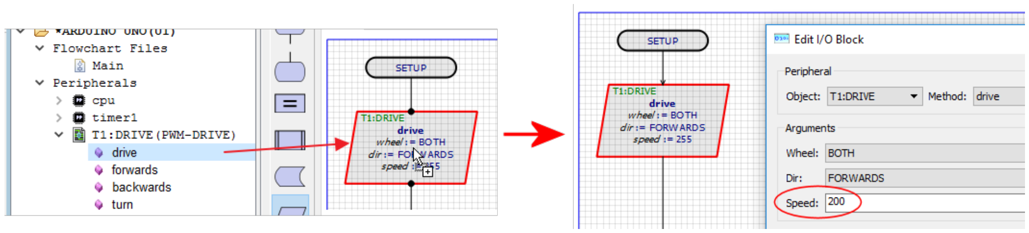


Motor Drive Control

The Drive methods give control over the left and right wheels of the motor. The user is presented with a series of simple control methods as follows.

Drive() Method

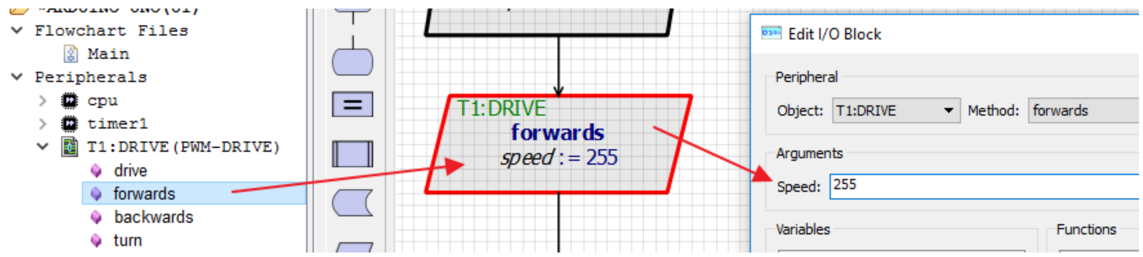
The drive method allows you to specify the wheel(s) to control, the direction of travel and the speed. The latter is a value from 0 -> 255 and represents the duty cycle of the PWM drive signal. For example, you might set up full speed ahead as both wheels forwards at around speed 200.



Peripheral
Object: T1:DRIVE Method: drive
Arguments
Wheel: BOTH
Dir: FORWARDS
Speed: 200

Forwards() Method

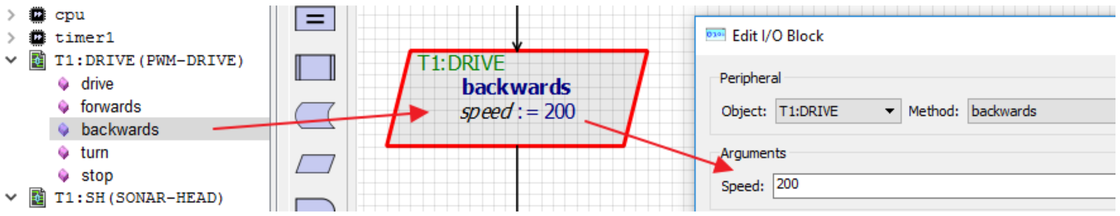
With the forwards() method the direction of travel is explicit and so all that is needed is the speed value.



Peripheral
Object: T1:DRIVE Method: forwards
Arguments
Speed: 255

Backwards() Method

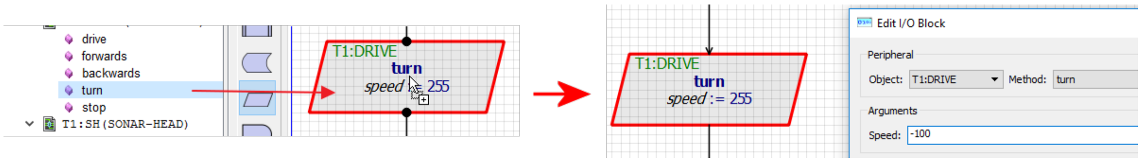
As with the forwards() method, all that is needed is the speed at which to reverse.



Peripheral
Object: T1:DRIVE Method: backwards
Arguments
Speed: 200

Turn() Method

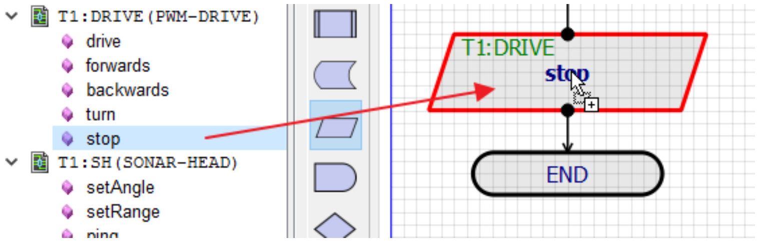
The turn() method simplifies the turning of the turtle by allowing you to specify a speed of turn. Negative values turn left while positive values turn right and the magnitude of the value is the speed of turn. In practise, this will execute a turn by turning one wheel forwards and the other backwards at the specified speed. It can therefore be thought of as a sharp turn where a soft turn involves stopping one wheel and turning the other.



The same effect can be achieved with greater precision by using the drive() command twice and controlling speed of wheels independently.

Stop() Method

The stop() method immediately stops drive of both wheels.



Zumo Turtle

Control of the Zumo turtle in Visual Designer falls into three basic categories :

- Using the line hunter sensors to detect position relative to a line.
- Using the gyro and compass to position or orientation the turtle.
- Driving the two motors attached to the tracks.

Line Follower Control

The Zumo comes with a mounted array of six IR reflectance sensors that allows the Zumo to detect contrasts in reflectivity directly beneath its blade, which can be used for following lines or detecting edges. Each reflectance sensor consists of an IR emitter coupled with a phototransistor that responds based on how much emitter light is reflected back to it. Unlike the Funduino therefore, the Zumo is well suited to Proportional Integral Derivative (PID) control algorithms when line following.

readLinePos() method

- Takes no parameters.

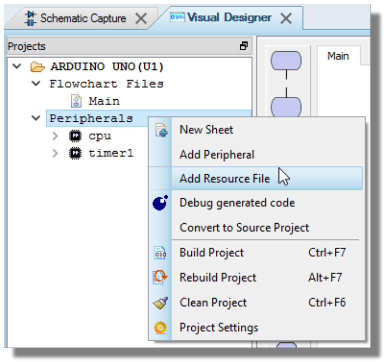
- Returns an integer between 0 and 6000 indicating the position of the line underneath the six IR sensors.

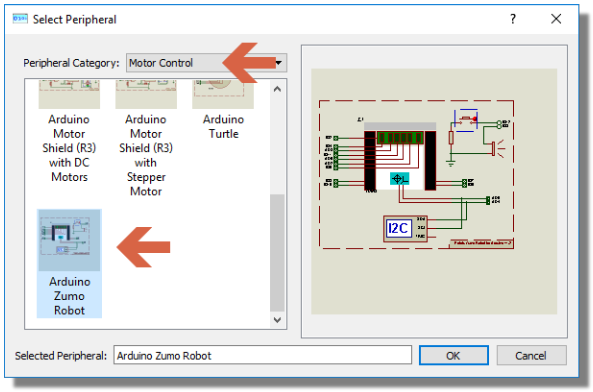
This method first reads the calibrated value from each of the six sensors into an array. Values per sensor range from 0 to 1000 where and are a measure of the reflectance in abstract units, with higher values corresponding to lower reflectance (e.g. a black surface or a void). An estimated position of the robot with respect to the line is then calculated used a weighted average of the sensor values multiplied by 1000. This means that a return value of 0 means that the line is directly under sensor 0, a return value of 1000 indicates that the line is directly below sensor 1 and so on. Intermediate values indicate that the line is between two sensors. The formula used is :

$$\frac{0*\text{value0} + 1000*\text{value1} + 2000*\text{value2} + \ldots\ldots}{\text{value0} + \text{value1} + \text{value2} + \ldots}$$

Example:

1) Add the Zumo turtle to the Visual Designer Project.

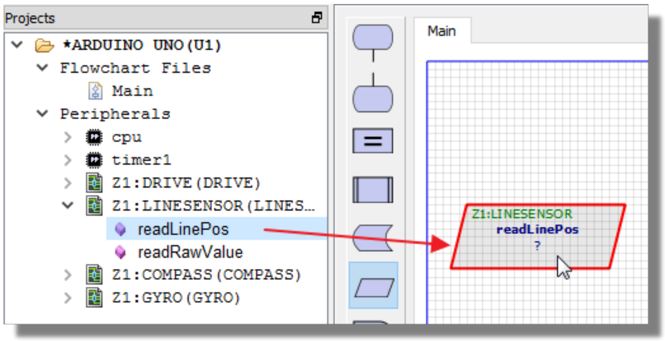



Select Peripheral
?
Arduino Motor Shield (R3) with DC Motors
Arduino Motor Shield (R3) with Stepper Motor
Arduino Turtle
Arduino Zumo Robot
I2C
OK
Cancel

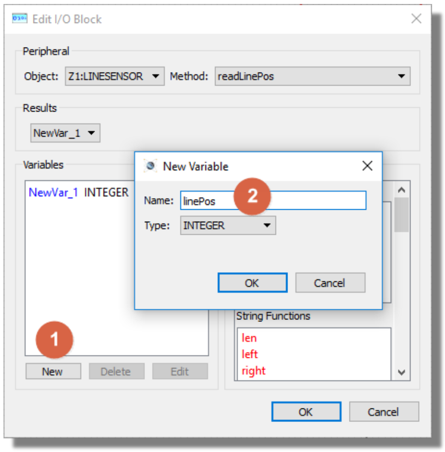
Peripheral Category: Motor Control

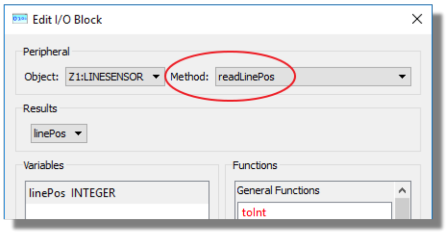
Selected Peripheral: Arduino Zumo Robot

2) Drag and drop the readLinePos() method onto the flowchart.



3) Edit the method and create an integer variable (e.g. linePos). Set the variable to store the return value from the method.





If you get a value of 0 the line is underneath the far left hand sensor, 1000 means the line is flush underneath the second sensor from the left, 2500 means that the line is right in the middle of the turtle between sensors 3 and 4 and a value of 6000 means that the line is underneath the far right sensor.

readRawValue() method

- Takes an integer parameter corresponding to the index of the IR sensor you want to read (0..5).

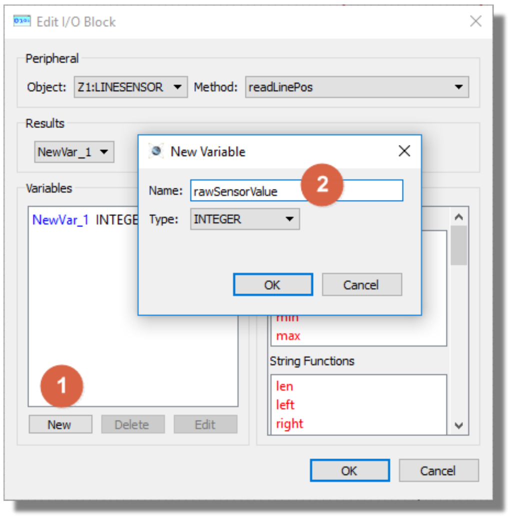
- Returns the raw (uncalibrated) value from the specified sensor.

The number returned is based on the reflectance. The higher the number, the less reflectance so a black line gives a high value. The numbers are not calibrated meaning that care is required with return values as they would change under different conditions in the real world.

Example:

1) Drag and drop the readRawValue() method onto the flowchart.

2) Edit the method and create an integer variable (e.g. rawSensorValue). Set the variable to store the return value from the method.



Edit I/O Block

Peripheral

Object: Z1:LINESENSOR Method: readLinePos

Results

NewVar_1

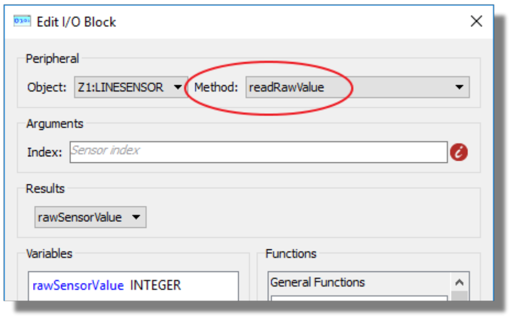
Variables

NewVar_1 INTEGE

New Variable

Name: rawSensorValue

Type: INTEGER



Edit I/O Block

Peripheral

Object: Z1:LINESENSOR Method: readRawValue

Arguments

Index: Sensor index

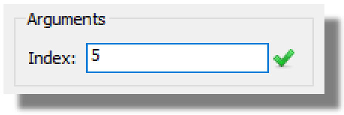
Results

rawSensorValue

Variables

rawSensorValue INTEGER

3) Set the index parameter to specify which sensor you want to query. 0 is the left hand sensor, 5 is the right hand sensor.



Arguments

Index: 5

Compass Control

The Zumo includes an integrated LSM303D chip which can be controlled by the following Visual Designer methods:

readMagneticField() method

*- Takes three floating point parameters (void readMagneticField (float *magX, float *magY, float *magZ))*

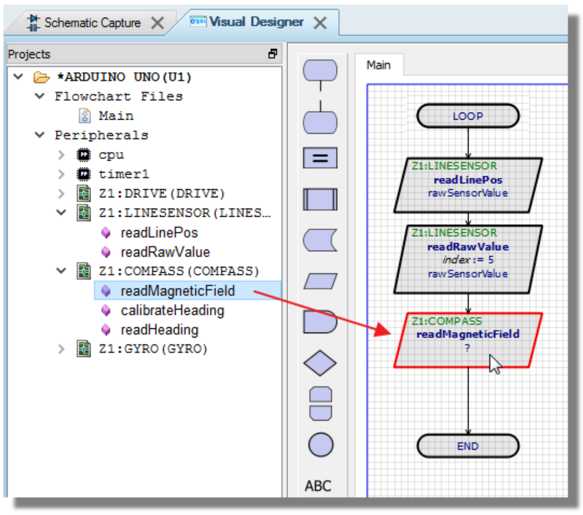
- Returns the X,Y,Z components of the magnetic field in units of Gauss.

This method passes in pointers to three floating point variables and the internal routine then populates them with the X,Y and Z components of the magnetic field. Note that the compass is widely inaccurate in relative and absolute terms. I.e. the vector is offset from the origin and North is impossible to determine without calibration.

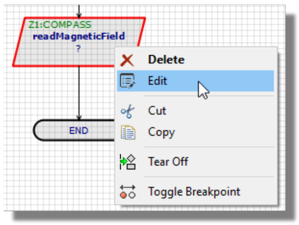
⚠ In the read device the compass gets a lot of interference from the motors, batteries, PCB, and its surroundings, so it is not generally useful for precision navigation. However, having been calibrated it can be used for rough orientation measuring in many environments.

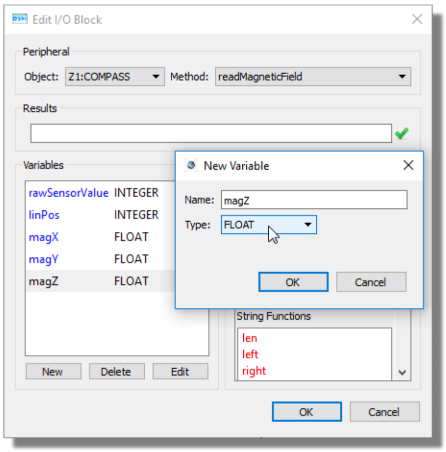
Example:

1) Drag and drop the method onto the flowchart.



2) Edit the method and create three new variables (e.g. magX, magY, magZ)




Edit I/O Block

Peripheral

Object: Z1:COMPASS Method: readMagneticField

Results

Variables

rawSensorValue	INTEGER
linPos	INTEGER
magX	FLOAT
magY	FLOAT
magZ	FLOAT

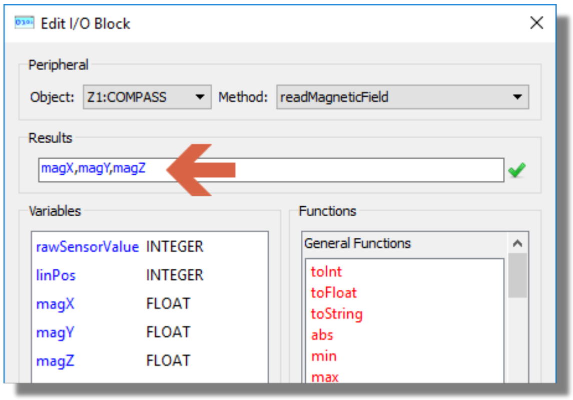
New Variable

Name: magZ

Type: FLOAT

String Functions: len, left, right

3) Enter the three variables in to the Results field separated by a comma. This passes the three variables as parameters to the method.


Edit I/O Block

Peripheral

Object: Z1:COMPASS Method: readMagneticField

Results

magX,magY,magZ

Variables

rawSensorValue	INTEGER
linPos	INTEGER
magX	FLOAT
magY	FLOAT
magZ	FLOAT

Functions

General Functions: toInt, toFloat, toString, abs, min, max

calibrateHeading() method

- Takes no parameters

- Returns no values

This function uses the drive method to spin the Zumo on the spot and take multiple magnetic readings. It takes the minimum and maximum X and Y magnetic values found and uses these to calculate the centre. It stops spinning when no new min or max values are set over a period of 500ms. **Before calling this function it assumes the Zumo is facing North**. Thus is uses the first magnetic reading to work out the angle of the vector that points to North.

readHeading() method

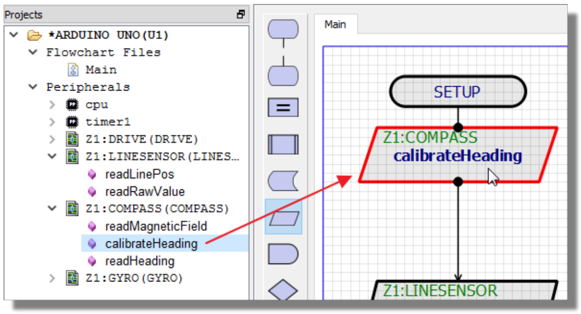
- Takes no parameters

- Returns a float representing the heading.

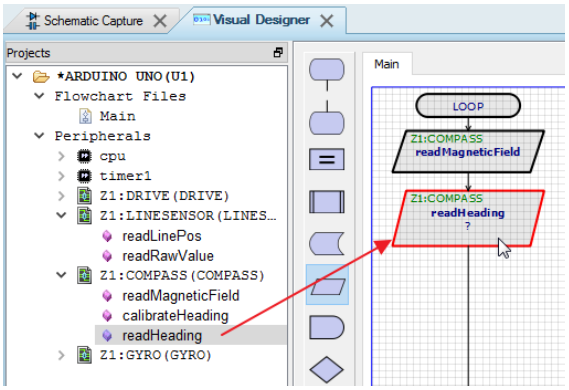
This function requires calibrateHeading() to have been called. If it hasn't then -1.0 is returned. Otherwise the heading angle in degrees limited to [-180.0, +180.0] is returned (0 is North).

Example:

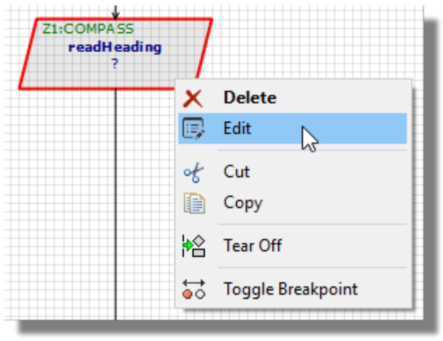
1) Drag and drop the calibrateHeading() method onto the flowchart. This is best done at the very beginning where the Zumo can be pointed north so the setup routine makes sense.

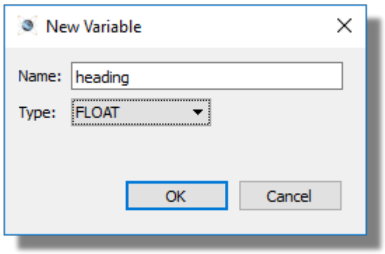


2) Drag and drop the readHeading() method anywhere after the calibration routine.



3) Edit the readHeading() method and create a variable of type float (e.g. heading) to receive the function return.





Name: heading
Type: FLOAT

A value of 0 means the Zumo is pointing north, a value of 180 means South, a value of -90 means West and so on.

Gyro Control

The Zumo (V1.2) includes an integrated L3GD20H 3-axis gyroscope that can be used to track rotation and, together with the LSM303D effectively provides an inertial measurement unit that programs can use. The Visual Designer wraps some of this functionality in two high level methods:

readAngularAcc() method

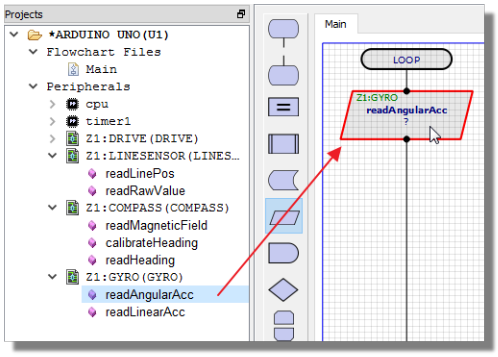
*- Takes three floating point parameters (void readAngularAcc(float *angX, float *angY, float *angZ))*

- Returns the X,Y,Z components angular acceleration in units of degrees per second.

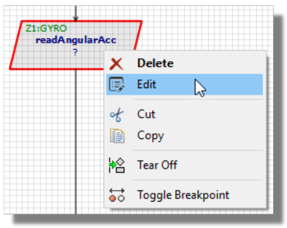
This function returns the angular acceleration of the Zumo in three dimensions. Note however, that due to the 2D nature of the simulation environment, only the angZ parameter will be non-zero.

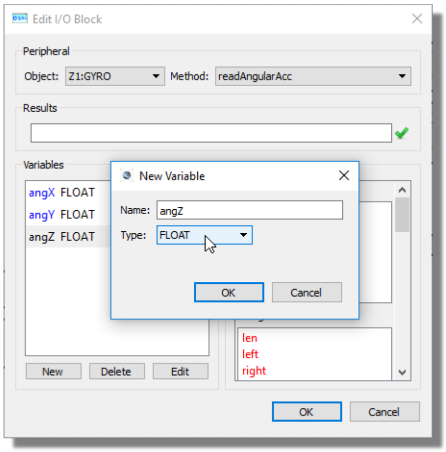
Example:

1) Drag and drop the method onto the flowchart.

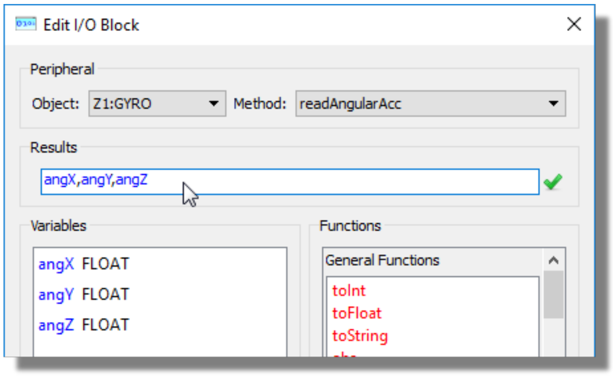


2) Edit the method and create three new variables (e.g. angX, angY, angZ)





3) Enter the three variables in to the Results field separated by a comma. This actually passes pointers to the three variables into the method which will then return the relevant values.



Edit I/O Block

Peripheral

Object: Z1:GYRO Method: readAngularAcc

Results

angX,angY,angZ

Variables

angX FLOAT
angY FLOAT
angZ FLOAT

readLinearAcc() method

*- Takes three floating point parameters (void readLinearAcc (float *linX, float *linY, float *linZ))*

- Returns the X,Y,Z components angular acceleration in units of G.

This function returns the linear acceleration in three dimensions.
The actual measurement of linear acceleration is on the LSM303 device whereas the angular acceleration is read from the L3GD20H device. Linear acceleration is a tricky one to measure in a simulation environment as it happens over such a short amount of time. The following is an explanation of the implementation:

The physics modelling does not simulate acceleration - when a the turtle is driven it sets the speed instantaneously. The reason for this is that the turtle and Zumo get up to speed so quickly there was any point in modelling it. However, this is no good if you're measuring linear acceleration. So acceleration has been modelled by looking at the change in velocity from one (simulation) frame to the next and limiting the change to a maximum of 1G for driven acceleration and 10G for collisions. Obviously, 1G accelerations occur over more frames than 10G accelerations and therefore are more successful to catch when polling the device for changes.

In practice, we assume the linear acceleration may be used in a tight loop to test against a collision with an obstacle. We welcome feedback on this - tell us what you are doing and if our implementation is lacking.

Motors Drive Control

The Zumo contains two integrated micro metal gearmotors with a DRV8835 dual motor driver to control them. The following methods are used in Visual Designer to control this hardware.

drive() method

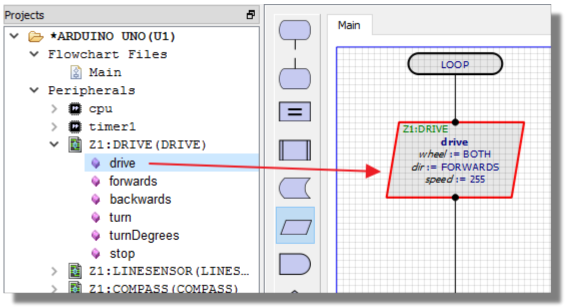
- Takes three parameters (which wheel, which direction and what speed).

- Has no return value.

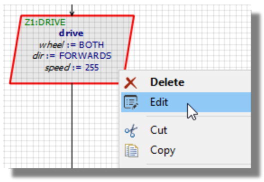
This is the workhorse method to move the Zumo turtle. It allows you to drive one or both of the wheels in a direction and speed of your choice.

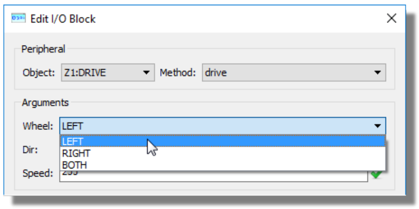
Example:

1) Drag and drop the drive method onto the flowchart.



2) Edit the method and specify the wheel and direction to drive.



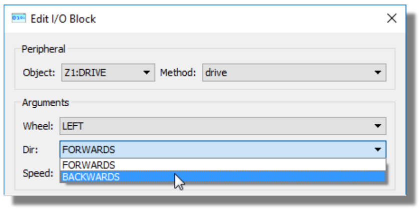


Object: Z1:DRIVE Method: drive

Wheel: LEFT

Dir:

Speed:



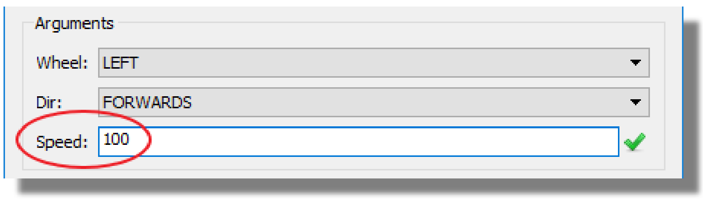
Object: Z1:DRIVE Method: drive

Wheel: LEFT

Dir: FORWARDS

Speed:

3) Specify the speed at which you want to drive the wheel. This is a value between 0 and 255 where 255 represents full speed.



Wheel: LEFT

Dir: FORWARDS

Speed: 100

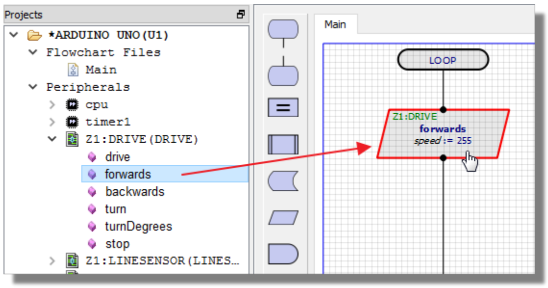
The range 0..255 are used because they represent a byte and at a lower level the motor is driven electronically by altering the duty cycle of a PWM signal (using the Arduino analogWrite() method).

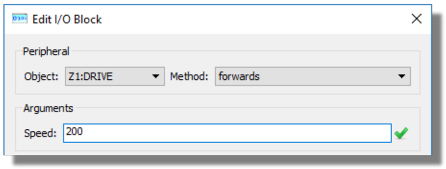
forwards() method

- Takes one parameter for speed.

- Has no return value.

With the forwards() method the direction of travel is explicit and so all that is needed is the speed value.





Object: Z1:DRIVE Method: forwards

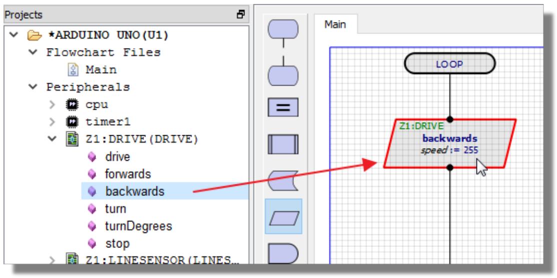
Speed: 200

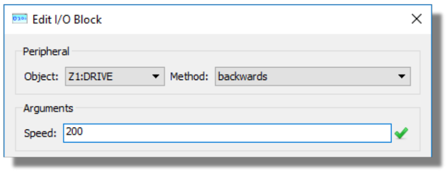
backwards() method

- Takes one parameter for speed.

- Has no return value.

As with the forwards method except driving the turtle straight backwards at the specified speed.





Object: Z1:DRIVE Method: backwards

Speed: 200

turn() method

- Takes one parameter for speed.

- Has no return value.

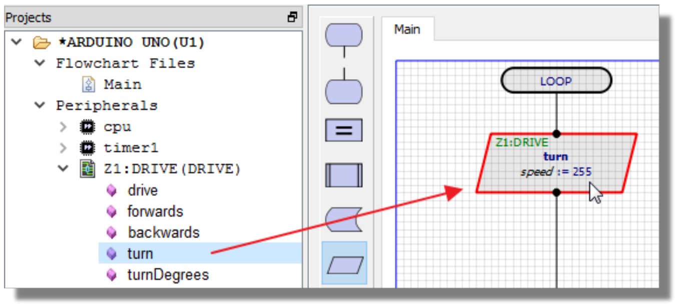
The turn command allows the Zumo to turn either clockwise or anti-clockwise at a given speed. The range of the speed parameter is -255..255 where:

-255 -> 0 : turn anti-clockwise by driving right wheel forwards and left wheel backwards.

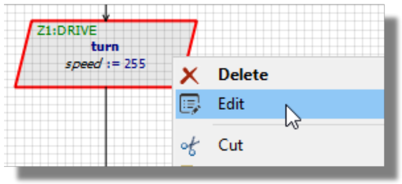
0->255 : turn clockwise by driving left wheel forwards and right wheel backwards.

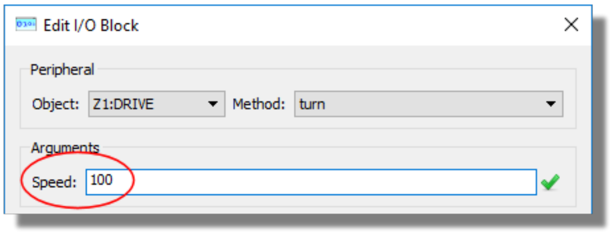
Example:

1) Drag and drop the method onto the flowchart.



2) Edit the method and set the speed value between 0 and 255. Prefix with a minus sign if you want to turn anti-clockwise.




Edit I/O Block

Peripheral

Object: Z1:DRIVE Method: turn

Arguments

Speed: 100

turnDegrees() method

- Takes two integer parameters, one for the number of degrees to turn and the second for the speed of turn.

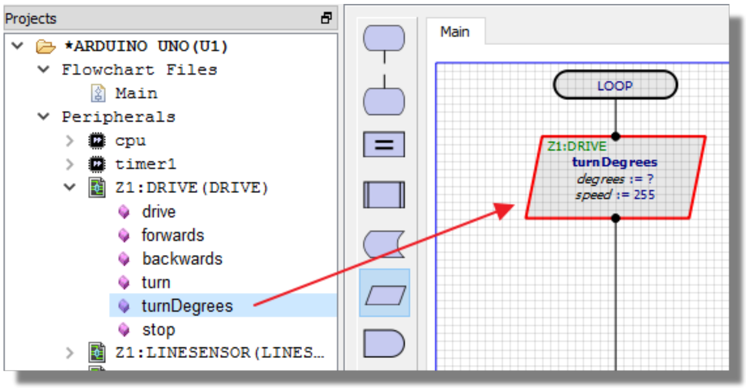
- Has no return value.

The turnDegrees() method allows you to control the amount of turn programmatically. You specify the number of degrees that you want to rotate (0..360) and the speed at which you wish to do so (-255...255).

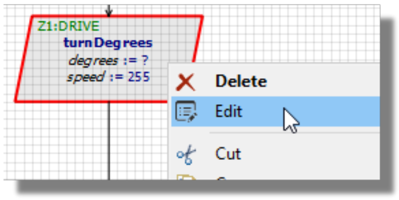
Internally, this method uses the gyro, reads the angular acceleration in Z and then calculates the time needed to turn the specified number of degrees. It follows that, while fairly accurate, it will be an approximation.

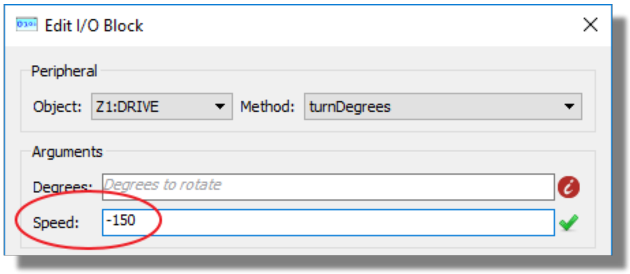
__Example:__

1) Drag and drop the method onto the flowchart.



2) Edit the method and set the speed value between 0 and 255. Prefix with a minus sign if you want to turn anti-clockwise.




Edit I/O Block

Peripheral

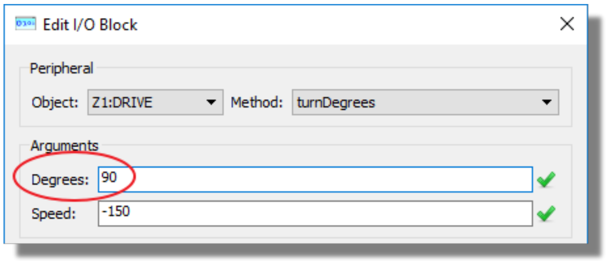
Object: Z1:DRIVE Method: turnDegrees

Arguments

Degrees: Degrees to rotate

Speed: -150

3) Set the number of degrees to turn.



Edit I/O Block

Peripheral

Object: Z1:DRIVE Method: turnDegrees

Arguments

Degrees: 90

Speed: -150

stop() method

- Takes no parameters

- Has no return value

This method will stop the motor drive and bring the turtle to a halt.

Mechanical Forces

It is important to understand that a Proteus simulation of the Virtual Turtle, while electrically accurate, is not a physics aware simulation. The motor control of the turtle is open loop and therefore the amount of distance travelled in a period of time **is not deterministic**. All sorts of physical effects such as inertia, momentum, friction, load/slope and battery power can have significant impact on distance travelled in the real device. In no way therefore should you rely on a timed value in the programming of the device unless you are confident that the real world conditions are not going to change.

In the Zumo turtle you can query the distance travelled via the gyro which helps enormously with writing the firmware but you still cannot determine the distance travelled over a period of time in advance.

For example, the turtle may travel 20 cm on the table but 10cm on carpet, it may lose 20% power as the batteries run down and may not start moving at all if the duty cycle of the PWM doesn't provide enough power to overcome inertia.

Similarly, there may be significant differences in behaviour with line sensing depending on the lighting conditions and the contrast of the line against the background.