Build Your Own Graphical LCD Component

This is a journey, that has taken me to a point where I am able to build and configure my very own graphical component. Having spent a lot of time studying the design specification of various LCD display functional capabilities, but unable to make any changes that I required. What I needed, was a way to take control of the gLCD. I would like to share that development journey with you. My journey started with a base gLCD component (*Thank You Matrixtsl team for allowing me to share this component*) with many of the Flowcode 7 simulation macro's intact, including the base functionality to produce text and graphics. The functions I was interested in changing; the communication port (the use of the hardware SPI port - that's already added for you) and the control macro's which initialises the gLCD via command codes (you get to change these). This of course would give greater control of the LCD as well as a much-needed speed boost, plus the ability to change the LCD component configurations.

> *Did you know, some gLCD displays have hardware scrolling built-in and LUT's for character generation! Yes, well now you can begin experimentation of those elusive commands!*

You are free to develop your very own component, but please share your successes.

The Base Graphical Component

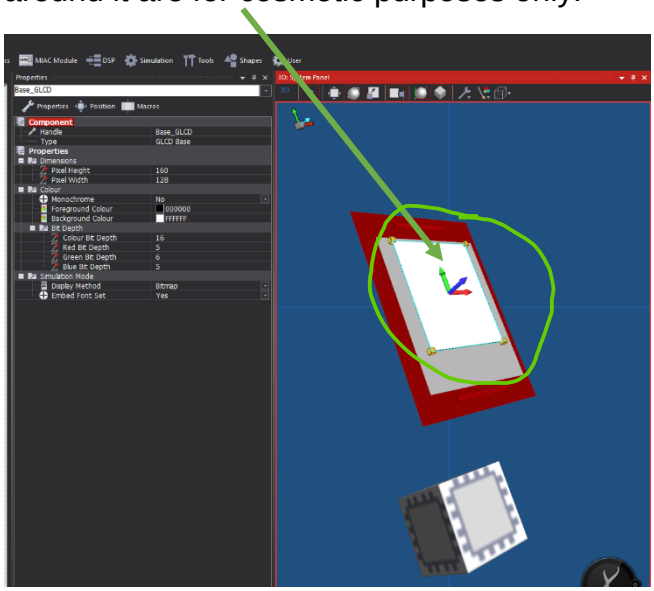
We will start by looking at the building blocks of the base graphical gLCD display component, that will enable you to build your very own gLCD component.

Download the file: - base_gLCD_spi.fcfx

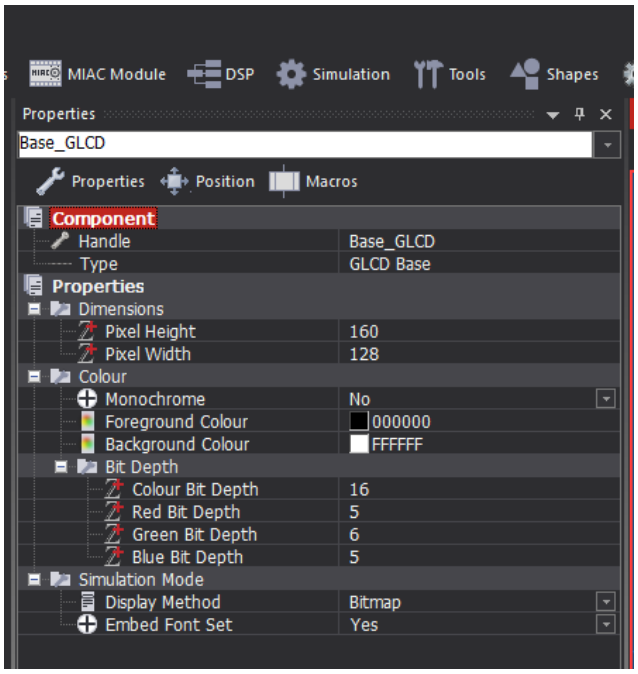
Now, you have the option to select the communication port - I2C or SPI, your choice. The SPI is already done for you. I will, in time, be developing a base gLCD with I2C.

First - The Base Canvas Layer (*pick your own size*)– used for both simulation and component construction.

This is the main part of the base component, adjustable in size to match your gLCD – the **base *canvas layer*** as shown here (highlighted in white). The other shapes around it are for cosmetic purposes only.



Everything is linked to this layer in one form or another, so it is worth noting, that the names of files; variables; properties; parameters and many of the simulation/functional macros are all associated to this layer. We must therefore, be aware that some names cannot be changed or the new component will not work, as I had discovered. I will give details of which ones you can change.



I have highlighted the layer, so the **Properties** panel on the left gives those details that we need to bring across to our LCD component. The obvious property is the canvas dimensions. You can set this to whatever the corresponding size of your display.

You will notice the main properties indicated are:

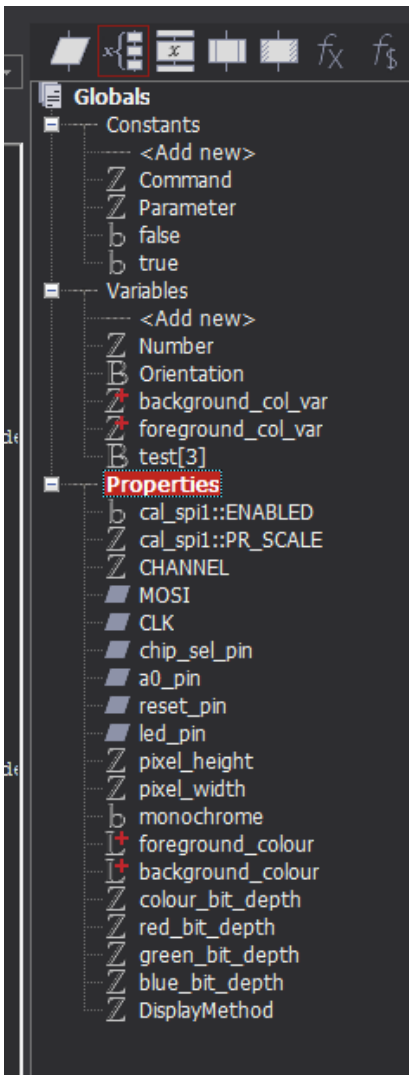
- Monochrome
- DisplayMethod
- Embed Font Set
- Colour_Bit_Depth
- Red
- Green Bit Depth
- Blue

However, there are some properties/variables which you need to be careful with, example:

- pixel _height
- pixel_width
- foreground_colour
- background_colour

Any changes to their names you make here will cause the component to fail, unless you change every occurrence within the component to the name you have changed them to. I would recommend you leave the variable names as they are.

Here are the main 'Global' properties that are used within the base components macros calculations.



Obviously, some properties you may wish not to use.

Example – what if you decide to go down the I2C route, then the a0_pin would not be needed. Yes, you can leave the Property in, it will not do any harm, but will never be implemented during execution of the component. This makes the base gLCD display component extremely versatile.

The CHANNEL; MOSI and CLK, I have added for this component, These can be safely removed if you use the I2C CAL.

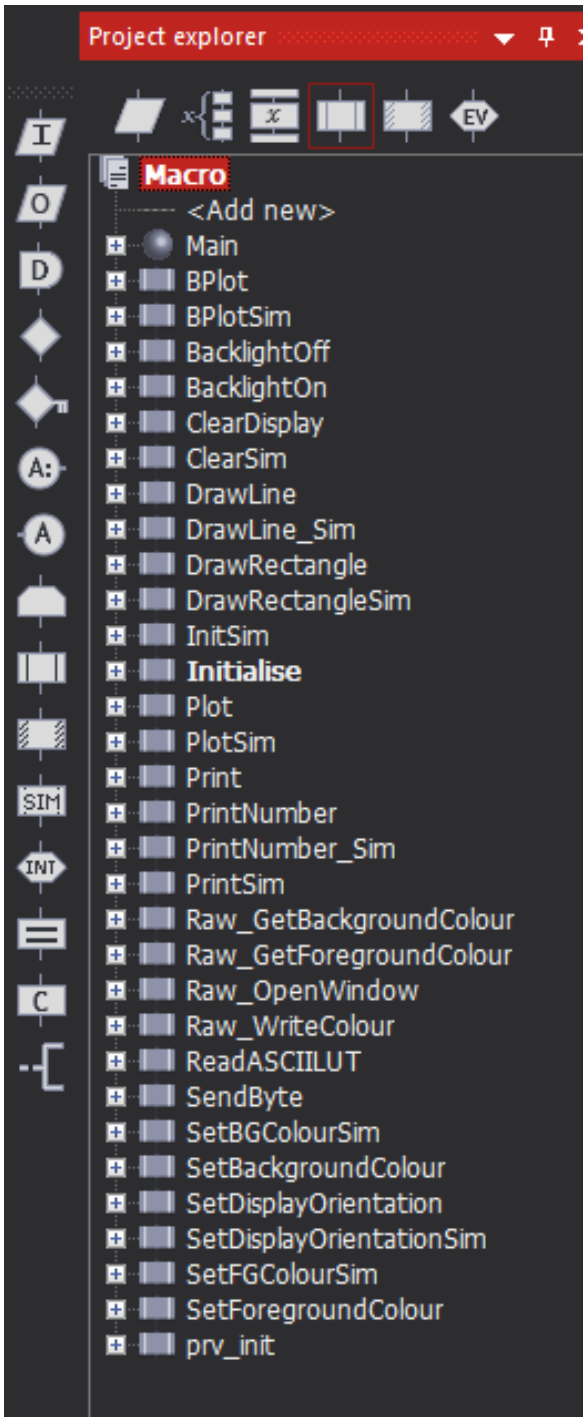
You may well wonder why I have put the 'CHANNEL' property in when I do not intend to use software – well, what about those 16 & 32 Bit pic's that have two SPI ports!

Yes, you can then implement the use of the other SPI port for a SD Card reader – for storing images perhaps. Or, maybe adding a routine for the (*chip select)* CS pin so the SD card shares the same SPI port. These options all become a possibility for experimentation.

Also, there are some 'Local' variables/properties for many of the macro's. Just leave them as they are, there's no need to change them.

Just remember, we are only concerned with just those for – canvas size; communication CAL and initialisation commands. So, the majority, we can leave well alone. Just concentrating on these three, will give you a much greater insight to what possibilities are waiting!

Base Components Macros – 32 gLCD Simulation/Functional macro's



This list shows all the specific macro's types used for the based gLCD component. Those shown in **green**, we need to change/edit/develop. Don't change their names though!

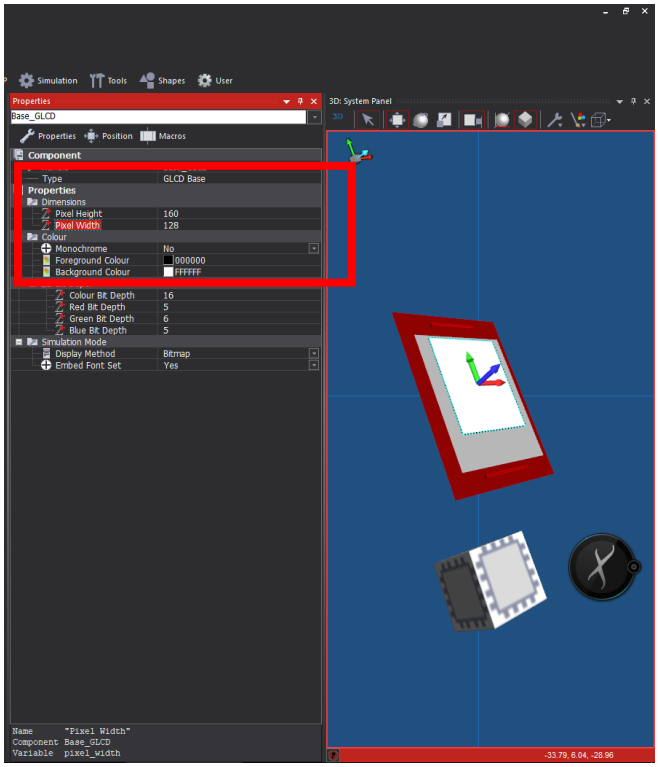
- **prv_init** – *this macro contains both component property events and simulation event*. You also get to choose which communication CAL to be used for you gLCD
- **Main –** this is the main component program
- **Initialise** – this macro is called first when the new component is started in your project. It has all the command codes that are needed to set up your gLCD
- **InitSim** – generates the components graphical/text output onto your computer screen during project development - simulation. They show what the actual component will do when your project is compiled to the chip.
- **SendByte –** sends data to your gLCD using whichever communication port you have chosen. Holds the communication protocol logic using digital output pins that you get to choose. I have remove the software option, but it could be added if you wish.
- **Functional** – the rest are functional routines for producing text and graphics to make the display work on your hardware.

Those with 'Sim' in their name are purely for simulation and **MUST NOT** be changed. The one called '**ReadASCIILUT**' looks like it has nothing in it, it's embedded with the API of Flowcode 7, and is compiled into the microcontroller ROM (character Look-up-tables). The rest, leave well alone.

Well, by changing just three macros, you end up with a completely new gLCD component. We will need to finalise what we have done by completing the new components configuration before exporting to our library. This is achieved by giving it a unique GUID component code; ICON; name and which macros are to be visible to

the user. All the work for SPI is done ready, needing only those changes as to which digital output control pins you need. So, let's get started.

Step 1 – Base gLCD Canvas Layer



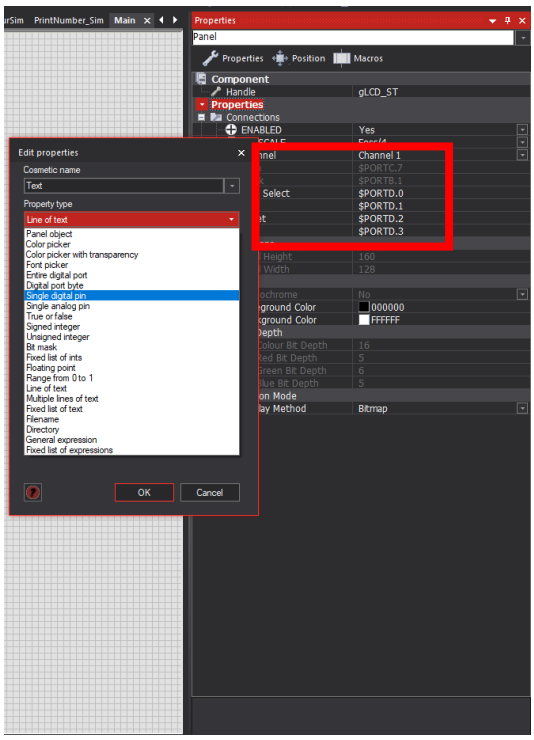
The only properties that require changing for most LCD displays are:

(Click onto the canvas layer and adjust to your displays requirements).

- **Dimensions** – set the canvas size to match your LCD. Whatever changes you make here are automatically transferred to your Component macros.
- Set the **canvas layer** colour to match your display (some are black; blue; yellow or green etc
- **Monochrome** should be set to **YES** if you are using a monochrome (single colour) display.

Leave the rest as they are. I have not experimented with changes made to each of the colour bit depths, may be worth investigating.

Step 2 – Digital Control Pins



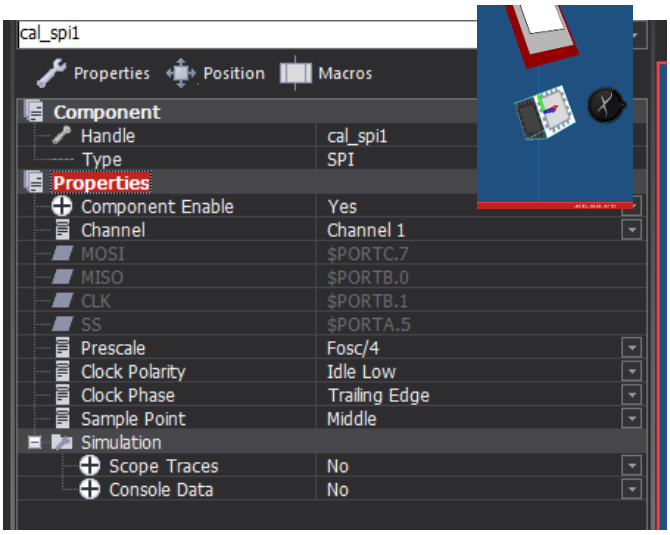
(Click anywhere on the 3D System Panel to select your component).

To install digital control pins such as RW or RD etc, then you can do this as follows:

Click onto the **Properties** tab and using the pull down menu, select **- Add new**

You will be presented with a list as shown opposite. Select from the property type – **Single Digital Pin** and give it a name that matches its purpose (the name is cosmetic). Next, give it a variable name that can be called in the **SendByte** macro. You can delete any pin that is not required, if it is remove from the **SendByte** macro first.

Step 3 – Communication CAL **SPI**

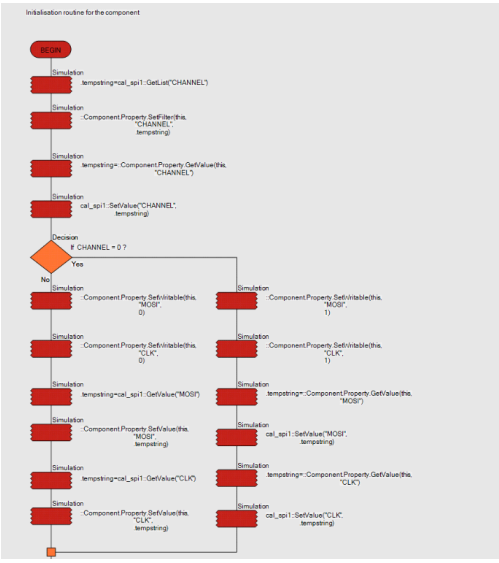


This base gLCD has been designed around the SPI communication port. I would recommend that, for your first attempt into developing your own component, would be to use a gLCD that has this method of communication. The development will be much easier to build and test, giving confidence of further development when you have something that works.

Not all the CAL SPI properties have been used, only those for: (*already set up for you*)

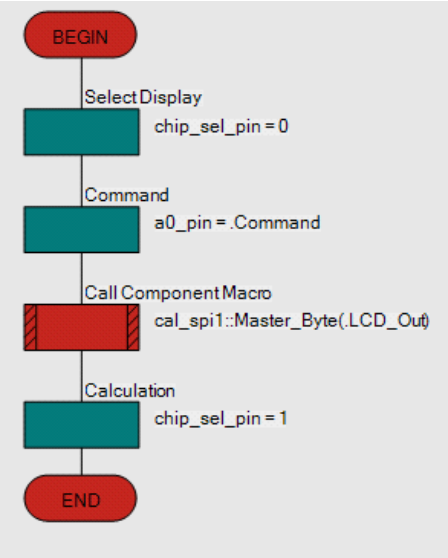
- CHANNEL
- MOSI (changed Property cosmetic name to **data**)
- CLK (changed Property cosmetic name to **clock**)
- Prescale

To link those to our component, we do this by editing the **prv_init** macro. (*already done for the SPI com, port).*



I would recommend that you export the macro before you make any changes. That way, you can import it again if you wish. The first set of ***simulation*** icons are code that links your component to the base SPI UART CAL, those below the CHANNEL branch are for Flowcode 7 display simulation. Now, when you click anywhere on the 3D System panel, you will see those communication port connections. These simulation macros are only for the base communication CAL, not the digital control pins. Those are handled next in the SendByte Macro.

Step 4 – LCD Control – (**SendByte** macro)



This routine is called by each of the following macros;

- Print
- Plot
- BPlot
- Initialise
- ClearDisplay

Therefore, for this LCD, it is necessary to differentiate command codes from data, these are processed by this macro. This is an important function that this macro achieves using ***constants*** and ***parameters.*** In fact, everything that is sent to your LCD is handled by this macro.

Let's have a look at a gLCD Manual (ILI9163) which specifies how this should be performed. The digital pin ***a0_pin*** performs the function control labelled **D/C** for this device. You will notice, this **a0_pin** is controlled by the Parameter called .**Command** (not to be confused by a variable called Command).

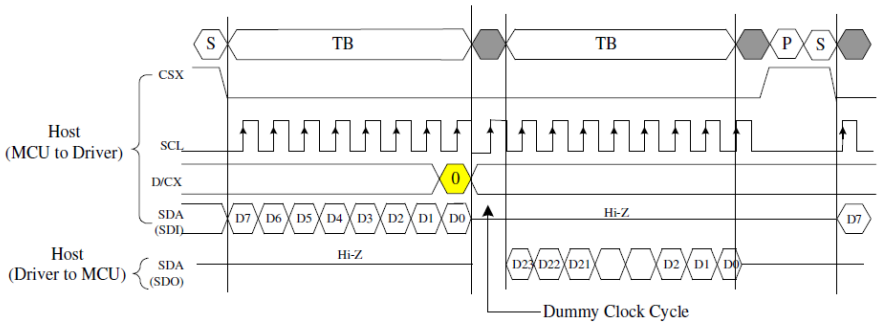
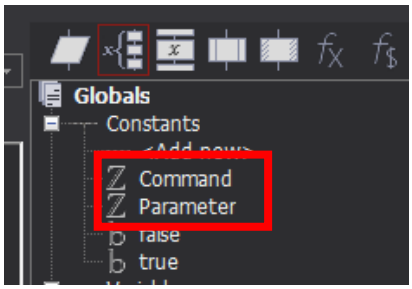
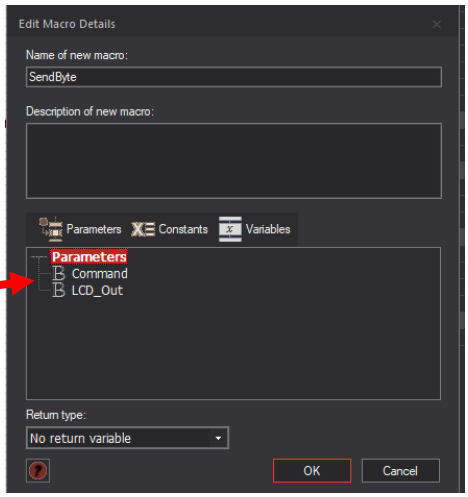


Figure7: 4-pins Serial Protocol (for RDID command: 24-bits read)



These two variable constants play a key role in the **SendByte** macro

When the **D/C** signal is LOW – the LCD expects – command codes
When the **D/C** signal is high – the LCD expect – data
It only needs to be in the correct state at the beginning of the packet!



So, how do we achieve this?

Let's have a look at the SendByte macro again.

This macro has two parameters:

- **Command**
- **LCD_Out**

So, when this macro is called, these two parameters are requested by the macro.

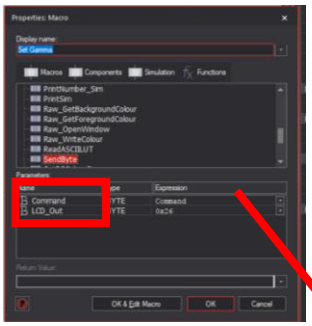
Looking at the macro **Initialise** that calls the **SendByte** macro:

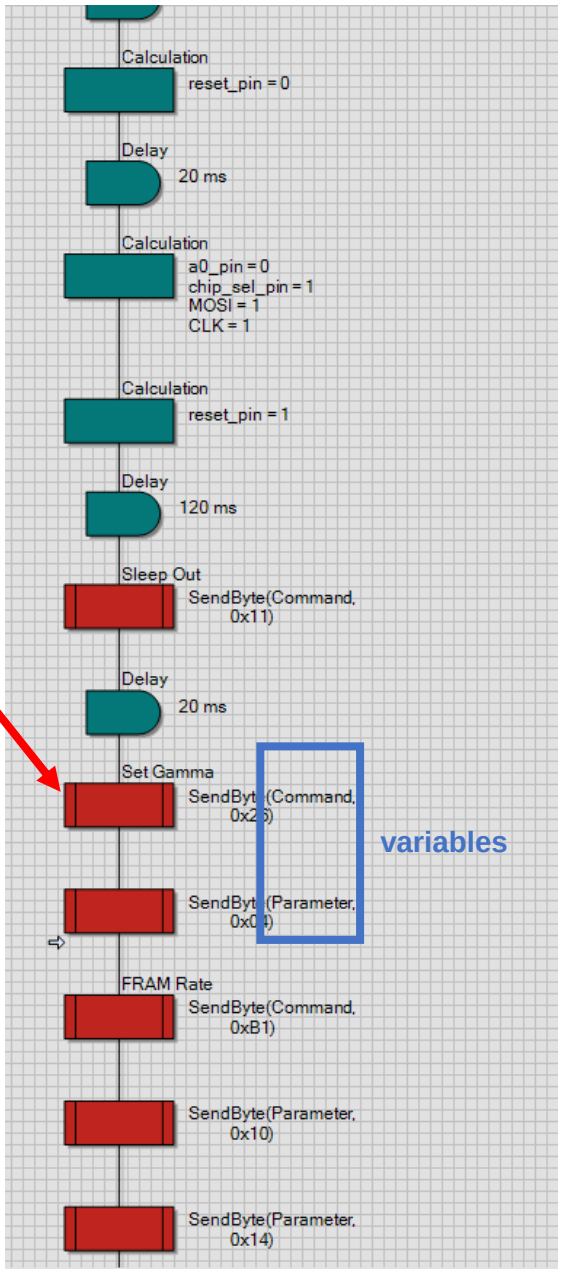
There is a long list of macro call's for the **SendByte** macro, as shown:

The **Parameters** on the left are those expected by the **SendByte**:

Command and **LCD_Out**

They must not be confused with variables of the same name.





The macro Label - **Expression** on the right is the expected variables called by the macro. There are two expected variables.

You can put whatever byte number you like in here, but to make life easy we use a readymade variable for the first Command Parameter and put the data we want sending in the LCD_Out parameter.

Variable constant called **Command = 0**

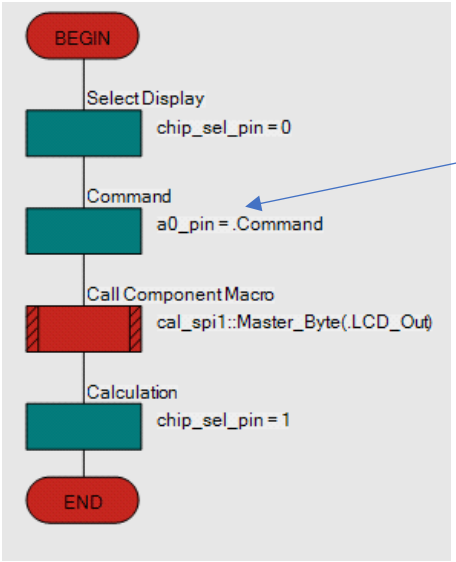
Variable constant called **Parameter = 1**

Therefore, if we send the variable called **Command**, we are sending the value = 0

And if we send the variable called **Parameter**, we are sending the value = 1

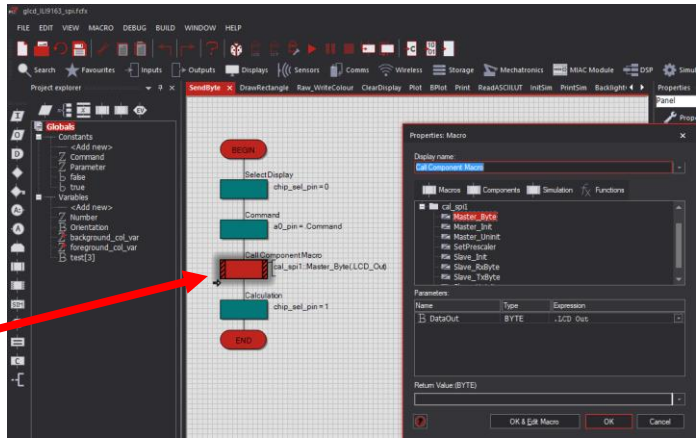
This variable is now carried into the **SendByte** macro, making it possible for you to use it to control the **a0_pin** set by the variable within this macro itself.

(Place your mouse over each of the variable constants in the Project explorer window to see their values) see above!



a0_pin is controlled by the **Parameter** called **.Command** which holds the variable value:

either 0 or 1



The base UART SPI CAL is called here and the data is sent to the device by obtaining data from the LCD_Out variable.

Therefore, if you wish to use the I2C base CAL, it only this macro call here, that will be changed.

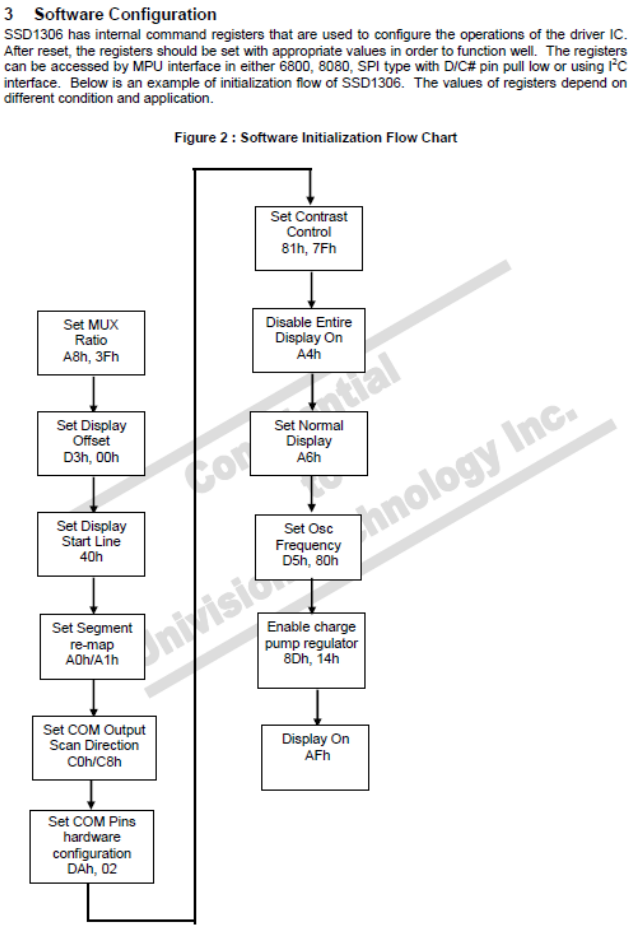
Compare this to the 4-Pins Serial protocol and you will then discover, that the routine is able to switch the a0_pin to whatever setting is required just by the parameter set by the variable it contains (low for commands and high for data).

If you wish to change the way the LCD is controlled, it is with this macro (**SendByte**) that you would implement the control strategy for your device, using digital control pins that you have installed/set-up in your components properties panel. I have only ever found, where commands and data need to specifically differentiated, there has only ever been one digital control pin used. Their names may change, but their function is fundamentally similar.

I believe the hardest part is getting the command codes set correctly. Everything else would possibly remain unchanged.

Step 5– Initialisation Commands.

To get your copy of the command codes to initialise for your chosen gLCD, it will of course involve thorough research, but it will be worth the effort. Here is an example of what you are looking for: example - ***SSD1306 specification***



3 **Software Configuration**

SSD1306 has internal command registers that are used to configure the operations of the driver IC. After reset, the registers should be set with appropriate values in order to function well. The registers can be accessed by MPU interface in either 6800, 8080, SPI type with D/C# pin pull low or using I^2C interface. Below is an example of initialization flow of SSD1306. The values of registers depend on different condition and application.

Figure 2 : Software Initialization Flow Chart

This one is for the SSD1306. Within the document, you will find what each of the code blocks (flowchart) are, for setting up the LCD display, helping you to locate the ones of interest to yourself. It will also help identify command codes from data settings for each code block. Yes, it will be frustrating, but it's what many coders are prepared to do for that special project.

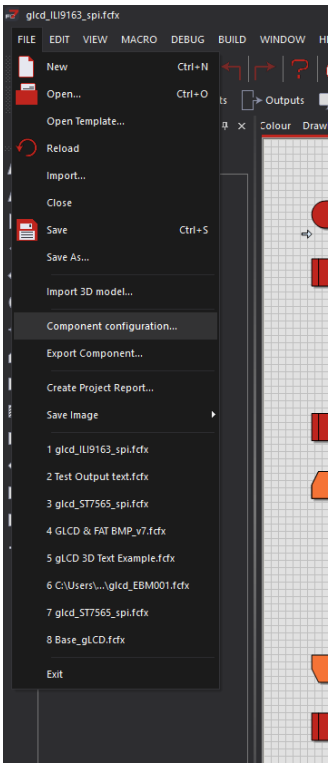
Flowcode 7 has many of the initialise commands already built in. You can find them as shown below.

Stat a new project and load the LCD component that matches the one you wish to develop. Using the ‘View C’ code under the Build menu, will allow inspection of the command codes. However, you may wish to research any further options available by obtaining the displays specification manual.

```
1494  /*=-----------------------------------------------------------------------=*\
1495     Use :The Init macro must be called once to initialise the Graphical LCD display before any other Graphical LCD cc
1496  \*=-----------------------------------------------------------------------=*/
1497  void FCD_0fc51_gLCD_SSD1306__Initialise()
1498  {
1499      //Local variable definitions
1500      MX_UINT8 FCL_RED;
1501      MX_UINT8 FCL_GREEN;
1502      MX_UINT8 FCL_BLUE;
1503
1504
1505      FCP_SET(B, A, 0x1, 0x0, 1);
1506      FCP_SET(B, A, 0x20, 0x5, 1);
1507      FCP_SET(B, A, 0x10, 0x4, 1);
1508      FCP_SET(B, A, 0x8, 0x3, 0);
1509      FCP_SET(B, A, 0x4, 0x2, 0);
1510      FCP_SET(B, A, 0x2, 0x1, 0);
1511
1512      FCI_DELAYBYTE_MS(200);
1513
1514      FCP_SET(B, A, 0x20, 0x5, 0);
1515
1516      FCI_DELAYBYTE_MS(200);
1517
1518      FCP_SET(B, A, 0x20, 0x5, 1);
1519
1520      FCI_DELAYBYTE_MS(200);
1521
1522      FCD_0fc51_gLCD_SSD1306__SendCommand(0xAE);
1523
1524      FCD_0fc51_gLCD_SSD1306__SendCommand(0xD5);
1525
1526      FCD_0fc51_gLCD_SSD1306__SendCommand(0x80);
1527
1528      FCD_0fc51_gLCD_SSD1306__SendCommand(0xA8);
1529
1530      #if (1) // 64 == 64
1531
1532          FCD_0fc51_gLCD_SSD1306__SendCommand(0x3F);
1533
1534      #else
1535
1536      //Code has been optimised out by the pre-processor
1537      #endif
1538
1539      FCD_0fc51_gLCD_SSD1306__SendCommand(0xD3);
1540
1541      FCD_0fc51_gLCD_SSD1306__SendCommand(0x00);
1542
1543      FCD_0fc51_gLCD_SSD1306__SendCommand(0x40);
1544
1545      FCD_0fc51_gLCD_SSD1306__SendCommand(0x8D);
1546
1547      FCD_0fc51_gLCD_SSD1306__SendCommand(0x14);
1548
1549      FCD_0fc51_gLCD_SSD1306__SendCommand(0xA1);
```

Once you have finished developing the control strategy for your display using the SendByte macro and set up the command codes, all that is required now is to configure your LCD display component ready for exporting to you library.

Step 6 –LCD component Configuration

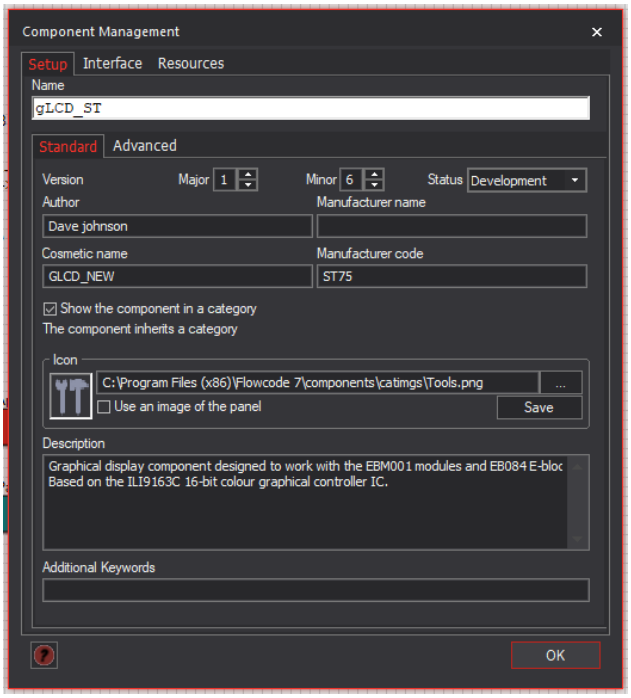


From the menu – File, select (***Component Configuration***)

All the options for how the component is used by Flowcode 7 are selected from here.

There are three-tab menus.

Setup – Interface - **Resources**



Setup -Standard

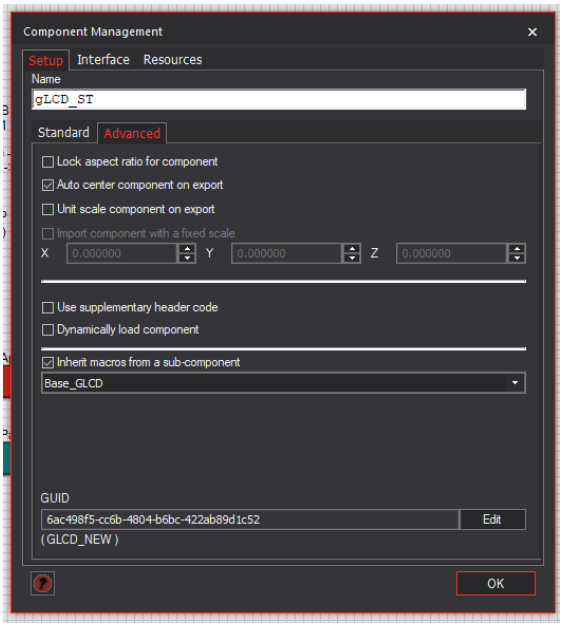
I would suggest you start with a status of – Development. It is up to you if you chose to place it in a category. The icon will help identify its status from other components.

Fill in as much details as possible.

Advanced

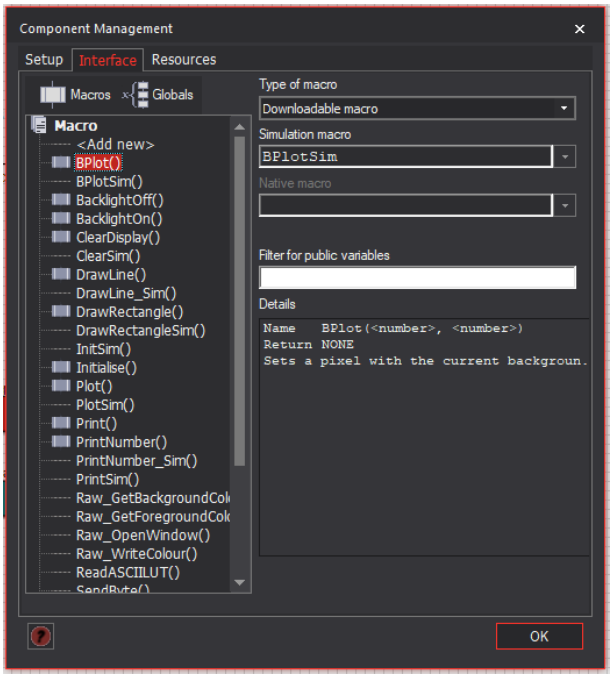
Tick the box – Inherit macros from base component.

The GUID **MUST** be updated by clicking on the edit and select **New**. You only need to do this once. When you are developing your component, changes can be made and will always be referenced to this new component with this GUID.



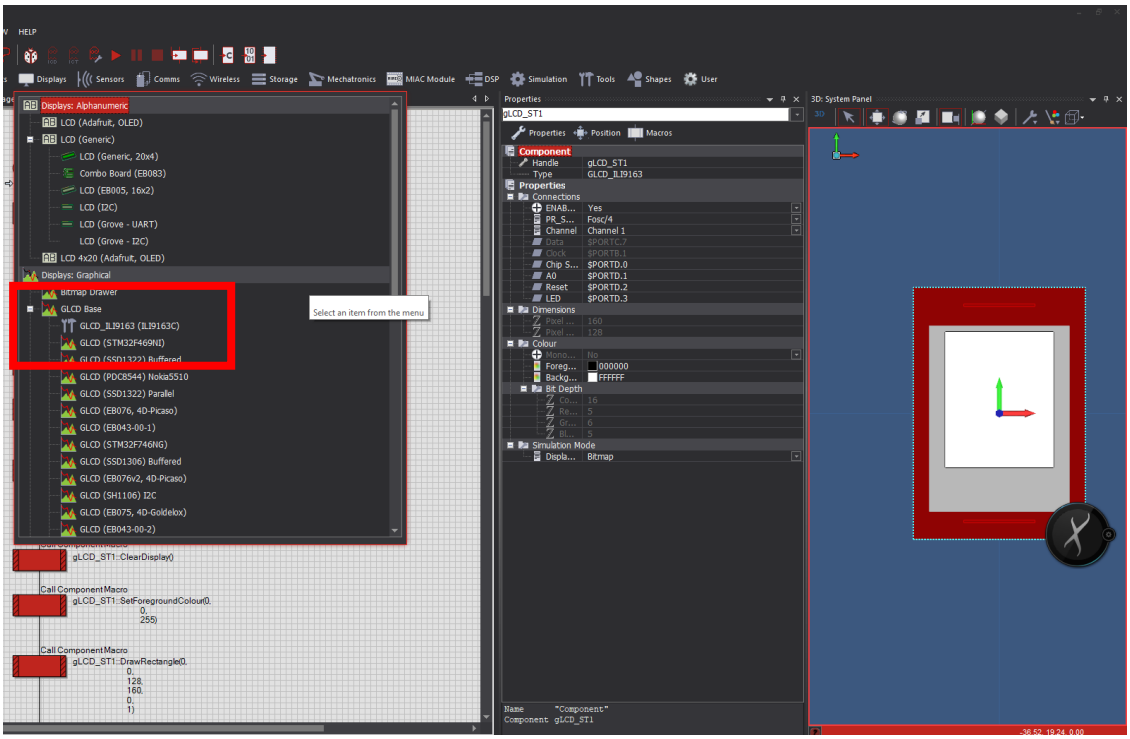
Interface

All the macros associated to this component are shown on the left. You need to select which macros will be set as (***downloadable***). The other macros should be set to ***Hidden***. They will be embedded with the finished component.

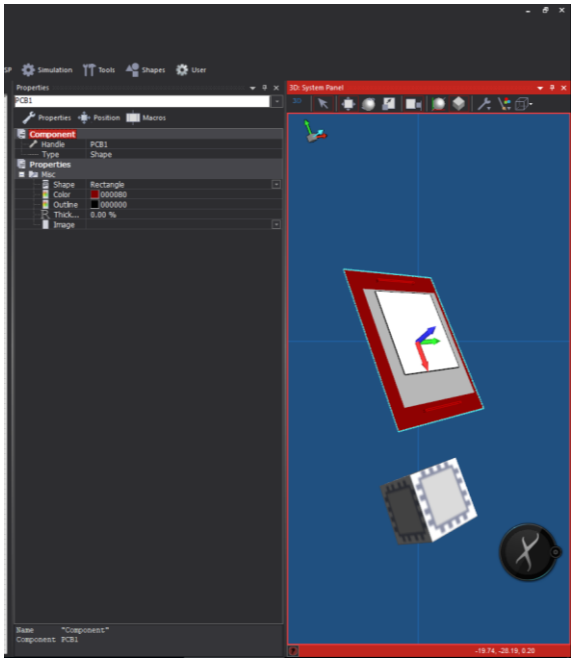


No changes are needed for the resources, or until you can develop your own 3D models and include them with your component.

When you have completed the components configuration, all that remains is to export the component to the library location you have set up in the global options.

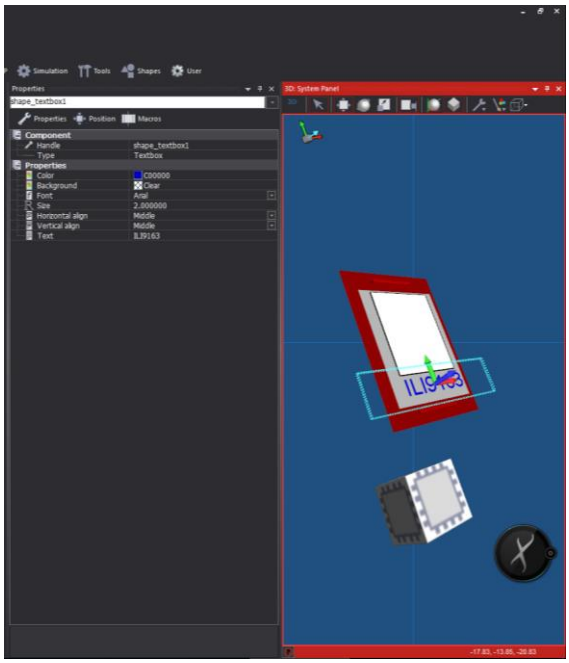


This is how your finished gLCD will look.



If you click onto the component to select the rectangle shape, you can change its appearance/colour using the colour properties. It just gives it that personal appearance to suit yourself.

You can even add text to further add more clarity to the devices identification.



Your finished component will look something like that shown above, minus the base UART SPI CAL of course.

That's it, your all done.