LCD Initialization

Character based LCD modules

The information in this section relates to Character based LCD modules, specifically those controlled by an HD44780 or equivalent.

Initializing by Internal Reset Circuit

This is the datasheet information regarding Initialization of the LCD controller.

Initializing by Internal Reset Circuit

An internal reset circuit automatically initializes the HD44780U when the power is turned on. The following instructions are executed during the initialization. The busy flag (BF) is kept in the busy state until the initialization ends (BF = 1). The busy state lasts for 10 ms after V_{CC} rises to 4.5 V.

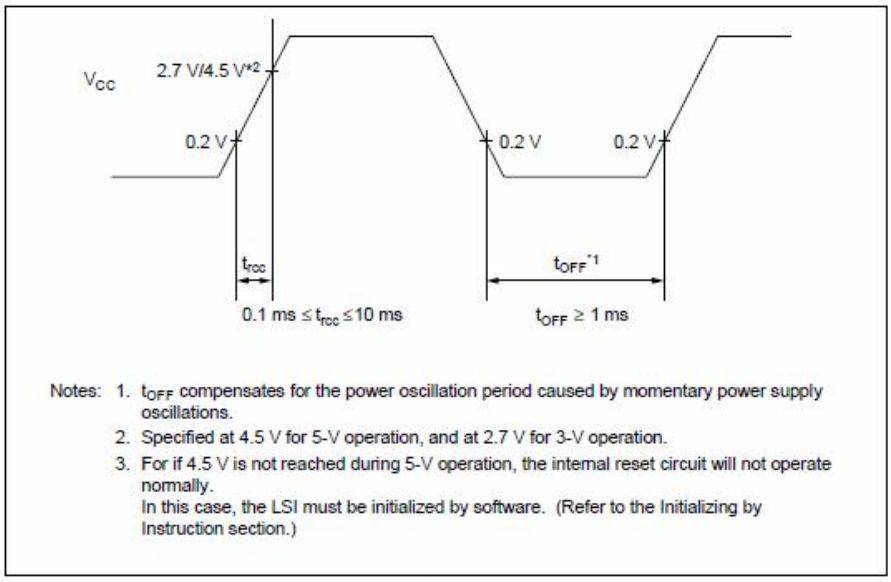
1. Display clear
2. Function set:
   DL = 1; 8-bit interface data
   N = 0; 1-line display
   F = 0; 5 × 8 dot character font
3. Display on/off control:
   D = 0; Display off
   C = 0; Cursor off
   B = 0; Blinking off
4. Entry mode set:
   I/D = 1; Increment by 1
   S = 0; No shift

Note: If the electrical characteristics conditions listed under the table Power Supply Conditions Using Internal Reset Circuit are not met, the internal reset circuit will not operate normally and will fail to initialize the HD44780U. For such a case, initial-ization must be performed by the MPU as explained in the section, Initializing by Instruction.

The 'Internal Reset' technique described above is relied upon by many programmers but, in my opinion, this is not a wise choice. Perhaps they are seduced by the first sentence with the promise of 'automatic' initialization. If you look at the result of this automatic initialization you will see that the controller is configured for a 1-line display when in fact the majority of LCD modules should be configured for a 2-line display. It also leaves the display off. This means that two of the four steps that were automatically performed are going to have to be redone.

But that is NOT the main reason that this technique should be avoided. The note at the bottom clearly states that this technique will fail if the power supply does not meet certain specifications, specifications that are buried elsewhere in the datasheet. In cases where the power supply cannot be guaranteed to meet those specification the datasheet recommends using "Initializing by Instruction".

Here are those power supply specifications. Do you think they are met by your wall wart?



Notes: 1. t_{OFF} compensates for the power oscillation period caused by momentary power supply oscillations.
2. Specified at 4.5 V for 5-V operation, and at 2.7 V for 3-V operation.
3. For if 4.5 V is not reached during 5-V operation, the internal reset circuit will not operate normally.
In this case, the LSI must be initialized by software. (Refer to the Initializing by Instruction section.)

Internal Power Supply Reset

Relying on this internal reset may be satisfactory for an LCD module that is part of a system that also includes the power supply, such as the display on a printer. It is not satisfactory for an LCD module that is going to be powered by a random power supply.

Most of the people reading this are probably tinkering with an LCD module that is connected to a microcontroller and is powered by the same power supply that is powering the microcontroller. Let's assume the best case, where the power supply does satisfy the requirements for an internal reset of the LCD controller, and where the program in the microcontroller doesn't attempt to send any information to the LCD module until that internal reset is finshed. What happens when the the reset button on the microcontroller is pushed? Answer: The program code will run again, but the LCD module will not be re-initialized since it's power was never interrupted. Here's the problem: The program code contains a **Function Set** instruction, and this instruction should only be executed once, immediately after the LCD module is initialized. Well it already ran once, when the power was applied, and here it is running it again, after the reset button is pushed and the microcomputer code restarts. When this instruction is executed a second time the status of the LCD controller may be indeterminate or, in plain language, the LCD controller may stop responding.

Initialization by Instruction

It really isn't that hard to use this technique once you decipher the flowcharts that describe the procedure. All of the flowcharts in the various datasheets seem to derive from the same source and they are all equally ambiguous in certain areas. Any differences are probably due to typographical or editing errors and those are easily spotted, providing you compare several different datasheets.

There are separate initialization flowcharts for the 8-bit interface and the 4-bit interface, but the actual sequence of instructions sent to the LCD controller is essentially the same in each case. First there are a series of what are technically **Function Set** instructions whose purpose is to effectively 'reset' the LCD controller. Next, if the 4-bit interface is desired, there is an additional **Function Set** instruction to change the interface from the default 8-bit configuration. Finally there are four more instructions, the 'real' **Function Set**, the **Display on/off Control**, the **Clear Display**, and the **Entry Mode Set**.

When power is applied to the LCD module the LCD controller always comes up in the 8-bit interface mode. This means that the LCD controller reads all eight of its data pins each time the Enable pin is pulsed. This is fine if an 8-bit data interface is actually being used, but what about the other possibility, where a 4-bit data interface is connected? In this second case there may be indeterminate data on the lower four bits, especially if those pins have not been grounded as recommended. The answer is that the controller has been set up to ignore those lower four bits throughout the early part of the initialization process, until the actual interface has been established by what I called the 'real' **Function Set** instruction in the previous paragraph.

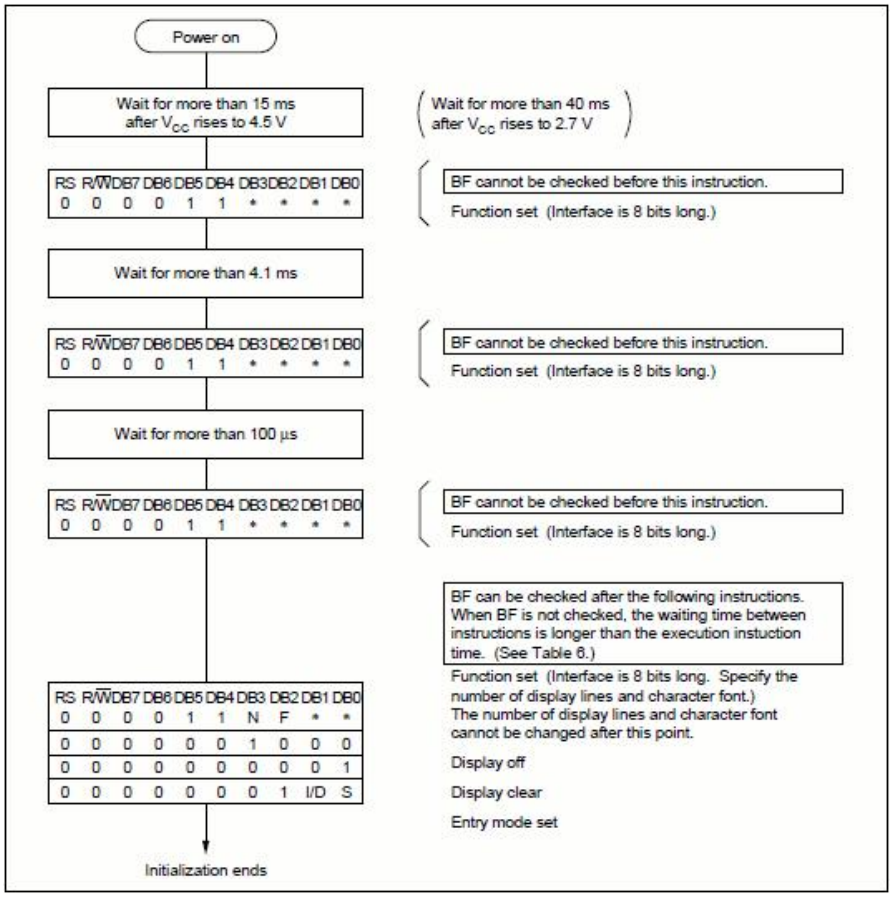
It is important to make sure that the LCD controller has finished executing an instruction before sending it another one, otherwise the second instruction will be ignored. The datasheets give specific times for the delays during the beginning what I call the 'reset' sequence. The datasheets are

ambiquious when it comes to the time delay associated with the end of this sequence and with the mode change in the 4-bit initialization. For the final sequence of instructions the datasheets all specify that you must either check the busy flag to see if the LCD controller is finished or wait a sufficient amount of time, a time that is longer than the instruction execution time. There is more about this near the end of this page.

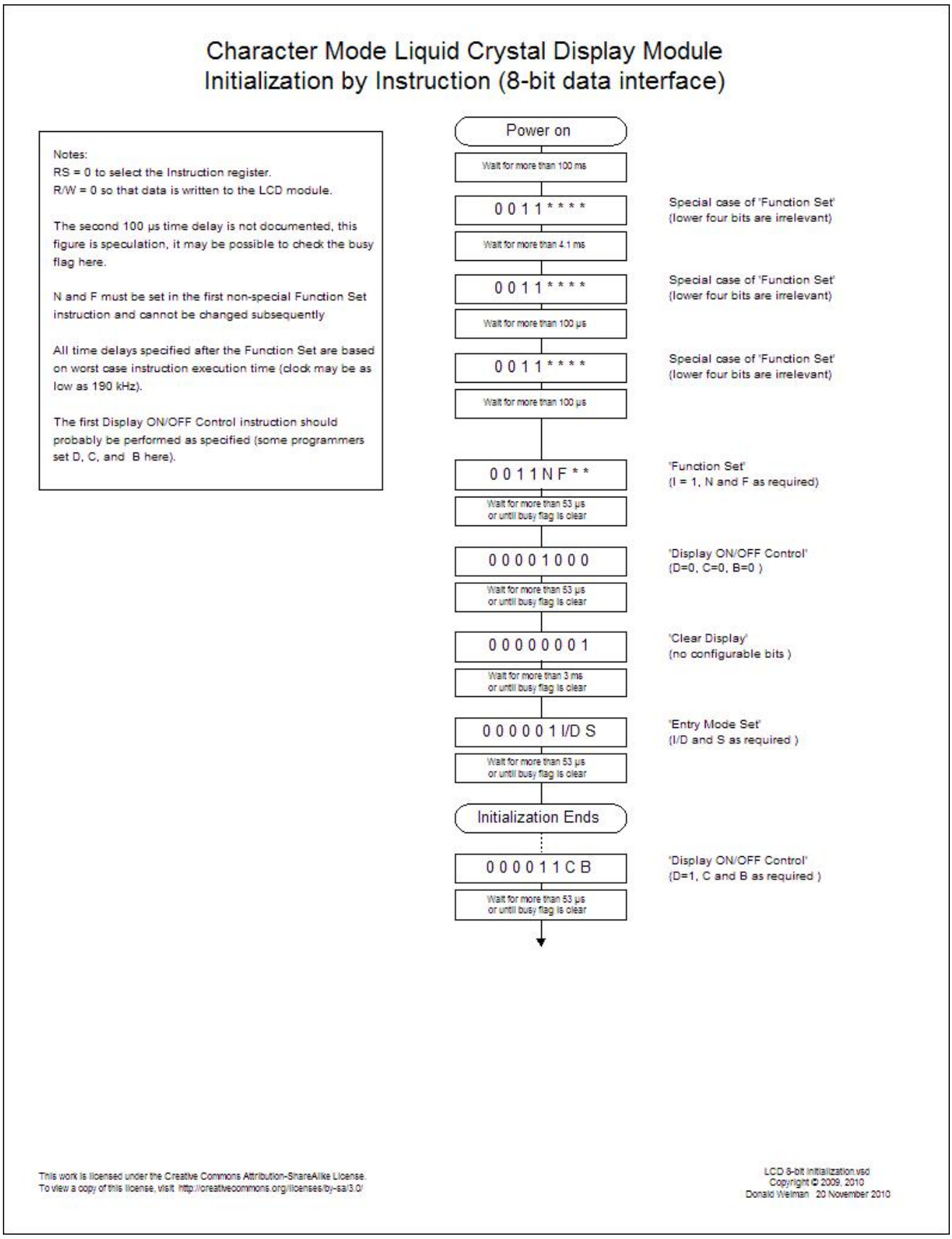
Below you will find a detailed explanation of the 8-bit initialization sequence followed by a detailed explanation of the 4-bit initialization sequence. In each case I present the data sheet version of the flowchart, my version of the flowchart, and finally some notes about each of the steps.

8-Bit Interface, Initialization by Instruction

Here's the flowchart as it appears in the Hitachi datasheet.



My version of the flowchart.



Download the 8-bit flowchart (PDF Document)

Notes

There is very little information available about what is really going on inside the LCD controller. I have made some guesses about what is happening and if anyone has any factual information that either confirms or repudiates my guesses I would appreciate hearing from you. Contact information is at the bottom of the page.

Step 1. Power on, then delay > 100 ms

There are two different values (with two different references) specified on the datasheet flowchart for this initial delay but neither reference is from when the power was first applied. The delay required from power-up must obviously be more than 40 mS and I have arbitrarily chosen to use 100 mS. Since this delay only occurs once it doesen't make sense to try to speed up program execution time by skimping on this delay.

Step 2. Instruction 00110000b (30h), then delay > 4.1 ms
This is a special case of the **Function Se**t instruction where the lower four bits are irrelevant. These four bits are shown as asterisks on flowcharts because the controller will ignore them. They are shown as '0's in these notes because that is how most programmers deal with irrelevant bits. This first instruction, for some unexplained reason, takes significantly longer to complete than the ones that come later.

Step 3. Instruction 00110000b (30h), then delay > 100 us
This is a second instance of the special case of the **Function Set** instruction. The controller does not normally expect to receive more than one 'Function Set' instruction so this may account for the longer than normal execution time.

Step 4. Instruction 00110000b (30h), then delay > 100 us
This is a third instance of the special case of the **Function Set** instruction. By now the LCD controller realizes that what is really intended is a 'reset', and it is now ready for the real **Function Set** instruction followed by the rest of the initialization instructions. The flowcharts do not specify what time delay belongs here. I have chosen 100 us to agree with the previous instruction. It may be possible to check the busy flag here.

Step 5. Instruction 00111000b (38h), then delay > 53 us or check BF
This is the real **Function Set** instruction. This is where the interface, the number of lines, and the font are specified. Since we are implementing the 8-bit interface we make D = 1. The number of lines being specified here is the number of 'logical' lines as perceived by the LCD controller, it is NOT the number of 'physical' lines (or rows) that appear on the actual display. This should almost always be two lines so we set N=1 (go figure). There are very few displays capable of displaying a 5x10 font so the 5x7 choice is almost always correct and we set F=0.

Step 6. Instruction 00001000b (08h), then delay > 53 us or check BF
This is the **Display on/off Control** instruction. This instruction is used to control several aspects of the display but now is NOT the time to set the display up the way we want it. The flow chart shows the instruction as 00001000, not 00001DCB which indicates that the Display (D), the Cursor (C), and the Blinking (B) should all be turned off by making the corresponding bits = 0.

Step 7. Instruction 00000001b (01h), then delay > 3 ms or check BF
This is the **Clear Display** instruction which, since it has to write information to all 80 DDRAM addresses, takes more time to execute than most of the other instructions. On some flow charts the comment is incorrectly labeled as 'Display on' but the instruction itself is correct.

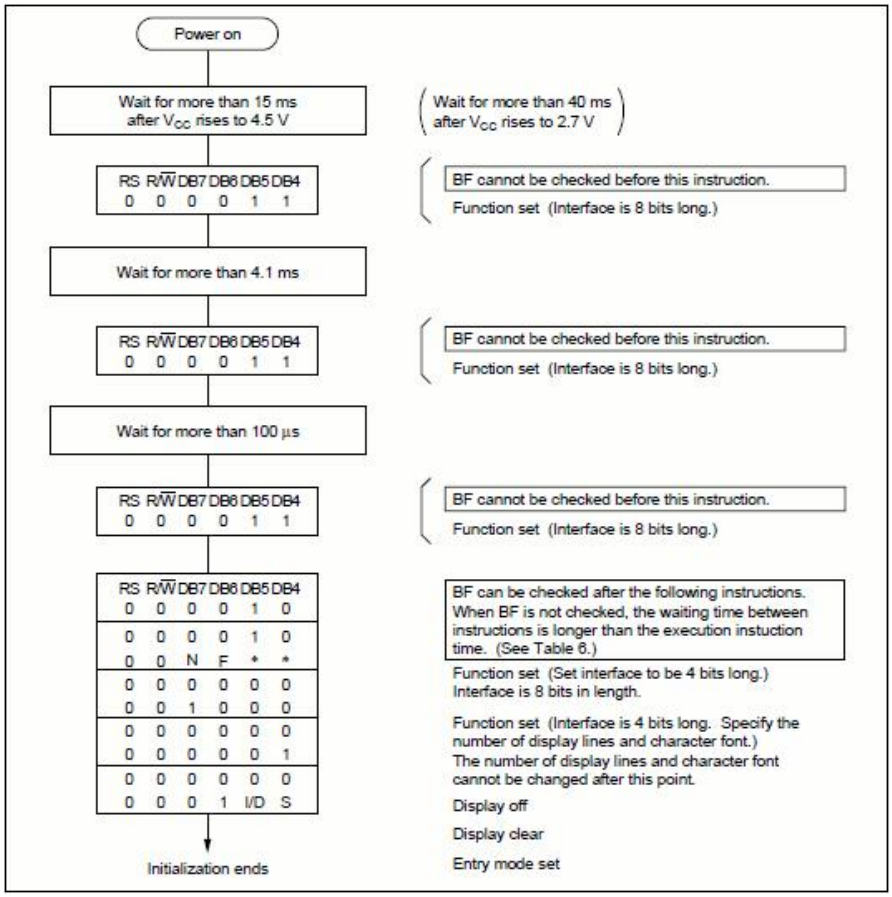
Step 8. Instruction 00000110b (06h), then delay > 53 us or check BF
This is the **Entry Mode Set** instruction. This instruction determines which way the cursor and/or the display moves when we enter a string of characters. We normally want the cursor to increment (move from left to right) and the display to not shift so we set I/D=1 and S=0. If your application requires a different configuration you could change this instruction, but my recommendation is to leave this instruction alone and just add another **Entry Mode Set** instruction where appropriate in your program.

Step 9. Initialization ends
This is the end of the actual intitalization sequence, but note that step 6 has left the display off.

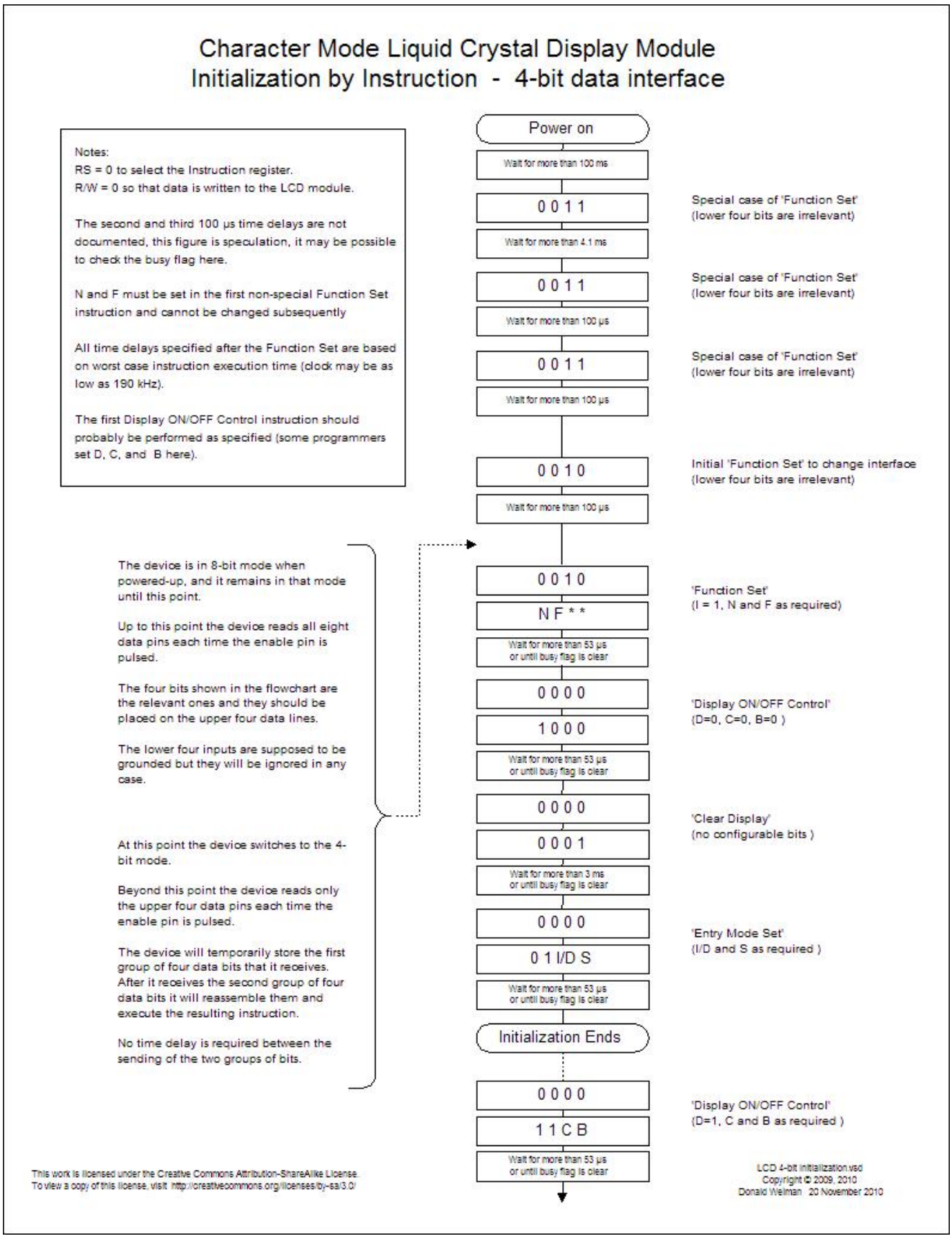
Step 10. Instruction 00001100b (0Ch), then delay > 53 us or check BF
This is another **Display on/off Control** instruction where the display is turned on and where the cursor can be made visible and/or the cursor location can be made to blink. This example shows the the display on and the other two options off, D=1, C=0, and B=0.

4-Bit Interface, Initialization by Instruction

Here's the flowchart as it appears in the Hitachi datasheet.



My version of the flowchart.



Download the 4-bit flowchart (PDF Document)

Notes

These notes are very similar to those for the 8-bit interface because the initialization procedure is very similar. The first four steps are identical because, as mentioned earlier, the LCD controller starts off in the 8-bit mode regardless of how many data lines are actually being used. Since this is an initialization sequence for the 4-bit mode there are only four data lines connected between the microcontroller and the LCD module, however the LCD controller is currently in the 8-bit mode and it expects data on all eight data pins. Fortunately, for this special case of the **Function Set** instruction,

the lower four bits are irrelevant so the fact that they are not connected to the microcontroller is also irrelevant. There is one extra step here, to change to the 4-bit interface, which throws off the step numbers after step 4. The remaining steps *look* different because they are implemented as two groups 4-bits but the actual instructions are the same.

Step 1. Power on, then delay > 100 ms
There are two different values (with two different references) specified on the datasheet flowchart for this initial delay but neither reference is from when the power was first applied. The delay required from power-up must obviously be more than 40 mS and I have arbitrarily chosen to use 100 mS. Since this delay only occurs once it doesen't make sense to try to speed up program execution time by skimping on this delay.

Step 2. Instruction 0011b (3h), then delay > 4.1 ms
This is a special case of the **Function Set** instruction where the lower four bits are irrelevant. These four bits are not shown on the flowcharts because the host microcontroller does not usually implement them at all (as opposed to the 8-bit mode wher they are implemented as '0's). This first instruction, for some unexplained reason, takes significantly longer to complete than the ones that come later.

Step 3. Instruction 0011b (3h), then delay > 100 us
This is a second instance of the special case of the **Function Set** instruction. The controller does not normally expect to receive more than one 'Function Set' instruction so this may account for the longer than normal execution time.

Step 4. Instruction 0011b (3h), then delay > 100 us
This is a third instance of the special case of the **Function Set** instruction. By now the LCD controller realizes that what is really intended is a 'reset', and it is now ready for the real **Function Set** instruction followed by the rest of the initialization instructions. The flowcharts do not specify what time delay belongs here. I have chosen 100 us to agree with the previous instruction. It may be possible to check the busy flag here.

Step 5. Instruction 0010b (2h), then delay > 100 us
Here is where the LCD controller is expecting the 'real' **Function Set** instruction which, in the 8-bit mode, would start with 0011. Instead, it gets a **Function Set** instruction starting with 0010. This is it's signal to again ignore the lower four bits, switch to the four-bit mode, and expect another 'real' **Function Set** instruction. Once again the required time delay is speculation.

The LCD controller is now in the 4-bit mode. This means that the LCD controller reads only the four high order data pins each time the Enable pin is pulsed . To accomodate this, the host microcontroller must put the four high bits on the data lines and pulse the enable pin, it must then put the four low bits on the data lines and again pulse the enable pin. There is no need for a delay between these two sequences because the LCD controller isn't processing the instruction yet. After the second group of data bits is received the LCD controller reconstructs and executes the instruction and this is when the delay is required.

Step 6. Instruction 0010b (2h), then 1000b (8h), then delay > 53 us or check BF
This is the real **Function Set** instruction. This is where the interface, the number of lines, and the font are specified. Since we are implementing the 4-bit interface we make D = 0. The number of lines being specified here is the number of 'logical' lines as perceived by the LCD controller, it is NOT the number of 'physical' lines (or rows) that appear on the actual display. This should almost always be two lines so we set N=1 (go figure). There are very few displays capable of displaying a 5x10 font so the 5x7 choice is almost always correct and we set F=0.

Step 7. Instruction 0000b (0h), then 1000b (8h) then delay > 53 us or check BF
This is the **Display on/off Control** instruction. This instruction is used to control several aspects of the display but now is NOT the time to set the display up the way we want it. The flow chart shows the instruction as 00001000, not 00001DCB which indicates that the Display (D), the Cursor (C), and the Blinking (B) should all be turned off by making the corresponding bits = 0.

Step 8. Instruction 0000b (0h), then 0001b (1h) then delay > 3 ms or check BF
This is the **Clear Display** instruction which, since it has to write information to all 80 DDRAM addresses, takes more time to execute than most of the other instructions. On some flow charts the comment is incorrectly labeled as 'Display on' but the instruction itself is correct.

Step 9. Instruction 0000b (0h), then 0110b (6h), then delay > 53 us or check BF
This is the **Entry Mode Set** instruction. This instruction determines which way the cursor and/or the display moves when we enter a string of characters. We normally want the cursor to increment (move

from left to right) and the display to not shift so we set I/D=1 and S=0. If your application requires a different configuration you could change this instruction, but my recommendation is to leave this instruction alone and just add another **Entry Mode Set** instruction where appropriate in your program.

Step 10. Initialization ends
This is the end of the actual intitalization sequence, but note that step 6 has left the display off.

Step 11. Instruction 0000b (0h), then 1100b (0Ch), then delay > 53 us or check BF
This is another **Display on/off Control** instruction where the display is turned on and where the cursor can be made visible and/or the cursor location can be made to blink. This example shows the the display on and the other two options off, D=1, C=0, and B=0.

About the Delays

It is important to make sure that the LCD controller has finished executing an instruction before sending it another one, otherwise the second instruction will be ignored. The data sheet gives specific times for the power up delay and for the first few instructions. After that it specifies that you must either check the busy flag to see if the LCD controller is finished or wait a sufficient amount of time, a time that is longer than the instruction execution time. The Instruction Set has a column which lists 'typical' instruction execution times and at the bottom of that column most data sheets indicate the clock speed (LCD controller clock) for which that time is valid. One data sheet that I have indicates that you should add 10% to the value when using software time delays but I have been a little more conservative. The datasheet from which I got the flowcharts above specifies that most instructions execute in 37 uS with a clock frequency of 270 KHz. Elsewhere in that same datasheet the clock specifications indicate that it may run as low as 190 KHz, which would increase the instruction execution time by more than 40%. It is this 'worst case' upon which I have based my recommended time delays.

Back to the index page.

Last modified on 29 September 2012