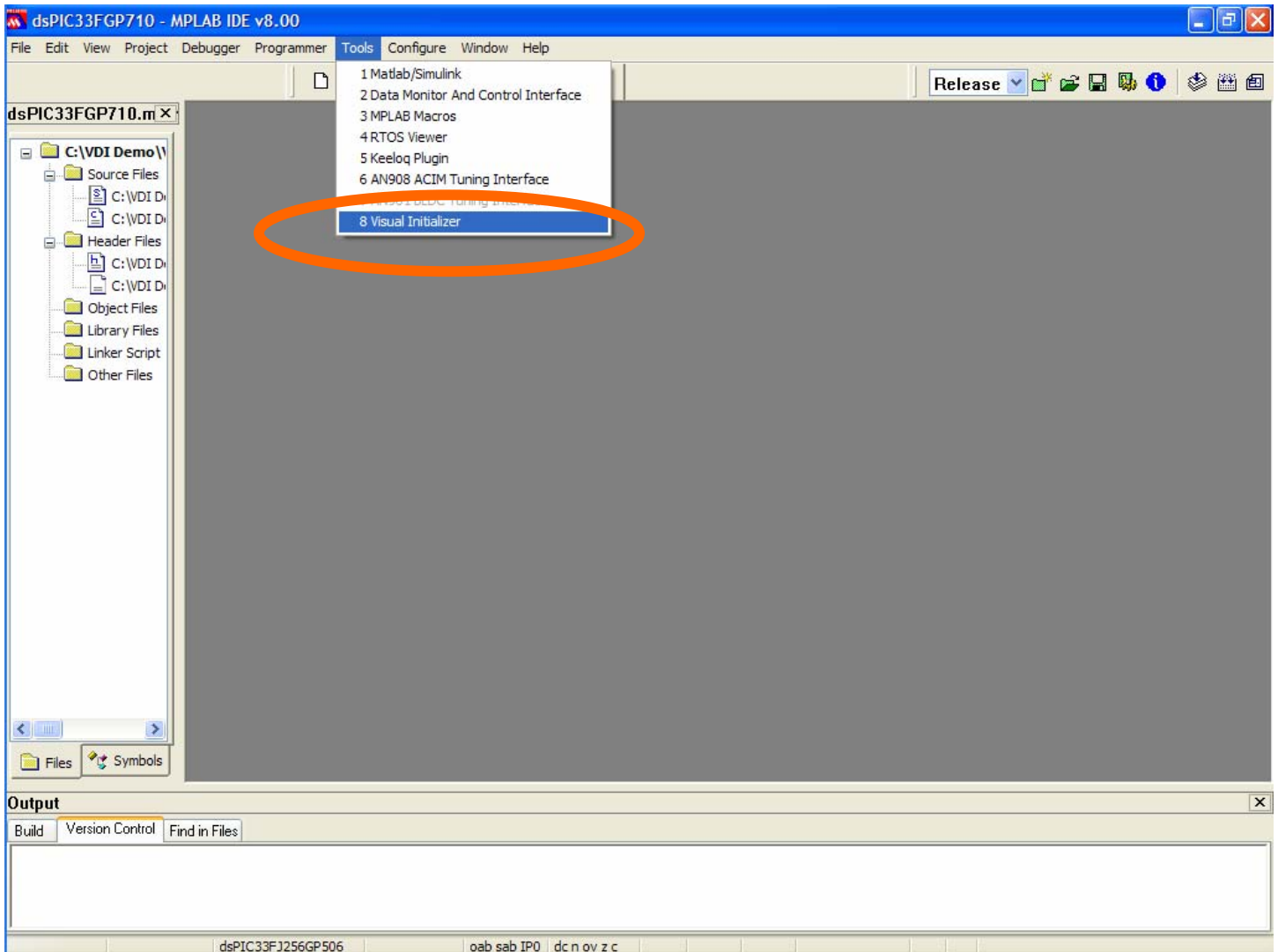


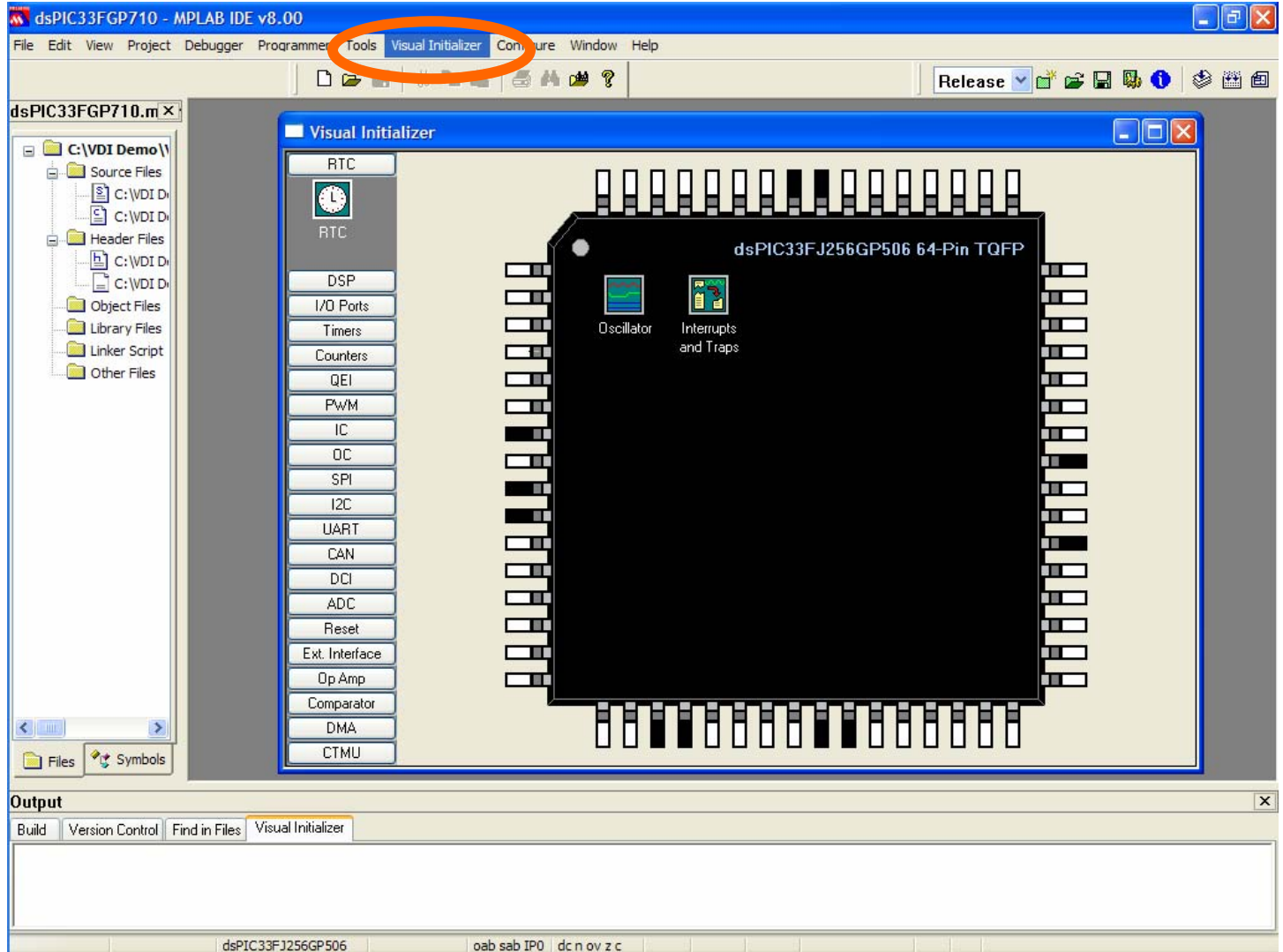
# MPLAB VISUAL DEVICE INITIALIZER

*The simplest, fastest way to initialize your PIC microcontroller*

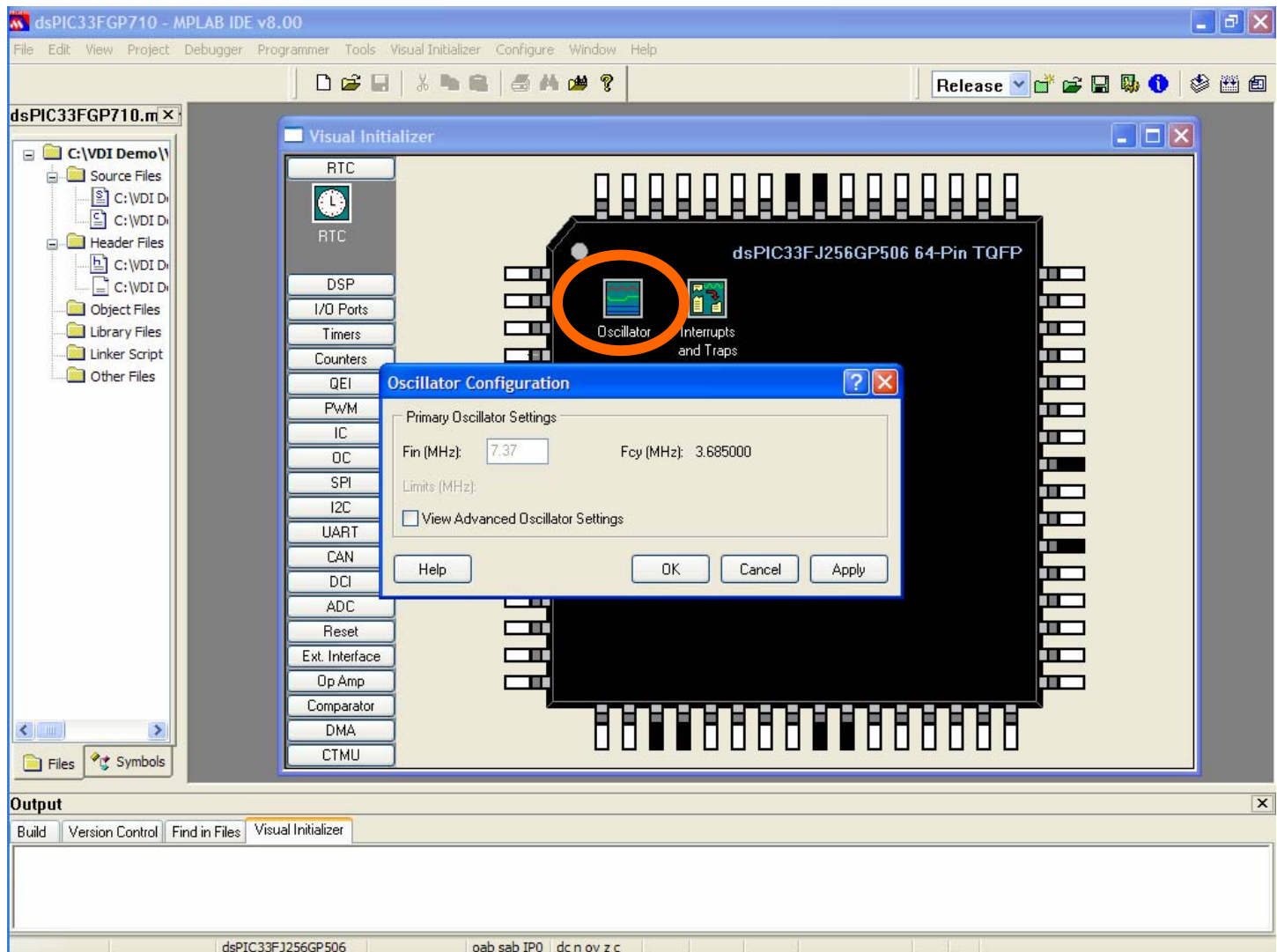
The MPLAB Visual Initializer is really easy to use. After you've used the Project Wizard to set up your project, pick Visual Initializer from the Tools menu.



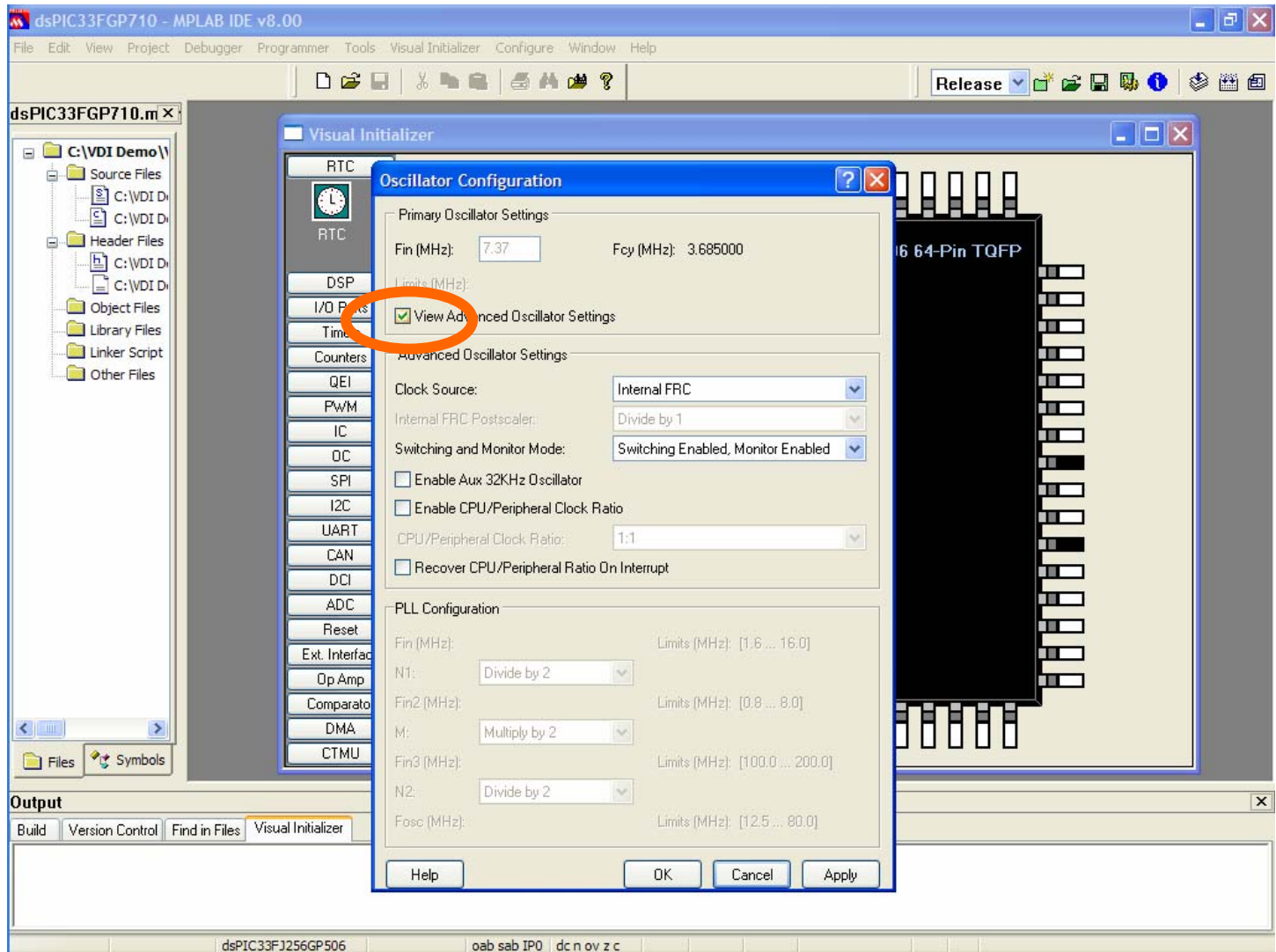
The first thing you'll see is a big picture of the device you are using. You may also notice that a "Visual Initializer" menu has been added to the menu bar.



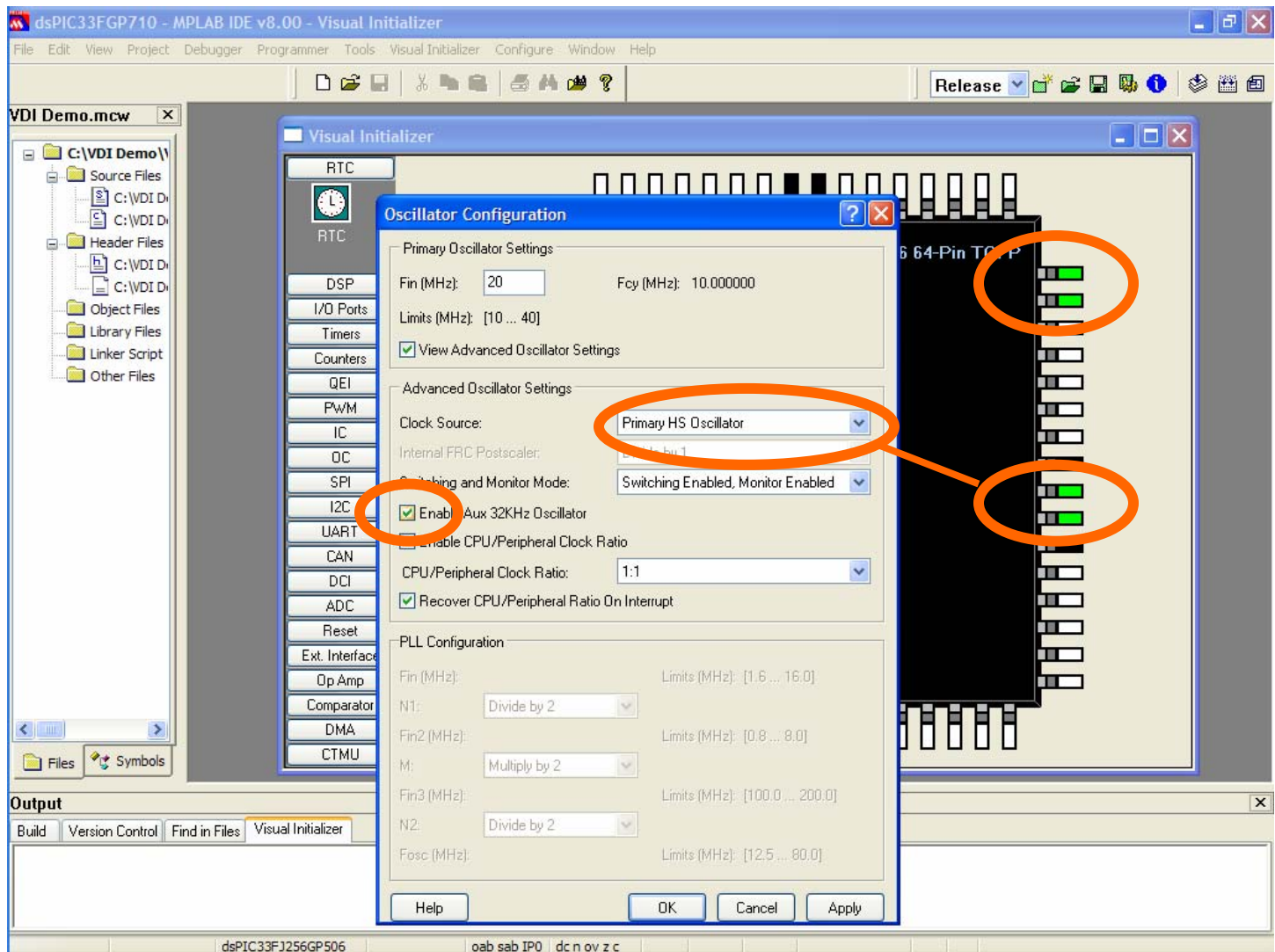
You will probably want to set up your system oscillator first. It is a logical first step, but it also allows the VDI to calculate baud rates, timer intervals and other oscillator-derived values for you. To configure the oscillator, just click it. A dialog shows you the default and simplest oscillator configuration, using the internal fast RC oscillator at 7.37 MHz in this case.



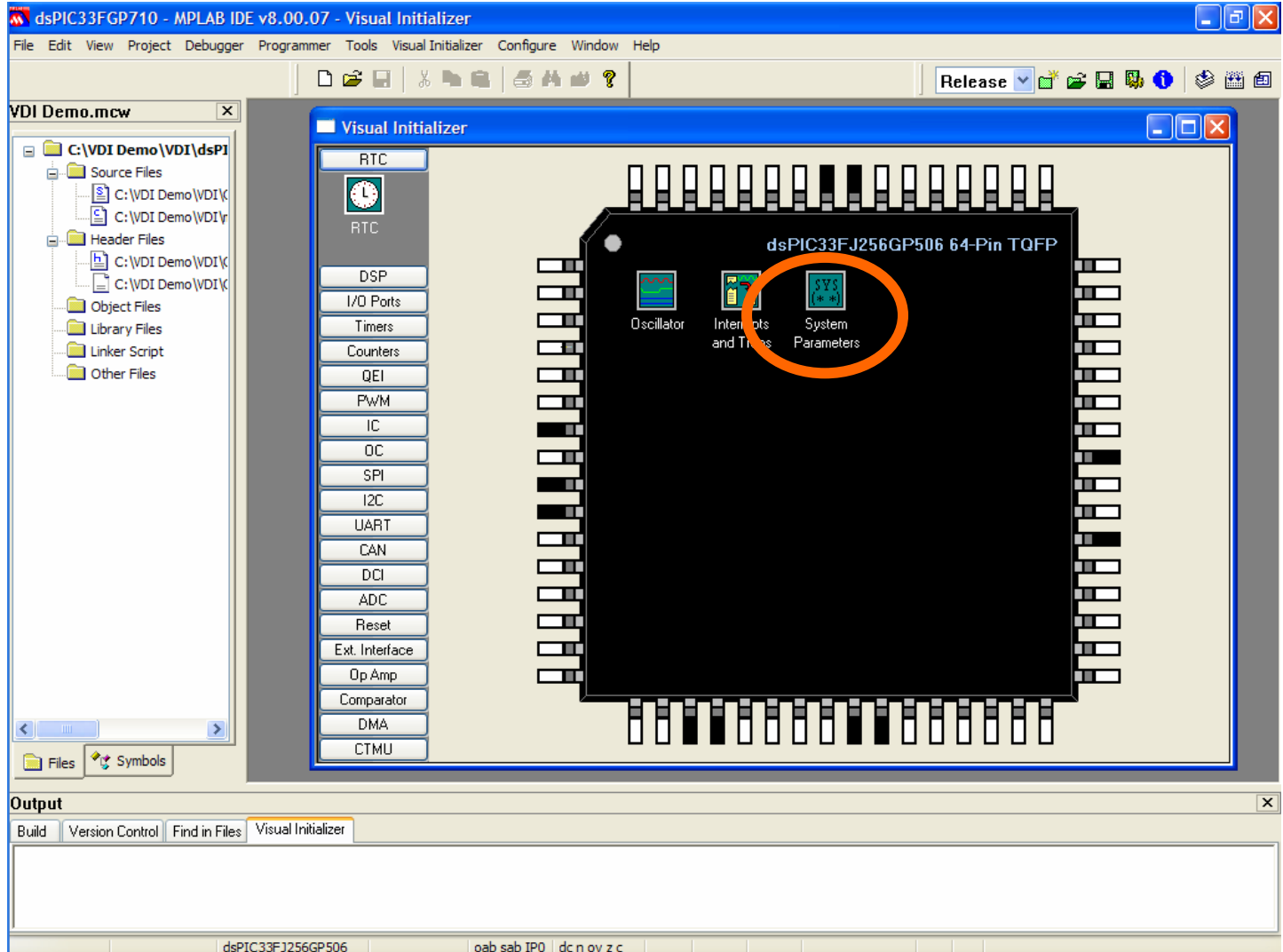
If you want to use more advanced oscillator settings in your project, check the “View Advanced Oscillator Settings” box and the great richness of dsPIC oscillator options is made available to you.



I picked the primary HS oscillator at 20 MHz, enabled the auxiliary 32 KHz oscillator, and enabled clock switching. Pushing the “Apply” button after each step lets you see the pins that are allocated to each function. Click OK when you’re done.

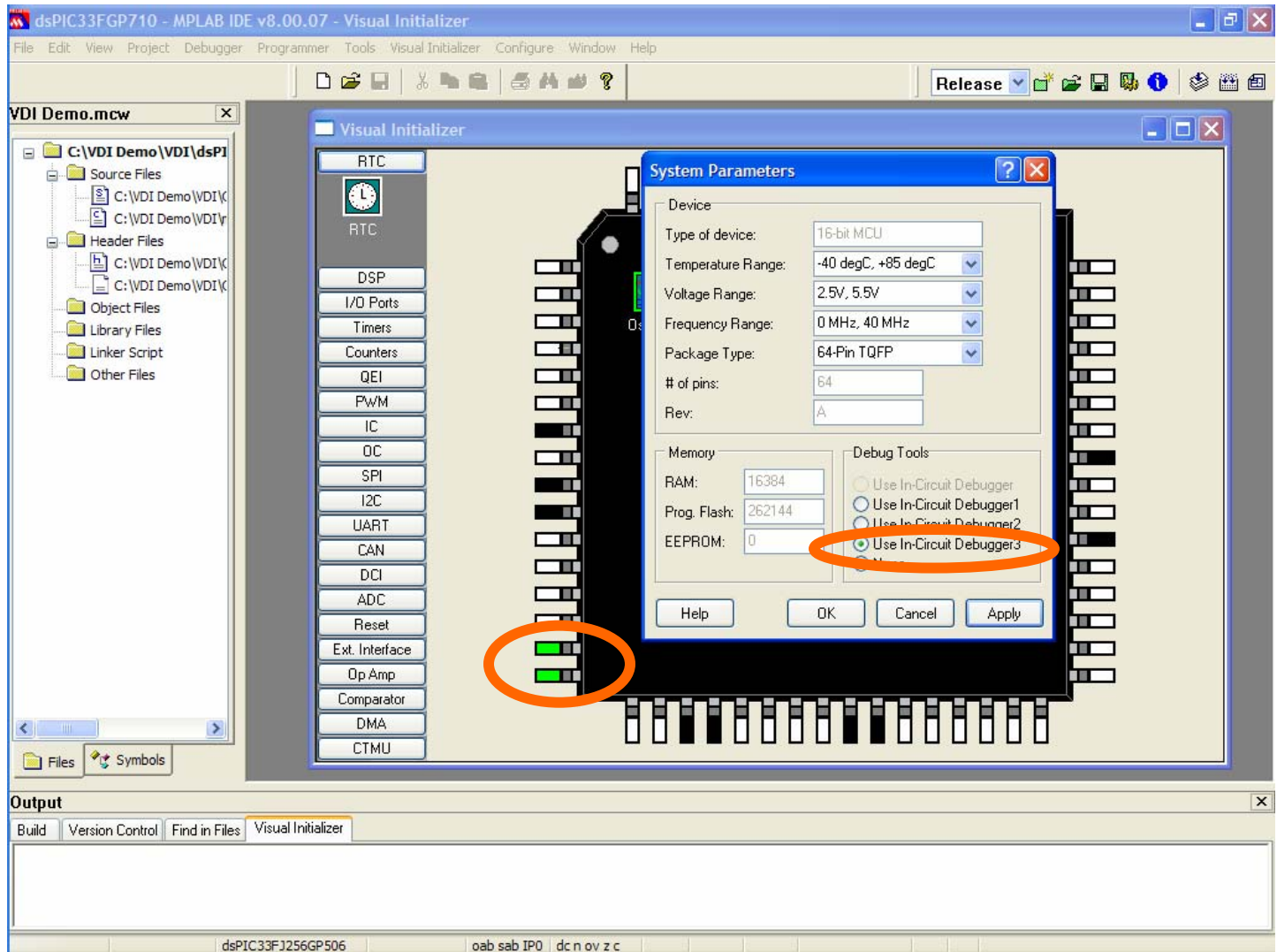


Since you'll probably be writing and debugging your application, you can use System Parameters to dedicate a debugger channel.





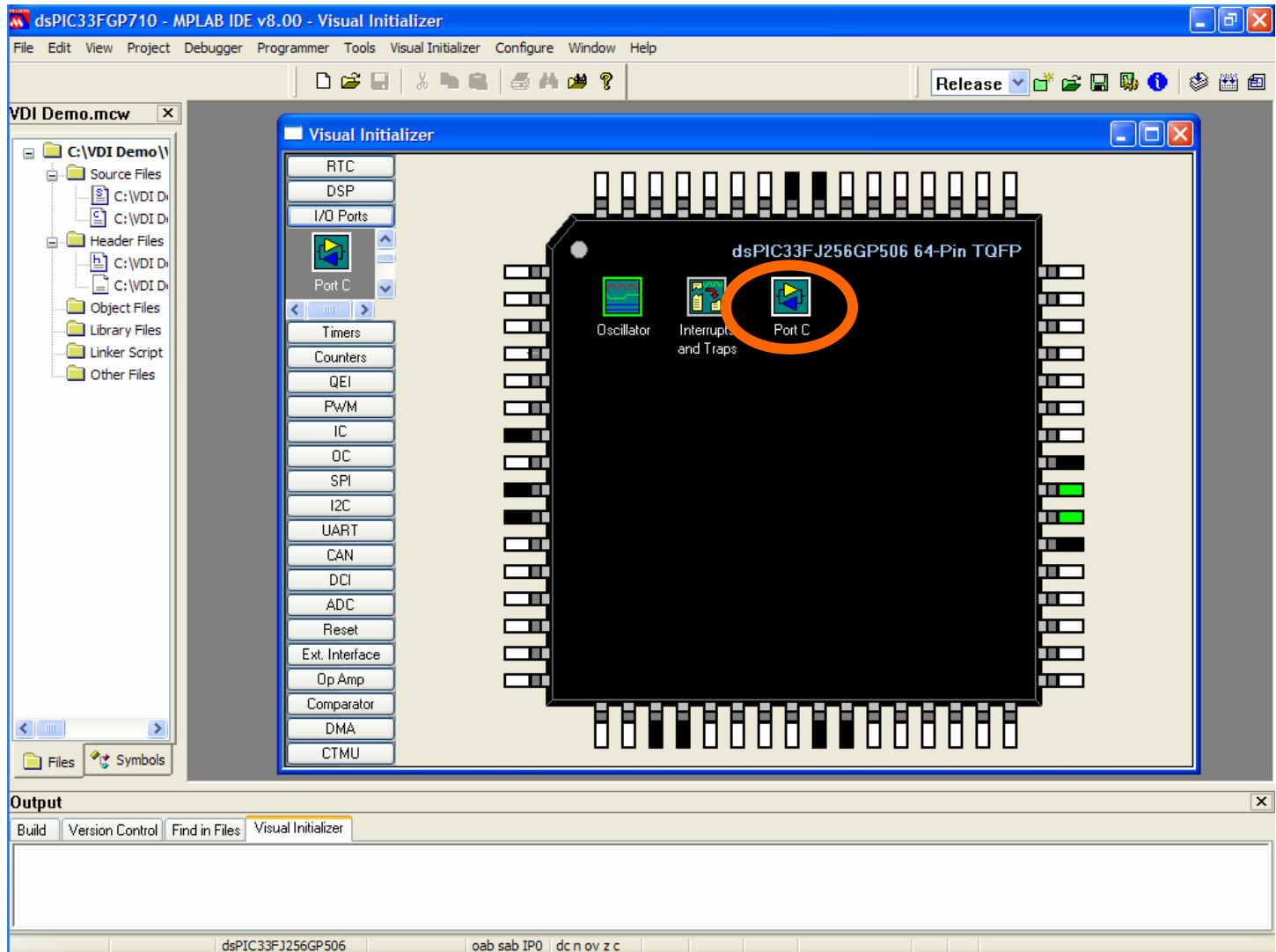
I chose debugger channel 3.



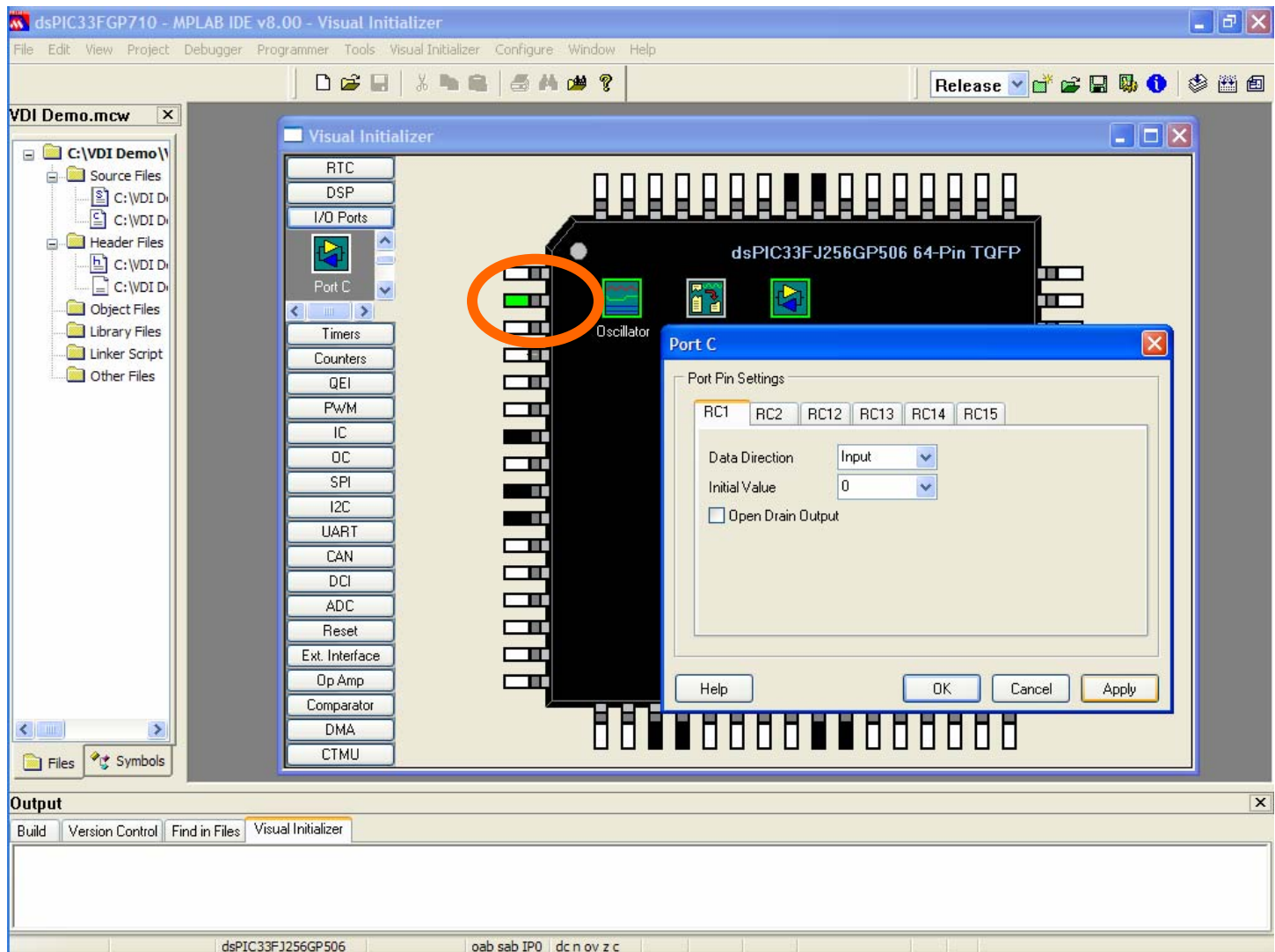
The screenshot shows the MPLAB IDE interface. The main window is the 'Visual Initializer' for a 'dsPIC33FJ256GP506 64-Pin TQFP'. On the left, a tree view shows the project structure. The 'I/O Ports' module is selected in the left pane and highlighted with an orange circle. Below it, 'Port C' is also highlighted. The main area shows a diagram of the microcontroller with pins 1 through 40 labeled. The 'Output' window at the bottom is empty.



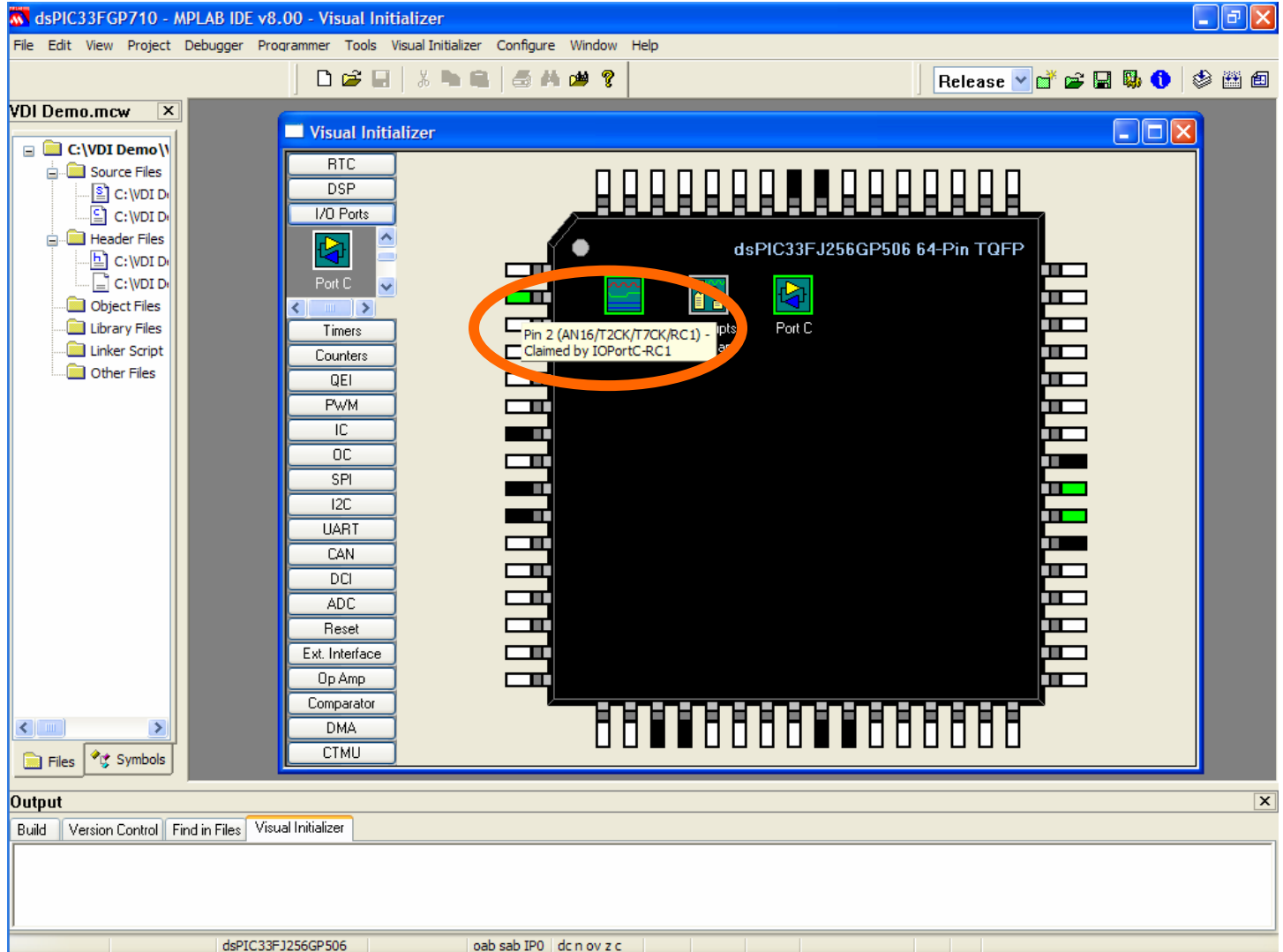
Just drag it (the Port C icon) off of the palette and drop it on the chip outline. Click it to configure it.



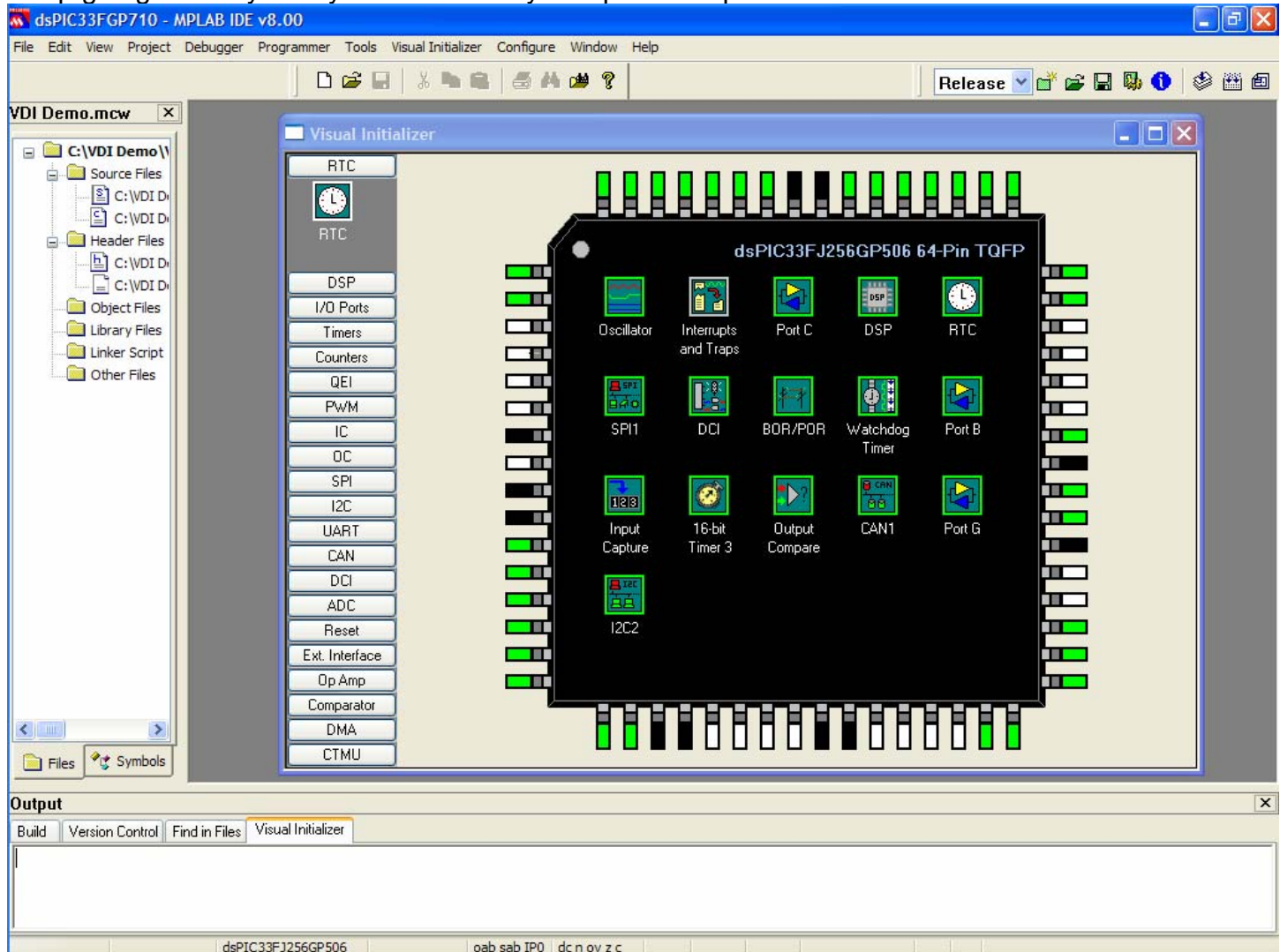
An easy to understand dialog shows the capabilities of the facility you choose. In this case, I set Port C pin RC1 to an input and clicked apply. The green pin (pin 2, second from top on the left) is RC1 and is “allocated” on the chip outline.



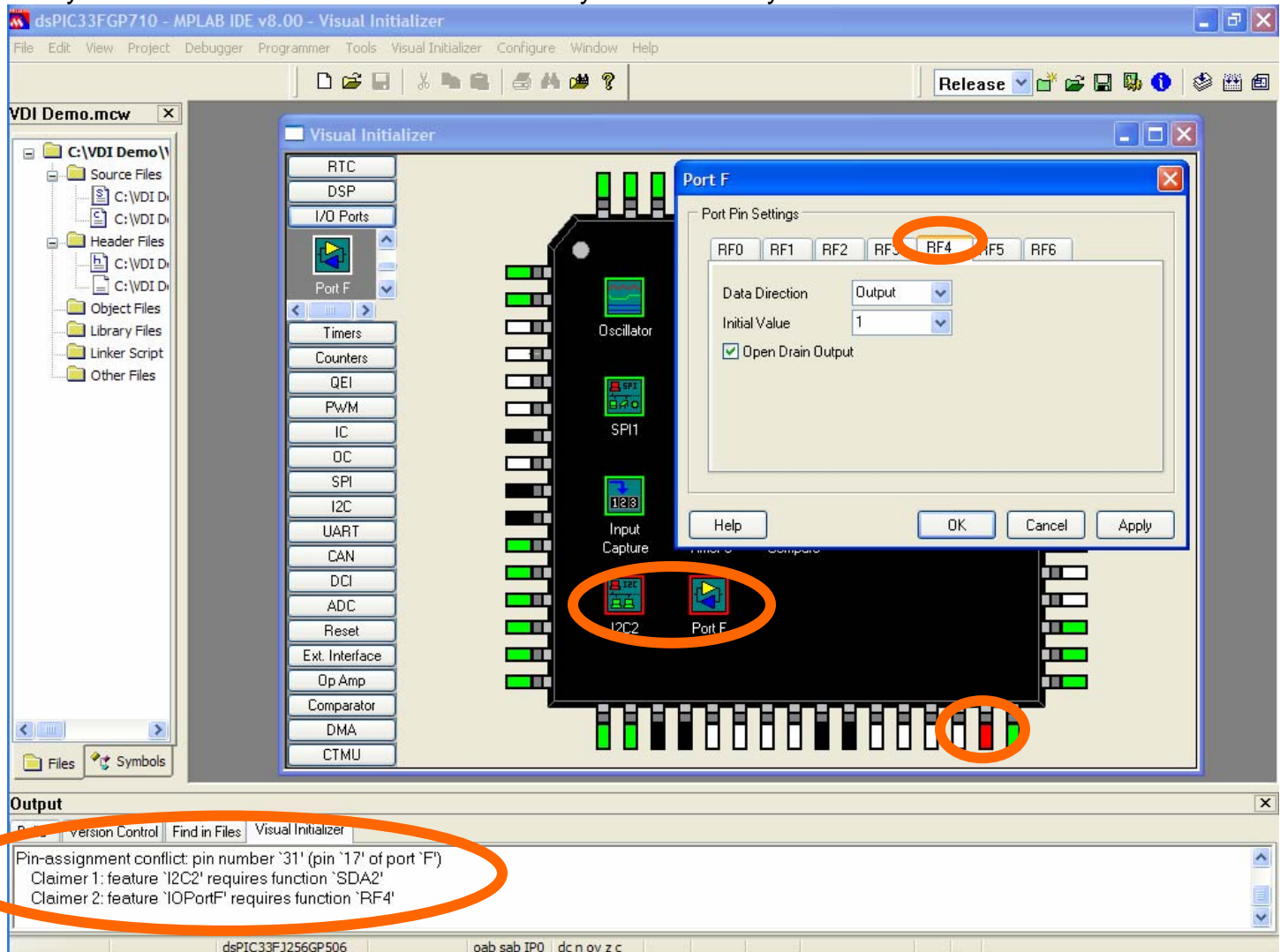
If you click OK to finish the I/O Port configuration and float your mouse cursor over a pin, a tool tip will pop up showing all of the potential pin assignments and the pin assignment you made.



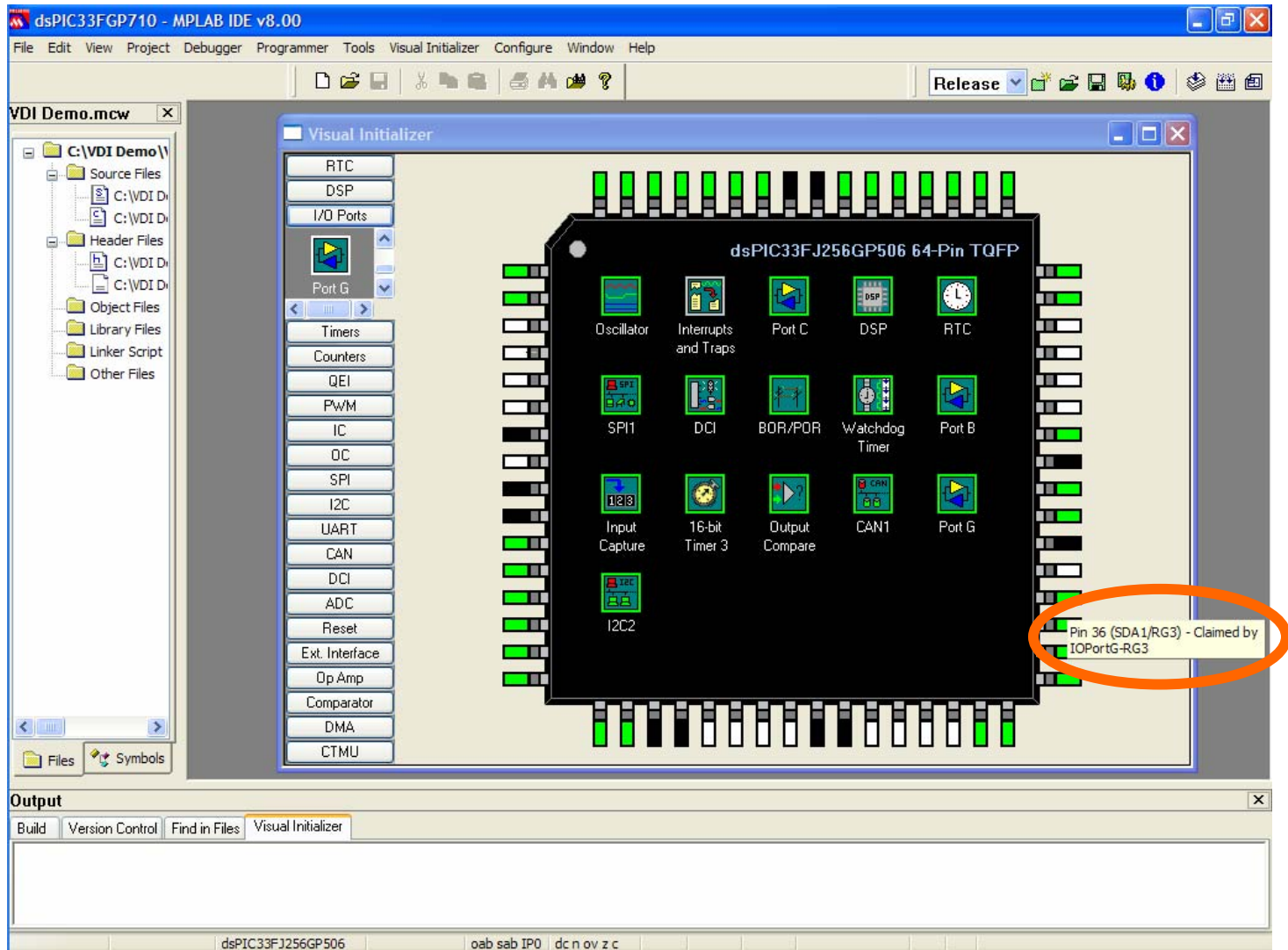
Keep going this way until your functionality and pinout requirements are met.



If you use Apply regularly, any pin allocation conflict becomes easy to fix. Here, I tried to make RF4 an open drain output, but pin 31 is already assigned the SDA function for I<sup>2</sup>C #2. Notice that the pin turns red, a message is printed in the output window and both icons have red collars. Since I/O pins are more plentiful than I<sup>2</sup>C functions, it's probably more effective to change the I/O pin, but *you* always make the choice. VDI doesn't force any decisions on you.

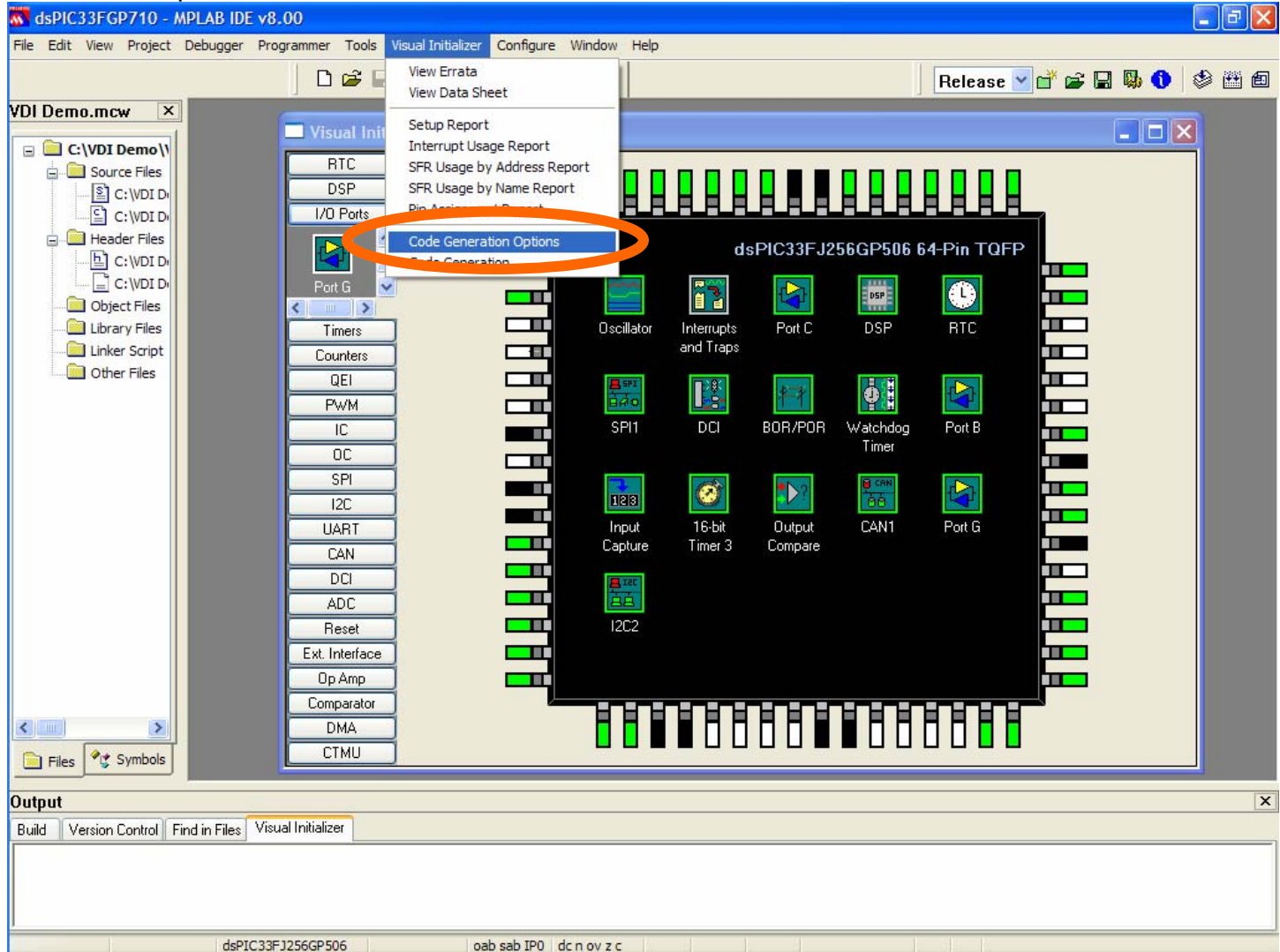


Port G has an open pin, RG3, which is nearby on pin 36, so I chose to use that instead.

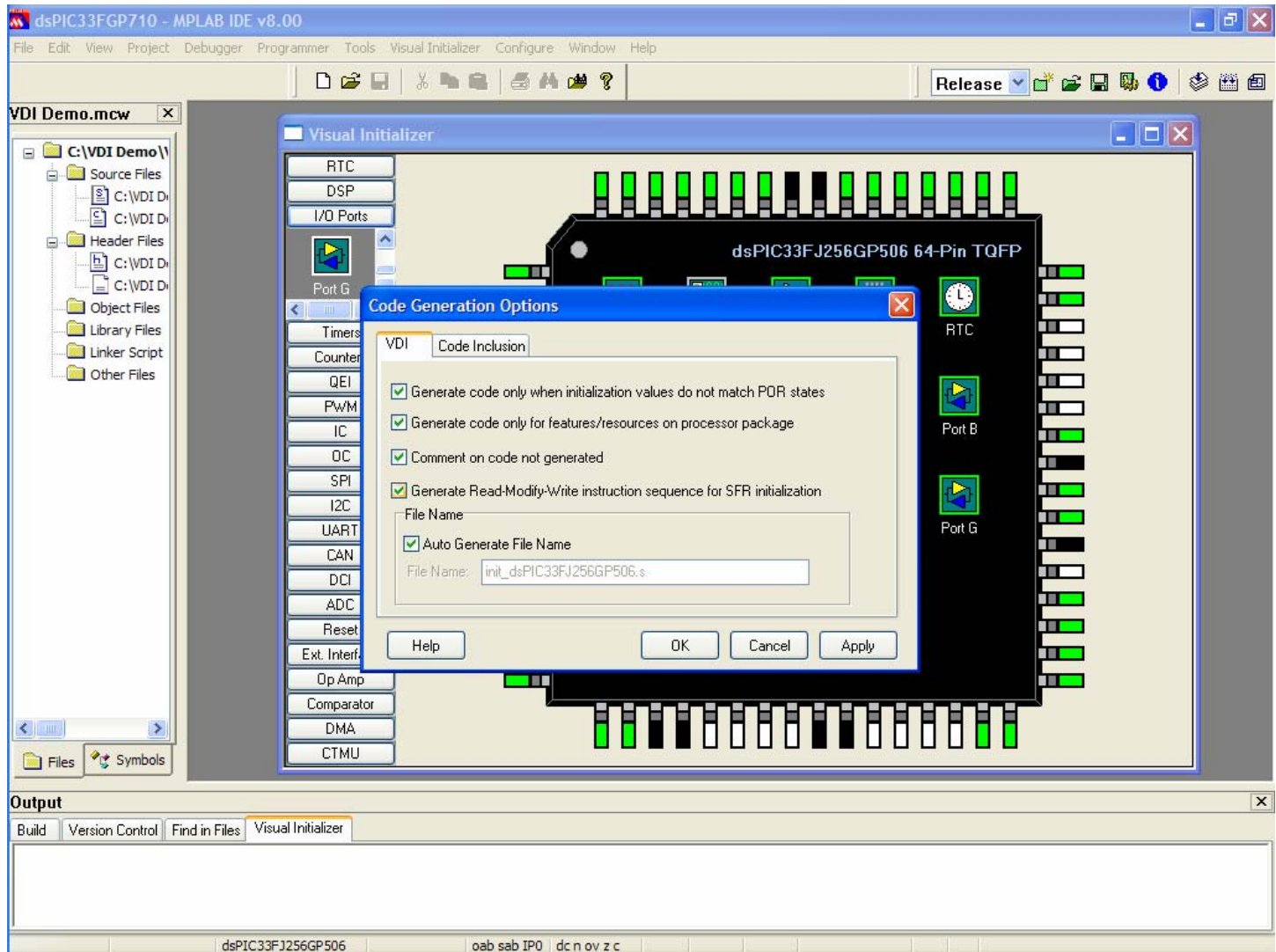




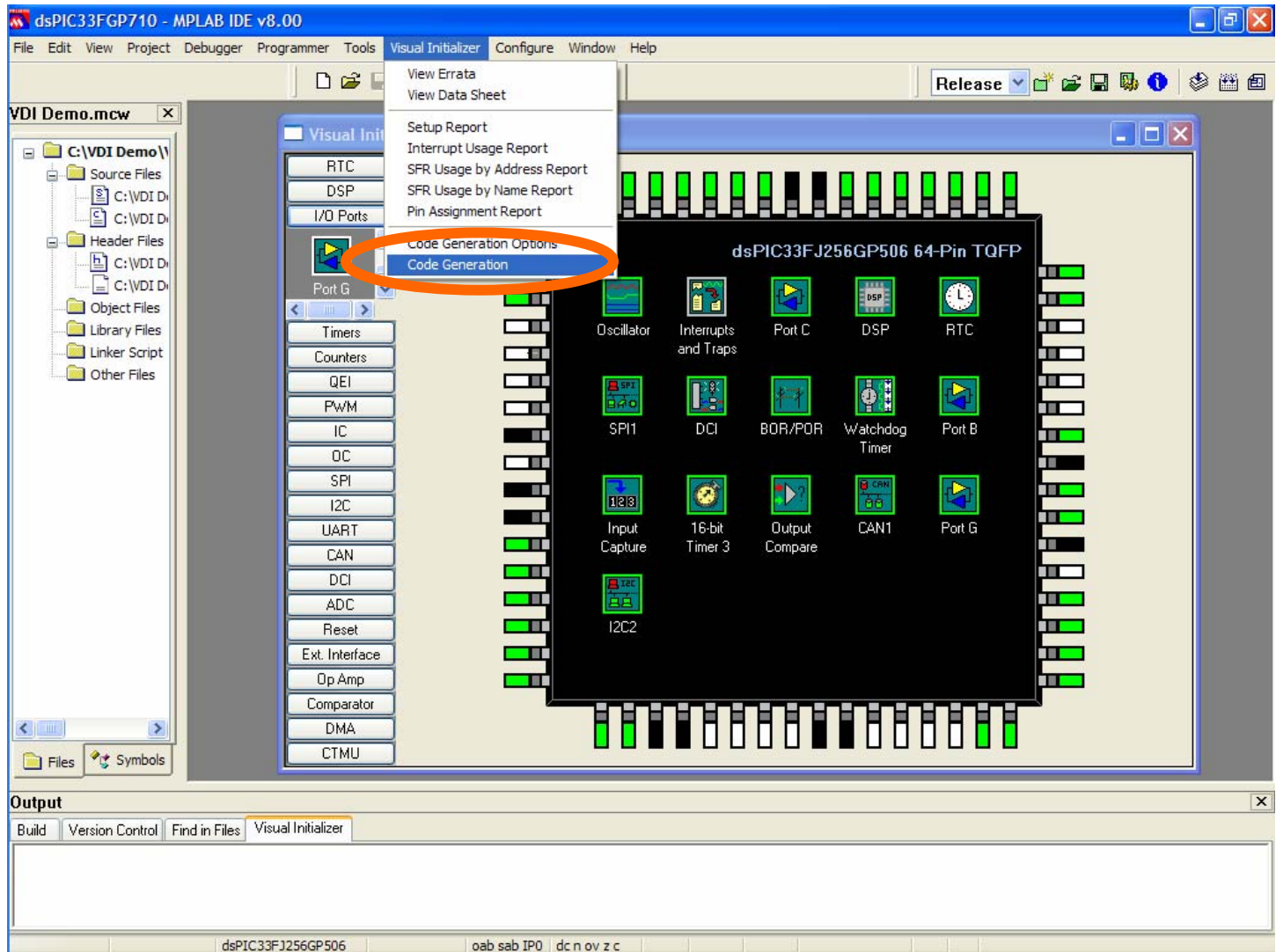
Now I'm done with my initialization, so I need to generate the initialization code. Sometimes it is important to tailor the code that is generated so I open the Visual Initializer menu and choose Code Generation Options.



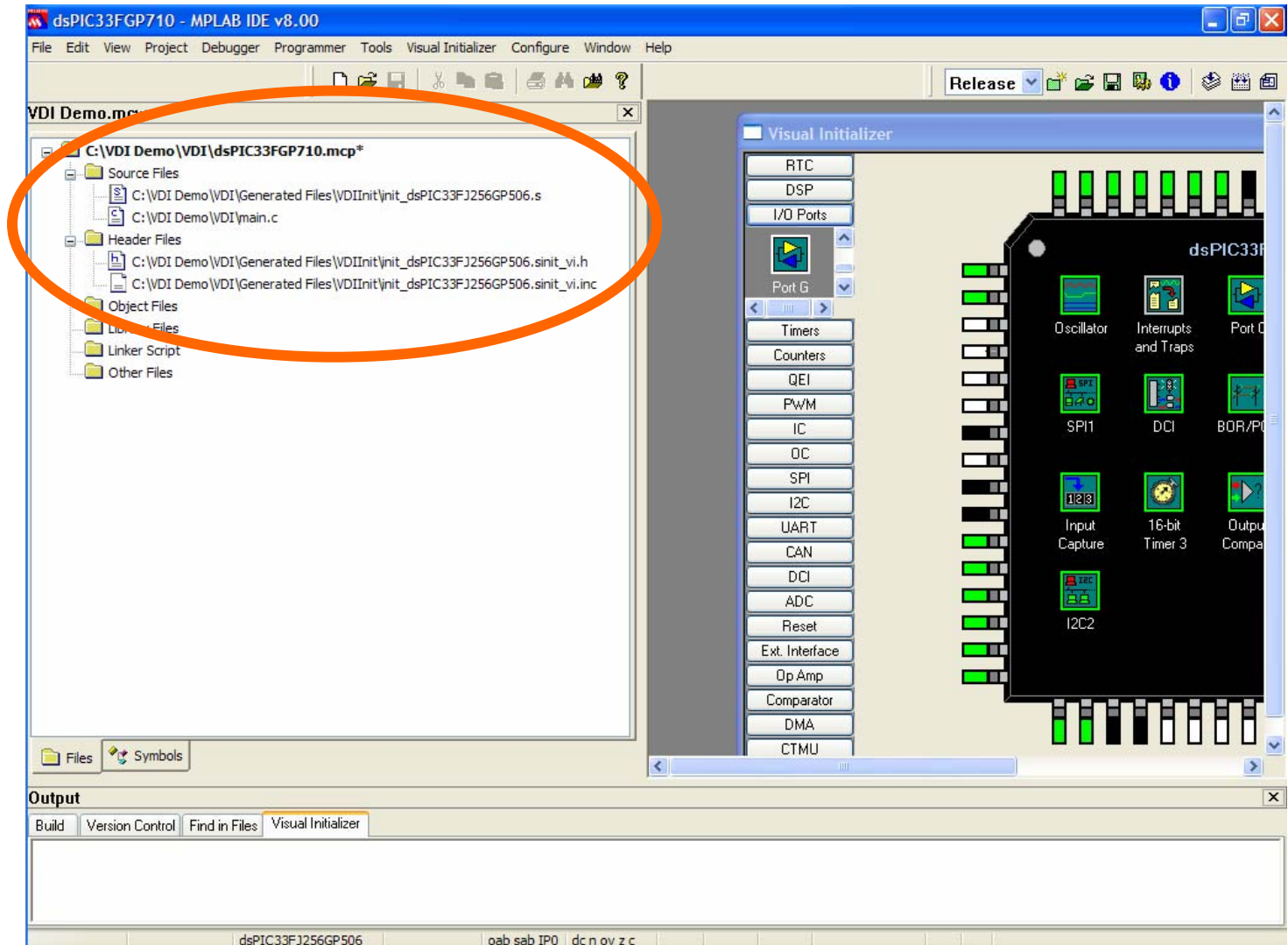
I can tailor the code in many ways to make it smaller or change how SFRs are initialized.



Finally, I'll generate the code implied by my setup and the code generation options I chose by selecting the Code Generation item in the Visual Initializer menu.



You'll notice that three new files have been added to your project: `init_dsPIC33FJ256GP506.s` (which has the initialization code in it), `init_dsPIC33FJ256GP506.sinit_vi.inc` (which has the assembly language "prototype") and `init_dsPIC33FJ256GP506.sinit_vi.h` (which has the 'C' language prototype).



Now all you have to do is call the initialization in your code and you're done.

