Using the xPC Target Driver Authoring Tool to Create a Delay Timer
December 2, 2008

Introduction

The xPC Target driver authoring tool, `xpcdrivertool`, helps you create Simulink S-function blocks to run in the xPC Target environment. Use the tool to create simple utility blocks and custom device drivers that include calls to xPC Target kernel functions[1].

This document describes how to create a time delay block using `xpcdrivertool`. When you add the block to a model, the delay block will wait a user-specified amount of time during each simulation time step. This is useful when you need to profile or investigate timing results as well as when you want to reserve or pre-allocate a specified amount of the simulation time step to account for future functionality. A test model is provided to demonstrate use of the delay block. Test results are also included[2].

This work was performed using xPC Target Version 4.0 (R2008b).

Using `xpcdrivertool` to Create a Time Delay Block

To invoke the xPC Target driver authoring tool, type the following in the MATLAB Command Window

```
>> xpcdrivertool
```

Figure 1 displays the user interface.

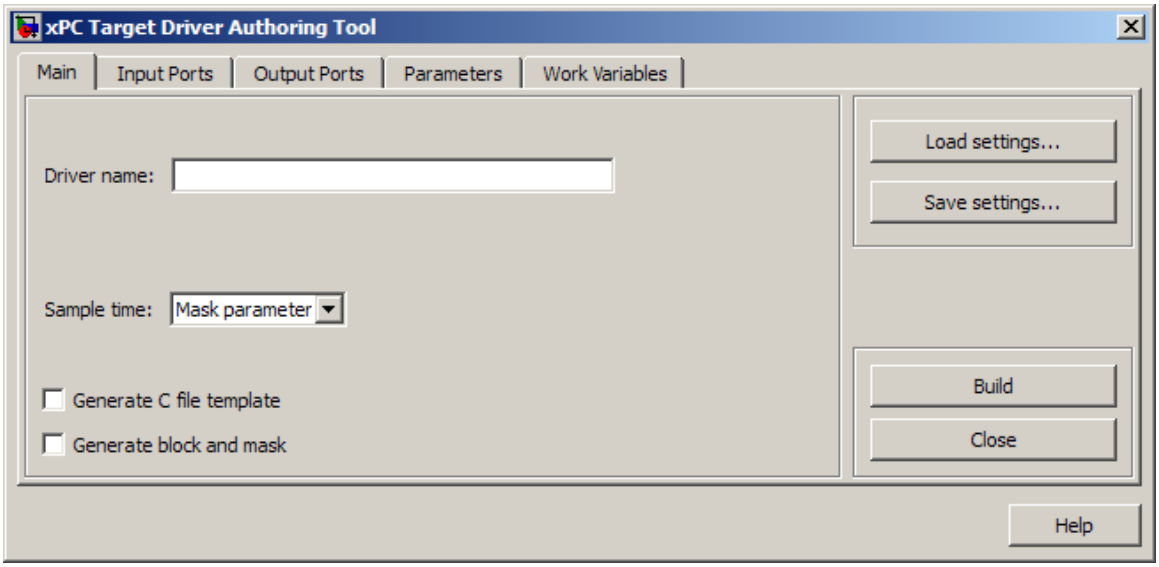

Figure 1 – xPC Target Driver Authoring Tool.

[1] For a complete list of kernel functions, refer to the xPC Target Function Library Quick Reference Sheet, `xPCTargetFunctionLibraryQuickReference.pdf`. This document is in the directory *matlabroot*\toolbox\rtw\targets\xpc\target\build\xpcblocks\thirdpartydrivers.

[2] A separate MATLAB Central File Exchange post, **xPC Target Delay Timer Block**, provides additional examples and detail on the use of this block.

The typical work flow to create an S-function block using `xpcdrivertool` is as follows:

1. Specify the block requirements (e.g., inputs, outputs, parameters, work variables).
2. Generate source code files, which will include the above requirements.
3. Modify the block template C file with your custom code.
4. Compile the source code and generate the Simulink block.
5. Customize the block mask.

Step 1: Specify the Block Requirements

The S-function in this example is named `delayTimer`. Start by entering the name in the **Driver name** edit box on the **Main** tab as shown in Figure 2. Change the **Sample time** option to `Inherited` so that the block inherits its sample time from the Simulink model.

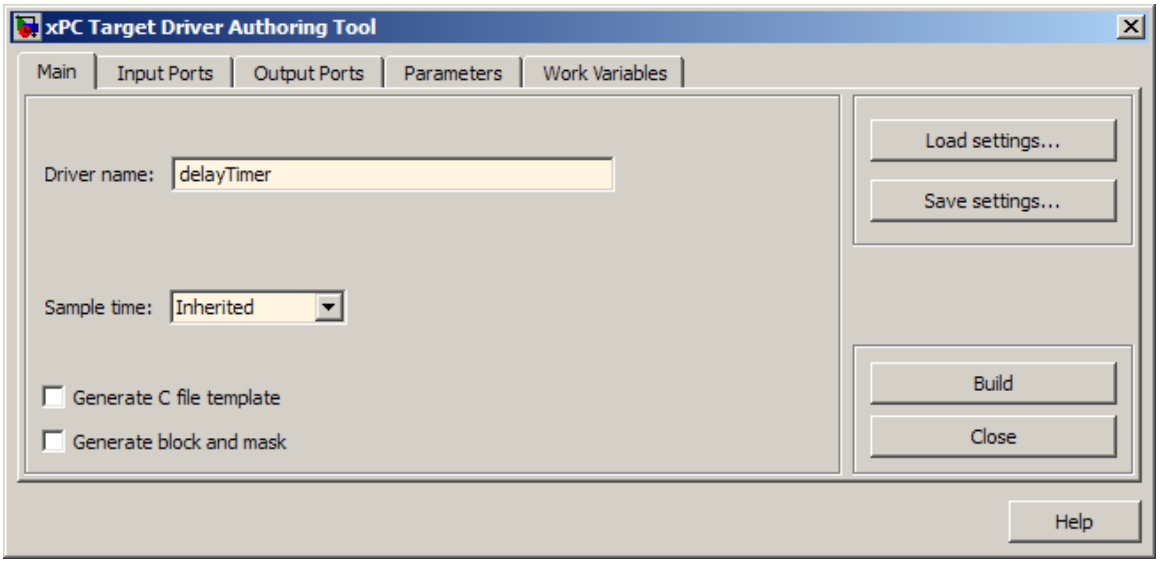

Figure 2 – Main tab.

Because the block implements a simple delay, no input or output ports are needed. As indicated in Figures 3 and 4, you can skip and leave unchanged the **Input Ports** and **Output Ports** tabs.

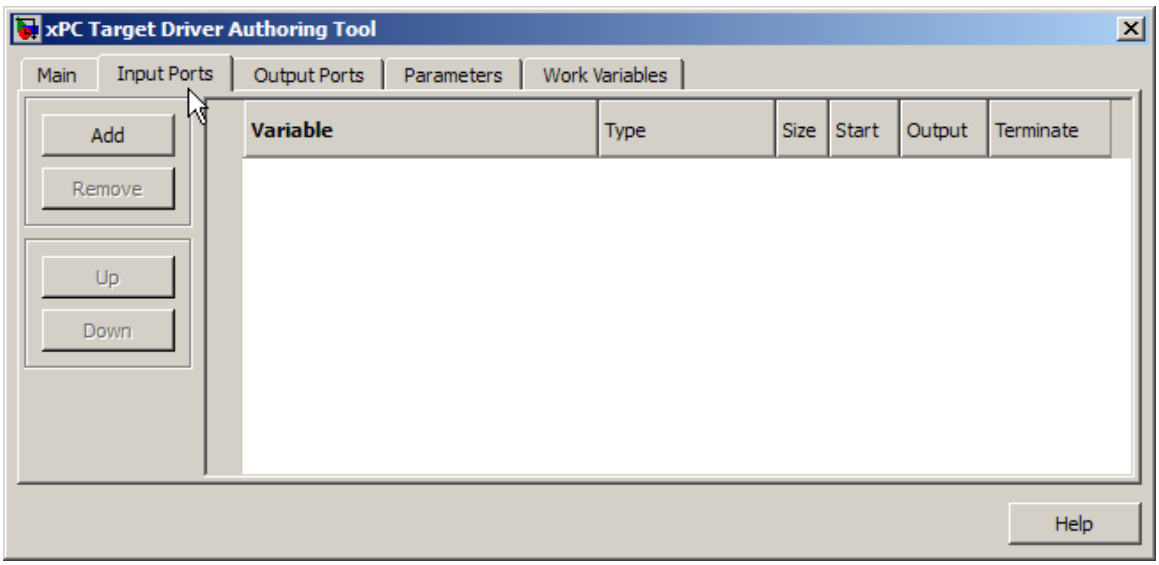

Figure 3 – Input Ports tab.

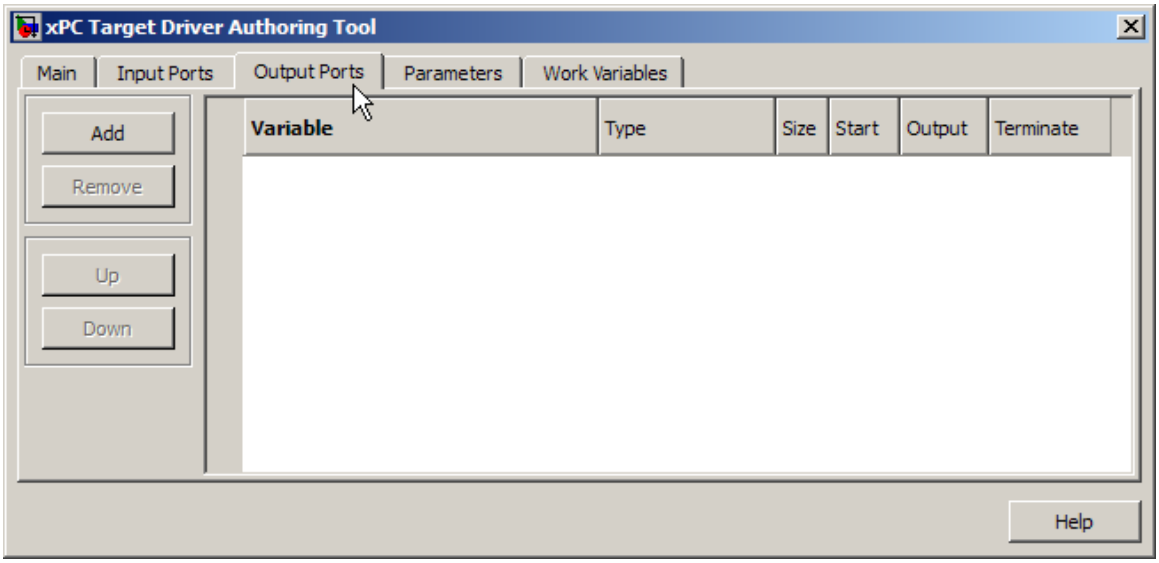

Figure 4 – Output Ports tab.

The `delayTimer` block needs one parameter to allow the user to specify the desired delay time in seconds. In the **Parameters** tab, add a `delay` variable as shown in Figure 5. The S-function uses this variable to pass the time delay value to the model's `mdlOutputs` routine. Inside of `mdlOutputs`, the xPC Target kernel function `xpcBusyWait` is called with argument `delay` to generate the requested delay.

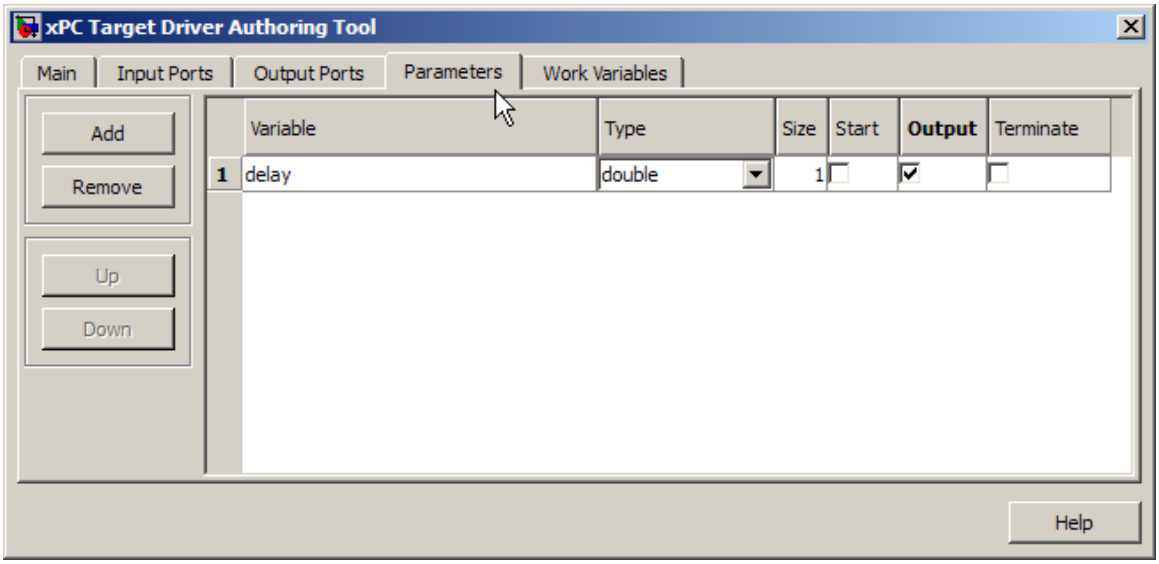

Figure 5 – Parameters tab.

Work variables are required when you need to share data between model routines (e.g., `mdlStart`, `mdlOutputs`, `mdlTerminate`). In this example, no work variables are needed so you can leave the **Work Variables** tab unchanged (see Figure 6).

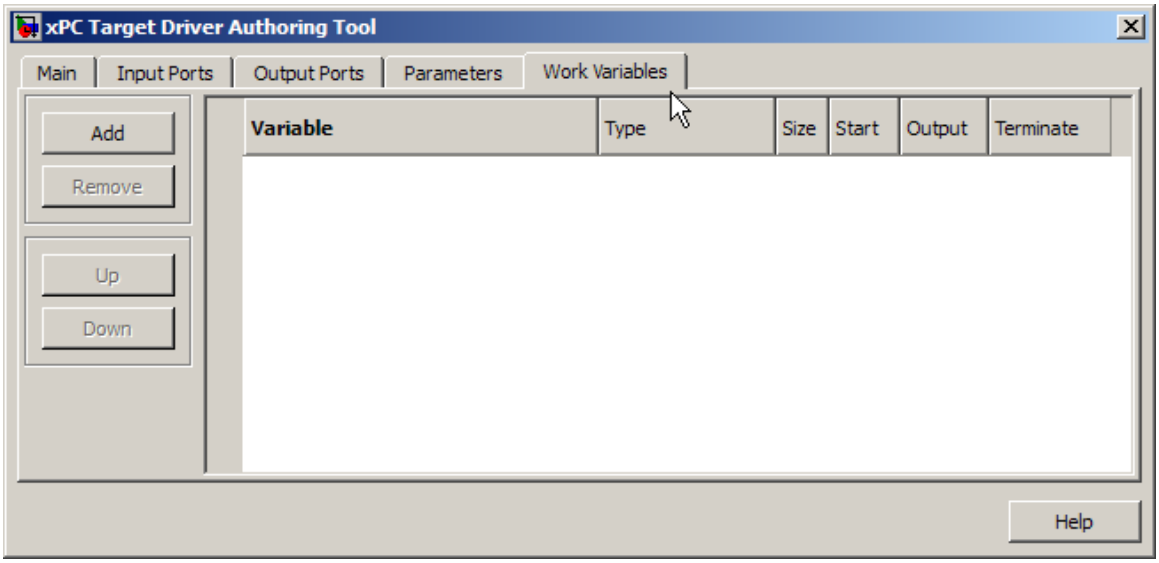

Figure 6 – Work Variables tab.

Step 2: Generate Source Code Files

Return to the **Main** tab and select the **Generate C file template** checkbox then click the **Build** button. This creates the C file template (`delayTimer.c`),which will contain the block functionality, the associated header file (`delayTimer.h`), and the Simulink S-function wrapper files (`sfcn_delayTimer.*`) as shown in Figure 7.

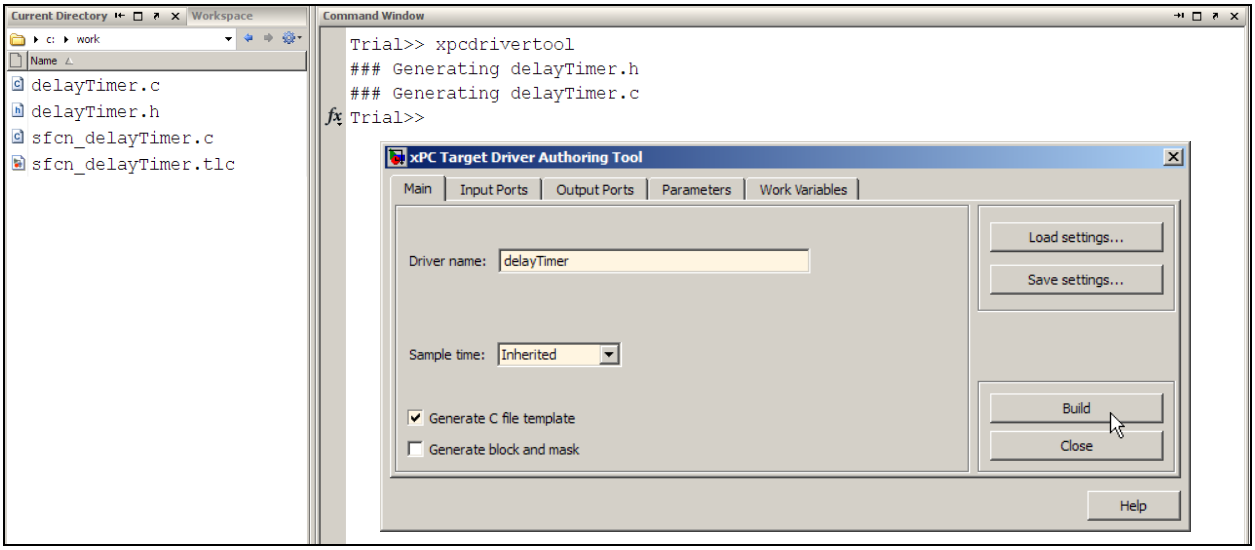

Figure 7 – Build process.

Step 3: Modify the Template C File – `delayTimer.c`

`delayTimer.c` contains a `void` function `delayTimerOutput` that is invoked by `mdlOutputs` at each simulation time step. Within `delayTimerOutput`, insert the call to the function `xpcBusyWait`. This is the kernel function used to create the delay. Specify the `delay` parameter as the input argument to `xpcBusyWait` as shown in Figure 8. Save `delayTimer.c`.

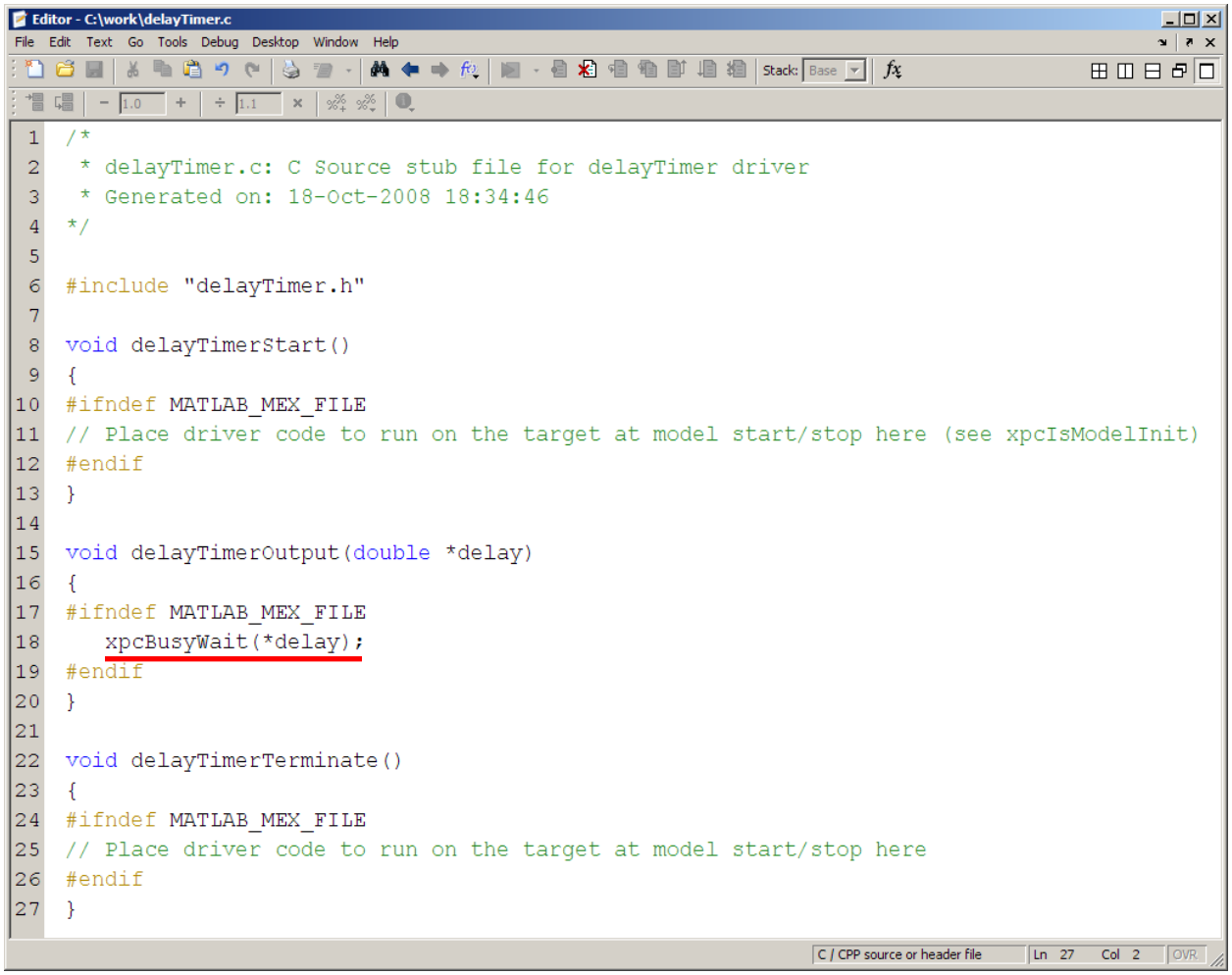

Figure 8 – C function `delayTimer.c`.

Step 4: Compile Source Code and Generate the Simulink Block

With code changes complete, return to the **Main** tab of `xpcdrivertool`. Clear the **Generate C file template** checkbox and select **Generate block and mask**. Click the **Build** button. This creates the Simulink S-function block, compiles all the source code, and generates the block binary MEX-file `sfcn_delayTimer.mexw32` as shown in Figure 9.

Note 1: During this step, ensure that the checkbox for **Generate C file template** is cleared; otherwise, the build command will create a new C file template and overwrite the file you previously edited.

Note 2: You can save your `xpcdrivertool` session at any time to a MAT-file (e.g., `delayTimer.mat`) by checking the **Save settings** button. Afterwards, you can exit `xpcdrivertool` and perform other tasks if desired. This allows you to break your work into multiple sessions. To return to and restore your work, run `xpcdrivertool` and click the **Load setting** button. Choose the MAT-file you previously saved.

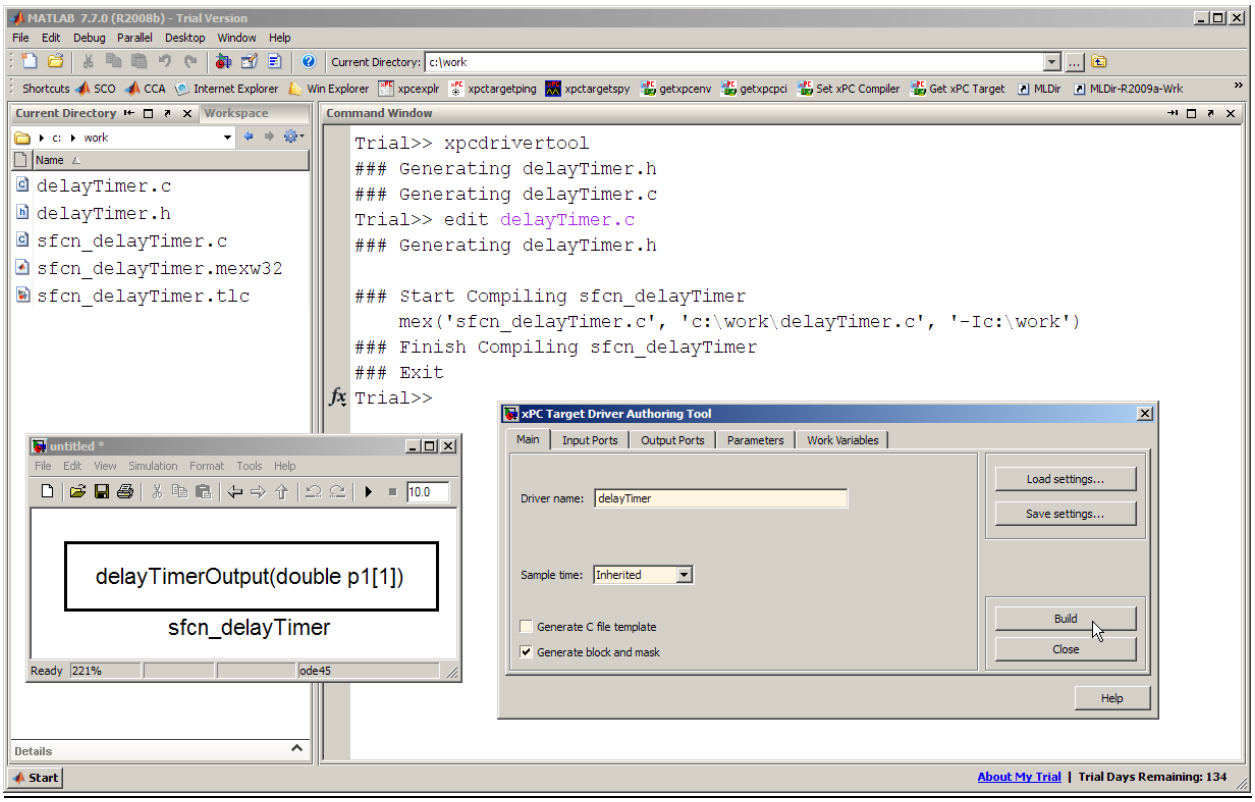

Figure 9 – Creating the delay timer model block.

Step 5: Customize the Block Mask

Customize the block created by `xpcdrivertool` (see Figure 10).

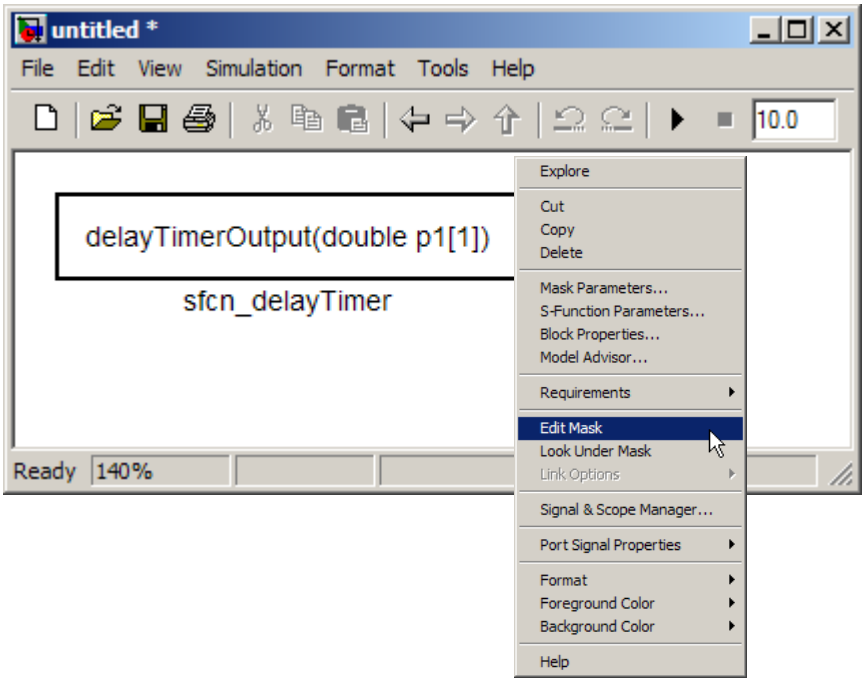

Figure 10 – Editing the delay timer model block.

Right-click on the block and select **Edit Mask**. Modify the **Icon**, **Parameters**, and **Documentation** tabs as illustrated in Figures 11 – 13.

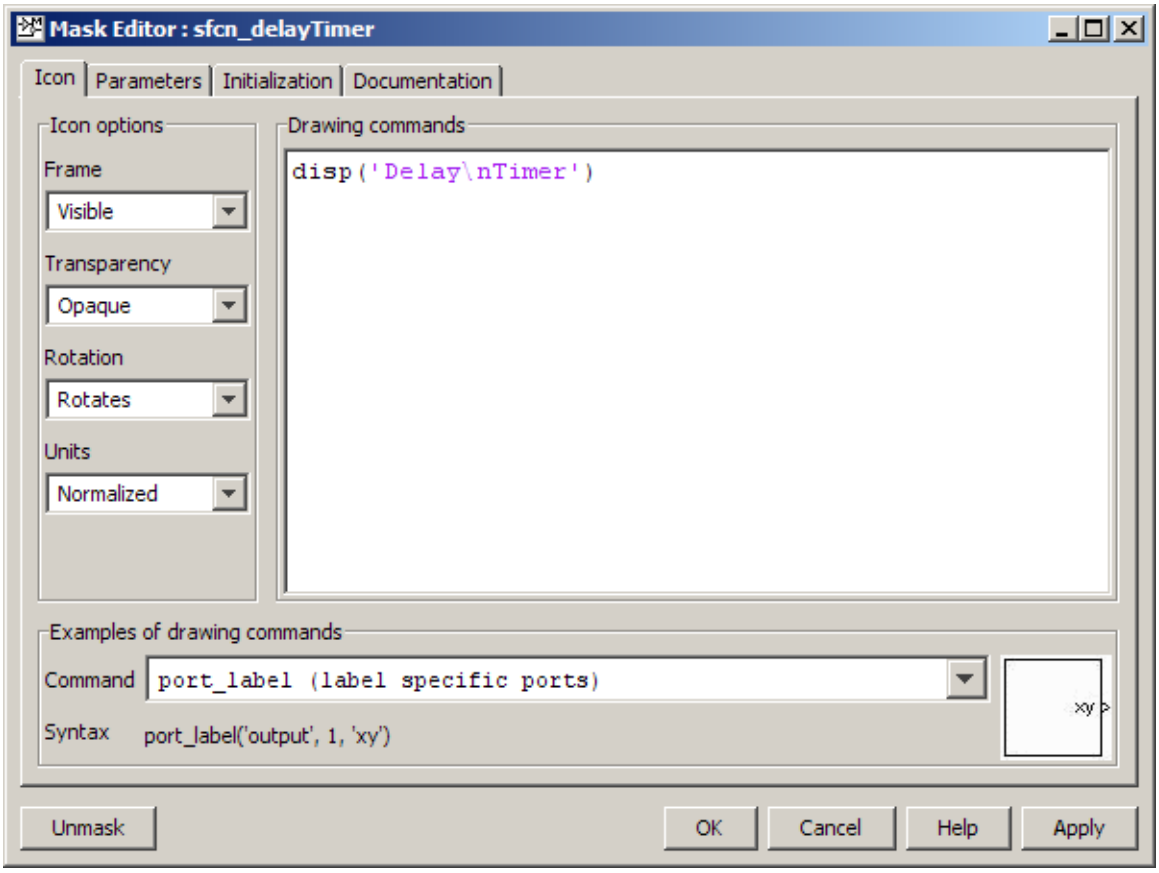

Figure 11 – Changing the block label using the Mask Editor.

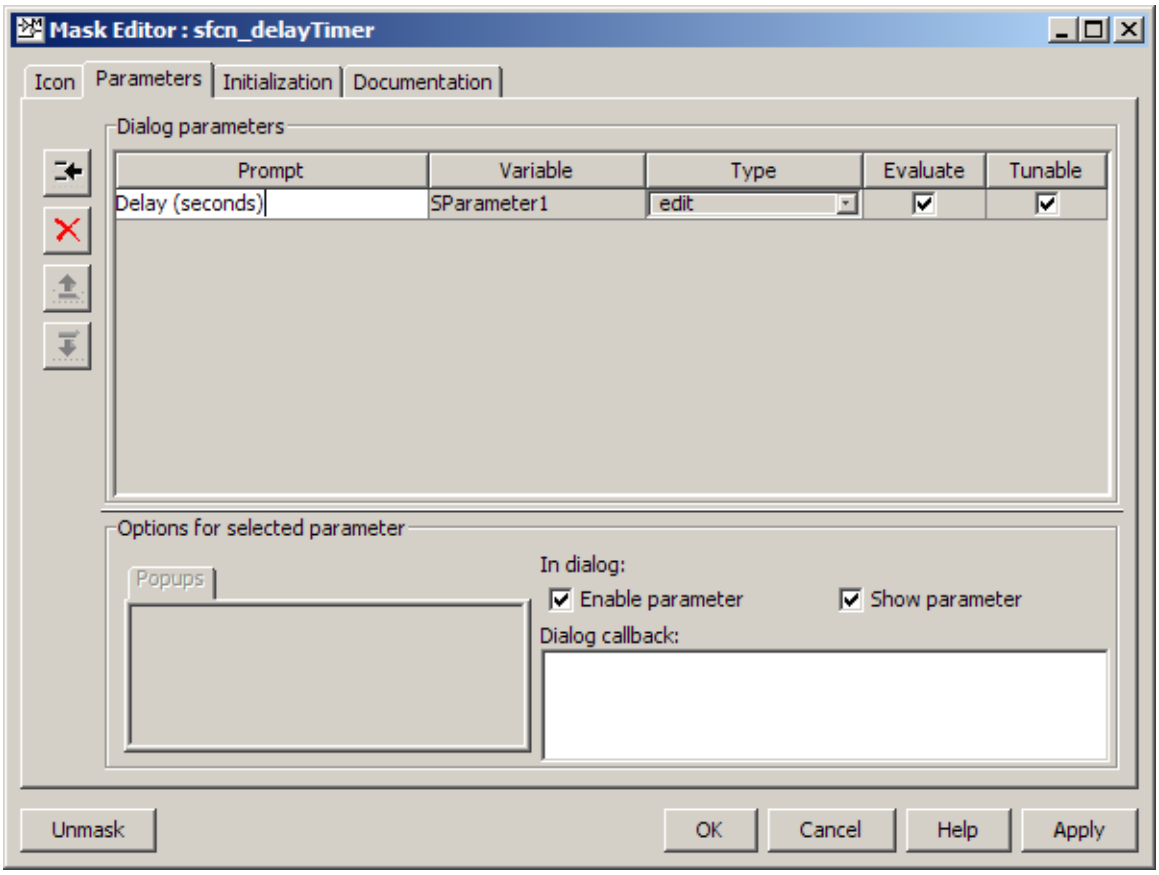

Figure 12 – Changing block parameters using the Mask Editor.

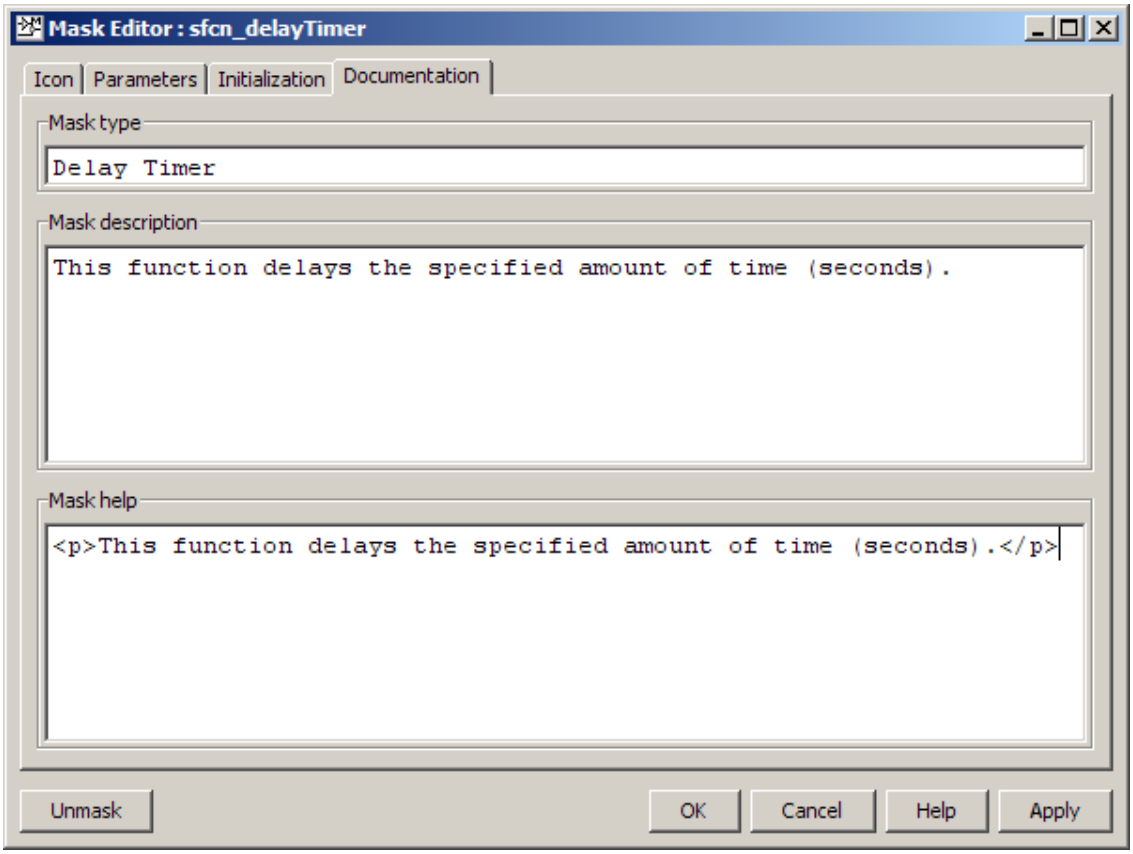

Figure 13 – Changing block documentation using the Mask Editor.

When done, click the **OK** button. Double-click the block to display the Block Parameters dialog so that you can specify the delay time as shown in Figure 14. Save the model as `delayTimerLib.mdl`. This model now contains the single block, `Delay Timer`. To use this block, drag it into a model.

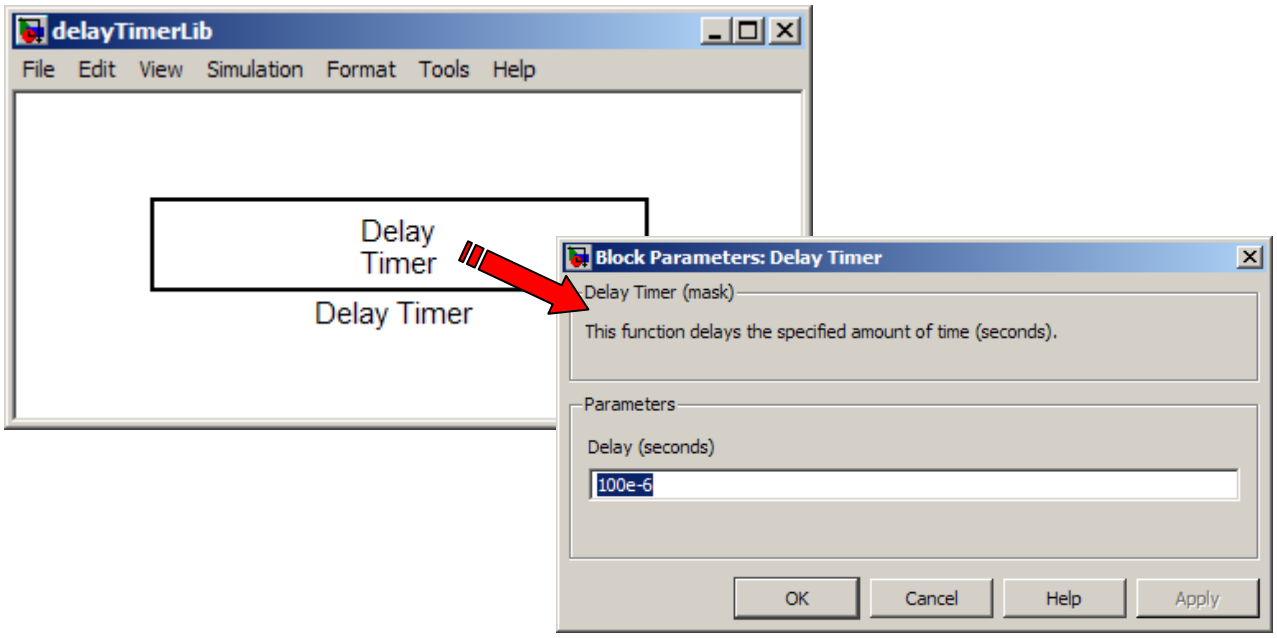

Figure 14 – Delay Timer and Parameters dialog.

Optional Step: Create `rtwmakecfg.m`

You can optionally place `delayTimerLib.mdl` (and accompanying support files) in a "library" or "utility" directory to keep it separate from application models that use the block. To do this, create an `rtwmakecfg.m` file as shown in Figure 15. The M-file contains path statements that specify the location of the block source and include files. Set `makeInfo.includePath` and `makeInfo.sourcePath` accordingly. These are needed during the application build process. `rtwmakecfg.m` is typically placed in the same directory as the files `delayTimer.h` and `delayTimer.c`. Add this directory to your MATLAB path.

```
>> path(path,'C:\work')
```

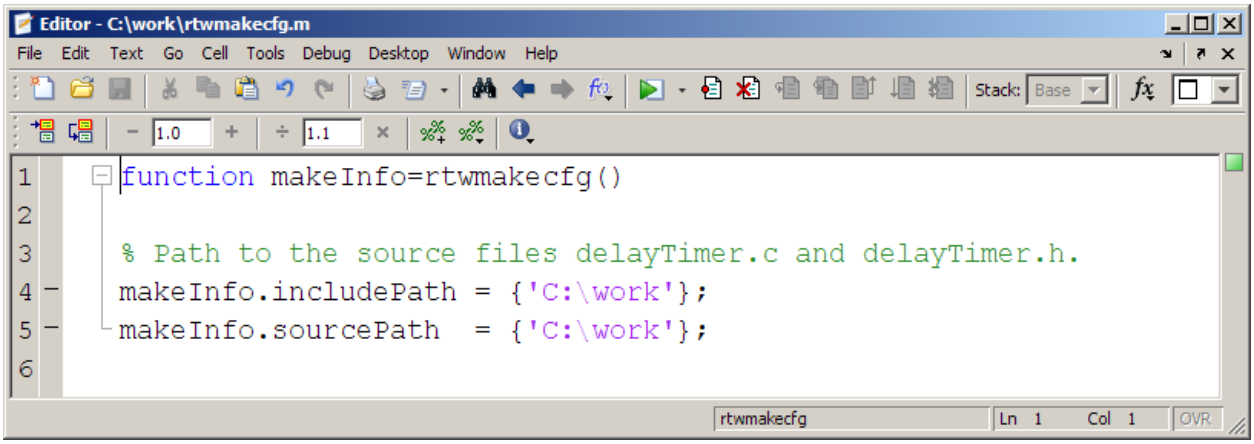

Figure 15 – `rtwmakecfg.m`.

Test Case

To demonstrate the `Delay Timer` block, the example `xpcDelayTimerTest.mdl` shown in Figure 16 is provided. The sample time for `xpcDelayTimerTest.mdl` is 250 µsec. This model includes a transfer function driven by a square wave signal. It also contains a circular counter that counts from 0 to 100, and then repeats. The example uses the counter to provide a trigger to the `Busy Time Delay` subsystem containing the time delay block. When the counter reaches a value of 100, it triggers the subsystem. This causes the simulation to delay by the time specified in the `Delay Timer` Parameter dialog box.

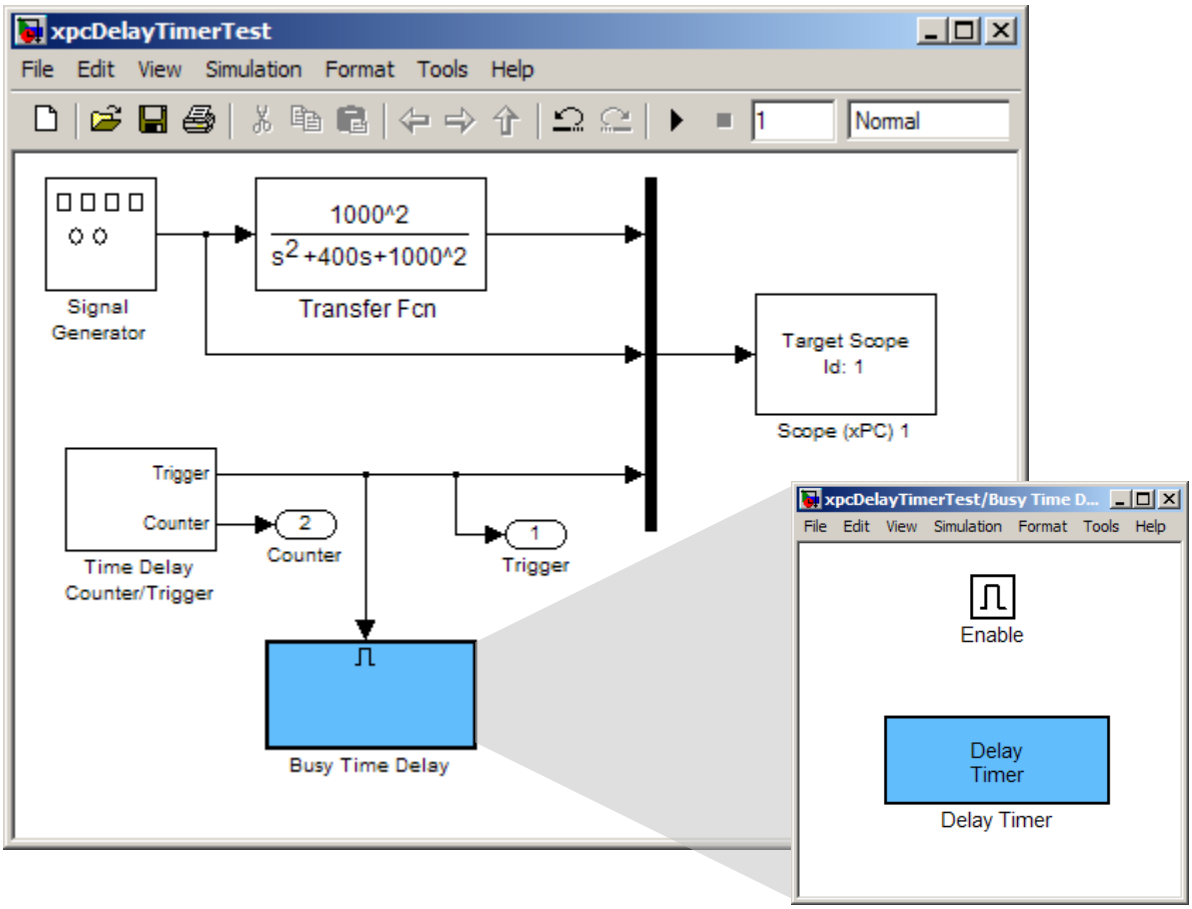

Figure 16 – Test model.

The counter, trigger, and Task Execution Time (TET) are logged as the simulation runs. TET is the amount of time it takes to execute the model code during the sample step. As a test case, the model was configured with the delay time set to 100 µsec. The example uses the MATLAB script

```
xpcDelayTimerTestResults.m
```

to run the application and generate the plot results shown in Figures 17. Note that the TET is very small (~ 6 µsec) when executing the model except when the counter equals 100. When this occurs, the `Delay Timer` block is triggered and the model takes an extra 100 µsec to complete – as reported in the TET.

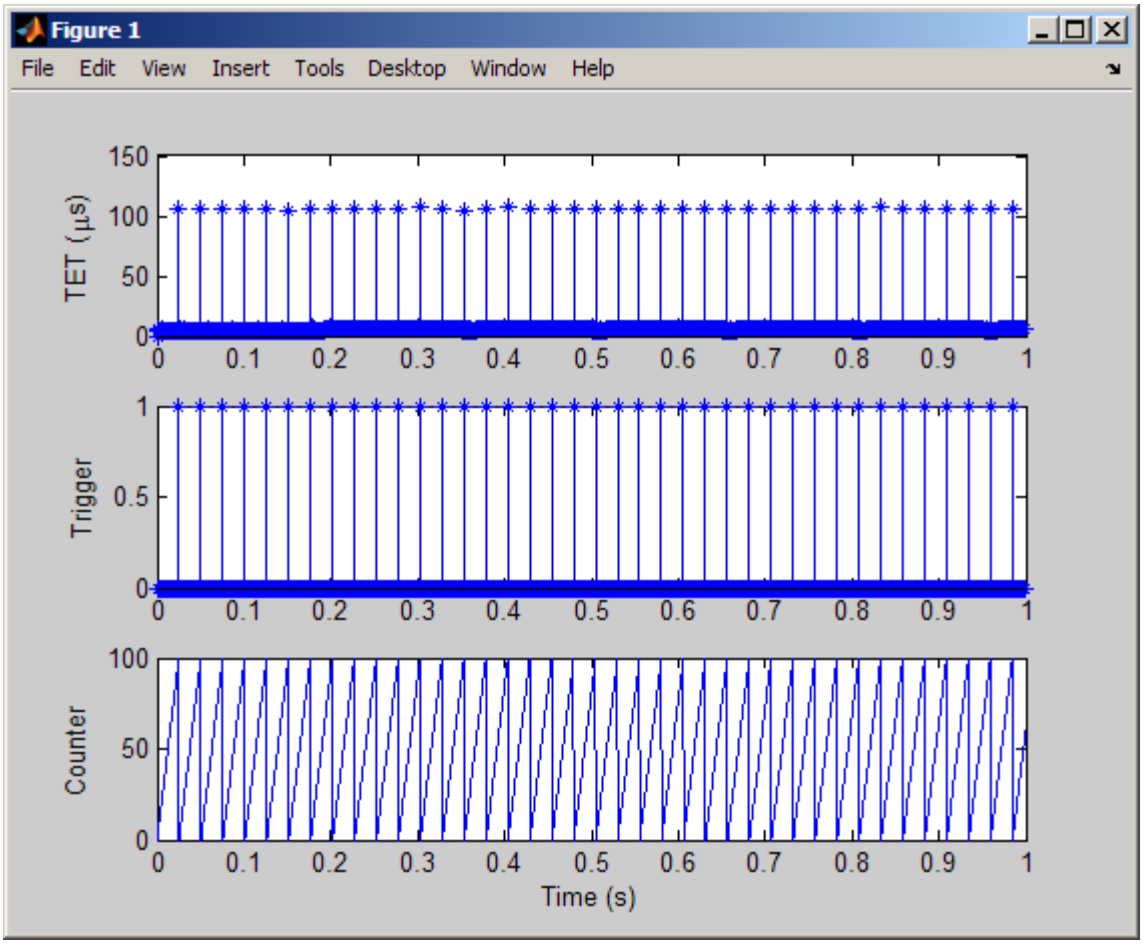

Figure 17 – Test results (100 µsec delay).

Conclusion

This document describes the xPC Target driver authoring tool and shows how it can be used to create a simple time delay block. In addition, a test model is provided that demonstrates the block's utility. All files referenced in this document are listed below and included in the compressed file `xPCTDN_DriverToolExample_R2008b.zip`.

Contents of `xPCTDN_DriverToolExample_R2008b.zip`[3]

```
delayTimer.c
delayTimer.h
delayTimer.mat
delayTimerLib.mdl
ReadMe.pdf (this document)
rtwmakecfg.m
sfcn_delayTimer.c
sfcn_delayTimer.tlc
xpcDelayTimerTest.mdl
xpcDelayTimerTestResults.m
```

[3] The file `sfcn_delayTimer.mexw32` is not included. This file is created by following the steps above. Alternately, it can be created by invoking the `mex` command as follows:

```
>> mex('sfcn_delayTimer.c', 'delayTimer.c')
```