An Introduction to Writing S-Functions

University of Michigan
EECS 461

Overview

MATLAB S-functions are an effective way to embed object code into a Simulink model. These S functions can be written in a few languages, C being the one most relevant for our purposes. This tutorial is meant to serve as a guide – almost a walk-through – that explains what all you need to know regarding S functions and how to use them in your final project Simulink models.

In lab 1, we wrote a simple adder program. We followed this up in lab 8 where we used Simulink blocks to represent the Digital Input and Output blocks and to shift and add integers. We will now combine concepts from labs 1 and 8 and utilize the best of both worlds. Understanding how and when to use S-functions in your final project is almost guaranteed to make your project significantly more understandable to both you and the GSI who helps you debug your project.

Tutorial

1. The first step is to make sure you are in the correct directory.

   Make sure you are working from the N drive. Create a folder called Adder anywhere on your H drive.

2. The second step is to select a C compiler in MATLAB.

   Follow the steps below if you are using a CAEN computer
   1. At the MATLAB prompt, enter "mex -setup"
   2. Select "Microsoft Visual C++ 2015 Professional (C)" as the compiler.

   This Microsoft compiler cannot generate object code for the NXP S32K. It should, however, allow you to complete most of your debugging outside the EECS 461 lab.

   Follow the steps below if you are using an EECS 461 lab computer
   1. At the MATLAB prompt, enter "mex -setup"
   2. Select MinGW64 compiler

3. Create a new Simulink model in Matlab 2018a. Insert 8 Constant Blocks, 2 Muxs, 1 Demux, and 5 display blocks. Insert the S-Function Builder block located under “User-Defined Functions”. Your Simulink model should look something like Figure 2.

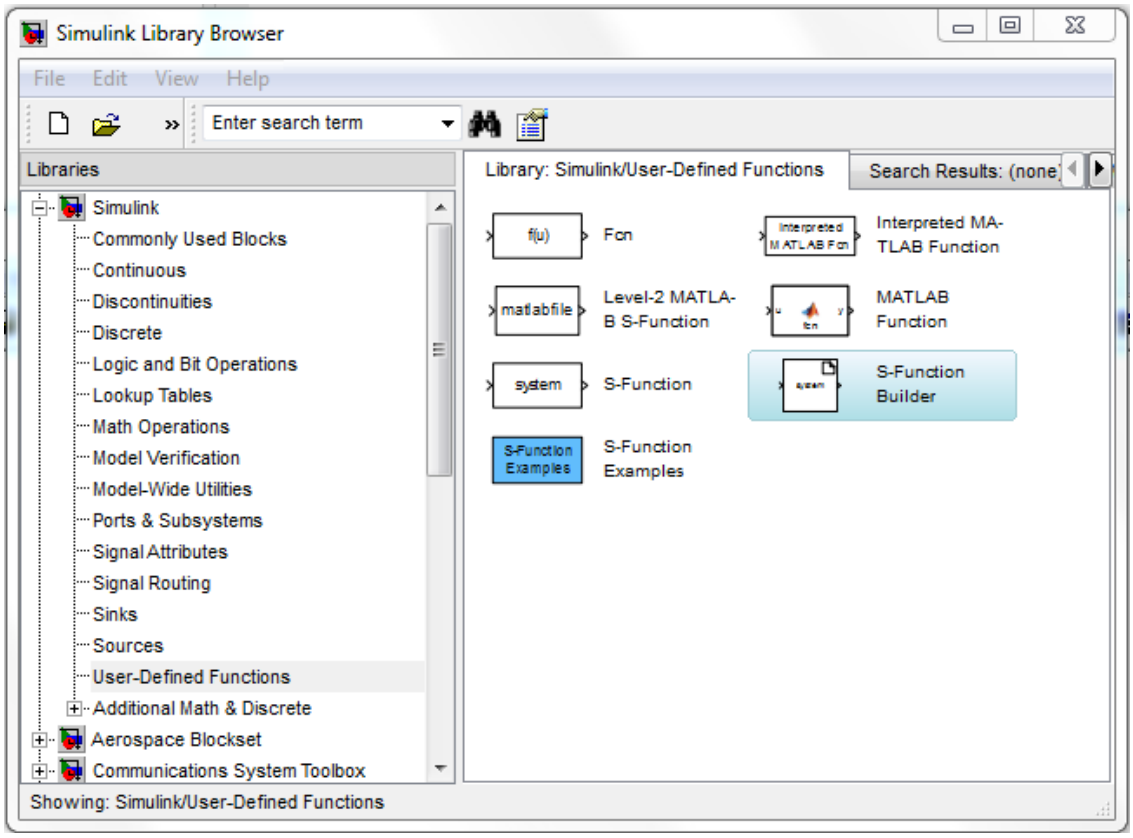

Figure 1

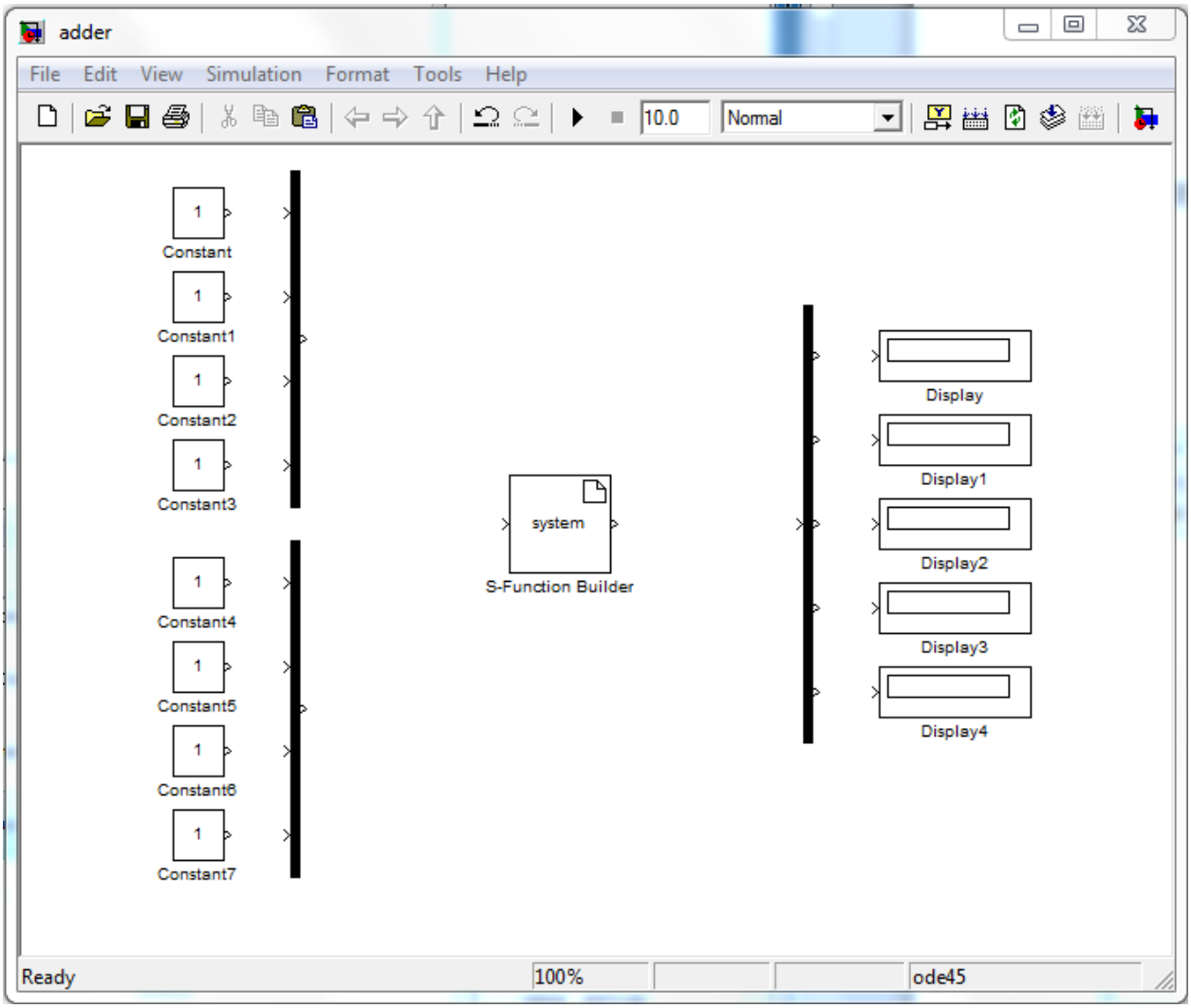
adder
File Edit View Simulation Format Tools Help
10.0
Normal
1
Constant
1
Constant1
1
Constant2
1
Constant3
1
Constant4
1
Constant5
1
Constant6
1
Constant7
system
S-Function Builder
Display
Display1
Display2
Display3
Display4
Ready
100%
ode45

Figure 2

4. Double-click on the S-Function Builder block. In the text box labeled “S-function name,” give your new S-Function a suitable name. This name has to be different from your model name.
5. Click on the “Data Properties” tab and then on the “Input ports” tab. This is where you will specify the names, types and sizes of your function’s inputs. Create two input ports, num1 and num2. These should be 1-D vectors and have 4 rows, as seen in the figure below. These settings mean that your S-Function will receive 2 vector inputs, each containing 4 elements. See figure 3 for reference.

   In the Final Project you will have many input signals to your S-functions. You may choose to group lines together and have multiple rows for each vector input.

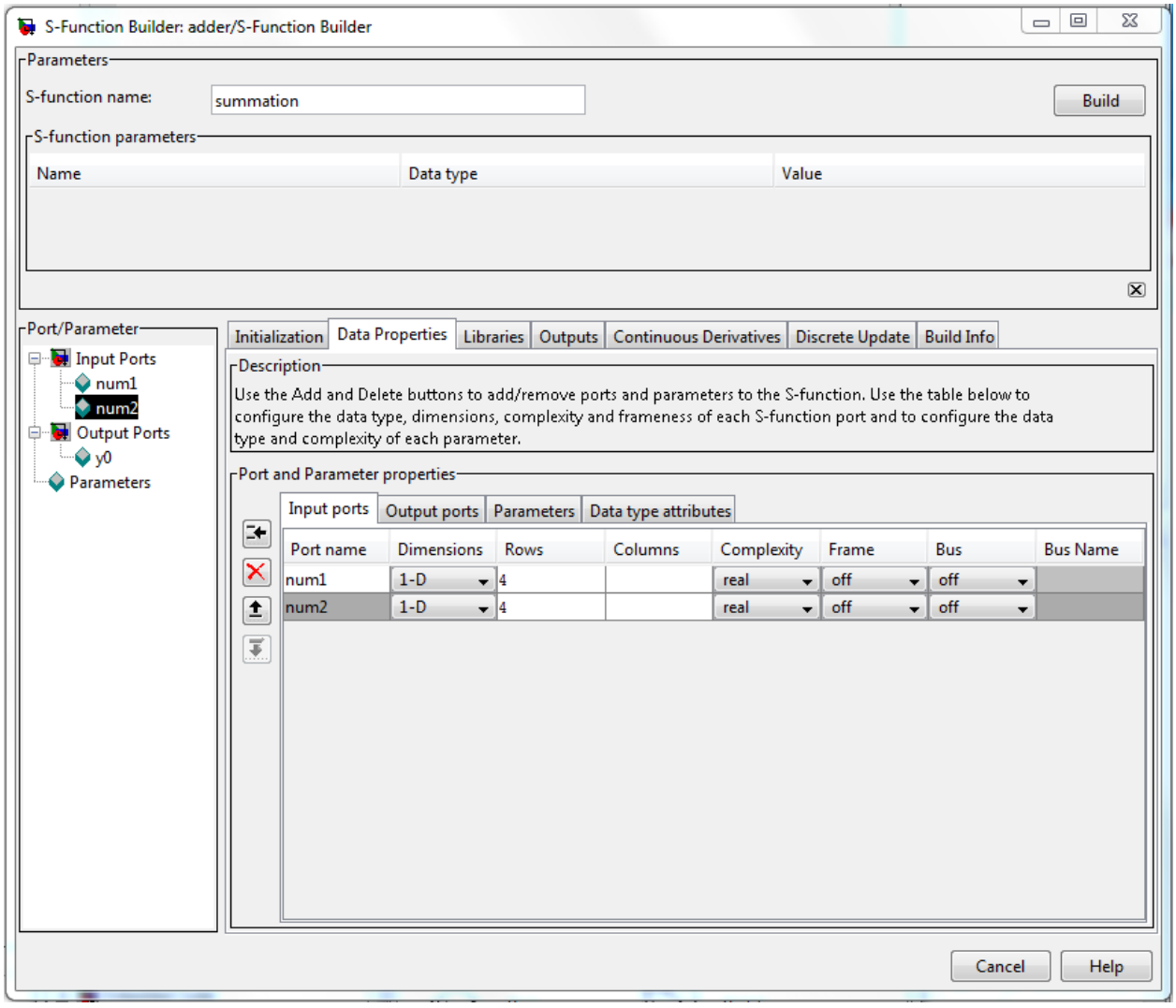

Figure 3

6. Now click on the “Output ports” tab. This is where you will specify the names, types and sizes of your function’s outputs. Create an output port, sum, a vector containing 5 elements.
7. Click on the “Data type attributes” tab and select a data type of “boolean” for inputs and outputs. See figure 4 for reference.

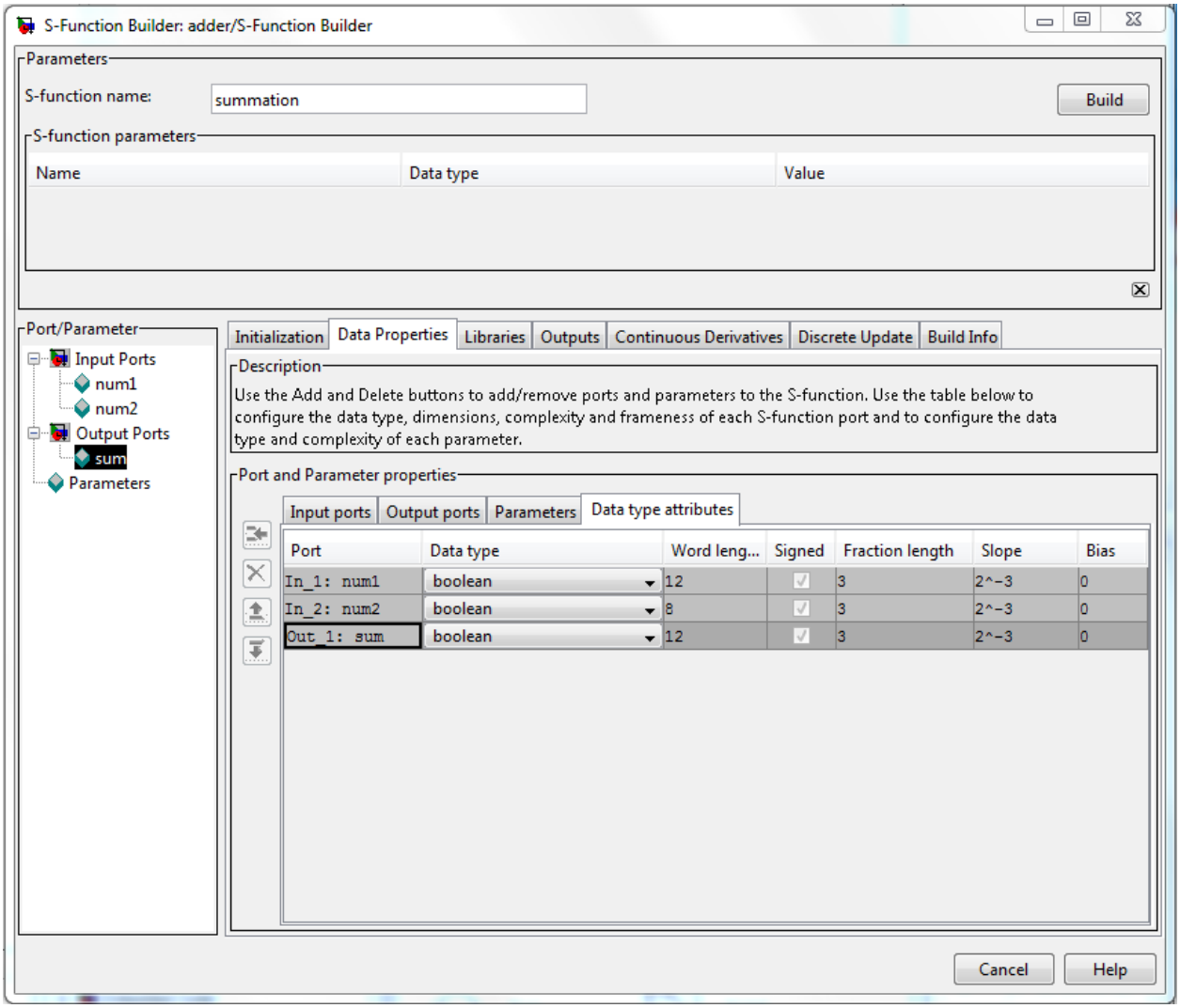

Figure 4

8. Click on the “Outputs” tab. This is where you will write the C code that connects your function’s inputs to its outputs. Each element of the input vector can be accessed by indexing into the vector the same way you would do in C. For example, if you wanted the first output element to be the logical OR of the first input element of each input, you would write:

   sum[0] = num1[0] | num2[0];

   Now implement your code for a 4 bit adder. See figure 5 for reference.

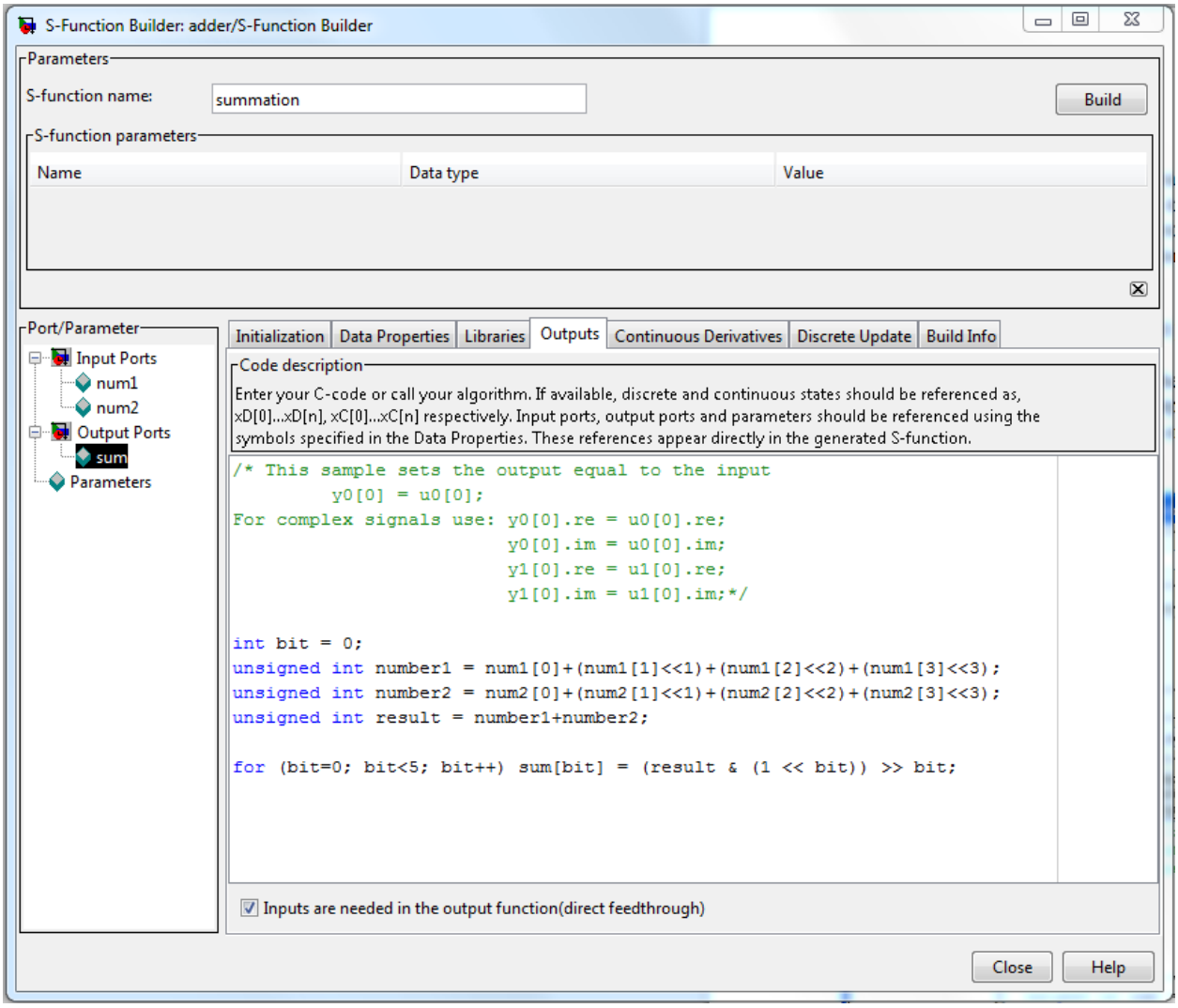

Figure 5

9. You are now ready to build your function. Click on the “Build Info” tab. Check the boxes named “Show compile steps,” “Create a debuggable MEX-file” and “Generate wrapper TLC”. Now click “Build.” If the build process was successful, your window should look similar to figure 6.

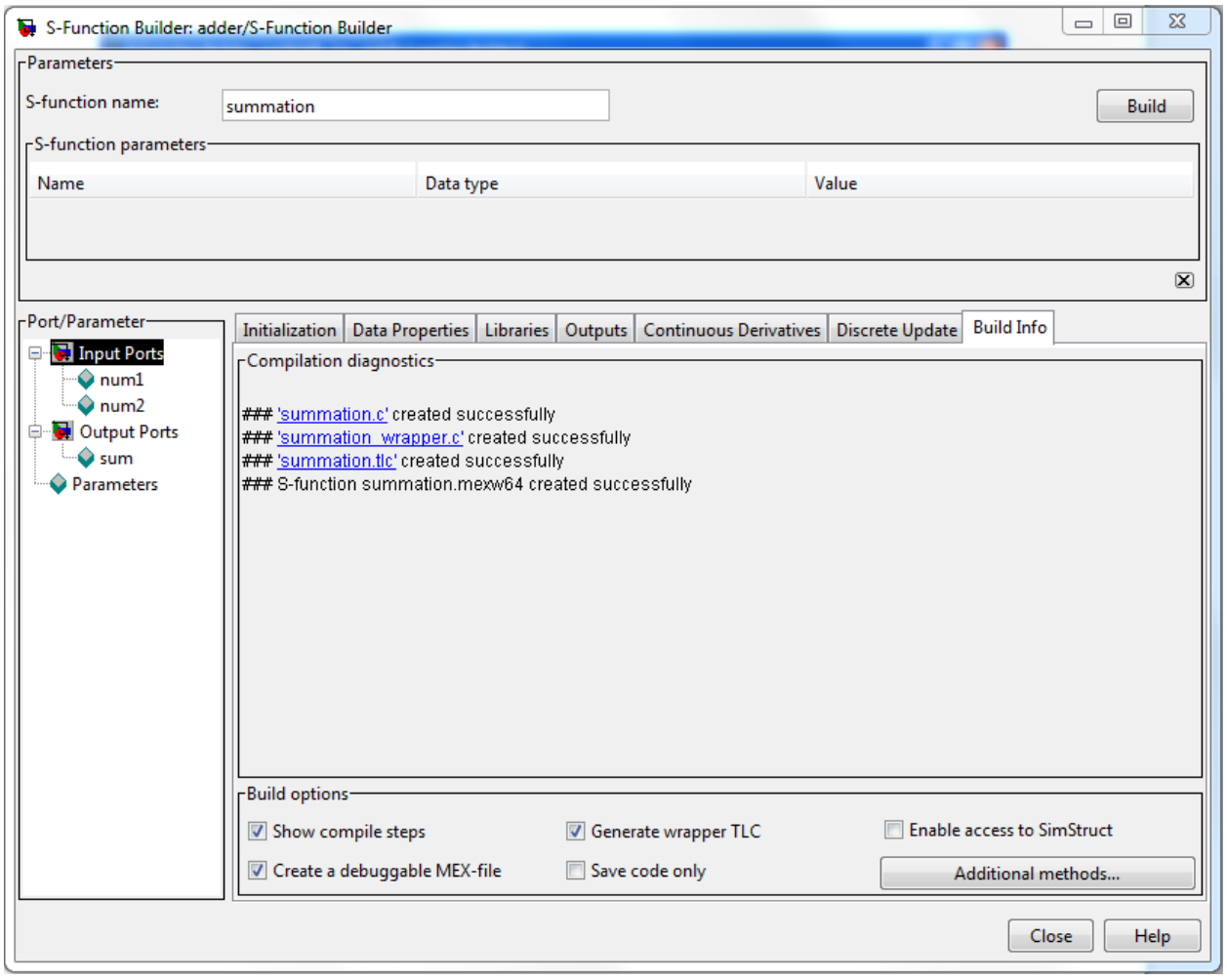

Figure 6

10. Your S-Function block should now be updated with the input and output ports. Now complete the model as shown in figure 7. You will have to change the data types of the constant blocks to "boolean". See figure 8 for reference.

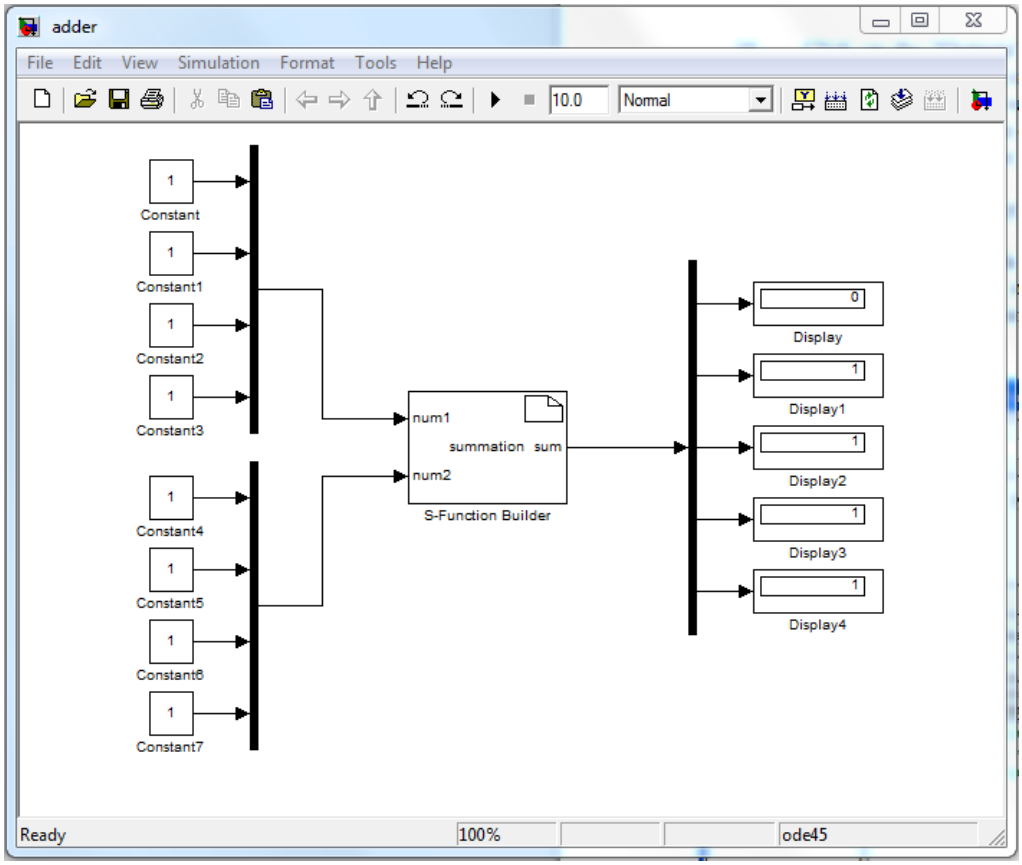

Figure 7

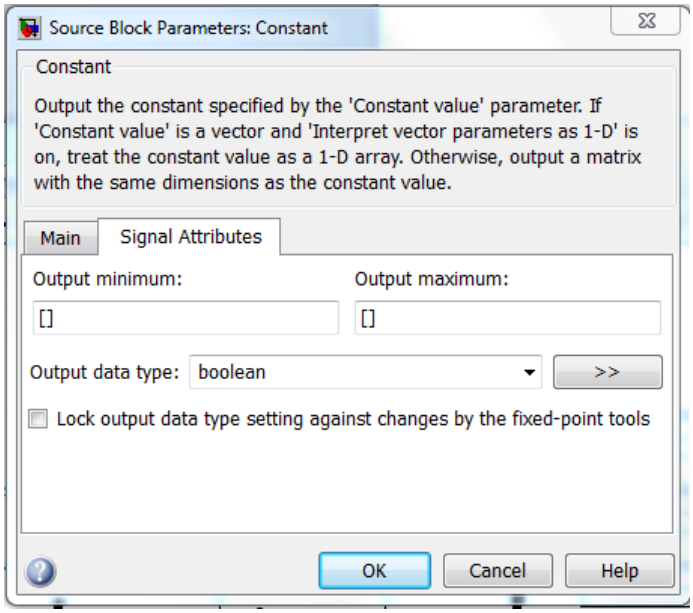

Figure 8

The preceding steps can all be done either in a CAEN lab, or in the EECS 461 lab. In order to create an S-function that may be used for automatically generating code that may be downloaded to the S32K144, you must be in the EECS 461 lab and using the GCC compiler.

OPTIONAL: Using Model Based Design toolbox

11. Incorporate the NXP Model Based Design blocks provided on the lab computers in your model. First, add the MBD_S32K14x_Config_Information Block to your model. Next, replace all of the Constant Blocks with Digital_Input Blocks. Replace all the Display Blocks with digital_Output Blocks. Select the appropriate pin for each block by double clicking on each block. See Figure 9 for reference. You can find the blocks in the Simulink Library Browser:
    1. NXP Model-Based Design Toolbox for S32K MCUs/
        - S32K14x MCUs/
            - S32K14x Core, System, Peripherals and Utilities/
                - GPIO Blocks
12. In the MBD_S32K14x_Config_Information block, modify the compile options (Build Toolchain -> MinGW64-> Compile Options) and append `-include stdbool.h`. Also add `-DRT` and `-DmwSize=size_t`, see Figure 10. This is necessary because the S function builder uses booleans when it creates a .c file (in mux blocks, for example); `mwSize` is a Mathworks defined type; `-DRT` specifies that the S-function is being built with the Simulink Coder product for a real-time application using a fixed-step solver.
13. You will need to set a fixed time step. Go to Simulation and select "Configuration Parameters...". For Fixed-step size, enter ".001". Click "OK".
14. Build your model by pressing *ctrl-b*. This will generate an executable linkable file (.elf). Use the S32 Design Studio IDE to upload the file to the S32K144 as follows:
    1. In your Matlab file browser you should see a new folder ending in "_mbd_rtw" (See Figure 11).
    2. Inside this folder you wil find a file with extension .elf. If you can't find the folder or .elf file, it's likely that you received compile errors after yo.u pressed *ctl-b.*
    3. Open S32 Design Studio from the desktop (use the workspace you have been using in-lab) and click the drop down next to the beatle and select Debug Configurations (see Figure 12).
    4. Right click and duplicate eecs461_Debug as in Figure 13.
    5. Change the name from "eecs461_Debug (1)" to "eecs461_Debug_elfUpload"
    6. Under C/C++ Application click browse and select the .elf file we found earlier.
    7. Disable auto build and click Apply. The code is now running on your board. See Figure 14.

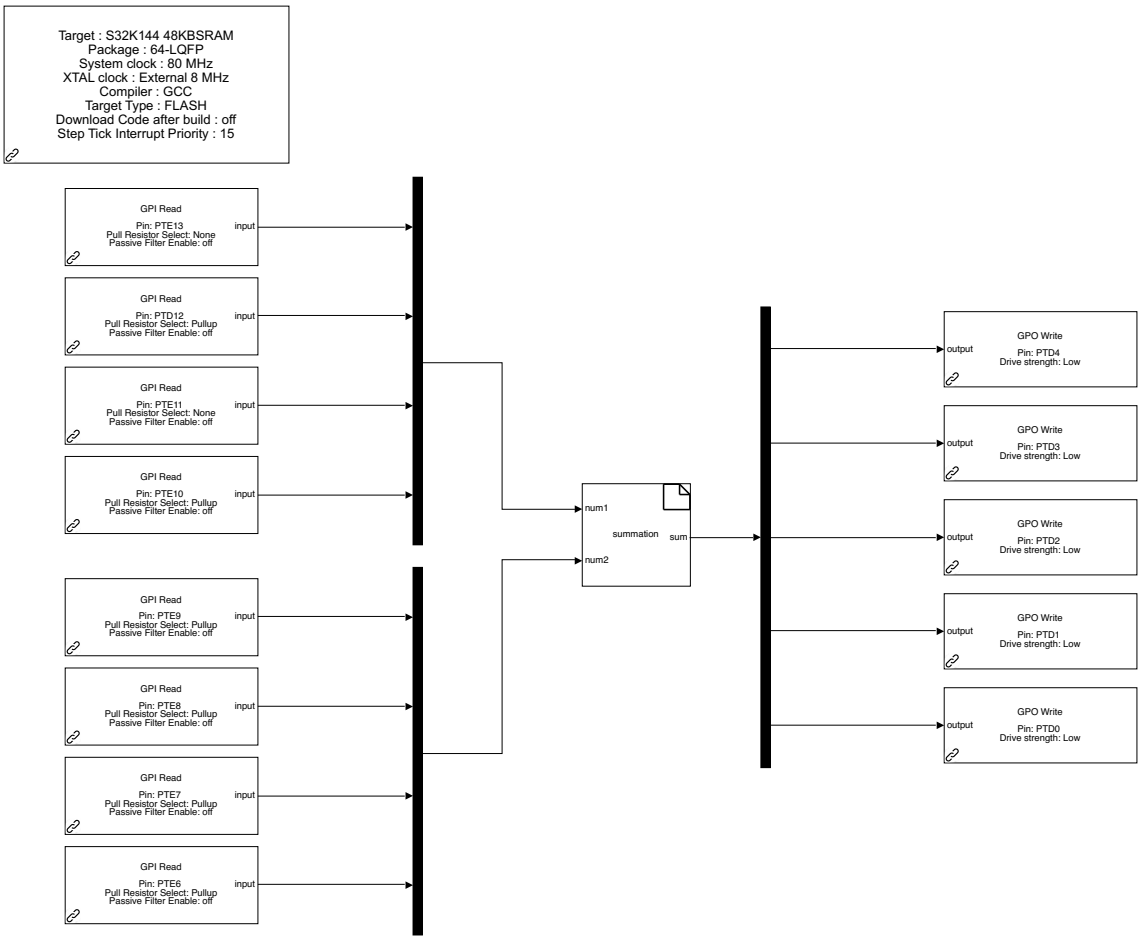
Target : S32K144 48KBSRAM
Package : 64-LQFP
System clock : 80 MHz
XTAL clock : External 8 MHz
Compiler : GCC
Target Type : FLASH
Download Code after build : off
Step Tick Interrupt Priority : 15
GPI Read
Pin: PTE13
Pull Resistor Select: None
Passive Filter Enable: off
input
GPI Read
Pin: PTD12
Pull Resistor Select: Pullup
Passive Filter Enable: off
input
GPI Read
Pin: PTE11
Pull Resistor Select: None
Passive Filter Enable: off
input
GPI Read
Pin: PTE10
Pull Resistor Select: Pullup
Passive Filter Enable: off
input
GPI Read
Pin: PTE9
Pull Resistor Select: Pullup
Passive Filter Enable: off
input
GPI Read
Pin: PTE8
Pull Resistor Select: Pullup
Passive Filter Enable: off
input
GPI Read
Pin: PTE7
Pull Resistor Select: Pullup
Passive Filter Enable: off
input
GPI Read
Pin: PTE6
Pull Resistor Select: Pullup
Passive Filter Enable: off
input
num1
summation
sum
num2
GPO Write
Pin: PTD4
Drive strength: Low
output
GPO Write
Pin: PTD3
Drive strength: Low
output
GPO Write
Pin: PTD2
Drive strength: Low
output
GPO Write
Pin: PTD1
Drive strength: Low
output
GPO Write
Pin: PTD0
Drive strength: Low
output

Figure 9

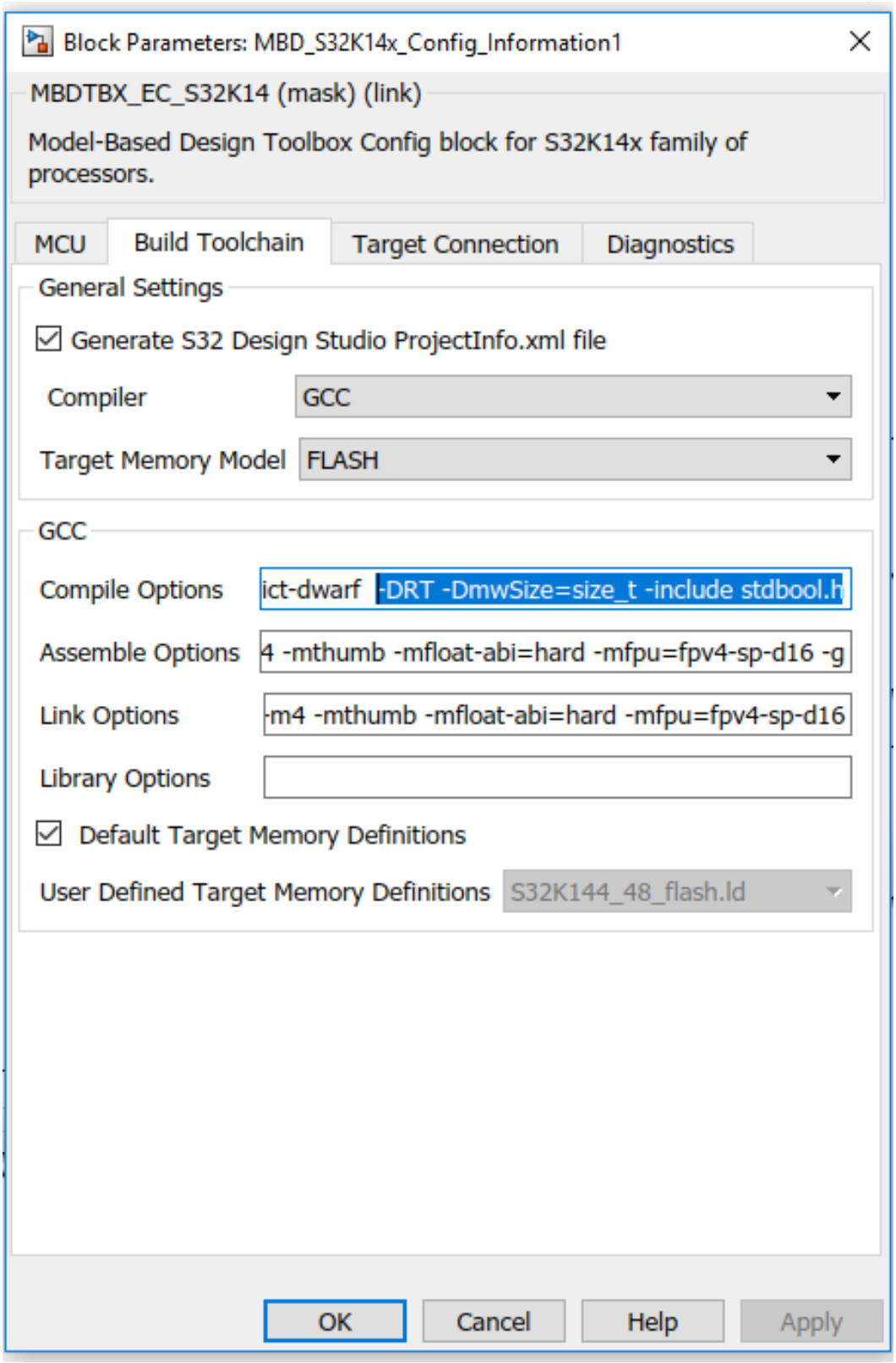
Block Parameters: MBD_S32K14x_Config_Information1
MBDTBX_EC_S32K14 (mask) (link)
Model-Based Design Toolbox Config block for S32K14x family of processors.
MCU
Build Toolchain
Target Connection
Diagnostics
General Settings
Generate S32 Design Studio ProjectInfo.xml file
Compiler
GCC
Target Memory Model
FLASH
GCC
Compile Options
ict-dwarf -DRT -DmwSize=size_t -include stdbool.h
Assemble Options
4 -mthumb -mfloat-abi=hard -mfpu=fpv4-sp-d16 -g
Link Options
-m4 -mthumb -mfloat-abi=hard -mfpu=fpv4-sp-d16
Library Options
Default Target Memory Definitions
User Defined Target Memory Definitions
S32K144_48_flash.ld
OK
Cancel
Help
Apply

Figure 10

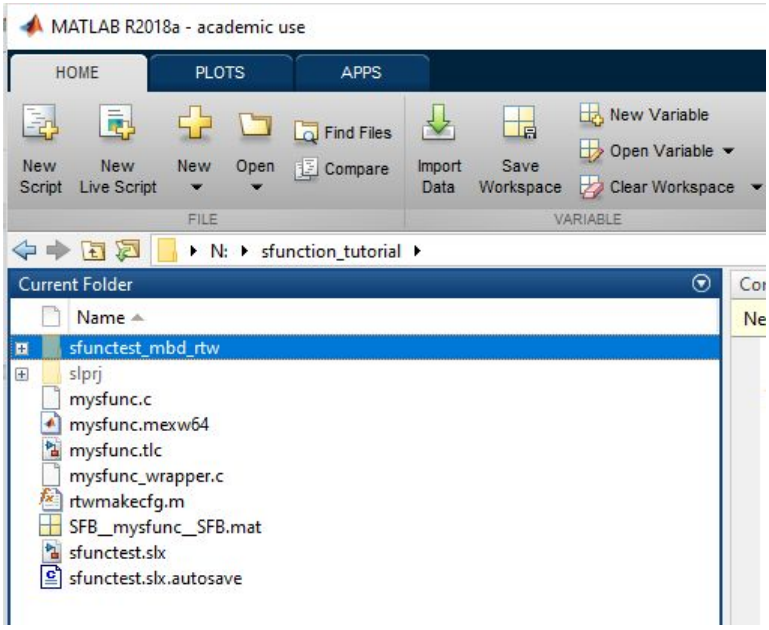

Figure 11

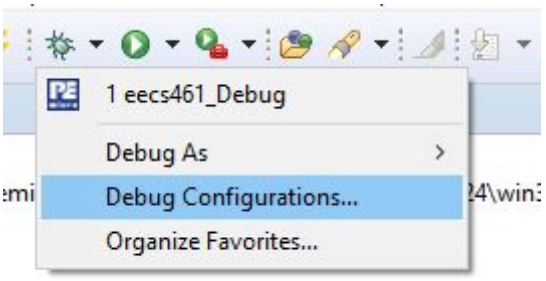

Figure 12

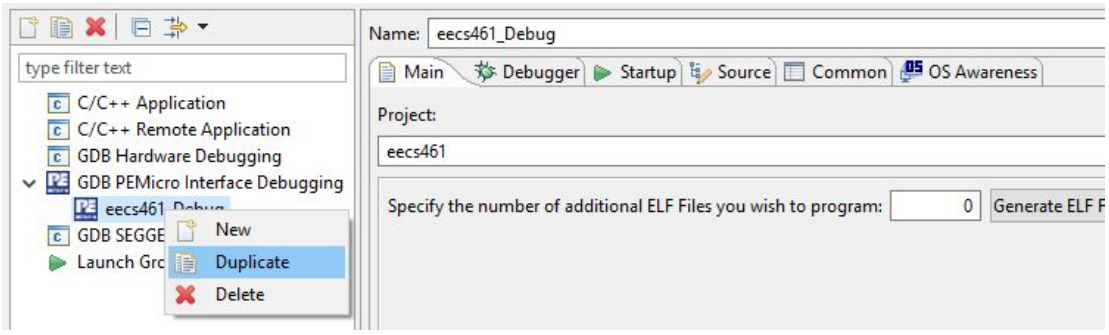

Figure 13

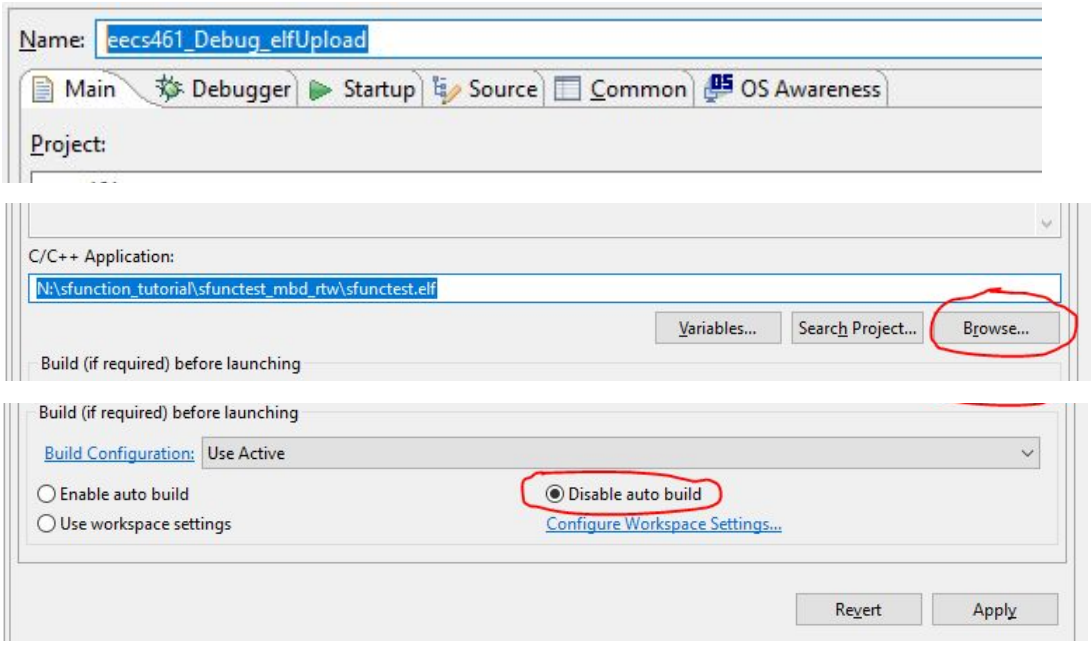
Name: eecs461_Debug_elfUpload
Main
Debugger
Startup
Source
Common
OS Awareness
Project:
C/C++ Application:
N:\sfunction_tutorial\sfunctest_mbd_rtw\sfunctest.elf
Variables...
Search Project...
Browse...
Build (if required) before launching
Build (if required) before launching
Build Configuration:
Use Active
Enable auto build
Disable auto build
Use workspace settings
Configure Workspace Settings...
Revert
Apply

Figure 14