This document has a basic C-S Function example broken down into parts and instructions for how to run build and execute this in simulation.

The documents we will look at first is *cfun_FB_checkout.c*, which is the C-Code S-Function that runs in Simulink. To work with external C-Code models, use this S-Function as a wrapper that handles the interfaces with the Simulink Engine.

The following segments are in order…

```
#define S_FUNCTION_NAME cfun_FB_checkout
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"

#include "FB.h"

#define NPARAMS 4

// Define Parameters
struct SigsFB ax1_sigsFB;
struct ParamsFB ax1_paramsFB;

double SIM_dt;
```

cfun_FB_checkout.c 1-Declarations

This first segment **MUST** include the first 3 lines where the S-Function is named and defined as level 2 (C-Code…) and the “simstruc.h” header file is included. I am also including my feedback module header file and #define(ing) NPARAMS which will be used later.

We will now utilize several built in methods to setup the S-Function.

```
/* Function: mdlInitializeSizes
=============================================
 * Abstract:
 *    The sizes information is used to determine the S-function
 *    block's characteristics (number of inputs, outputs, states, etc.).
 */
static void mdlInitializeSizes(SimStruct *S){
    ssSetNumSFcnParams(S, NPARAMS);   /* Number of S-function prameters */
    ssSetNumContStates(S, 0);   /* Number of continuous states */
    ssSetNumDiscStates(S, 0);   /* Number of discrete states   */

    /* Initialize one input port with direct feedthrough */
    // Define 2 Input Ports
    if (!ssSetNumInputPorts(S, 3)) return;
    // For Each Port, set the width
    ssSetInputPortWidth(S, 0, 1);
    ssSetInputPortWidth(S, 1, 1);
    ssSetInputPortWidth(S, 2, 1);

    // For each port, set direct feedthrough
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortDirectFeedThrough(S, 1, 1);
    ssSetInputPortDirectFeedThrough(S, 2, 1);

   /* Initialize one output port */
    if (!ssSetNumOutputPorts(S,1)) return;
    ssSetOutputPortWidth(S, 0, 1);

    /* Initialize one sample time */
    ssSetNumSampleTimes(S, 1);

    SIM_dt = ( mxGetPr(ssGetSFcnParam(S,0)) )[0];

    ax1_paramsFB.Kp  = ( mxGetPr(ssGetSFcnParam(S,1)) )[0];
    ax1_paramsFB.Ki  = ( mxGetPr(ssGetSFcnParam(S,2)) )[0];
    ax1_paramsFB.Kd  = ( mxGetPr(ssGetSFcnParam(S,3)) )[0];

    ax1_sigsFB.cmd = 0.0;
    ax1_sigsFB.eff = 0.0;
    ax1_sigsFB.error[0] = 0.0;
    ax1_sigsFB.error[1] = 0.0;
    ax1_sigsFB.fb = 0.0;
    ax1_sigsFB.intError[0] = 0.0;
    ax1_sigsFB.intError[1] = 0.0;
    ax1_sigsFB.difError = 0.0;
    ax1_sigsFB.ff = 0.0;

}
```

cfun_FB_checkout.c 2. mdlInitializeSizes

This section defines and initialized a lot of stuff (in order):

1. Def. Number of parameters

2. Def. Number of Continuous States
3. Def. Number of Discrete States
4. Init. Input Port Quantity, Width, and Feedthrough
5. Init. Output Port Quantity, Width
6. Init. Quantity of Sample Times

Once all that is complete, in the input parameters are registered and any other variable initializations can occur. In this example the parameters Kp, Ki, and Kd are all pulled in from the Simulink dialog box and the structure which contains them is initialized. The signal structure is also initialized at this time. This initialization happens each time the S-Function is simulated so the initiations will overwrite there previous states.

```
/* Function: mdlInitializeSampleTimes
=======================================
 * Abstract:
 *    This function is used to specify the sample time(s) for your
 *    S-function. You must register the same number of sample times as
 *    specified in ssSetNumSampleTimes.
 */
static void mdlInitializeSampleTimes(SimStruct *S){
    ssSetSampleTime(S, 0, SIM_dt);
    ssSetOffsetTime(S, 0, 0);
}
```

cfun_FB_checkout.c 3. mdlInitializeSampleTimes

Here the sample time is initialized to the variable *SIM_dt* (which was pulled from a dialog box parameter). An offset time can also be initialized.

```
/* Function: mdlOutputs
=======================================================
 * Abstract:
 *    Calls the doubleIt.c function to multiple the input by 2.
 */
static void mdlOutputs(SimStruct *S, int tid){
    InputRealPtrsType cmd = ssGetInputPortRealSignalPtrs(S,0);
    InputRealPtrsType ff = ssGetInputPortRealSignalPtrs(S, 1);
    InputRealPtrsType fb = ssGetInputPortRealSignalPtrs(S,2);

    real_T            *curr    = ssGetOutputPortRealSignal(S,0);

    // Update FF, FB and CMD
    ax1_sigsFB.ff = *ff[0];
    ax1_sigsFB.fb = *fb[0];
    ax1_sigsFB.cmd = *cmd[0];

    // Run FB Algorithm
    FB(&ax1_paramsFB, &ax1_sigsFB, SIM_dt);

    // Update the output
    *curr = ax1_sigsFB.eff;
}
```

cfun_FB_checkout.c 4. mdlOutputs

This is the equivalent of a main() function, which gets called once every time step. In this example, the three input ports are accessed and their current time step values are assigned. Then the FB signals are updated and the module algorithm is called. Finally, the output port value is assigned the appropriate value.

```
/* Function: mdlTerminate
=======================================================
 * Abstract:
 *    In this function, you should perform any actions that are necessary
 *    at the termination of a simulation.  For example, if memory was
 *    allocated in mdlStart, this is the place to free it.
 */
static void mdlTerminate(SimStruct *S){
    UNUSED_ARG(S); /* unused input argument */

}
```

cfun_FB_checkout.c 5. mdlTerminate

This method is only called once and cleans up the S-Function at the end of simulation.

```
#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

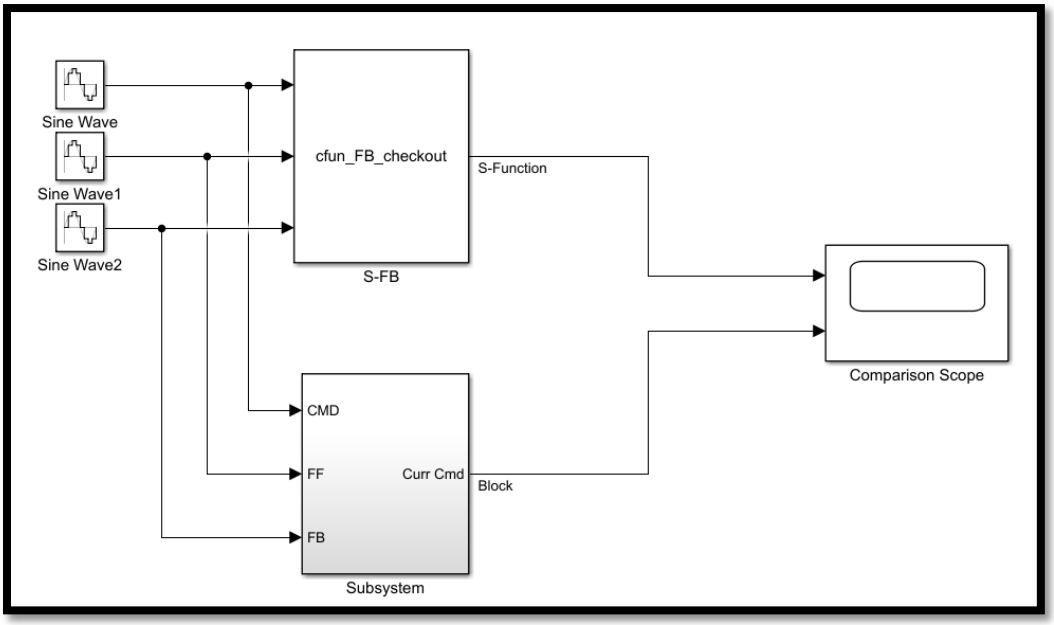
This has to go at the end of the file…

Once this file and all referenced files are written they can be MEX'd for use by the Simulink Engine. The call in the Matlab terminal (in the correct directory of course) should look like this. A –g can be added prior to the first file name to create a debuggable MEX file, which will be discussed later. Note that the first file listed should be the S-Function.

```
>> mex cfun_FB_checkout.c FB.c CNTR_UTIL.c
Building with 'Microsoft Visual C++ 2017 (C)'.
MEX completed successfully.
```

Command Terminal 1. Call to MEX the required C-Code

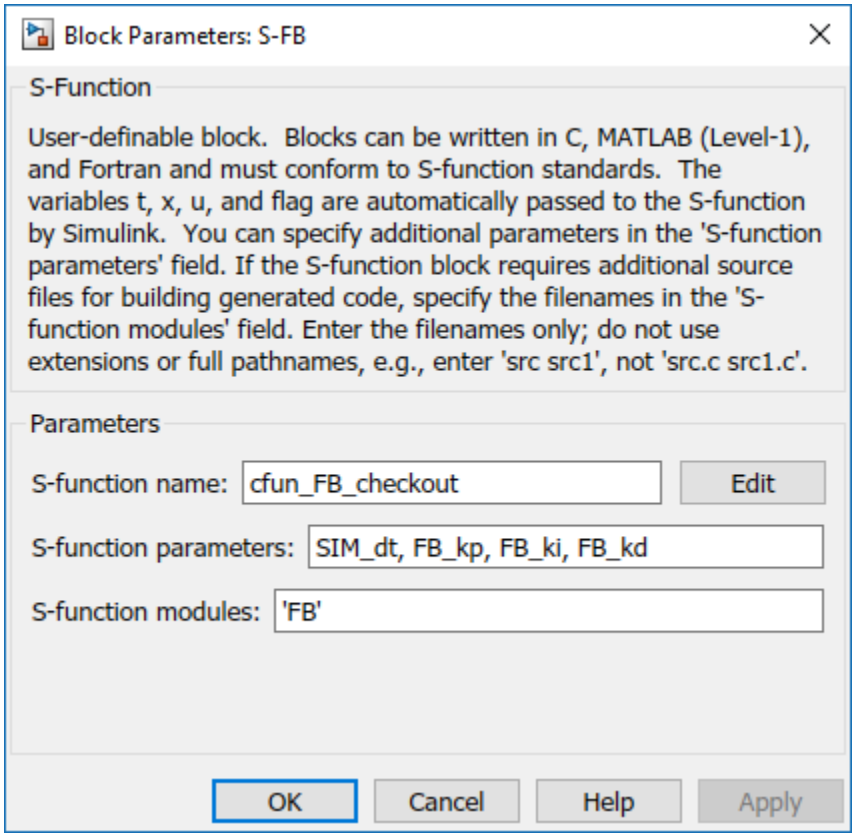
Now a Simulink model can be built to implement the S-Function that has just been created.



FB_checkout.mdl 1. Simulink Schematic showing S-Function and Block Based Subsystem

This "checkout" Simulink model is used to verify that the C-Code version of the FB Controller is identical to the Simulink block version. Both algorithms receive the same inputs (which can be somewhat arbitrary as long as they are identical) and the outputs are compared in the scope.

Double clicking the S-FB S-Function to access the parameter dialog box results in this…



FB_checkout.mdl 2. S-Function Dialog Box

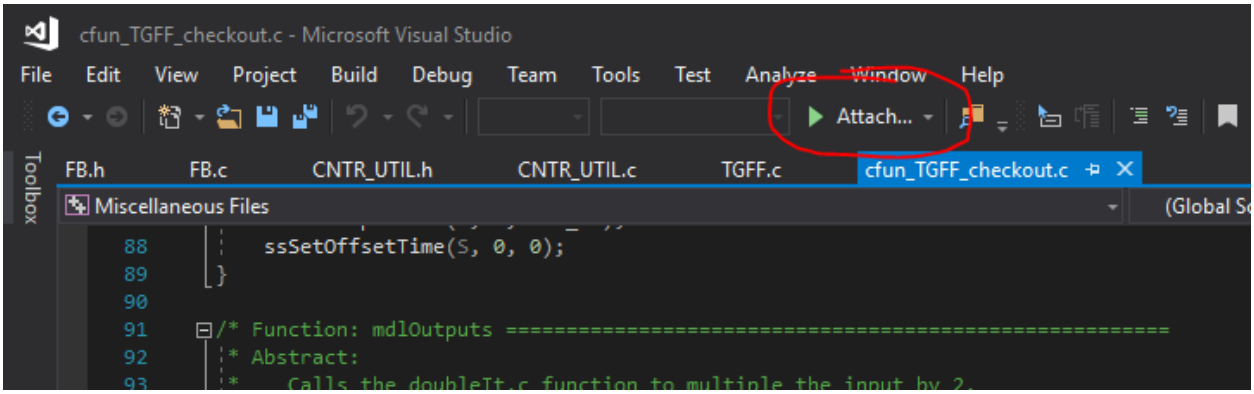
cfun_FB_checkout is the name of the C-File that is discussed above. There are 4 parameters which are imported from the Matlab workspace (this list must be the exact length defined in cfun_FB_checkout. 'FB' indicates that there is additional C-Code required to compile and execute this S-Function. FB.h and the associated FB.c are included in the body of cfun_FB_checkout, thus it is listed here.

Simply run the Simulink model at this point.

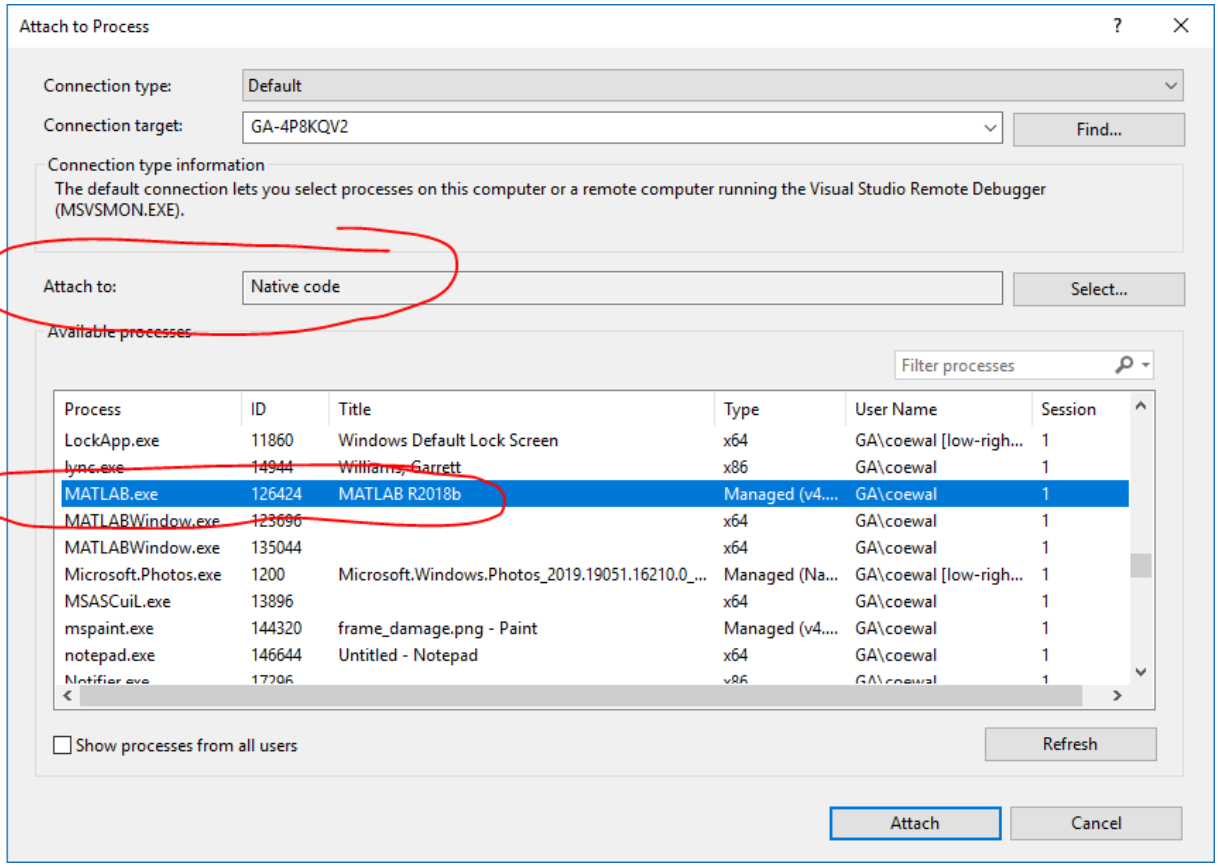
Debugging:

Debugging a C-Code S-Function can be achieved using Visual Studio. Matlab has

1. Open the Simulink Model
2. Ensure the C-Code was MEX'd with the -g option
3. Run the Simulink model to verify if functions
4. Without closing the MATLAB environment, start Visual Studios
5. In VS, click Attach



6. Select “Native Code” in the attach to drop down and “MATLAB.exe” in the process list.



7. Click “Attach” and close the dialog box if necessary
8. In the MATLAB command terminal, type “clear mex”
9. Open the desired C-Code file in Visual Studio and insert a breakpoint
10. Start the simulation in Simulink and wait for the program to stop at the breakpoint