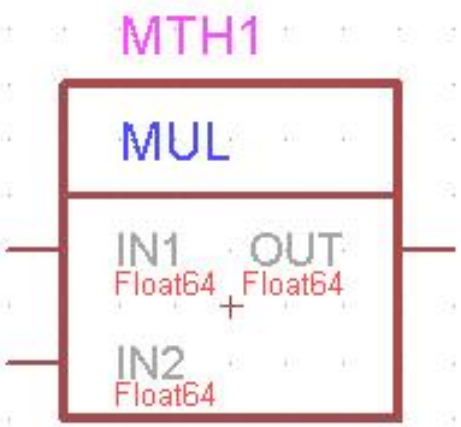
ADVANCED MOTION CONTROLS

Click & Move Function Blocks

Function Blocks

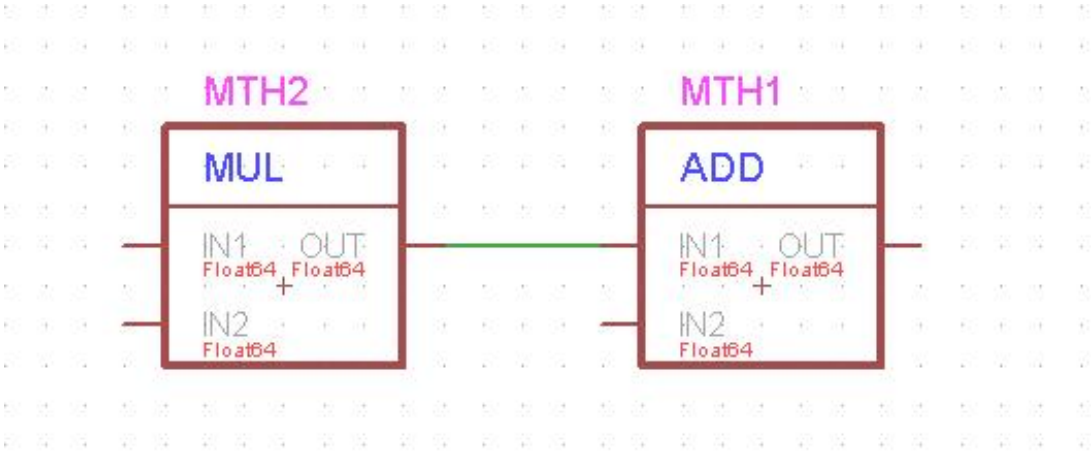
To the user, the FBs are crucial. These are the software equivalent of electronic chips. They contain inputs and outputs, with associated names and Data types. Each FB contains code (like a small program) to give it its functionality. The user only sees the interface, being the inputs and outputs. The code itself is hidden - this data encapsulation and hiding is crucial to separate the different levels of programming and maintenance.



Function Block Parts

The name of the function block is located inside the upper section of the block. The instance of the block is shown above the block. The instance name is formed from the name of the library containing the block and the numerical order the block was placed on the schematic. The blocks below were taken from the MATH library.

Input pins are displayed along the left side of the block and output pins are displayed along the right side. The name of the pin is shown inside the block next to the pin and the data type of the pin is shown below the name of the pin.

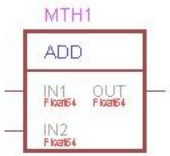


Function Block Library

The library contains a description of the function block, how it works and how it may be used.

The description indicates compatibility of the block with the various components of C&M and the data types supported by the function block pins.

Addition



Project library name: Math
Debug: NO
Functional init: YES
Target list: PCW PLA PCL MAL PCI
Compiler for PCW target: MGW, BCB, BC5, MVC
Compiler for PLA target: GMU
Compiler for PCL target: GXL
Compiler for MAL target: GAL
Compiler for PCI target: MXI

Note: PCW - PC with Microsoft Windows
PLA - AMC programmable servo drive platform A
PCL - PC with Linux
MAL - MACC (Motion Automation Controller Card) with Linux
MGW - MinGW32 gcc compiler (Version 3.4.2)
BCB - Borland 5.5 command line compiler
BC5 - Borland 5.5 command line compiler
GMU - MicroBlaze compiler
GXL - GCC Cross Compiler On Windows for x86 Linux
GAL - ARM CortexA8 compiler
GAU - ARM CortexM3 compiler

Pins:

Name	Direction	Init value	Data type	Const	Size	Debug mode	Description
In1	Input	-	TypeIn1	YES	-	NORMAL	
In2	Input	-	TypeIn2	YES	-	NORMAL	
Out	Output	-	TypeOut	YES	-	NORMAL	Out := In1 + In2

Note: 1. For more information see *C&M-MC help/Overview/Function block definition.*
2. For the range of data types see *C&M-MC help/Overview/Data types.*
3. "-" in the "Init value" column means default value, see *C&M-MC help/Overview/Data types.*

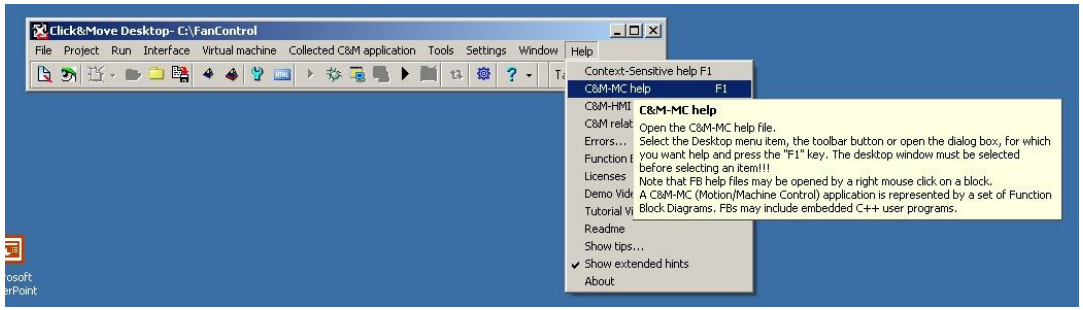
In general, the addition operation is commutative, but there is an exception, adding two strings is not commutative.
An operation is commutative if changing the order of the operands does not change the result.

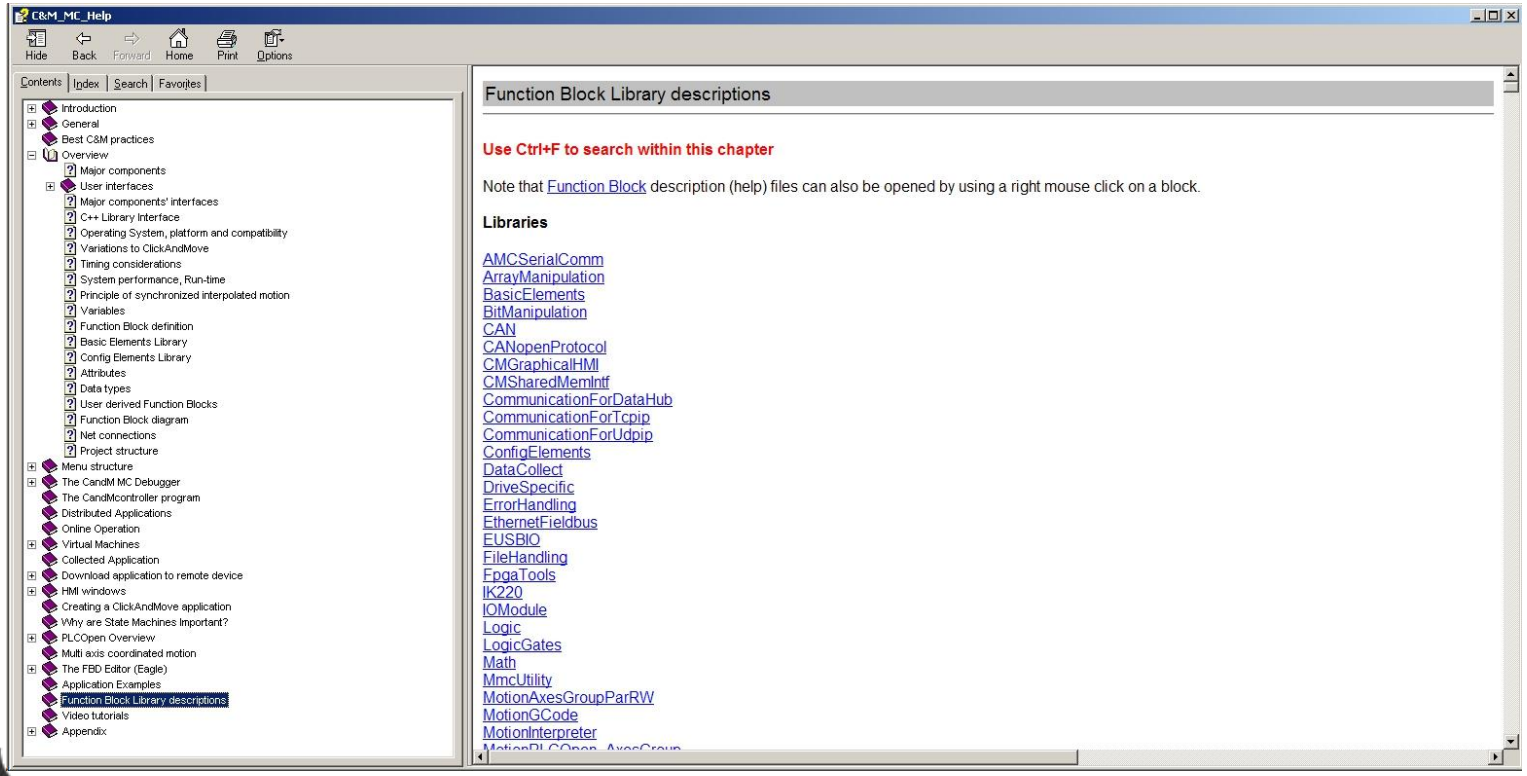
Allowable types for TypeIn1, TypeIn2 and TypeOut: Int, Int8, Int16, Int32, Int64, UInt8, UInt16, UInt32, UInt64, Float32, Float64, CMString,PString256,PString32,Array
TypeIn1, TypeIn2 and TypeOut can be different.

See also *C&M-MC help/Overview/Data types* in Desktop or start menu.

Library Help

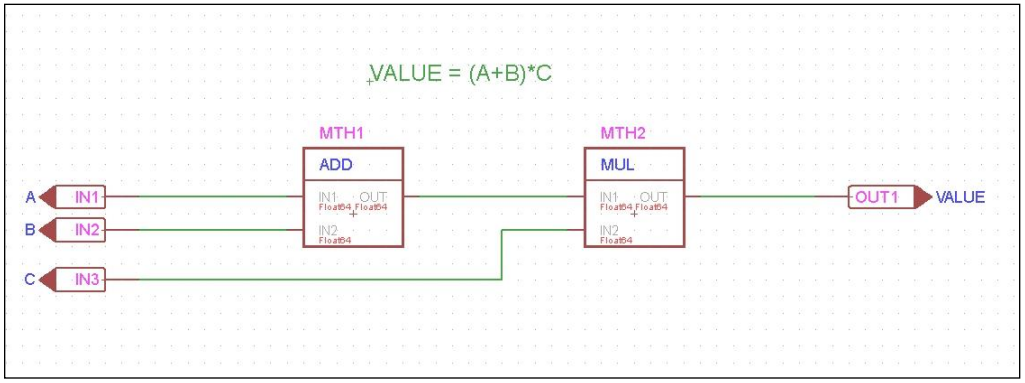
Use the C&M-MC Help selection on the C&M Desktop to list the C&M function blocks in the library.





UDFB Overview

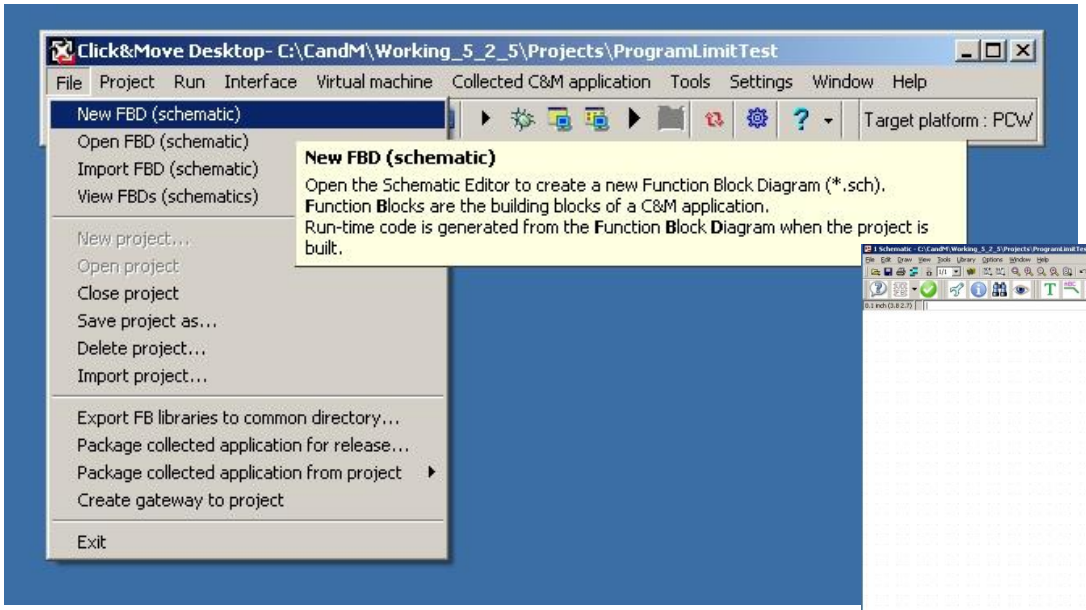
You can create a User Derived Function Block from a block of logic. The UDFB is added to the library and can be used in the same way as the C&M supplied function blocks. UDFB is convenient when the same logic is required in multiple locations and as an organizational tool to improve the readability of the project schematics.



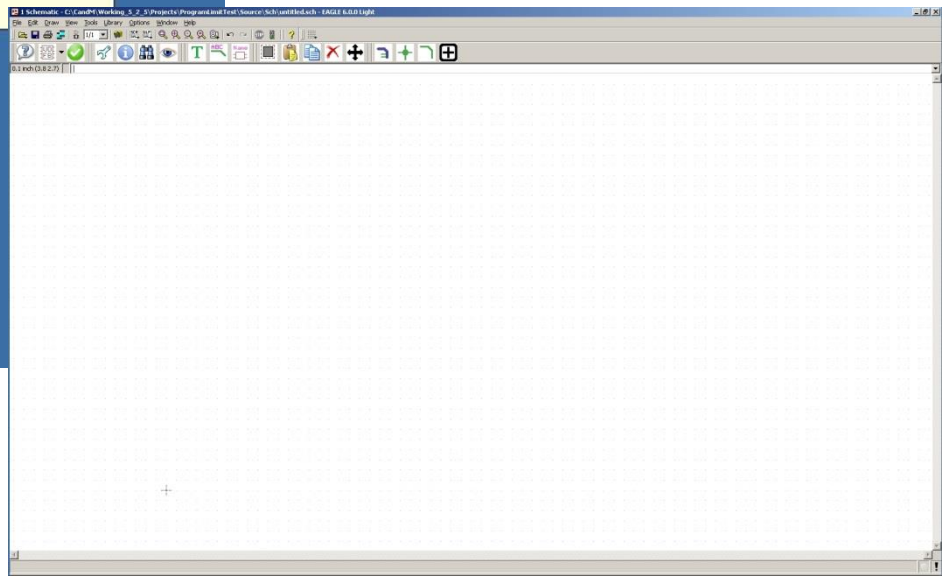
Create UDFB

To create the UDFB start with a new empty schematic.

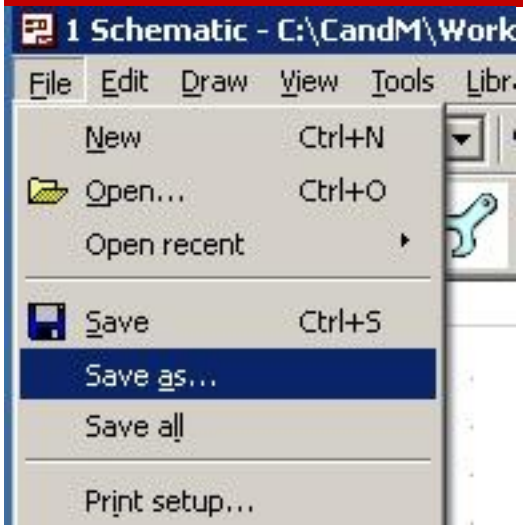
On the C&M Desktop click ‘File’, ‘New FBD (schematic)



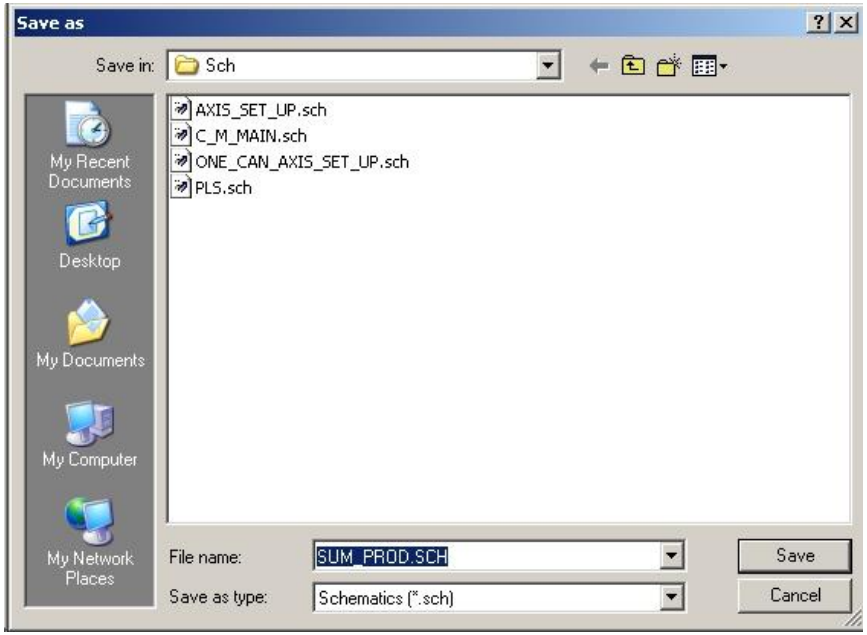
C&M opens a blank schematic in Eagle named ‘Untitled’.



Name UDFB



Choose ‘File’, ‘Save As’ in the top left corner of the Eagle window and give the new schematic a name. Standard convention is to use all upper case characters and under scores as spaces. The name you use will identify the UDFB in the library.

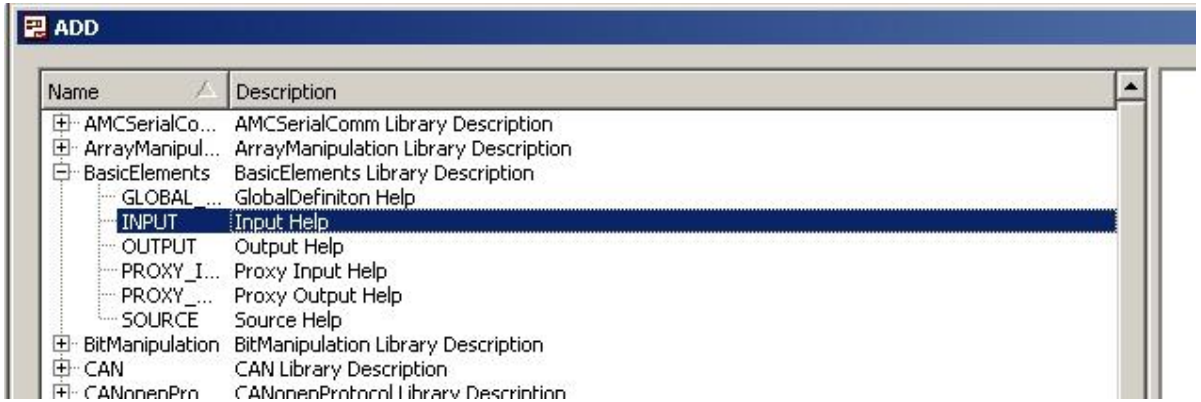


This schematic is named ‘SUM_PROD’.

Add Elements to the UDFB

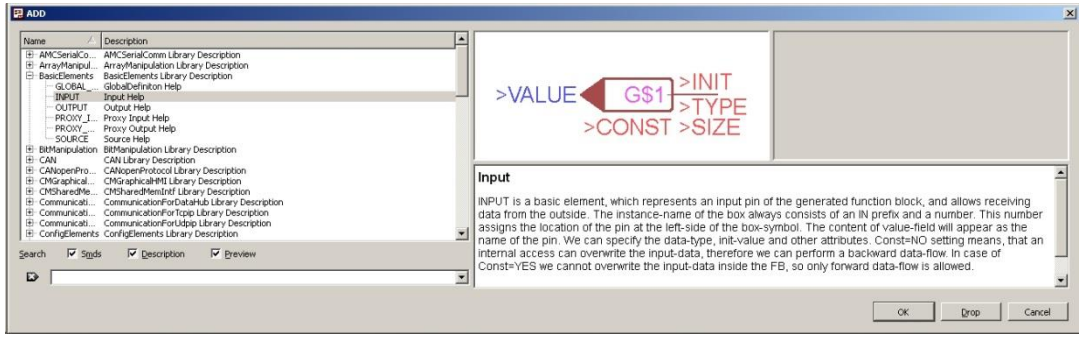
Add inputs, outputs, and logic elements to the UDFB from the library. You can even add other UDFB's to this schematic! For our example we need 3 inputs, one output, an addition block and a multiply block. Click the 'Edit' , 'Add...' or use the add button  to open the library.

You can find the inputs in the 'Basic Elements' section of the library.

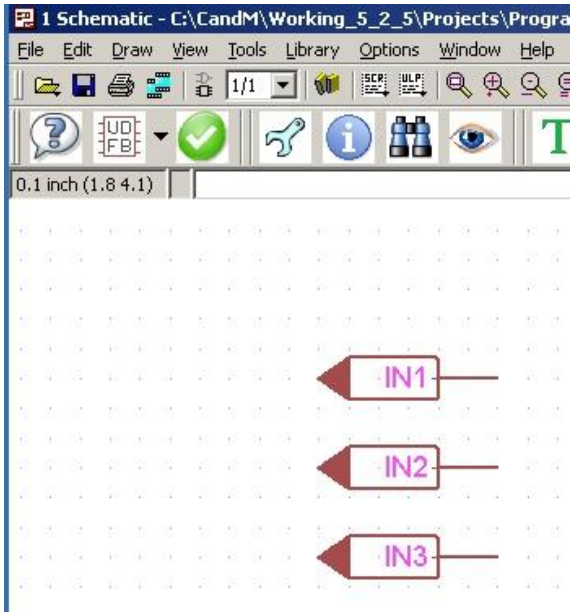


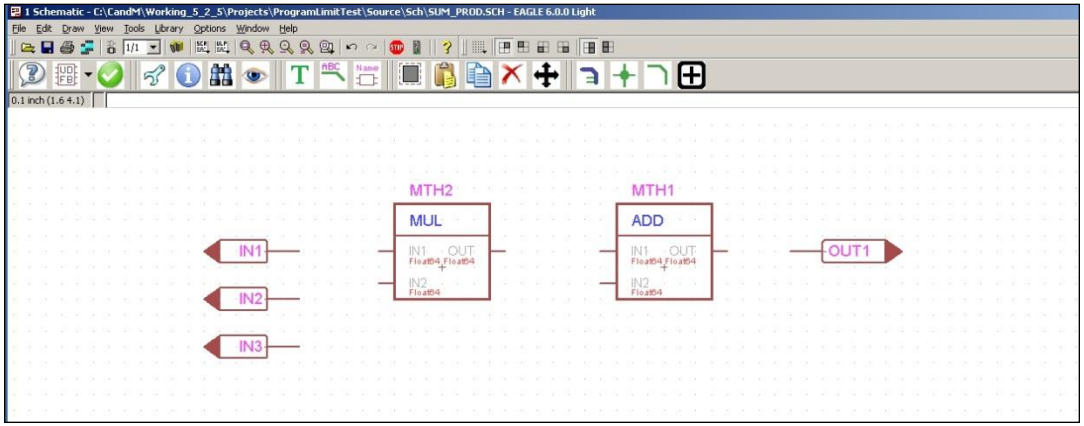
Choosing an Element

Double clicking on the ‘Input’ selection under ‘Basic Elements’ will grab the input item. Alternately, you can single click ‘Input’ and then click the ‘OK’ button at the bottom of the window. Caution: Do not click the ‘Drop’ button, it will temporarily remove the selected item from the menu.



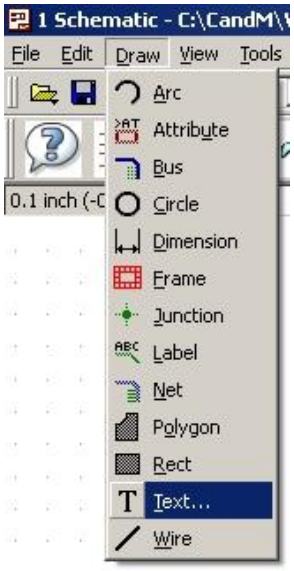
Once the element is grabbed, Eagle returns to the schematic. An element is dropped with each click of the mouse. Use the roller wheel on the mouse to zoom in and out of the schematic and press ‘Escape’ to end the insertion action.



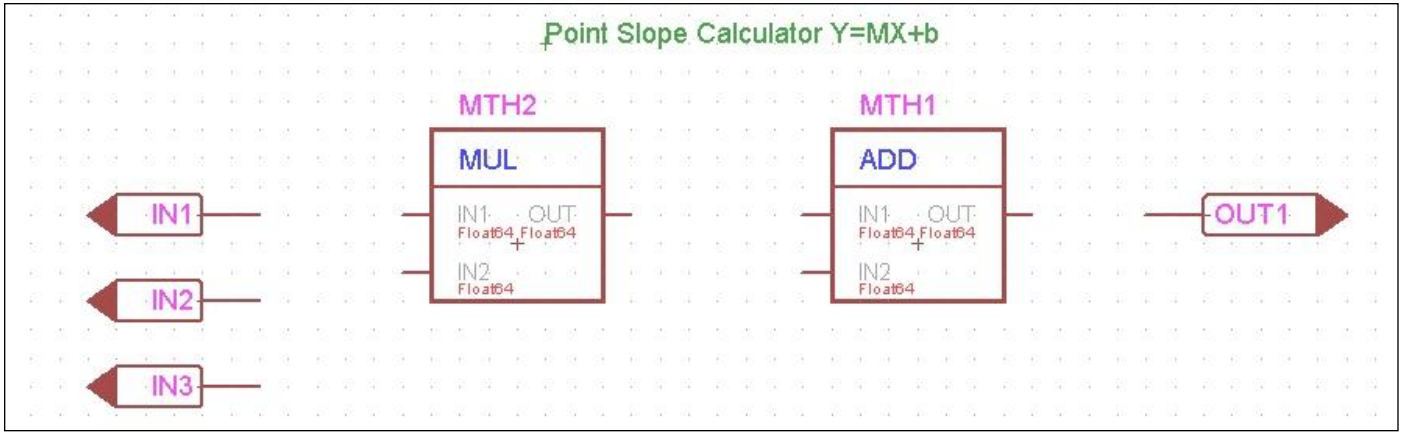


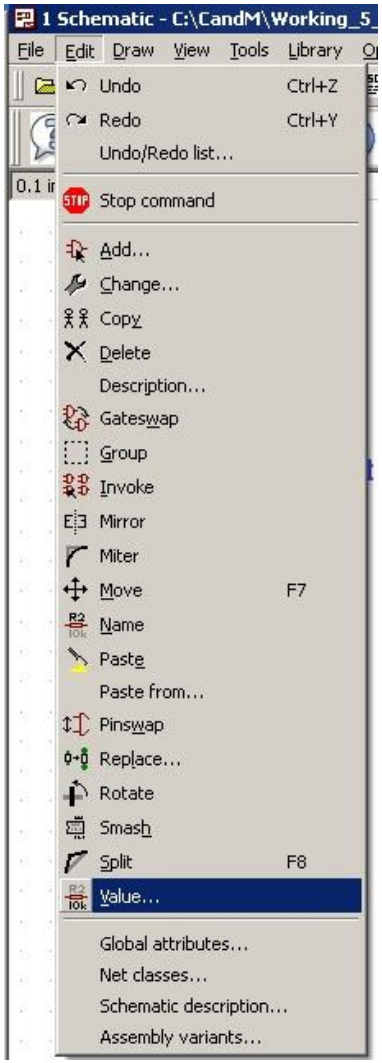
Add all the Elements

Continue the add process to insert one output, one addition block and one multiply block to the schematic. You can find ADD and MUL in the MATH section of the library.



Add some text to the schematic to describe the function of the UDFB. Click 'Draw', 'Text' or use the 'Text' button T to open the text tool. Add the text: "Point Slope Calculator Y=MX+b". Press escape to close the tool.

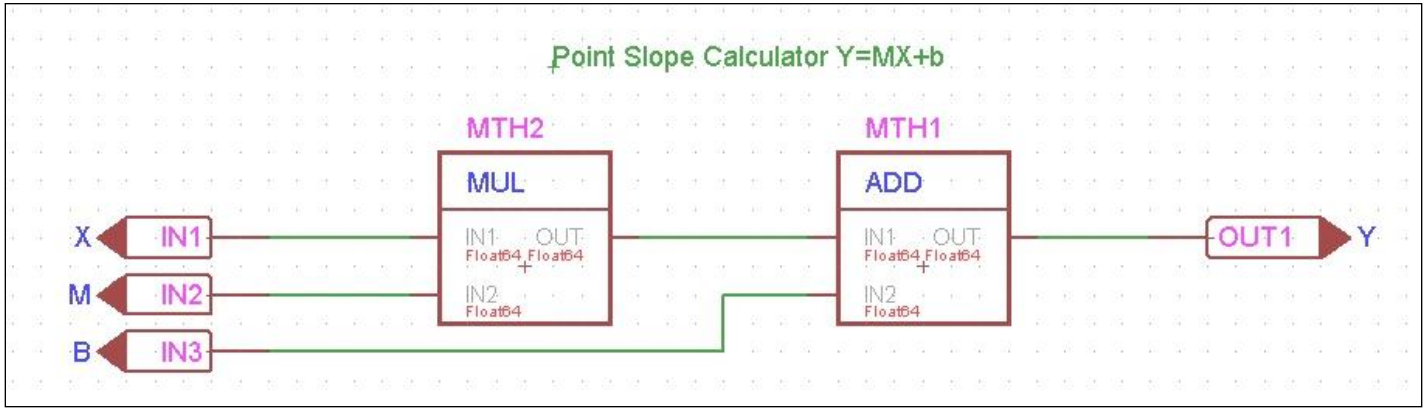




Name the Inputs and Outputs

Click 'Edit', 'Value' from the top left corner of the screen to enter the set value mode. Click the inputs and outputs one at a time to set the value (name) of each as shown.

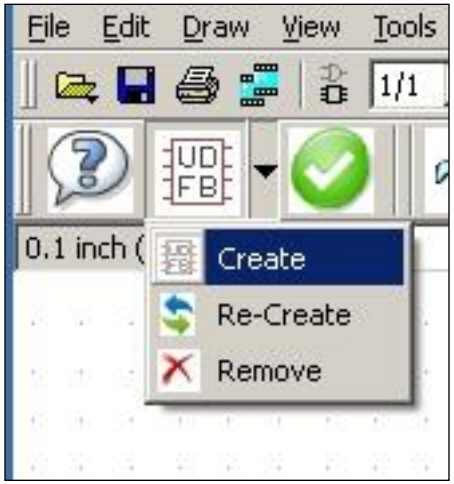
Click the add wire button  and connect the elements as shown. Click the end of a pin to start and end a wire.

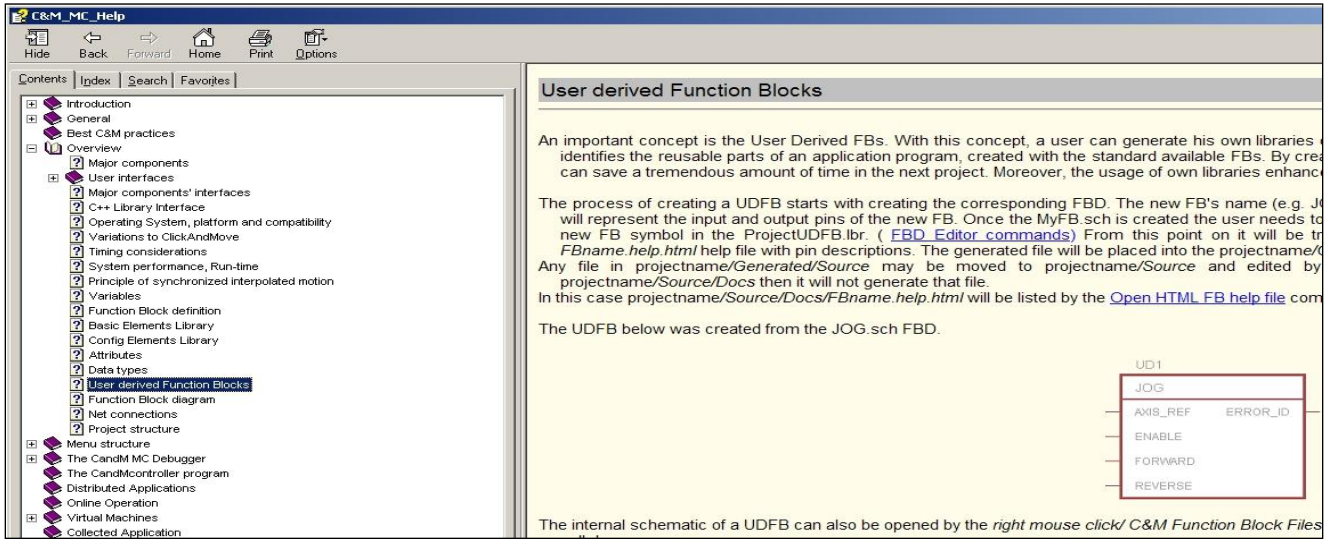


Add UDFB to the Library

Use ‘File’, ‘Save’ to save your changes. Click the UDFB button  and choose ‘Create’ to add the UDFB to the library.

Close the Eagle edit window, the UDFB is ready for use.





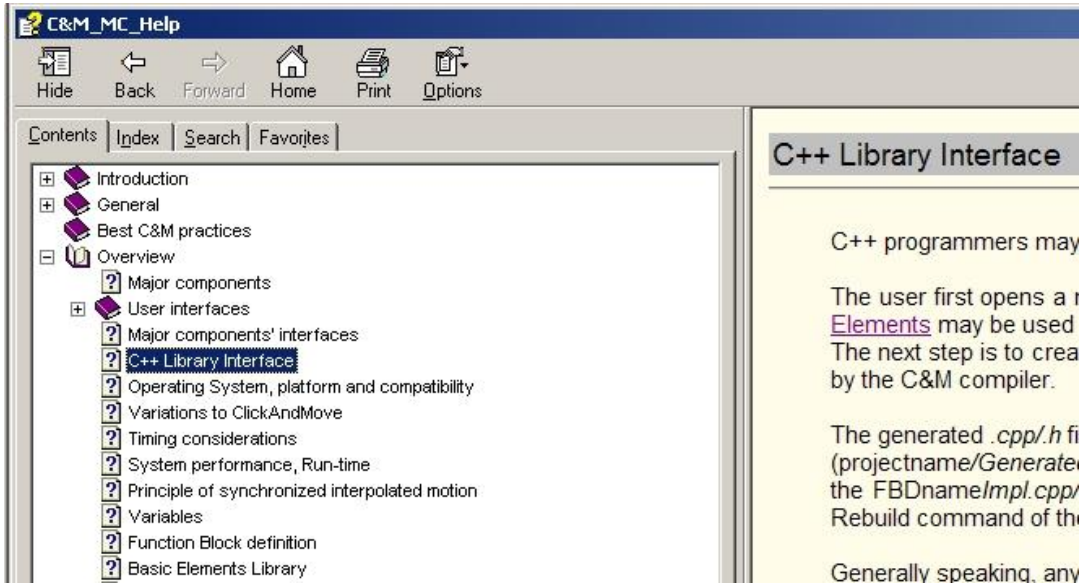
Additional UDFB information can be found in the C&M_MC_Help Overview section.

UDFB With CPP Overview

All function blocks have lines of code behind them. It is possible to create a UDFB containing your custom lines of CPP code.

To do this we will construct a UDFB containing only inputs and outputs. We build the project to have C&M create skeleton CPP files. We then edit the skeleton files adding our code.

The C&M_MC_HELP on this subject can be found in the Overview section under 'C++ Library Interfaces.

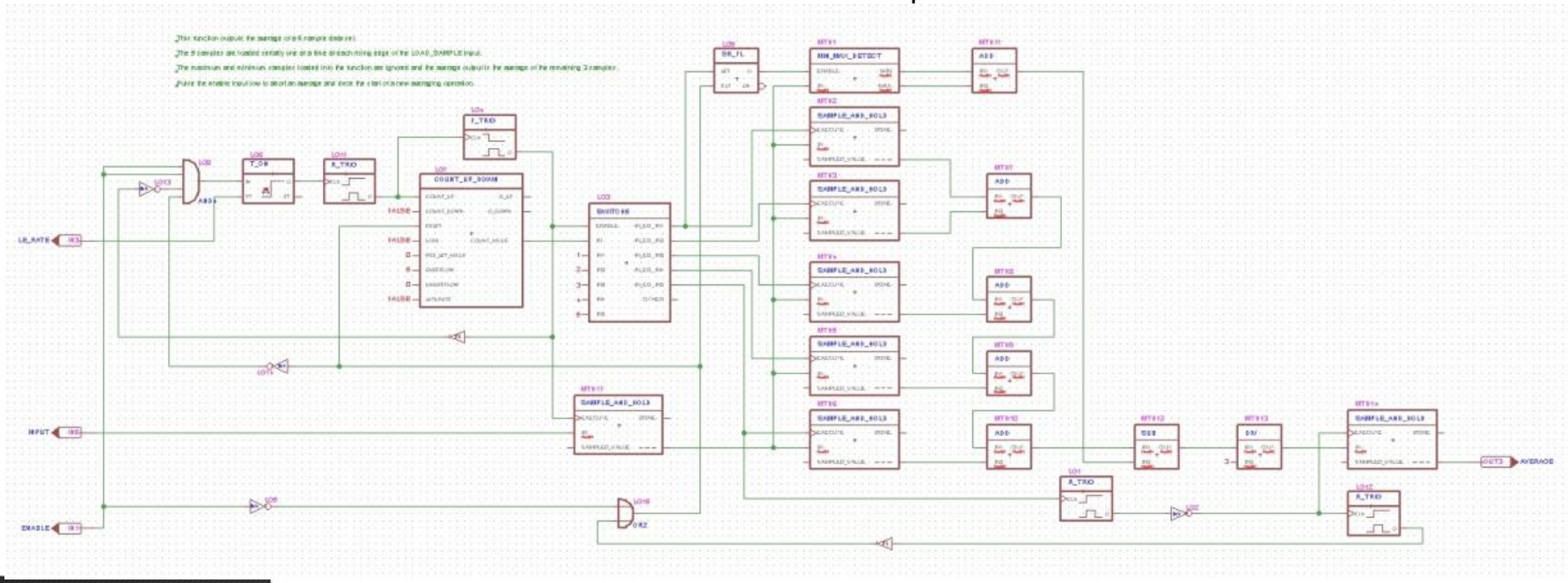


3 of 5 Averaging Problem

One of our customers needed a position averaging function. He wanted to average a set of 5 data points and not include the maximum and minimum samples in the average.

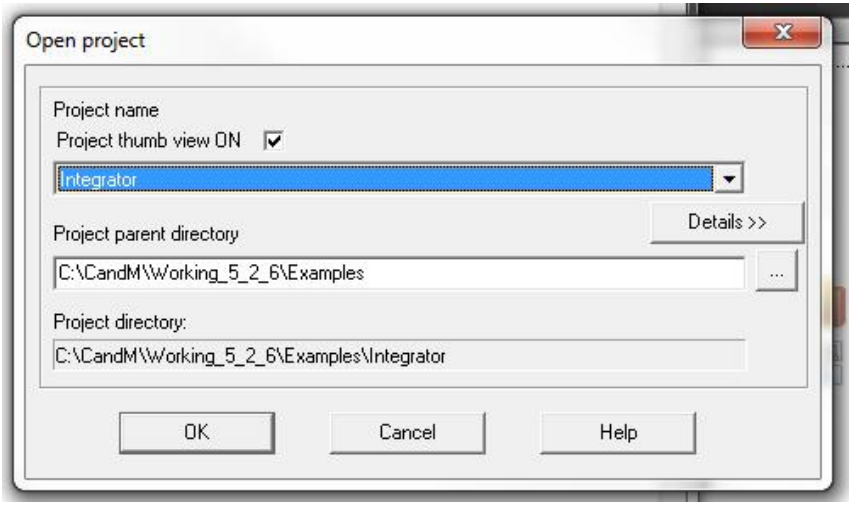
I came up with a solution based on the logic in this diagram. The function works but this might be better implemented in CPP.

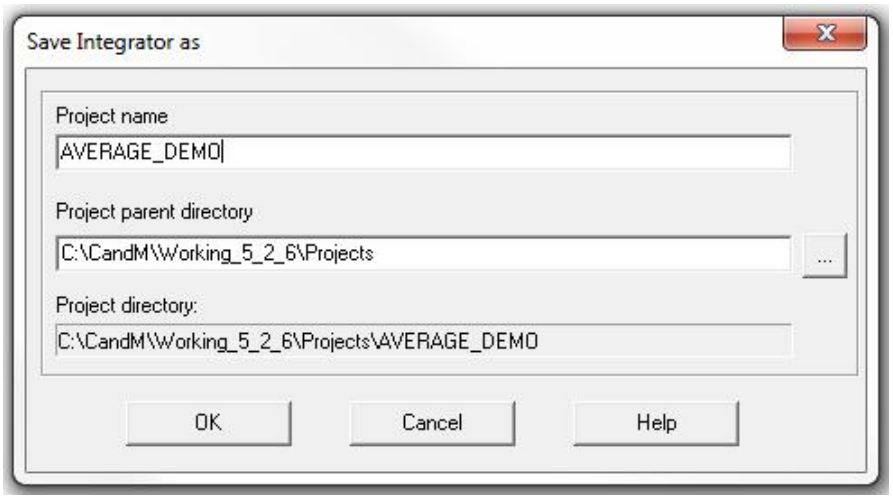
Lets create a UDFB from CPP code to solve the same problem.



Repurpose Existing Project

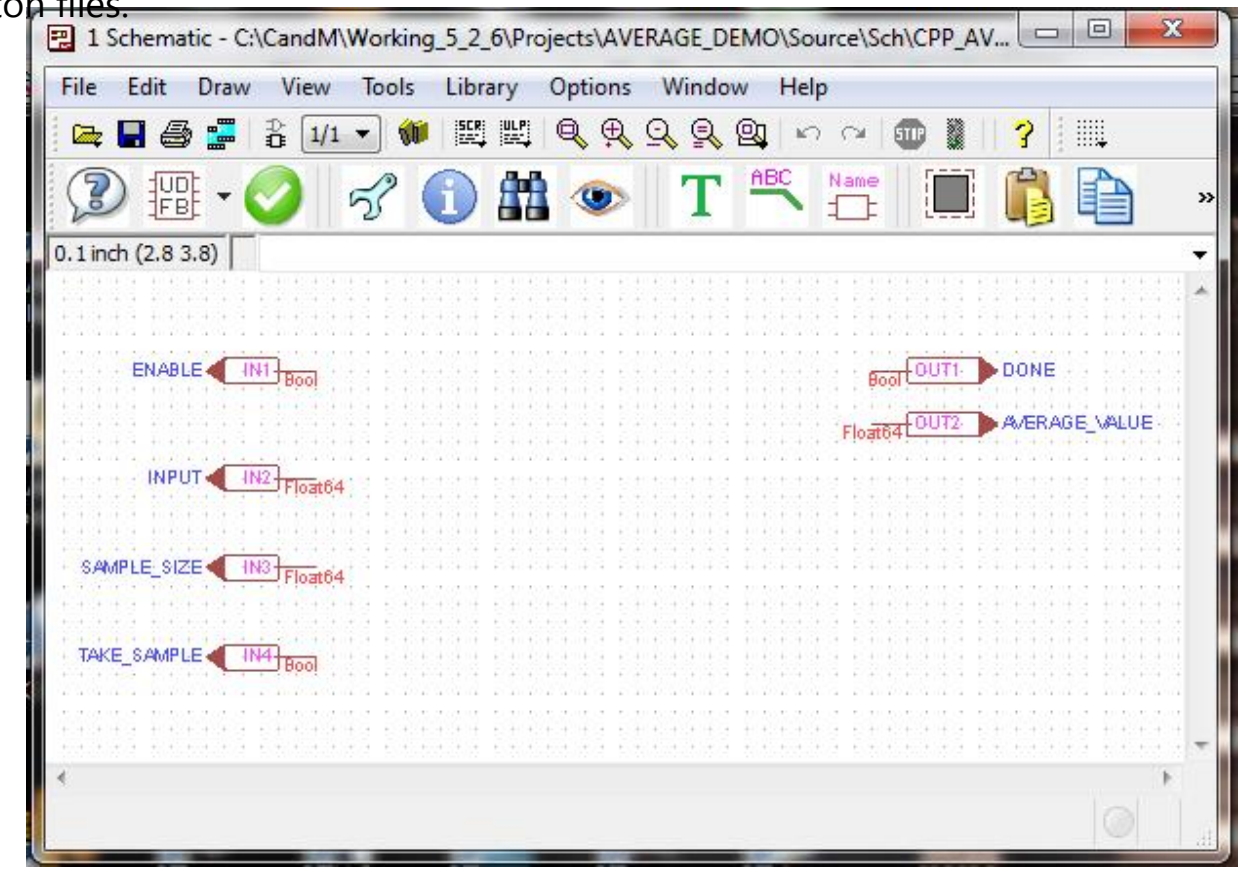
Open the Integrator project from the examples folder and save it to the project folder as AVERAGE_DEMO.



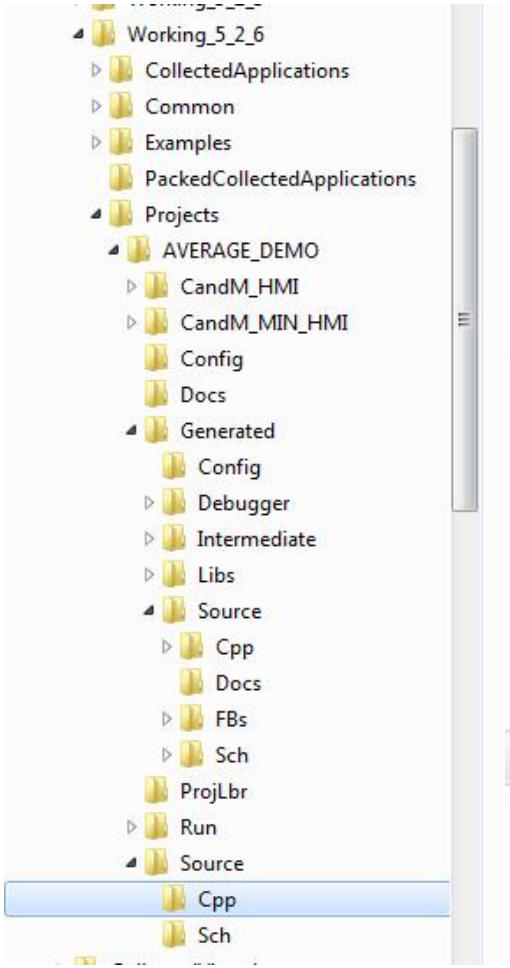


Add an Skeleton UDFB

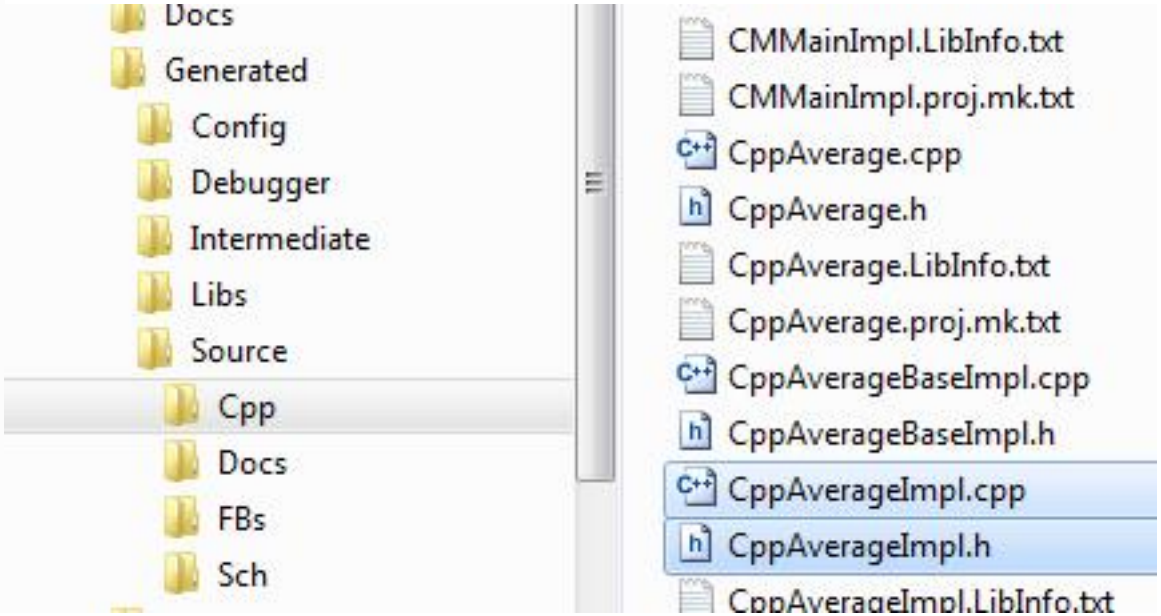
Add a User Derived Function Block to hold the CPP code. Add inputs and outputs shown in the image below. Use the exact names shown, they will become the names of the arguments and returned values for the CPP program. Update the library and save and close the UDFB. Rebuild the project to generate the skeleton files.



Add a Cpp folder to the project.



Create a folder named Cpp under the project\source folder. The new folder is shown in the image on the left. We must move the two files named CppAverageImpl.cpp and CppAverageImpl.h (Shown Below)from the project\Generated\Source\Cpp folder to the folder we just created. Be sure to move them, they must no longer exist in the Cpp folder under the Generated folder.



Add Code to the Header File

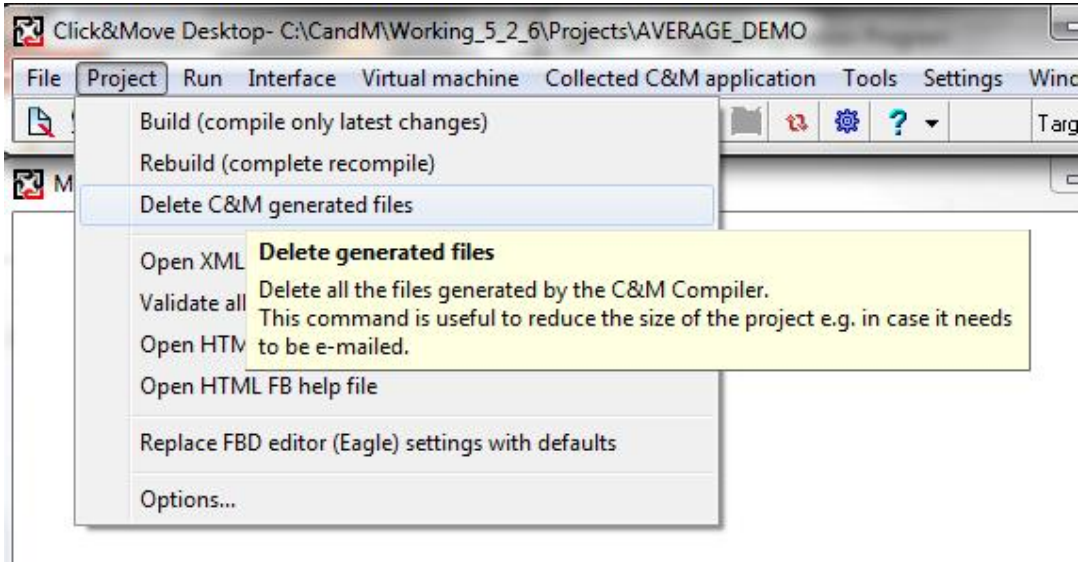
```
10
11  class CppAverage : public CppAverageBase {
12
13      public:
14          CppAverage(Char const * instanceName, const
15          ~CppAverage();
16          void body(void);
17      protected:
18      private:
19          // Object Copy Prevention
20          CppAverage(const CppAverage&);
21          CppAverage& operator=(const CppAverage&);
22
23      // The next lines are my variables.
24
25          Bool sampledEnable_;
26          Bool sampledTake_;
27          Float64 avgSum_;
28          Float64 minVal_;
29          Float64 maxVal_;
30          Int sampleNum_;
31  };
32
```

Add the hilighted line shown on the left to the CppAverageImpl.h file.

Save the changes and close the file.

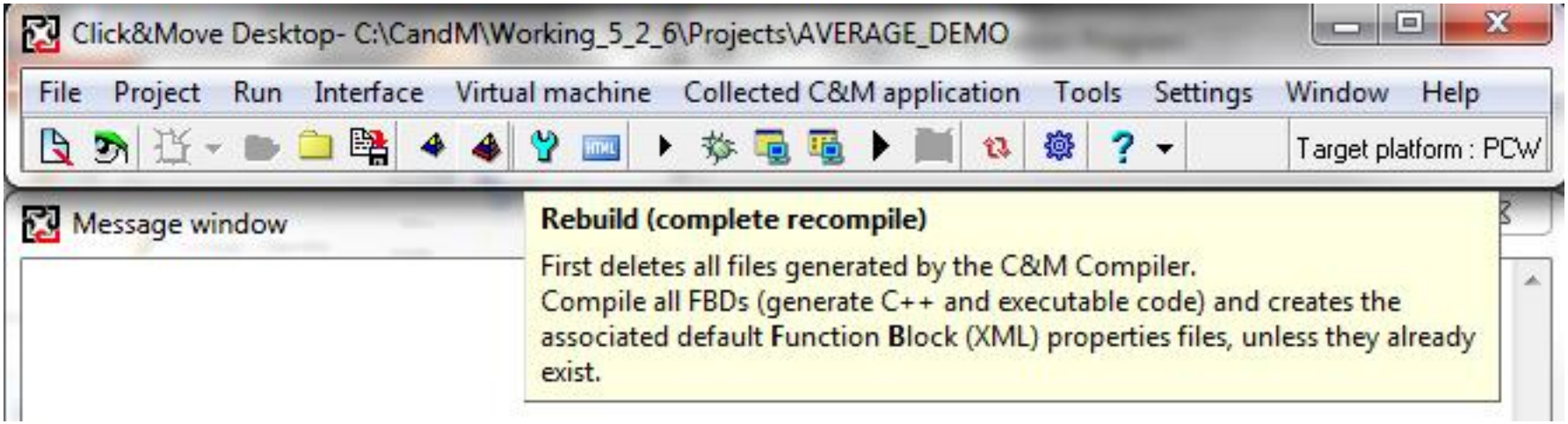
```
23
24  void ImplFBsCM::CppAverage::
25 body()
26  {
27    Float64 tmp;
28
29    if(*pin_Enable)                          // Is the enable pin set to true?
30    {
31      if(!sampledEnable_)                     // Are we detecting the rising edge of the enable pin?
32      {
33        sampleNum_=1;                         // Take the first sample.
34        maxVal_=(*pin_Input);
35        minVal_=maxVal_;
36        avgSum_=maxVal_;
37        sampledTake_=(*pin_TakeSample);       // Note the state of the take sample pin
38      }
39      else
40      {
41        if(sampleNum_<(*pin_SampleSize))      // We keep taking samples until all requested samples are done.
42        {
43          if(sampledTake_!=(*pin_TakeSample)) // Has the Take Sample pin changed state?
44          {
45            tmp = (*pin_Input);               // Read the Input pin value
46            avgSum_+=tmp;                     // Add the new sample inut to my accumulator
47            if(tmp>maxVal_)                   // Copy the sample to my MAX and MIN registers if is an extreme value
48            {
49              maxVal_= tmp;
50            }
51            if(tmp < minVal_)
52            {
53              minVal_ = tmp;
54            }
55            sampleNum_++;                     // Bump up my sample taken count
56          }
57        sampledTake_=(*pin_TakeSample);       // Update the last known value of the take sample input pin
58        }
59        else                                  // If we have taken all samples, calculate the average and signal Done.
60        {
61          src_AverageValue=(avgSum_ - (maxVal_ + minVal_))/((*pin_SampleSize)-2);
62          src_Done=TRUE;
63        }
64      }
65    }
66    else                                      // If the enable pin is turned off, then turn off output Done pin.
67    {
68      src_Done=FALSE;
69    }
70    sampledEnable_=(*pin_Enable);             // Update the last know value of the Enable input pin
71  }
72
```

Add the lines of code shown on the left to the body function in the CppAverageImpl.cpp file. Save and close the file.



Rebuild the Project

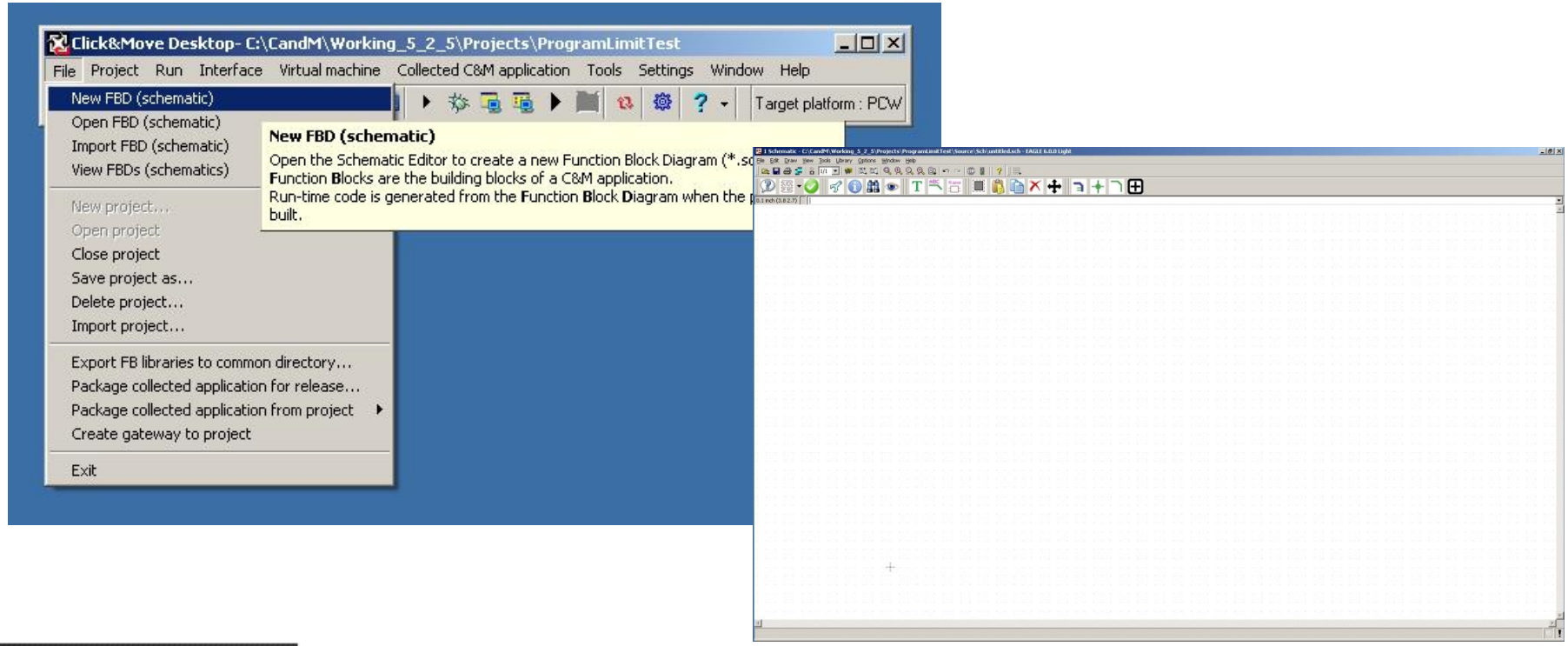
Clean the project and then do a complete rebuild of the project.



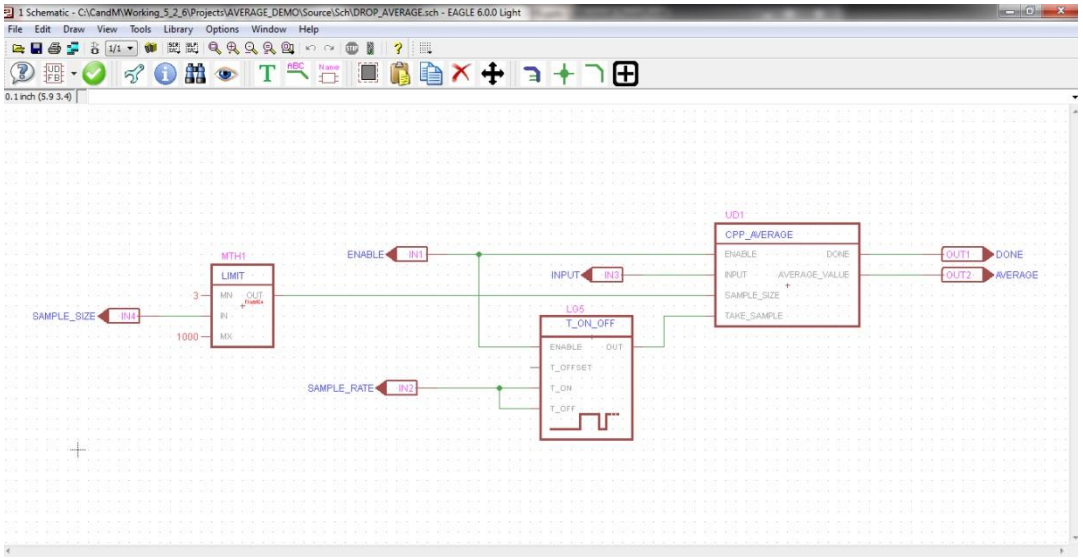
Add an Empty UDFB to the Project

To create the UDFB start with a new empty schematic.

On the C&M Desktop click ‘File’, ‘New FBD (schematic)

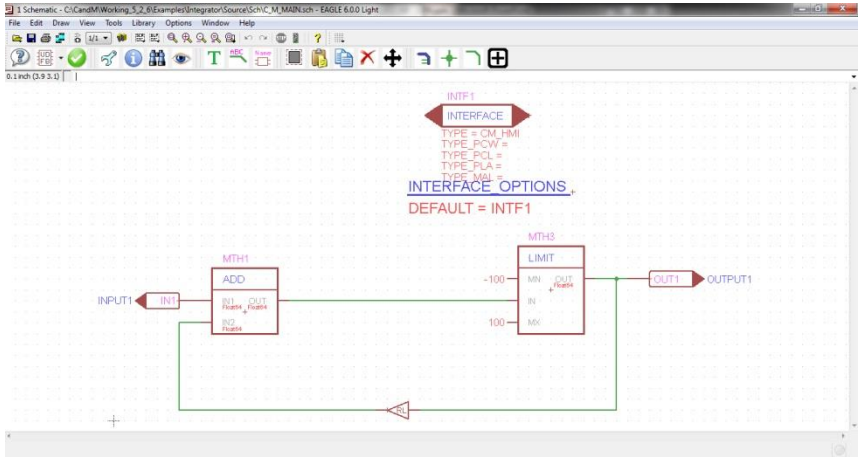


Add DROP_AVERAGE UDFB To the Project

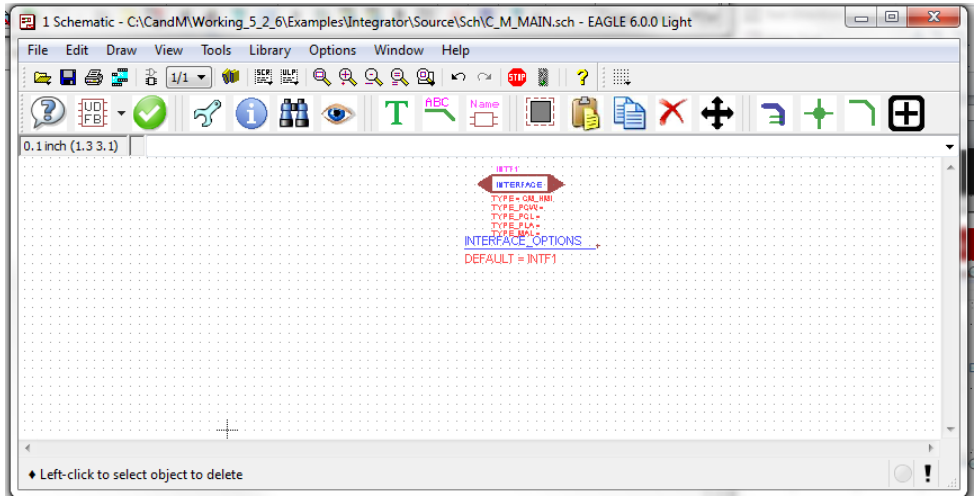


Add the elements shown on the left to the empty schematic just created. Save the schematic as DROP_AVERAGE.

Delete the Integrator schematic

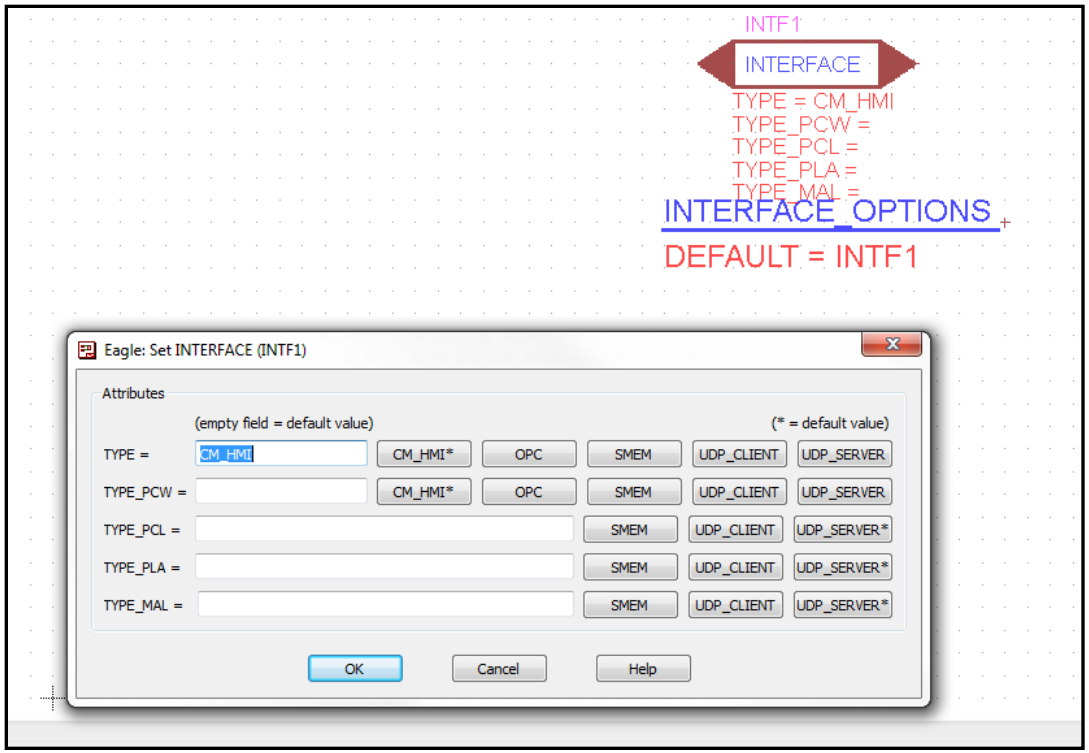


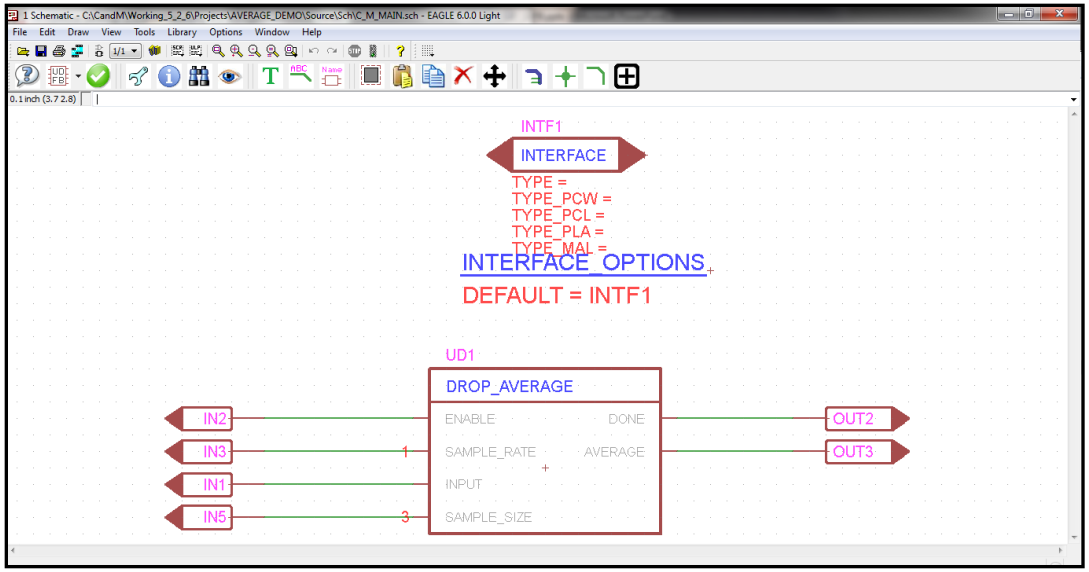
Open the Main schematic and delete all the logic blocks, wires, inputs and outputs.



Disconnect the Graphical HMI

Right click the INTERFACE object and choose Set Connect . Clear the field named “TYPE =“ and click “OK”





Add Averaging function

Add the inputs, outputs, the DROP_AVERAGE UDFB, and the wires as shown in the image at left.

Set the default values for SAMPLE_RATE and SAMPLE_SIZE as shown.

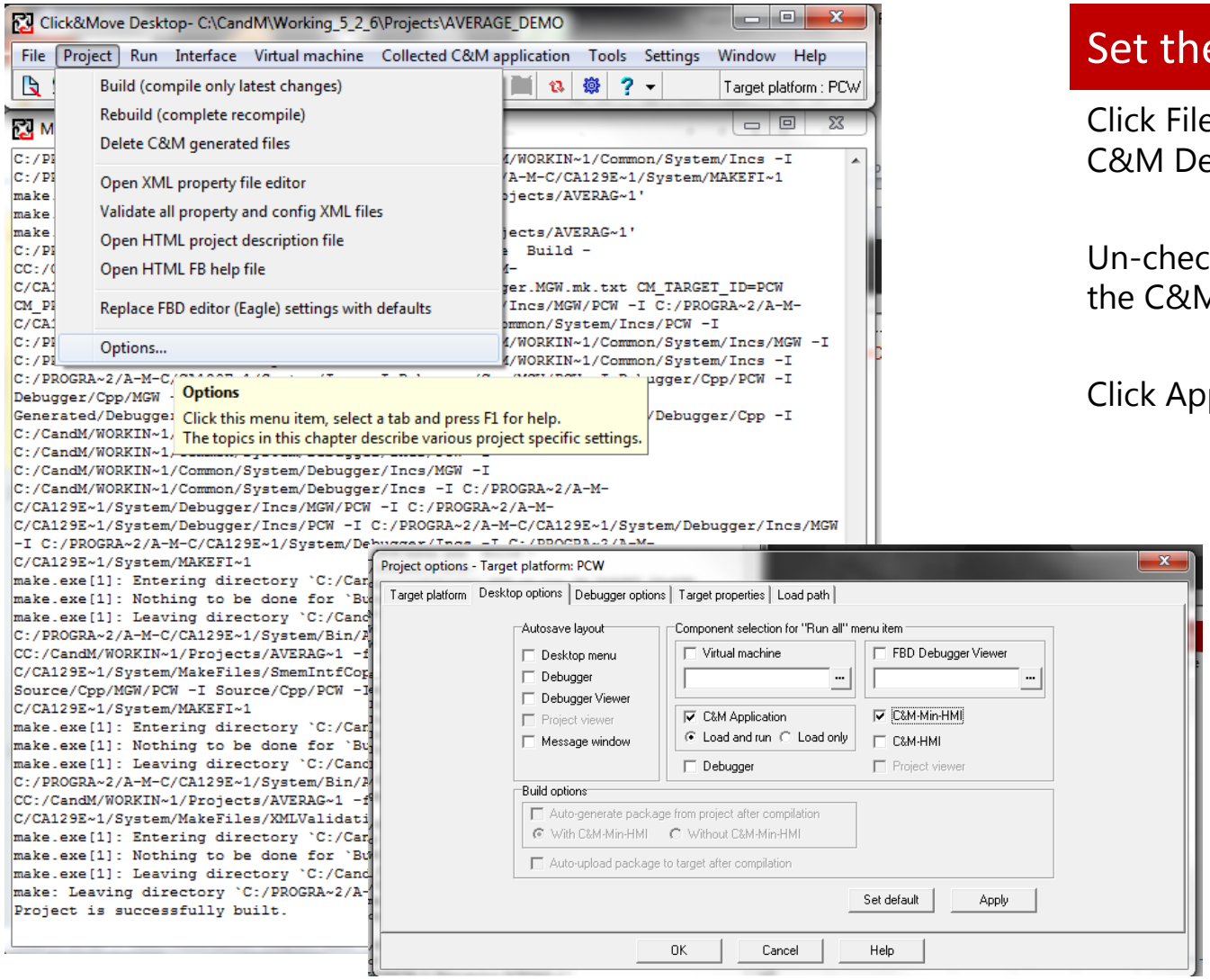
Close the schematic and rebuild the project.

Set the project Options

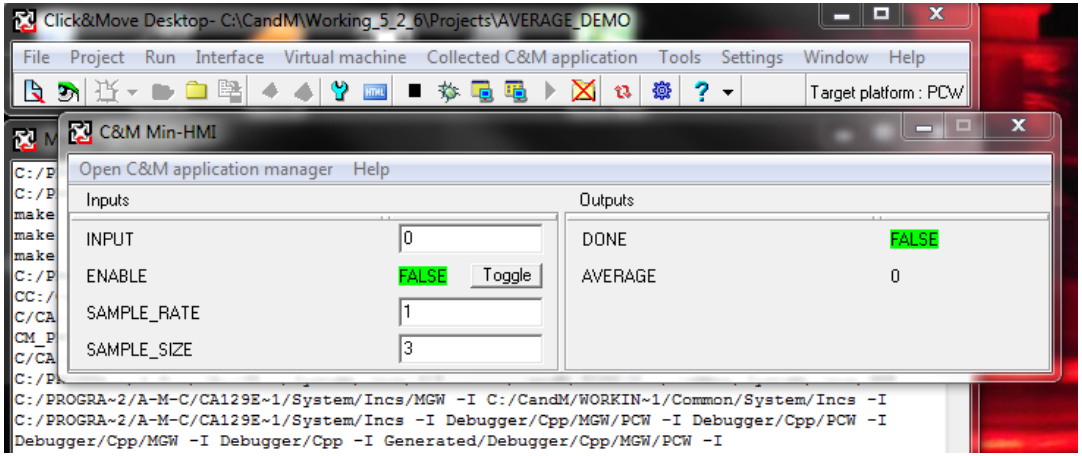
Click File and Options from the C&M Desktop.

Un-check the C&M Hmi and check the C&M Min-Hmi.

Click Apply and OK



Run the Program



At startup the Min-Hmi will open with controls and indicators for each input and output on the main schematic.

The program samples a stream of input data at the sample rate. After collecting the number of samples indicated by sample size the average is calculated.

The calculated average will not include the maximum and minimum valued samples taken. For that reason the actual number of samples in the average is always two less than the value of sample size.

To test, use a 10 second sample rate and the debugger to monitor the action and set the input value manually from the mini HMI. The first sample is taken when the enable input goes true and subsequent samples are taken each time the take sample input changes state.