



# Hands-On Workshop: How to Use Freescale's **FreeMASTER Tool** for **Development and Debugging**

AMF-ACC-T1244

Michal Hanak | Application Engineer  
John H. Floros | Field Application Engineer

OCT. 2015

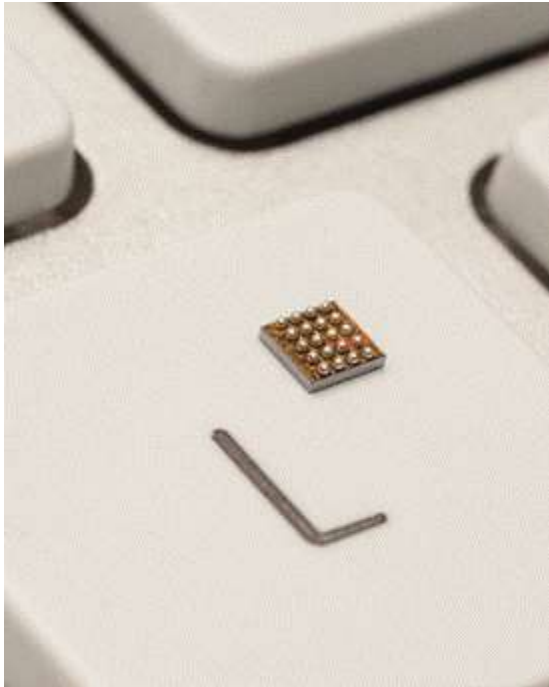


External Use

Freescale, the Freescale logo, AN/Vis, C-S, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetic, MagniV, motorKIT, PEG, PowerQUICC, Prosecc Expert, QorIQ, QorIQ Qonverge, Qorivos, Ready Files, SafeAssure, the SafeAssure logo, StarCore, Synchrify, Vortiga, Vybrid and Xilinx are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. AirMat, BeeKit, FreeStack, CoreNet, Flexis, LayerStack, MXC, Platform in a Package, QUICC Engine, SMARTMO2, Tower, TurboLink and UMEMS are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. © 2015 Freescale Semiconductor, Inc.



# Introduction: MOTIVATION FOR FREEMASTER



- FreeMASTER was created as an internal development tool by our Motor Control team - “by the engineers for the engineers” in year 2000. Today it is maintained in the SW Libs team in Freescale.
- The original motivation was to be able to visualize and tune parameters without stopping the MCU core in the debugger. Breakpoints in code are often a luxury which you cannot afford in real time applications.
- As it matured a customizable and scriptable HTML rendering engine was added to enable custom GUI pages to be created and used to control, demonstrate or sell embedded applications.



# FreeMASTER Overview

## A Real-Time Monitor for Your Freescale MCU



# Agenda

- What is FreeMASTER?
- FreeMASTER as a Real-Time Monitor
- FreeMASTER as a Control GUI
- FreeMASTER vs. Debugger
- FreeMASTER Replacing Custom GUI Applications
- FreeMASTER Internal Application Structure
- Summary



# Objectives

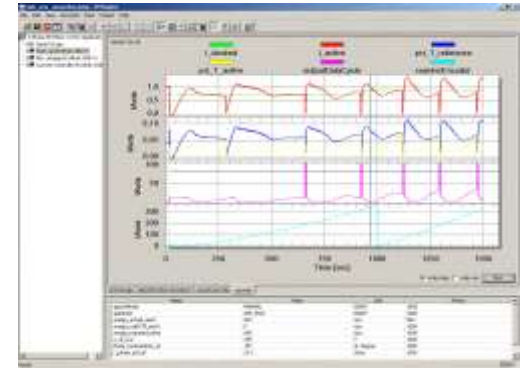
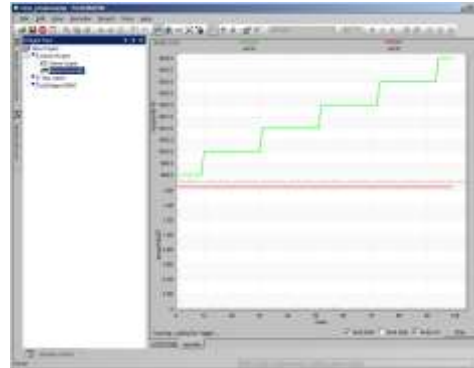
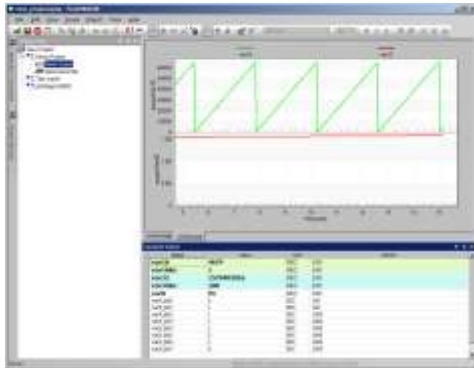
After this FreeMASTER Overview, you will know:

- ✓ What FreeMASTER is and what features it contains for real time monitoring of your application on the MCU
- ✓ How to configure some of the features available in the FreeMASTER user interface
- ✓ The steps required to enable FreeMASTER in your application at a high level



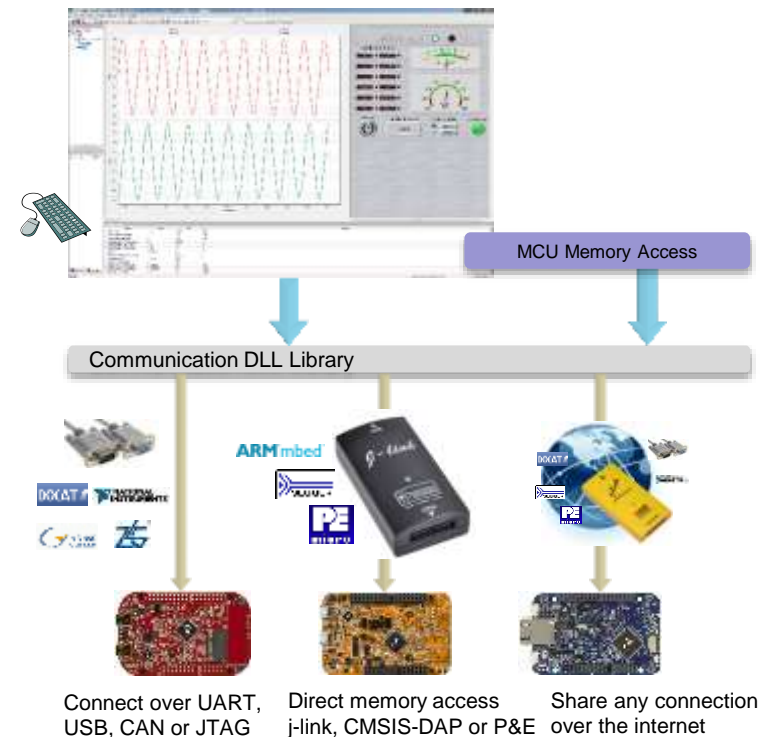
# FreeMASTER as a Real-Time Monitor

- **FreeMASTER can Real-time Monitor**
  - internal variables
  - processes & algorithms
  - application states



# FreeMASTER as a Real-Time Monitor

- **PC Host Connects to an embedded application over unified DLL library**
  - **SCI, UART**
  - **USB-CDC** - Kinetis, ColdFire V2
  - **CAN** - msCAN, FlexCAN with PC interface from IXXAT, Vector, NI, Glinker, ZLG
  - **JTAG/EOnCE** (56F8xxx only)
  - **BDM** - Kinetis, PowerPC, ColdFire, HCS with Segger, P&E Micro, CMSIS-DAP, iSystem, ...
  - Any of the above remotely over the IP network
- **Enables access to application memory**
  - Parses ELF application executable file
  - Parses DWARF debugging information in the ELF file
  - Knows addresses of global and static C-variables
  - Knows variable sizes, structure types, array dimensions etc.





# FreeMASTER Features

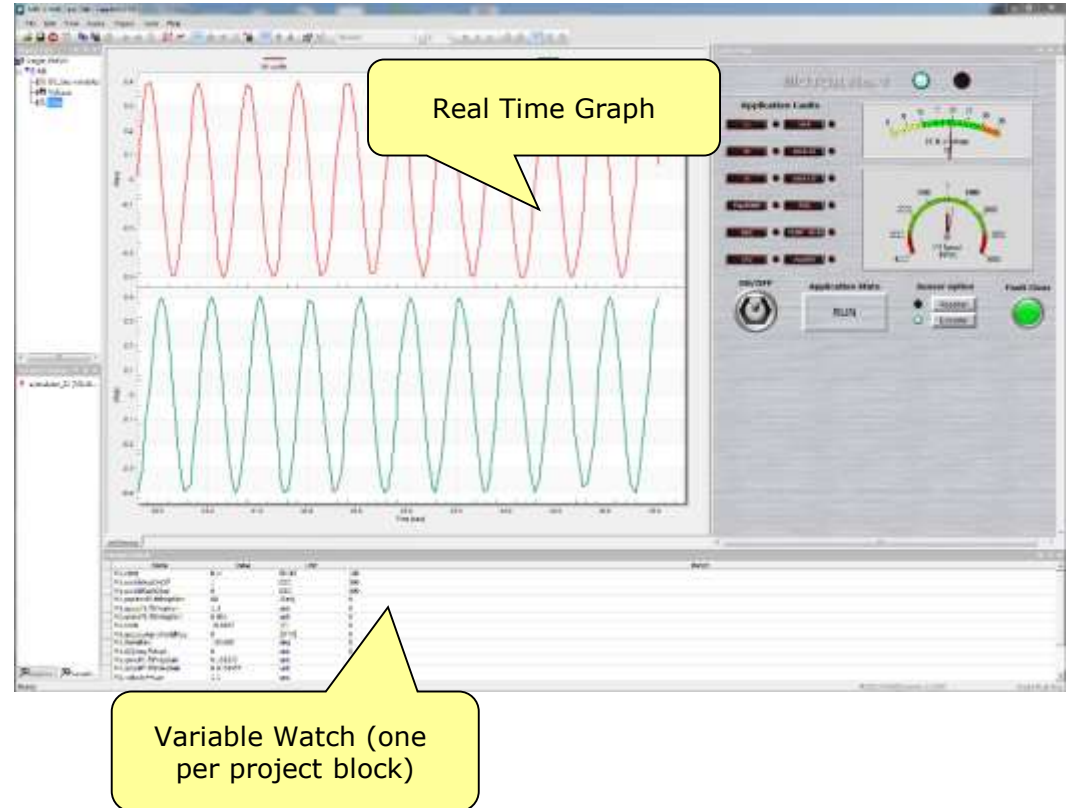
- Graphical environment
- Easy to understand navigation
- Real-time access to embedded-side C variables
- Visualization of real-time data in the Scope window
- Acquisition of fast data changes using the on-target Recorder
- Built-in support for standard variable types (integer, floating point, bit-fields)
- Demo mode with password protection support
- HTML-based description or navigation pages
- ActiveX interface to enable VBScript or JScript control over embedded applications
- Remote Communication Server enabling a connection to target board over a network, including the Internet



# FreeMASTER as a Real-Time Monitor

Display the variable values in various formats:

- **Text**, tabular grid
  - variable name
  - numeric value
  - peak detector
  - number-to-text enumeration
- **Real-time waveforms**
  - up to 8 variables simultaneously in an oscilloscope-like graph
- **High-speed recorded data**
  - up to 8 variables in on-board memory **transient recorder**



# FreeMASTER as a Real-Time Monitor

- **Variable Transformations**

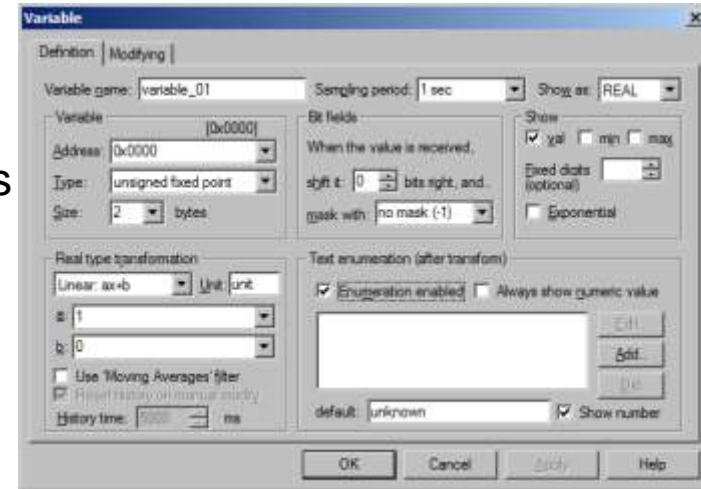
- Value can be transformed to custom units
- Transformations may reference other variable values
- Inverse-transformation applied when writing a new value to the variable

- **Ability to protect memory regions (TSA)**

- Describing variables visible to FreeMASTER
- Declaring variables as read-write to read-only for FreeMASTER - the access is guarded by the embedded-side driver

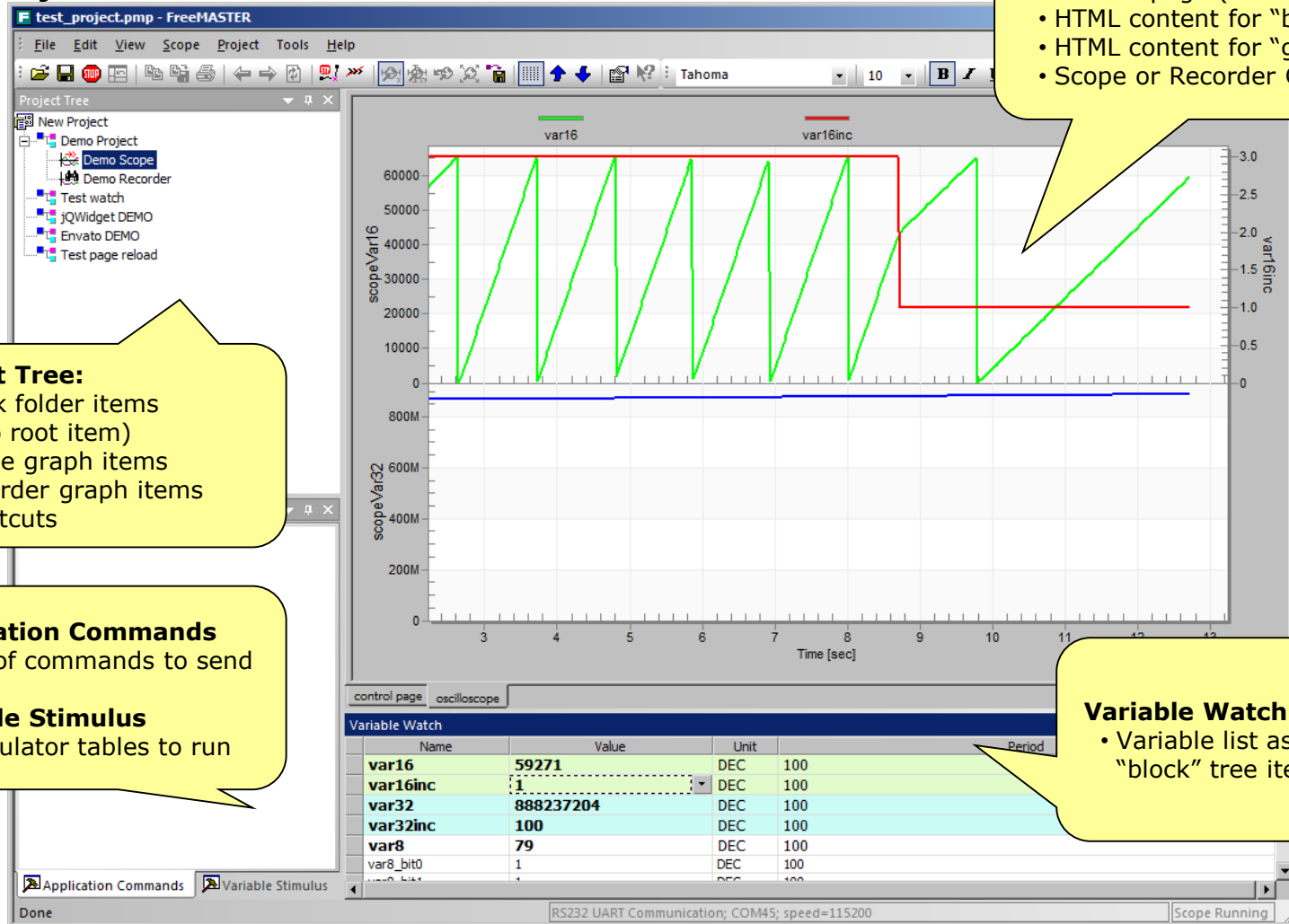
- **Application Commands**

- Command code and parameters are delivered to an application for arbitrary processing
- After processed (asynchronously to a command delivery) the command result code is returned to the PC
- Legacy feature, not used in today's applications (requires target-side driver)



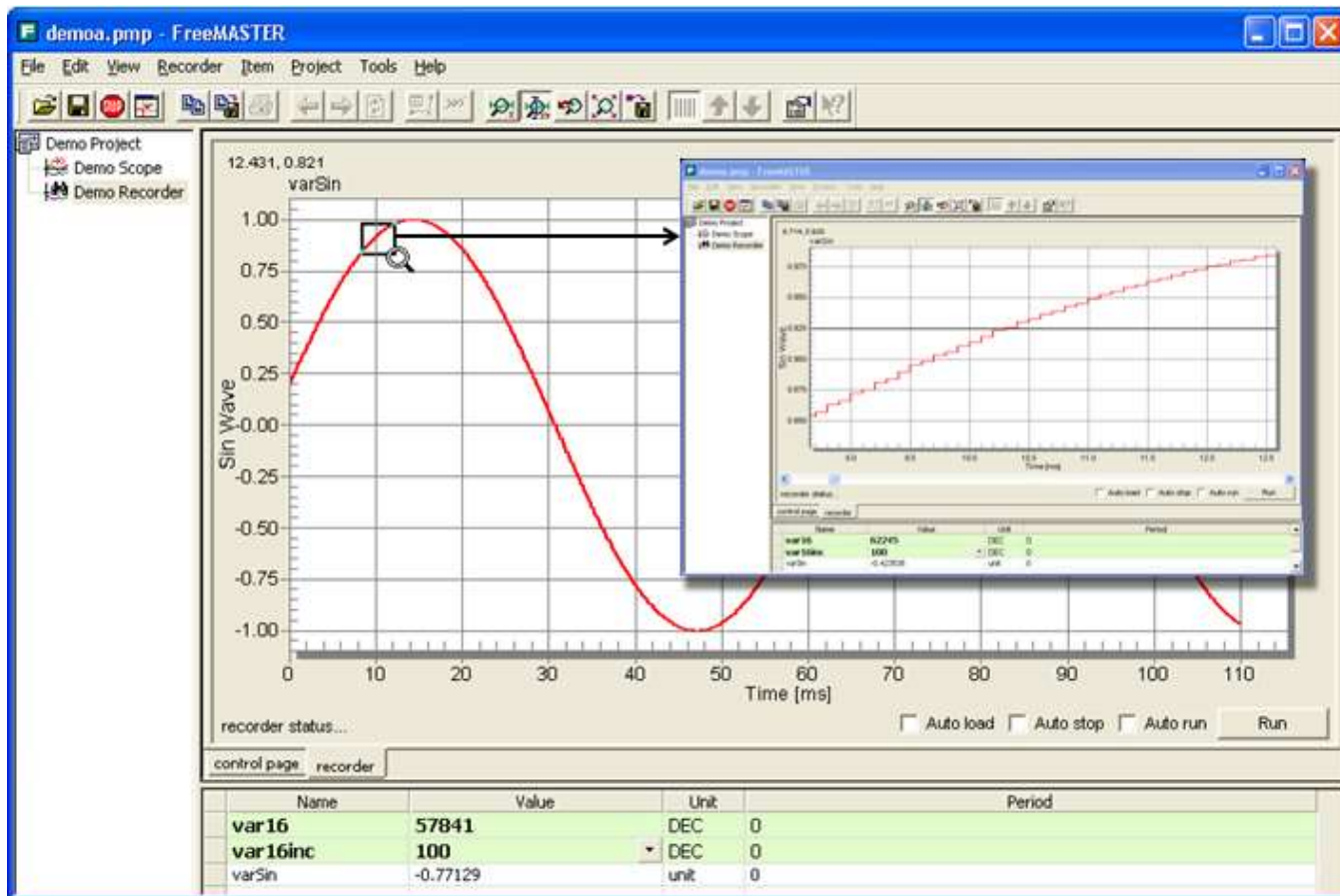
# FreeMASTER as a Real-Time Monitor

## Anatomy of the main window

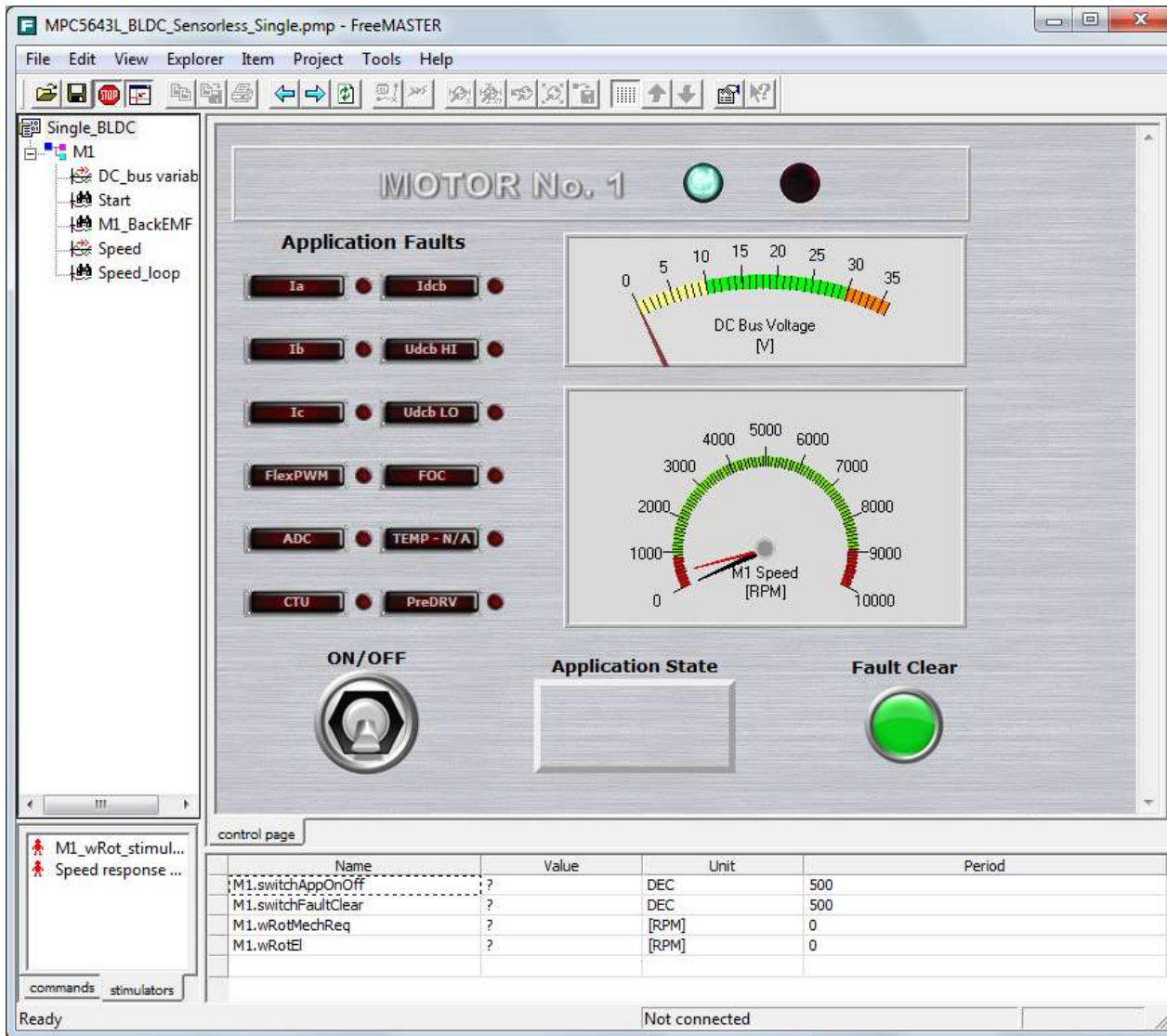


# FreeMASTER as a Real-Time Monitor

- **Establish a Data Trace on Target**
  - Set up buffer (up to 64KB), sampling rate and trigger



# FreeMASTER as a Real-Time Monitor



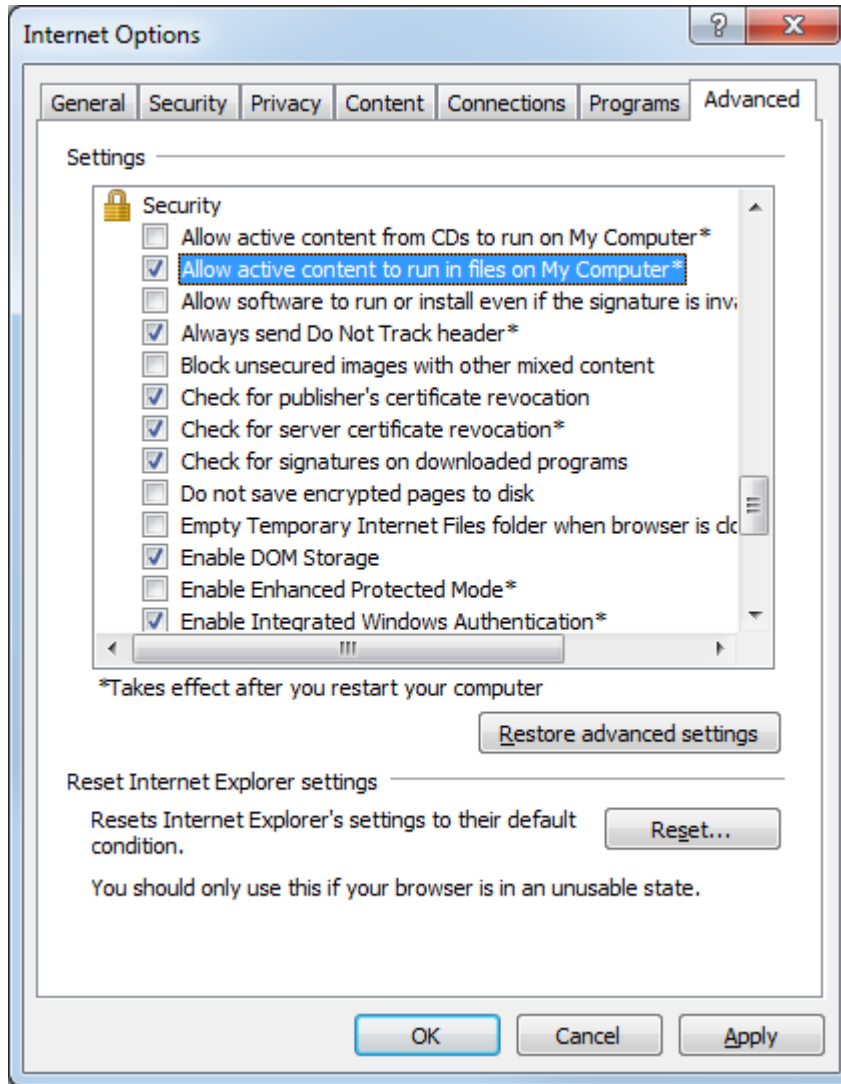
The HTML-based data visualization area. The user can provide any collection of ActiveX-based instrumentation to create custom visual dashboards as complex or elegant as desired.

The data visualization area may also be used to display arbitrary information, such as presentations, help files and fact sheets.





# FreeMASTER as a Real-Time Monitor



- In order to allow the ActiveX – based instrumentation to run it may be necessary to set you Internet options to allow the active content to run.

# FreeMASTER as a Real-Time Monitor

## Demo Mode

- To prevent modification, the project's author can lock the project against changes by switching it into the *Demo Mode*.
- An important part of the FreeMASTER's capabilities is the demonstration and description of the target board application. It is essential that the demonstration project, once prepared, is not accidentally modified.
- In the Demo Mode, the user cannot change the *Project Tree item properties, cannot add or remove the tree items, and cannot change any project options.*



# FreeMASTER as a Real-Time Monitor

## FreeMaster Communication Driver

- Go to [www.freescale.com/freemaster](http://www.freescale.com/freemaster)
  - Go to the “downloads” tab and look for “FreeMASTER Communication Driver”
  - In the CodeWarrior project window, paste the FreeMASTER folder into the “Project\_Headers” folder of your project
  - Once the package is installed, there are several options to interface with the target device, using CAN, SCI, or JTAG

For additional information, refer to Freescale’s Application Note AN4752

# FreeMASTER as a Real-Time Monitor

## Adding the FreeMaster Communication Driver

- The corresponding header and C files from the unpacked folder are added to the project header files.
- The paths containing the FreeMaster files must be incorporated into the project:
  - From the CW menu bar, go to Project > Properties
  - Go to “C/C++ Build”> “Settings”
  - Look for the item “Access Paths” under S12Z Compiler
  - Add the following paths under: ”Search User Paths”:
    - "\${ProjDirPath}\Project\_Headers\FreeMASTER“
    - "\${ProjDirPath}\Project\_Headers\FreeMASTER\S12ZVM“
    - "\${ProjDirPath}\Project\_Headers\FreeMASTER\src\_common“

# FreeMASTER as a Real-Time Monitor

## Using the FreeMaster Serial Driver

- At the top of your project, we have included the freemaster header file:

```
#include "freemaster.h"
```

- The “main” routine now includes a FreeMaster initialization (must be always after the comms initialization; in this case, the SCI):

```
FMSTR_Init();
```

- The infinite for loop now includes a function that continuously sends the variable values to FreeMaster

```
FMSTR_Poll();
```

# FreeMASTER as a Real-Time Monitor

## Steps to integrate FreeMASTER in your Application

1. Include the files under the **FreeMASTER Serial Communication Vxx\src\_common** in your application code project with no changes.
2. One file changed in **FreeMASTER Serial Communication V1.6\src\_platforms\MPC56xx** directory:
  - a) renamed freemaster\_cfg.h.example to freemaster\_cgh.h
  - b) Configure FreeMASTER by changing macro definitions
3. Addition to main.c
  - a) Add function call FMSTR\_Init() after system init
  - b) Add function call FMSTR\_Poll(); in main loop
4. To build with FreeMASTER support for MPC56xx, include all files under **FreeMASTER Serial Communication V1.6\src\_platforms\MPC56xx** and **FreeMASTER Serial Communication V1.6\src\_platforms\MPC56xx** directories.

# FreeMASTER as a Real-Time Monitor

- Highlights
  - Access to target variables, symbols and data types
  - Safe access over UART, CAN or USB with target-side driver
  - Transparent access over BDM (no target-side driver needed)
  - Addresses parsed from ELF file or provided by target (TSA)
  - Fine tuning parameters or direct control via variable modifications
  - Scope graphs with real time data in [ms] resolution
  - Recorder visualization transitions close to 10[us] resolution





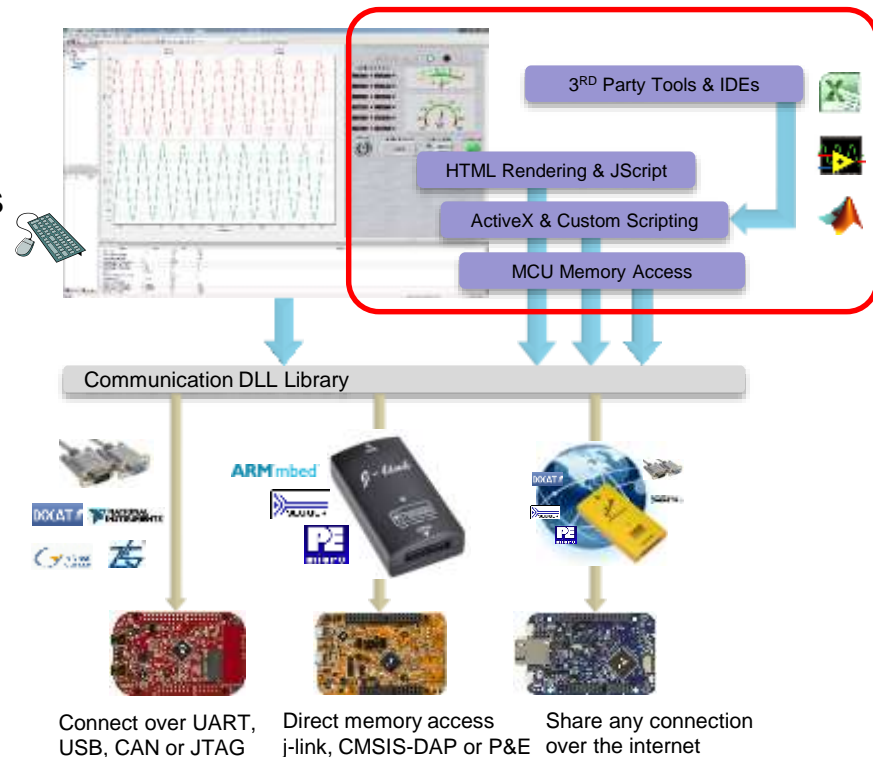
# FreeMASTER as a Control GUI

- **Variable access and modification**

- Manually in the Watch pane
- Time-tables & stimuli modification
- Script-based read/write directly from GUI
  - mouse-clicks and keyboard control
  - push buttons and forms
  - sliders, gauges or other ActiveX/HTML5 widgets
  - custom intelligence and control algorithms
- ActiveX clients external to FreeMASTER
  - Excel or Matlab – typical programmable clients
  - FreeMASTER enables HW-in-loop simulations
- Works over all communication interfaces

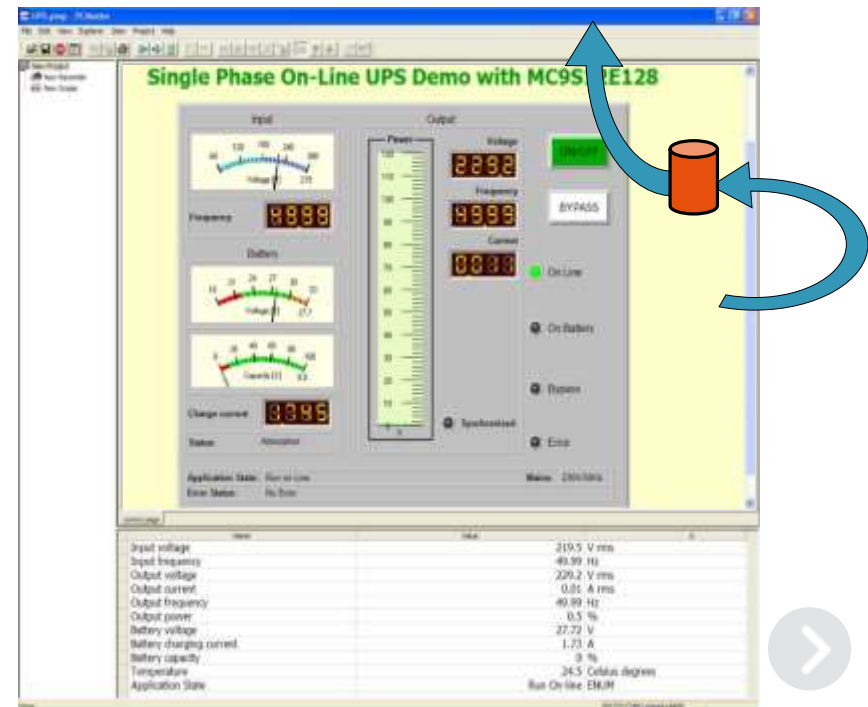
- **Sending Application Commands**

- “Traditional” control approach
- Not recommended as it is limited to systems with target-side agents (UART & CAN).



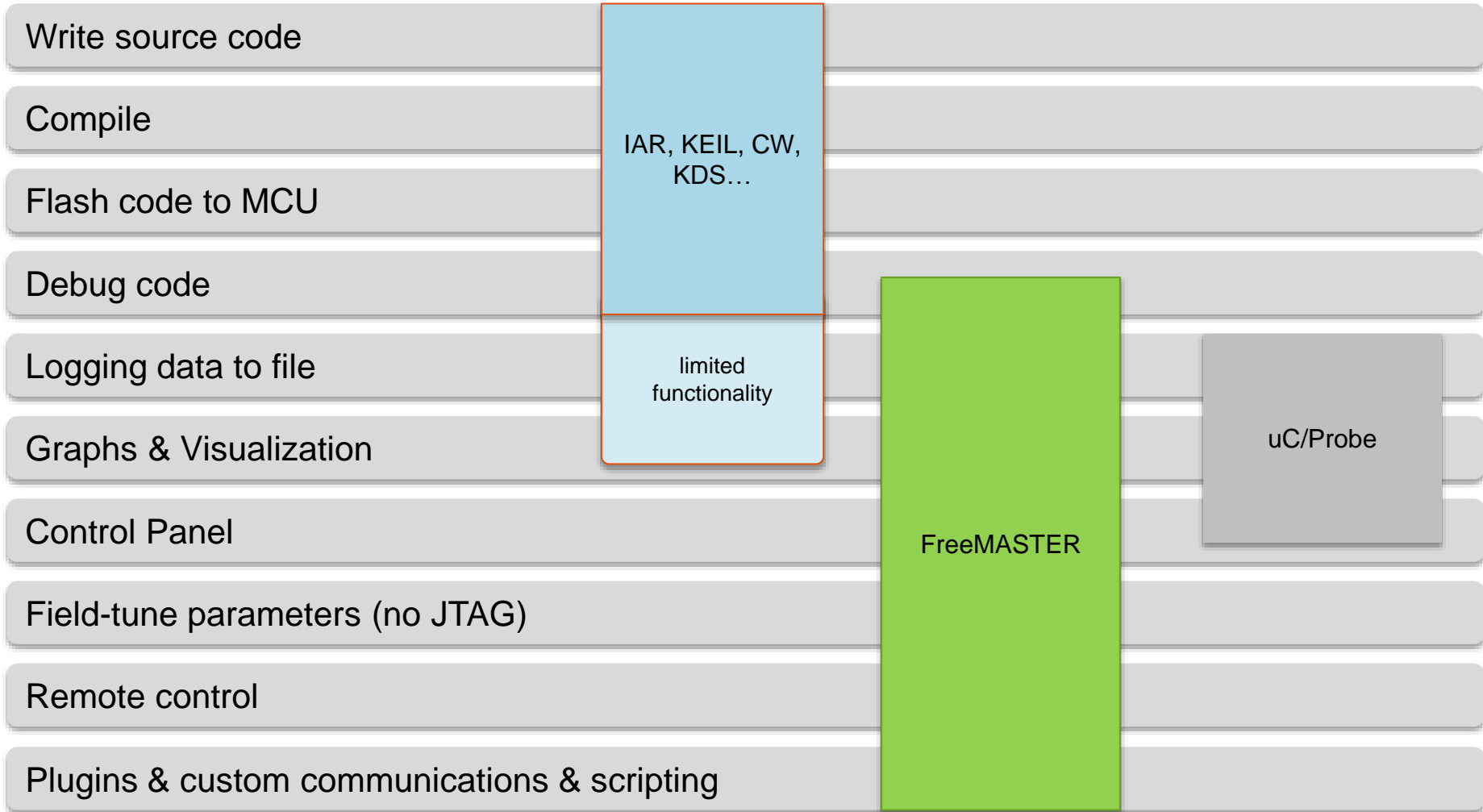
# FreeMASTER as a Control GUI

- Scripting in FreeMASTER
  - HTML pages are displayed directly in the FreeMASTER window
  - InternetExplorer v10 used as the rendering engine
  - HTML may contain scripts and ActiveX objects
- FreeMASTER invisible ActiveX object
  - Script accesses the FreeMASTER functionality through this object
  - Variable access
  - Direct memory access
  - Stimulator access
  - Application Commands
  - Recorder Data
  - Symbol and data type information



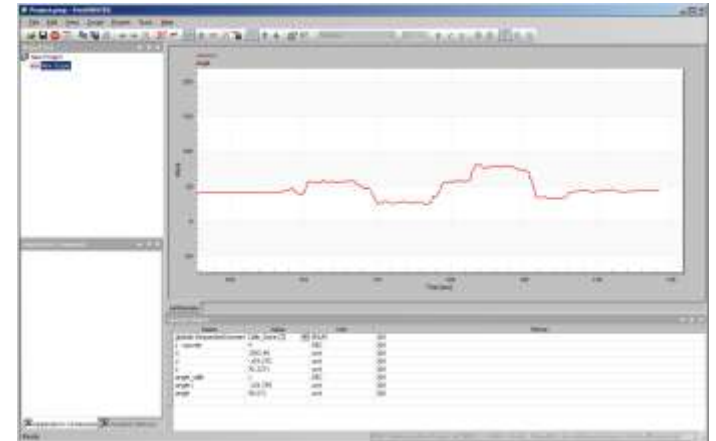
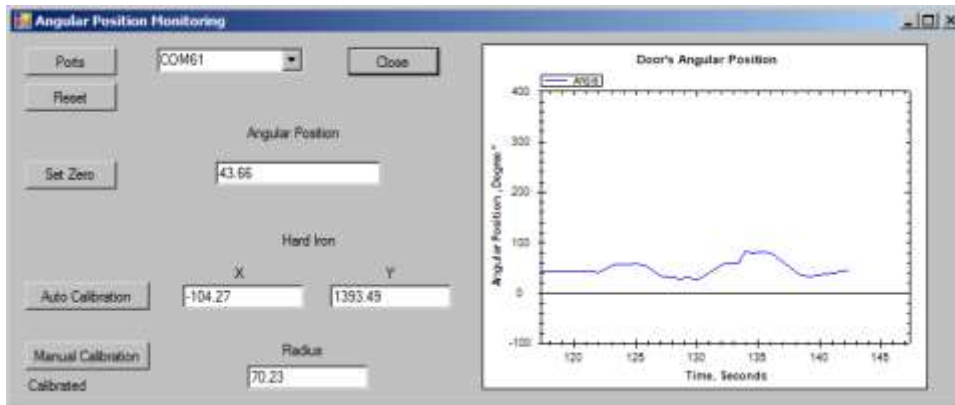


# FreeMASTER vs. Debugger



# FreeMASTER Replacing Custom GUI Applications

- **FreeMASTER instead of Custom GUIs**
  - comparing FreeMASTER with custom GUI approach
  - typical use cases



# From Custom GUI to FreeMASTER

- **Typical pitfalls of using custom GUI**

- Requires PC Host programming tools and skills
- Never enough communication interfaces, communication issues over and over again
- Time to develop a robust PC Host application
- Deploying GUI to host PC
- Using custom GUI with modified user application

- **Benefits of FreeMASTER**

- uniform approach – application control by variable modification
- works over UART/CAN but also over non-intrusive BDM
- one tool used with variety of GUIs
- GUI easily extended by multimedia content (charts, documentation) local, online or embedded
- Can be used with user-modified content too. User able to mix “our” data with “his” data in common charts.
- GUI project can be extended by user to cover more functionality

# From Custom GUI to FreeMASTER

- **Typical custom GUI approach**

communication driven data collection, custom protocol

- PC sends request, Target processes and replies with data
  - pro: under full control of developer, may be shielded from the rest of application logic
  - con: communication development just for sake of GUI, typically not used for any other purpose
  - con: migration to different communication media is typically hard
  - con: user modifications of firmware makes the GUI to stop working

- **FreeMASTER approach**

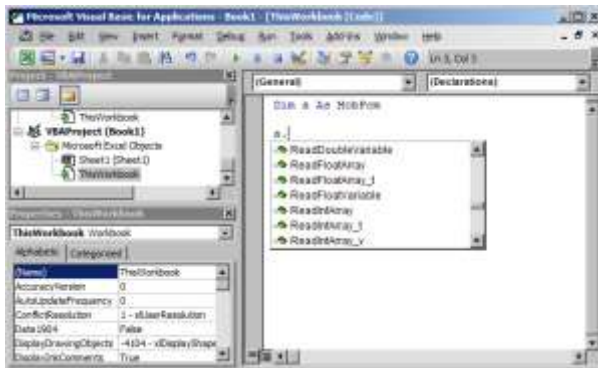
control by modifying variables

- use either artificial variables dedicated for GUI control
- or modify state variables used also by the general application algorithm
  - con: typically requires to change existing applications with custom GUI
  - pro: works over standardized protocol or with BDM direct memory access
  - pro: easy to protect or restrict functionality
  - pro: easy to integrate this approach with additional user modifications to firmware
  - pro: the TSA feature – self-describing and automatic board discovery (FreeMASTER 2.0 in 2015)

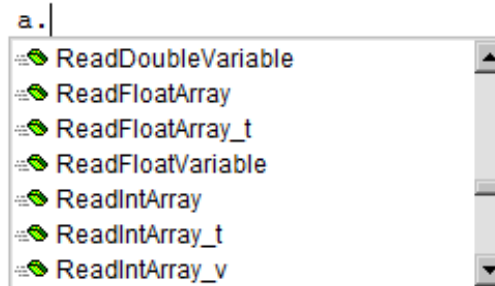
# FreeMASTER Internal Application Structure

- **Inside FreeMASTER**

- how to get maximum out of FreeMASTER
- linking with other executables
- reusing communication layer

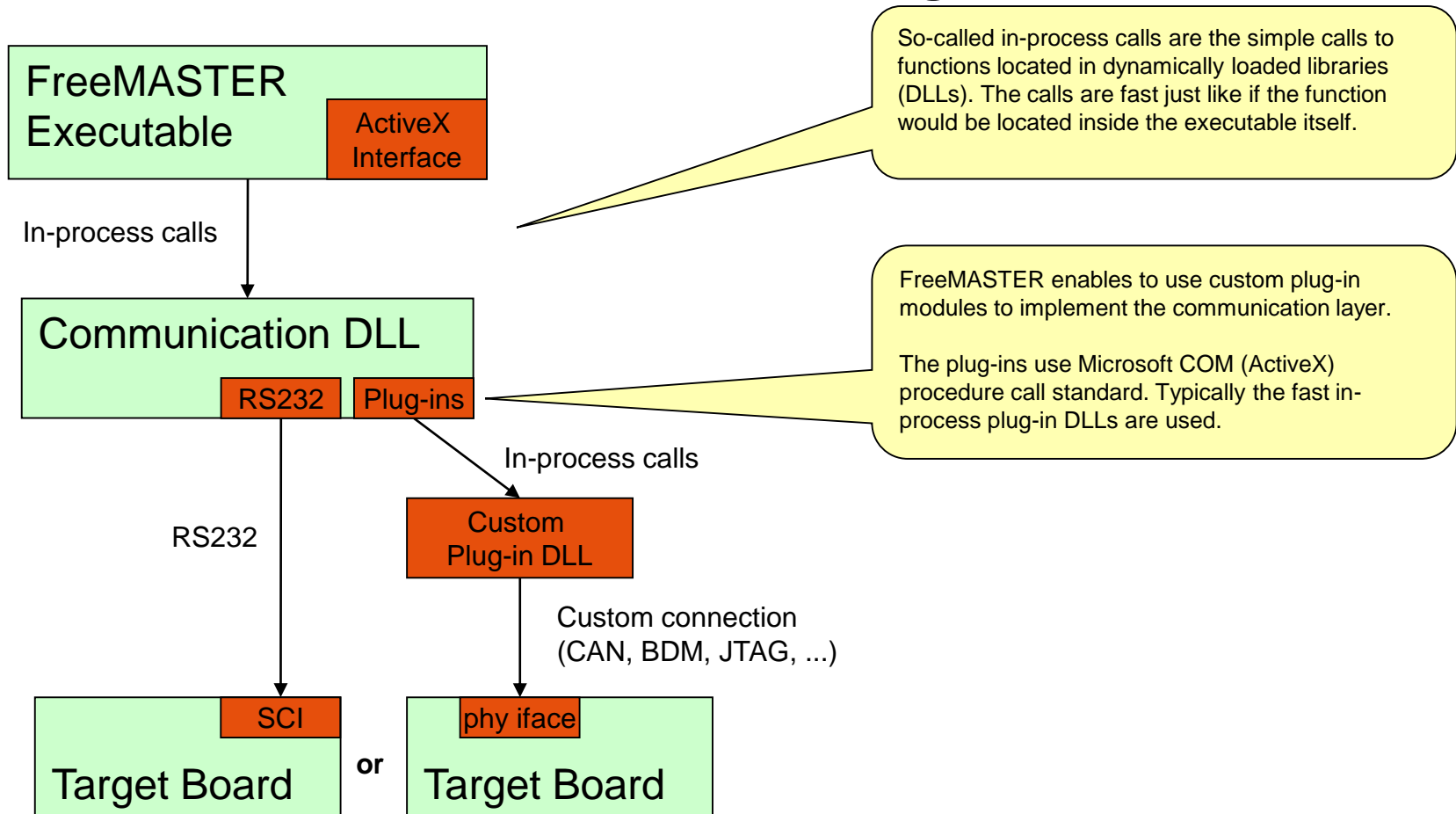


```
Dim a As McbPcm
```



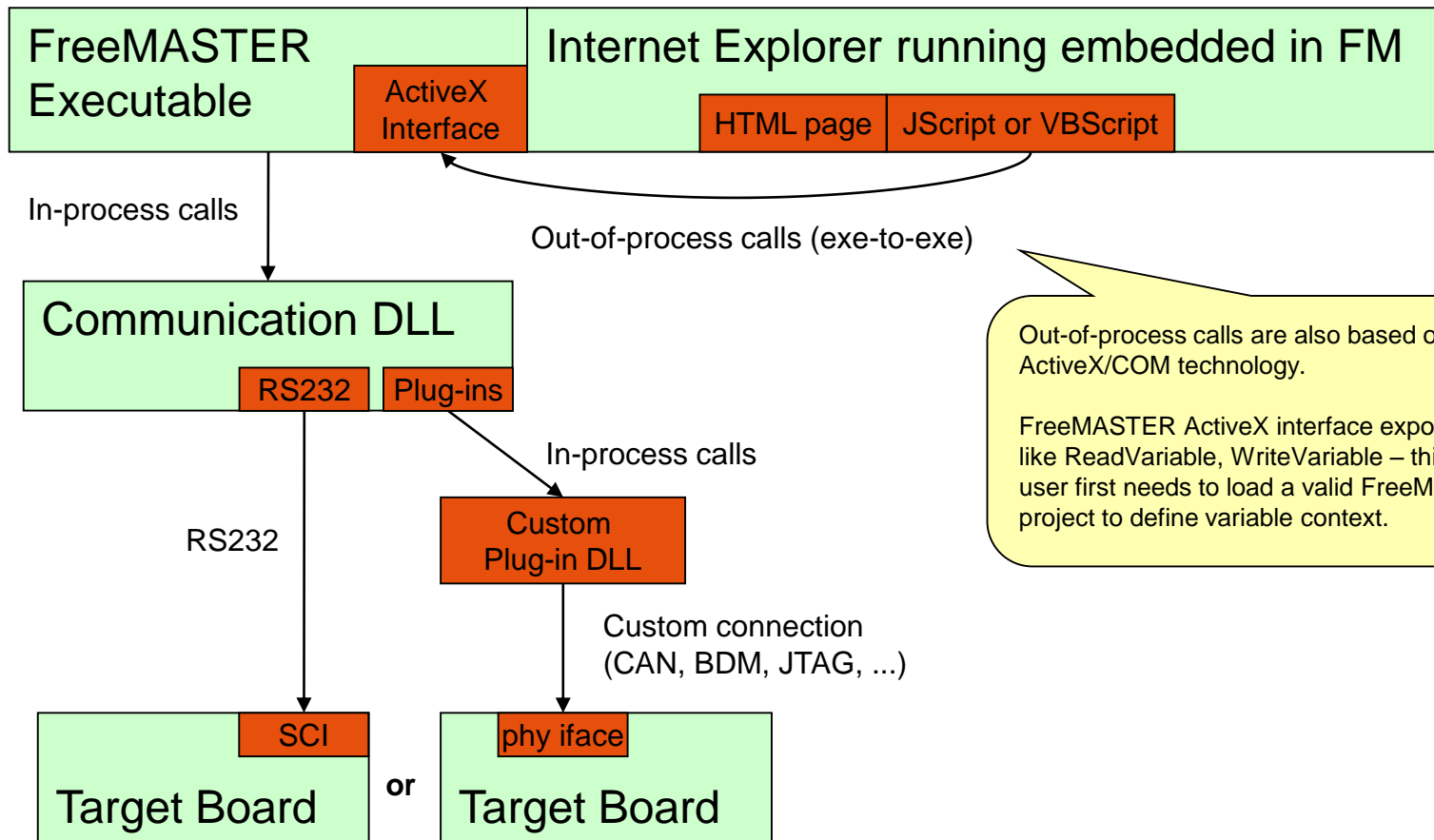
# FreeMASTER Internal Application Structure

## Basic FreeMASTER Communication Diagram



# FreeMASTER Internal Application Structure

## FreeMASTER Communication with HTML/JScript Pages

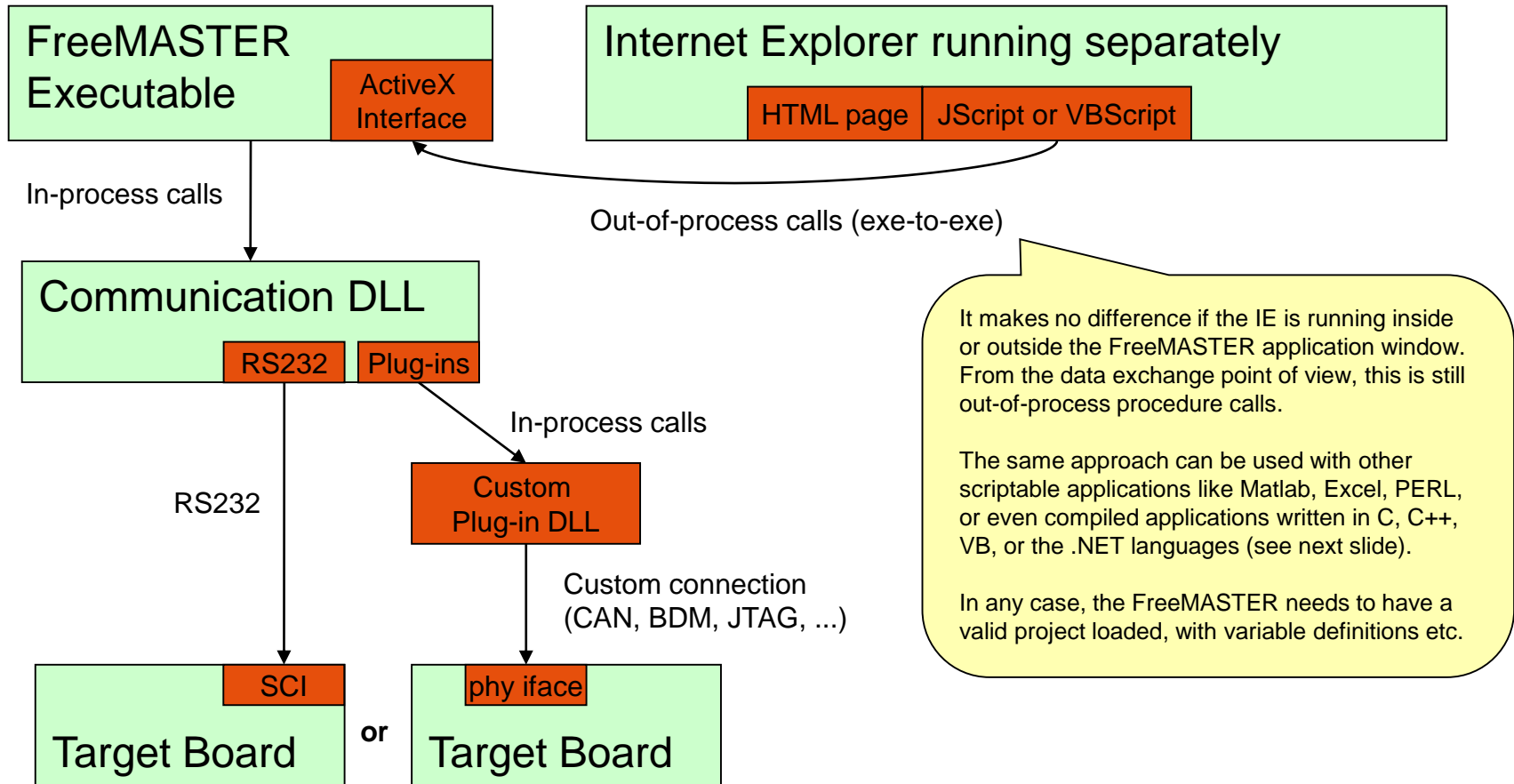


Out-of-process calls are also based on Microsoft ActiveX/COM technology.

FreeMASTER ActiveX interface exports methods like ReadVariable, WriteVariable – this means the user first needs to load a valid FreeMASTER project to define variable context.

# FreeMASTER Internal Application Structure

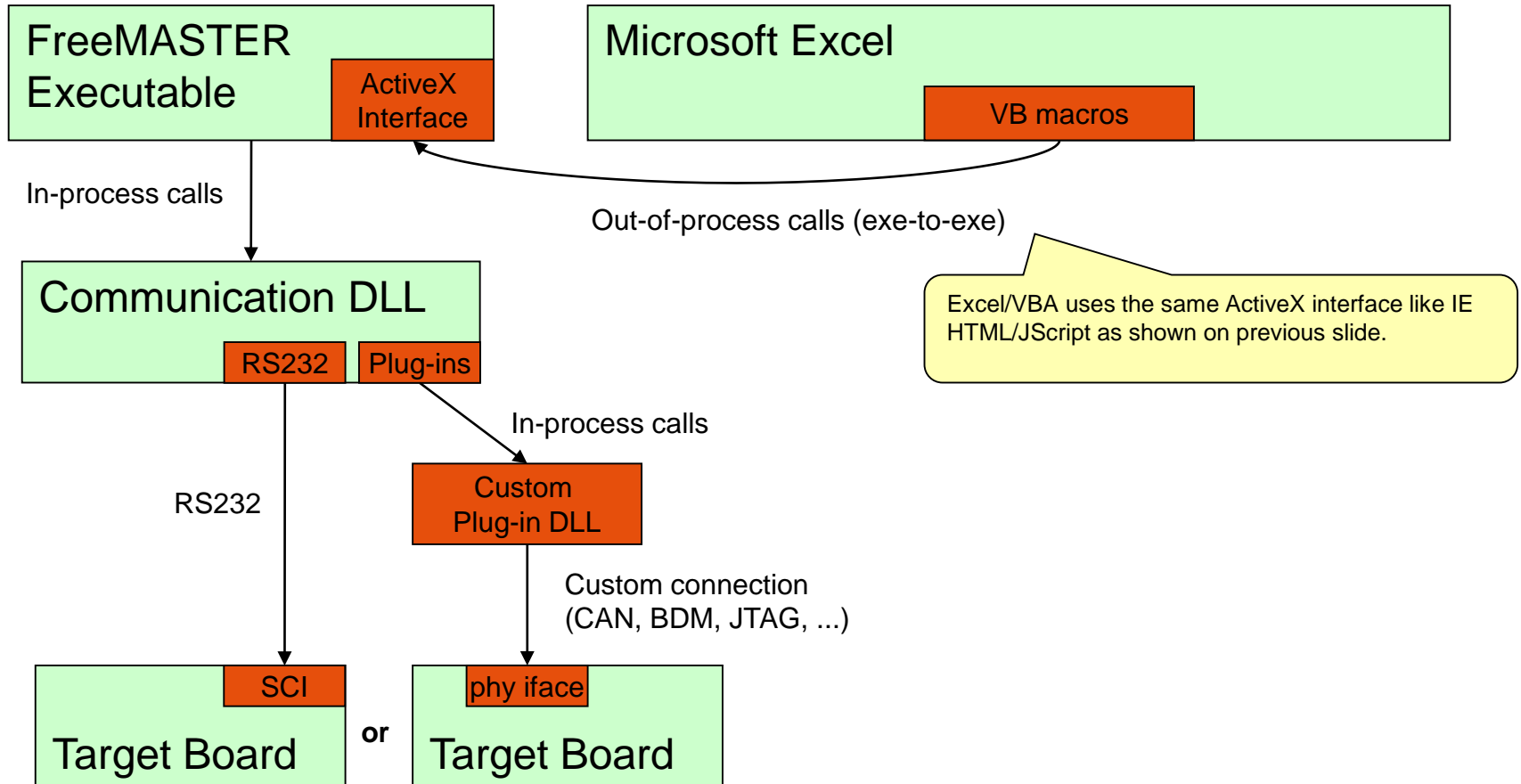
## Internet Explorer Running Separately (no difference)





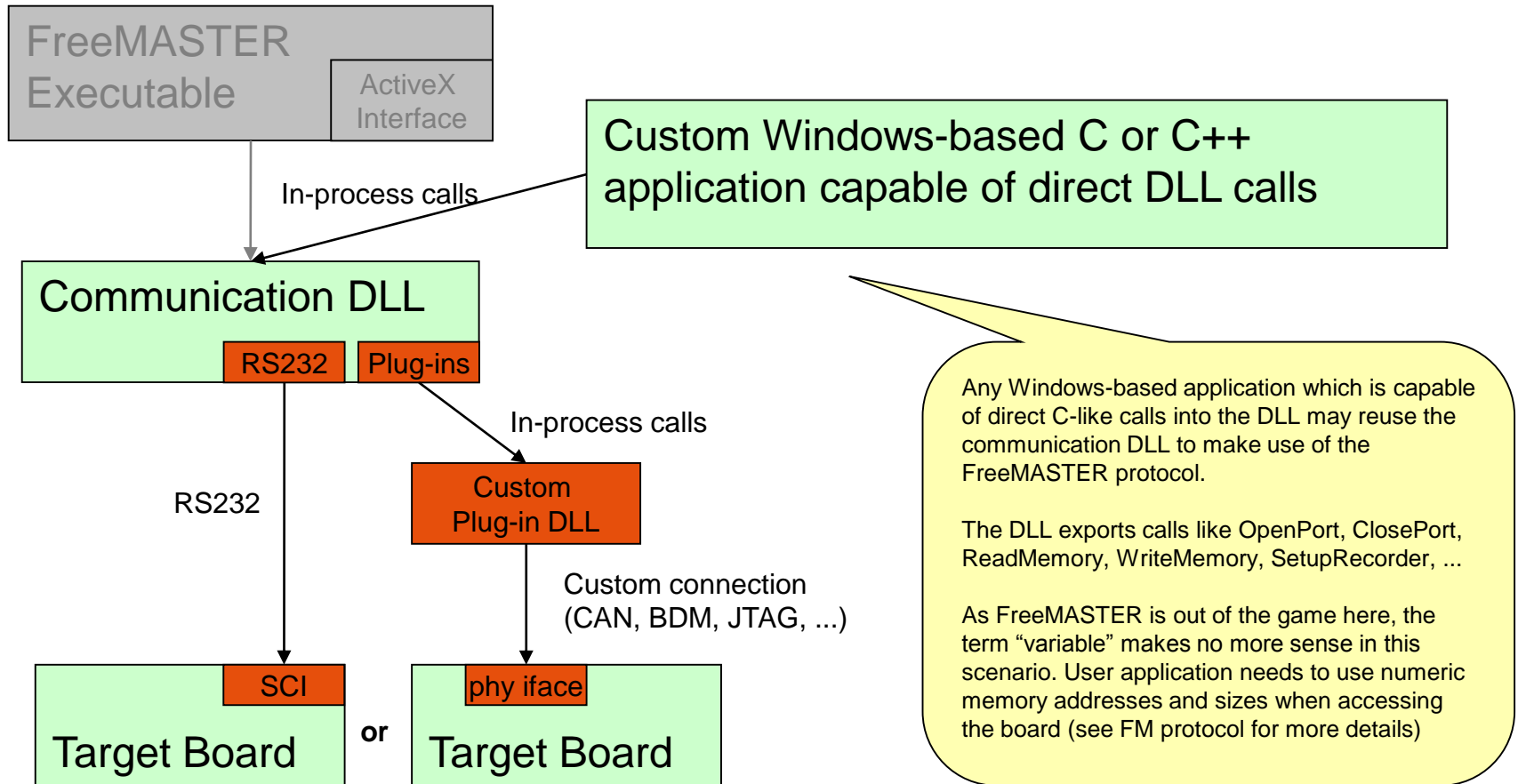
# FreeMASTER Internal Application Structure

## Excel (or other application) accessing FM ActiveX



# FreeMASTER Internal Application Structure

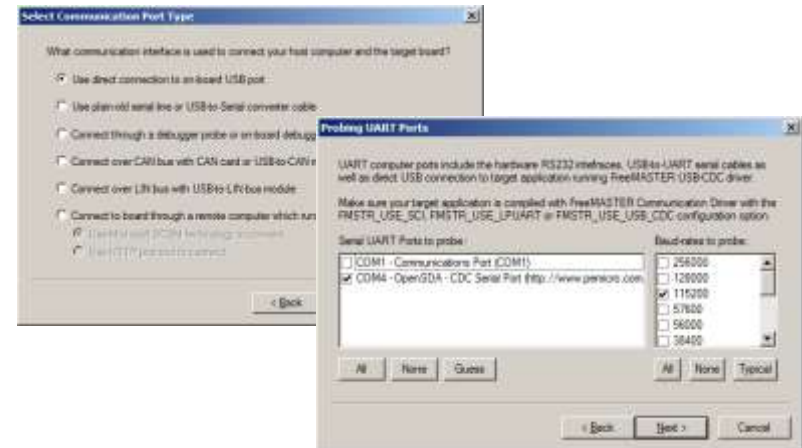
## Other Ways to Access Target Microprocessor: C, C++



# FreeMASTER — New in Version 2.0

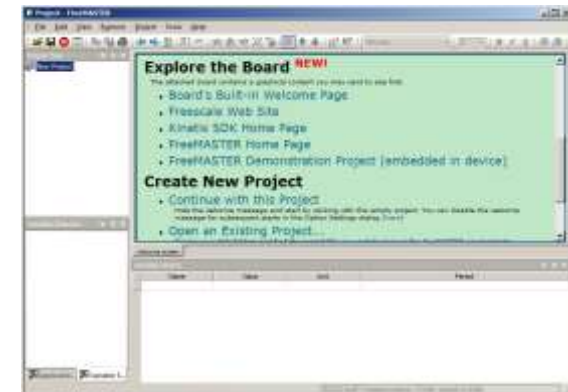
## • Connection Wizard

- Simplifies the connection to target
- Scans available COM ports and speeds
- Helps to select communication plug-in
- True plug & play experience when combined with Active Content

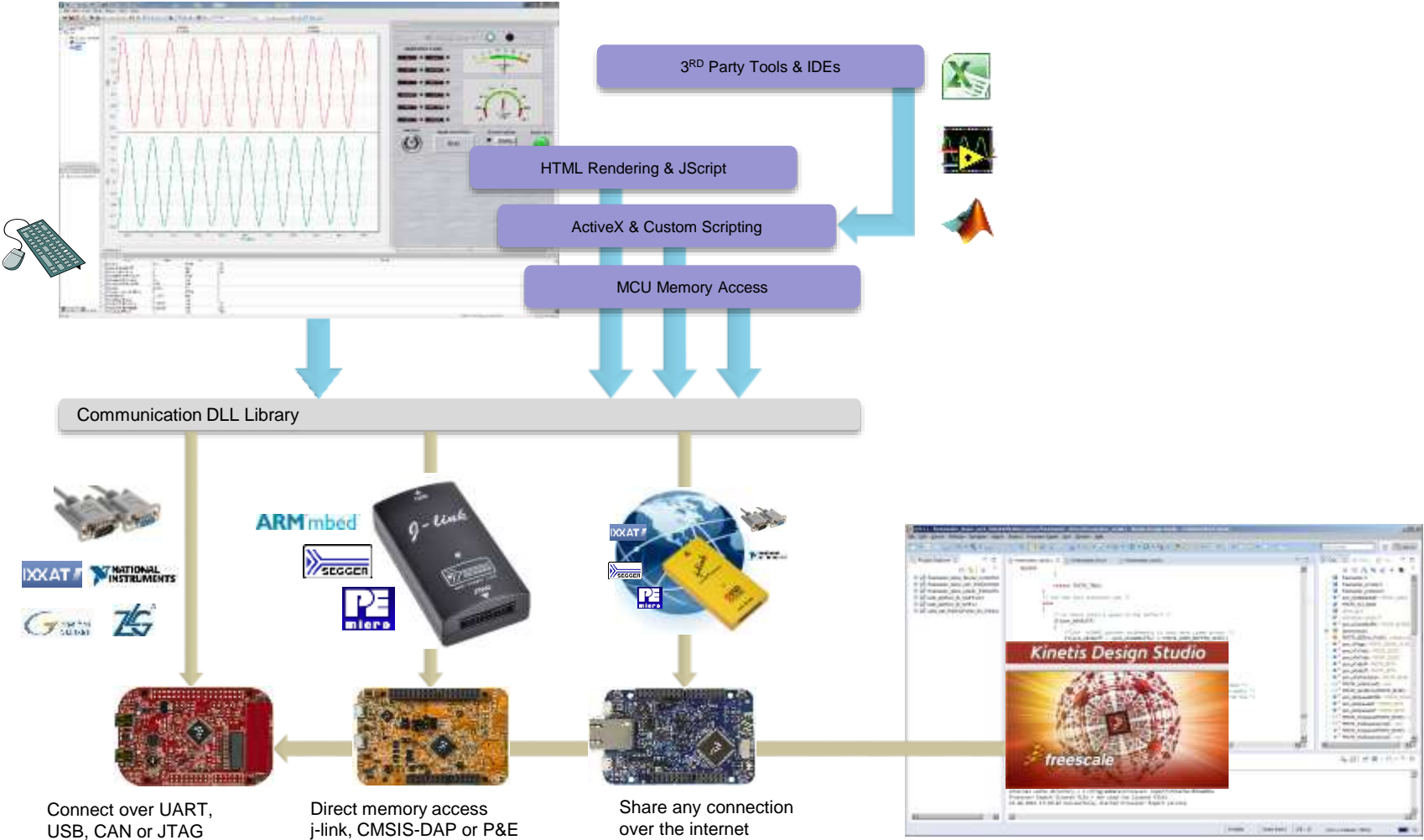


## • TSA with memory safety and **Active Content** stored in MCU Flash

- Enables variables to be defined without parsing ELF file
- Enables MCU to protect memory and enable access to “safe” variables only
- v2.0: TSA enables GUI files, pictures, scripts, project files and web links to be stored in MCU Flash
- User may “browse” available pages and project as soon as the board is connected



# Thank you - Questions?

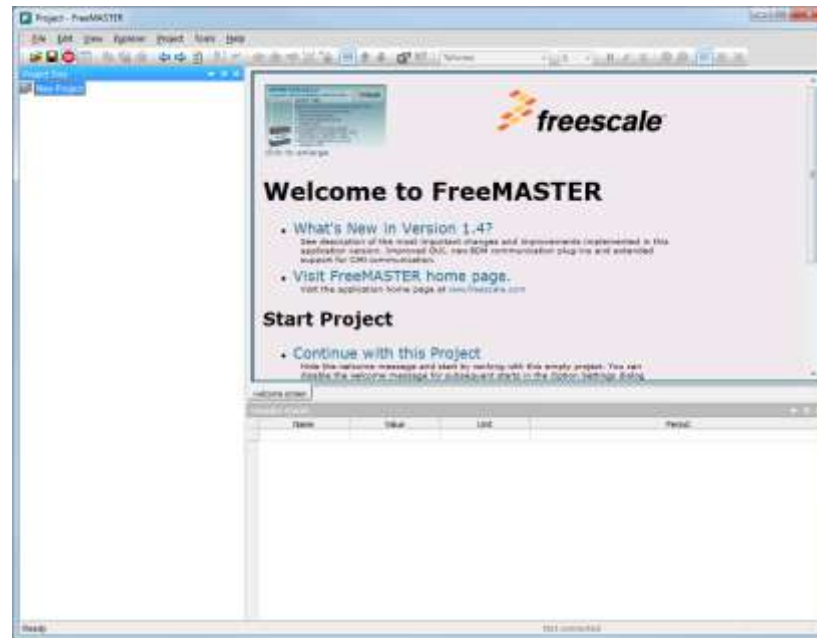


# Getting Started With Freemaster



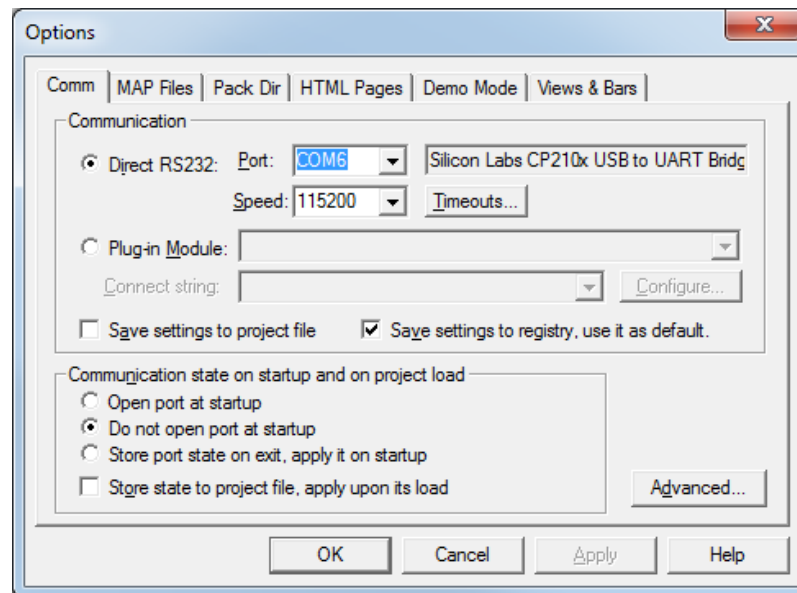
# Start FreeMASTER Interface

- From the Start Menu in Windows, go to
  - Start > All Programs > FreeMaster 1.4
- The FreeMASTER tool will start
  - ignore all the warnings and error messages, they are most probably caused by incorrectly assigned serial port.



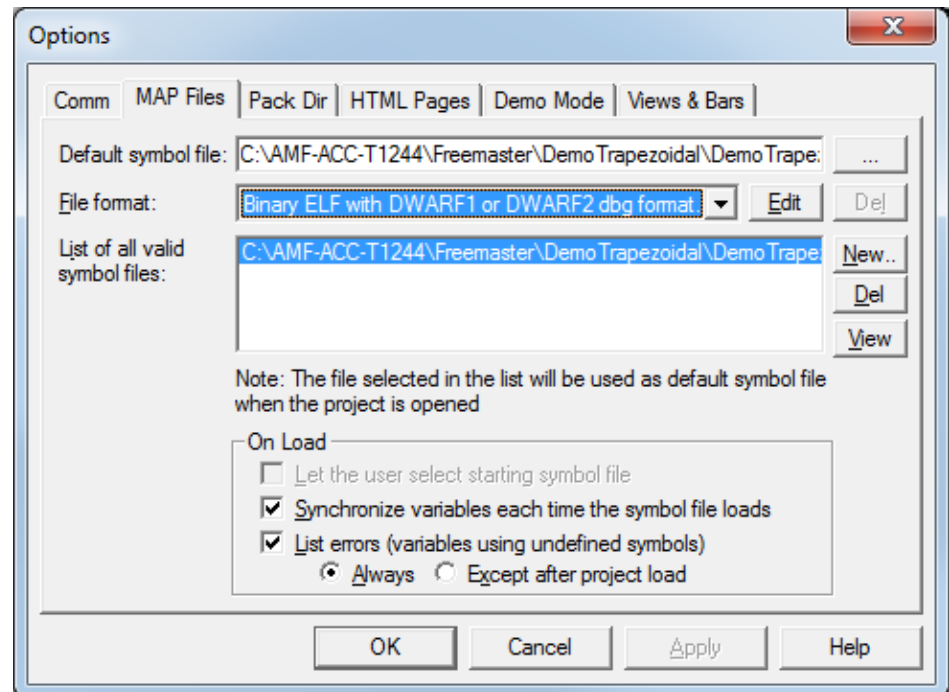
# FreeMaster – Configuring the Serial Port

- On the menu bar, go to Project > Options
- Select the correct COM port, with a speed setting of 115200 (this is the value we will use in the SCI initialization for the UART)



# FreeMaster – Loading the MAP file

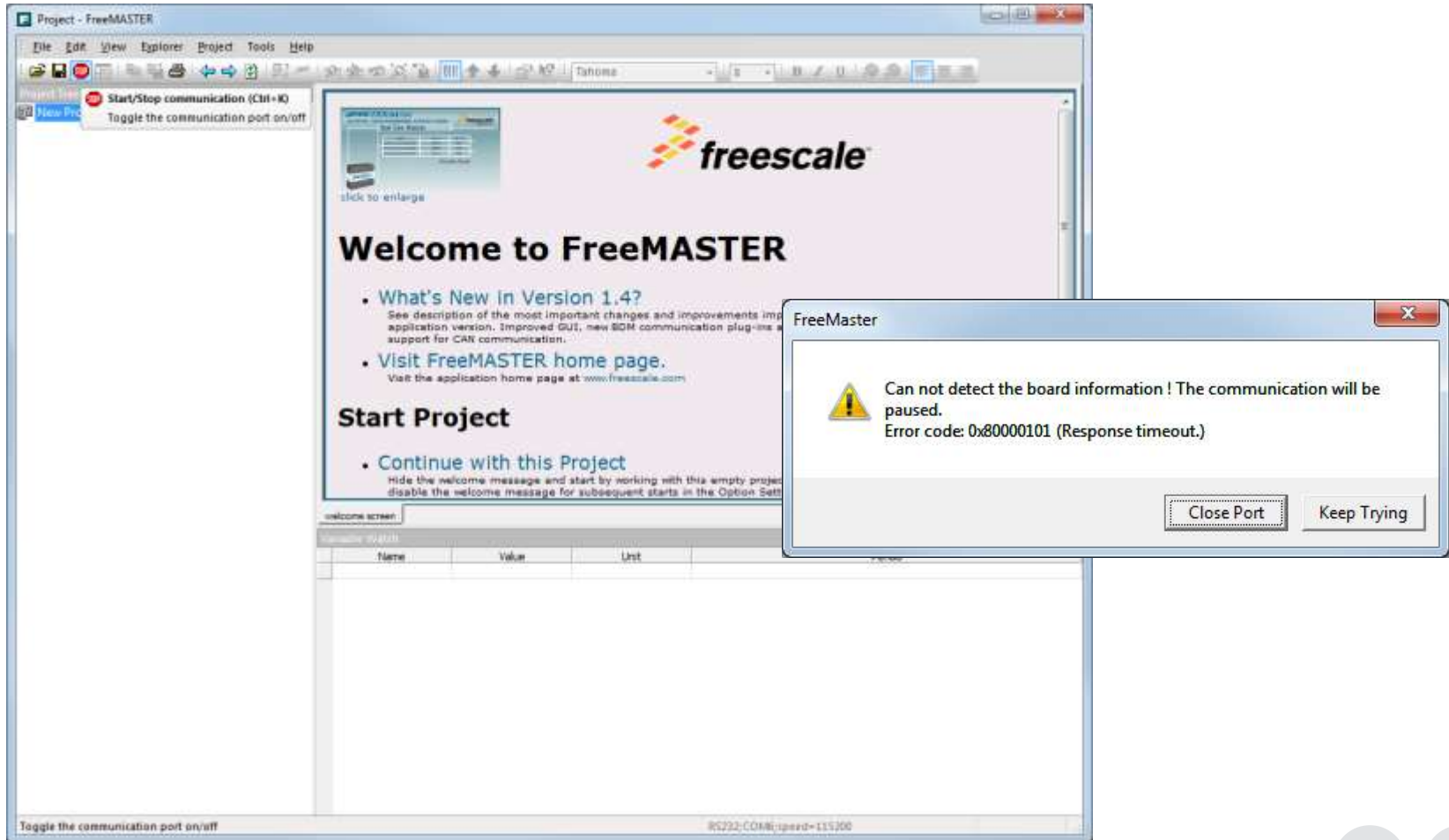
- From the options window, go to tab “MAP Files”
- Select the default symbol file:
  - Select the file  
“C:\AMF-ACC-T1244\Freemaster\DemoTrapezoidal\DemoTrapezoidalS12ZVM\DemoTrapezoidalS12ZVM.elf”
- Select the file format:
  - Binary ELF with DWARF1  
or DWARF2 dbg format
- Click OK





# FreeMaster – Start Communication

- Select Start/Stop Button to begin communication (you will see and error). Just select “Close Port”



# Need to integrate FreeMASTER into our Application

1. Copy the files under the **FreeMASTER Serial Communication Vxx\src\_common** where the DemoTrapezoidal application code is with no changes.
2. Copy the files under **FreeMASTER Serial Communication Vxx\src\_platforms\HC12** where the DemoTrapezoidal application code.
3. One file needs to change from the **FreeMASTER Serial Communication Vxx\src\_platforms\HC12** directory:
  - a) renamed freemaster\_cfg.h.example to freemaster\_cgh.h
  - b) Configure FreeMASTER by changing macro definitions in freemaster\_cgh.h (see following):

```

/*****
 * Select interrupt or poll-driven serial communication
 *****/

#define FMSTR_POLL_DRIVEN      1          /* no interrupt needed, polling only */
#define FMSTR_SCI_BASE        0x710     /* SCI1 base on S12ZVM */
#define FMSTR_USE_SCI         1          /* To select SCI communication interface */
#define FMSTR_USE_APPCMD      0          /* enable/disable App.Commands support */

```

# Need to integrate FreeMASTER into our Application (changes to mcd\_main.c)

```
/* Model's headers */
#include "DemoTrapezoidalS12ZVM.h"
#include "mc9s12zvm.h"
#include "mcd_s12z_compiler.h"
#include "tim_param.h"
#include "tim_s12zvm_library.h"
#include "clk_init.h"
#include "isr_lvl.h"
#include "freemaster.h"

void main(void)
{
    /* Initialize PLL clock */
    cmpu_s12zvm_init();                //Already initialized in the Boot
    isr_vect_init();

    /* System Timer Initialization */
    timer_s12zvm_base_init();
    timer_s12zvm_ch_init(0,0);

    /* SPI, SCI output pin routing */
    MODRR0 = MODRR0_SCI1RR_MASK; /* From Device Header file */

    /* Set SCI 1 baud rate divider*/
    SCI1BD = 434;

    /* PMF output pin routing */
    MODRR1 = 0;
```

# Need to integrate FreeMASTER into our Application (changes to mcd\_main.c)

```
/* PMF multiple timebase generators */
PMFCFG0_MTG = 1;

/* Initialize the processor. */
SYSTEM_INIT_TASK();

/* Attach SYSTEM_TASK to a timer or interrupt service routine with
 * period 0.001 seconds (the model's base sample time) here. The
 * call syntax for SYSTEM_TASK is SYSTEM_TASK();
 */

/* PTU Interrupt Enable */
PTUIEL = 0;
PTUIEH = 0;

/* TIM global enable */
TIM0TSCR1_TEN = 1;

/* Initialize Freemaster */
FMSTR_Init();

/* Enable interrupts */
Enable_Interrupts();
while (1) {
    FMSTR_Poll(); /* Poll for Freemaster Communication */
}
}
```

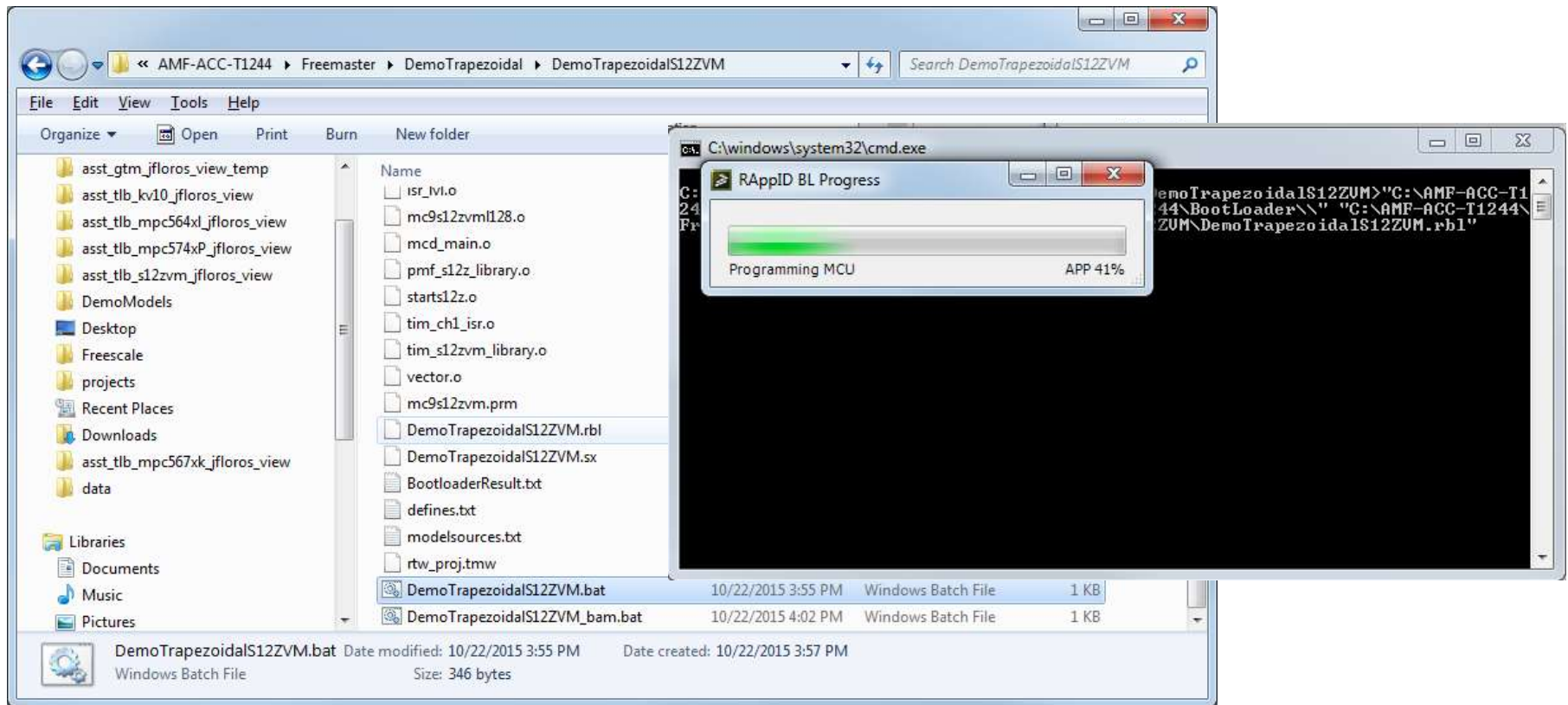
# Need to integrate FreeMASTER into our Application (changes to DemoTrapezoidalS12ZVM.mk)

```
# MEM_ALLOC      - Either RT_MALLOC or RT_STATIC indicating the style of the
#                 generated code. Statically allocated data is only useful
#                 for one instance of the model.
# COMPUTER      - Computer type. See the MATLAB computer command.
# BUILDARGS     - Options passed in at the command line.
# MULTITASKING  - yes (1) or no (0): Is solver mode multitasking
# MAT_FILE      - yes (1) or no (0): MAT file logging
# EXT_MODE      - yes (1) or no (0): Build for external mode
# TMW_EXTMODE_TESTING - yes (1) or no (0): Build ext_test.c for external mode
#                 testing.
# EXTMODE_TRANSPORT - Index of transport mechanism (e.g. tcpip, serial) for extmode
# EXTMODE_STATIC - yes (1) or no (0): Use static instead of dynamic mem alloc.
# EXTMODE_STATIC_SIZE - Size of static memory allocation buffer.

MODEL           = DemoTrapezoidalS12ZVM
MODULES         = clk_init.c freemaster_HC12.c freemaster_appcmd.c freemaster_protocol.c freemaster_rec.c freemaster_sco
#MODULES        = clk_init.c mc9s12zvm1128.c mcd main.c pmf s12z_library.c starts12z.c tim ch1_isr.c tim s12zvm_library
MAKEFILE        = DemoTrapezoidalS12ZVM.mk
ALT_MATLAB_ROOT = C:\PROGRA~2\MATLAB\R2013a
MATLAB_ROOT     = C:\Program Files (x86)\MATLAB\R2013a
$ FUNCTIONS     =
```

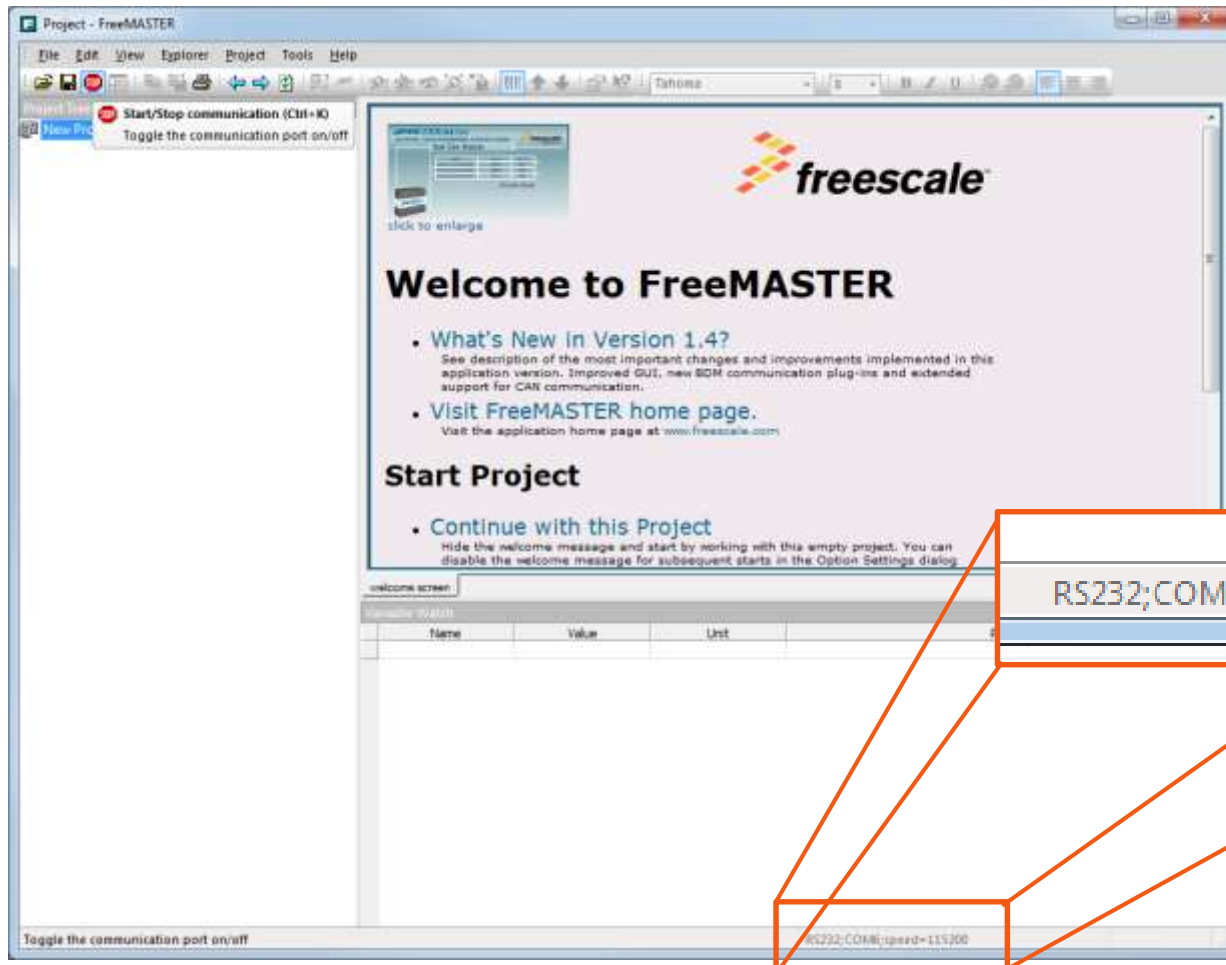
# Rebuild our Application code with the changes we made and reload software

1. Run DemoTrapezoidalS12ZVM.bat.
2. When build complete run DemoTrapezoidalS12ZVM\_bam.bat to download the code to the MCU.



# FreeMASTER – Start Communication

- Select Start/Stop Button to begin communication.



# FreeMASTER Interface

- In the FreeMASTER interface for “Empty Project” variable time is watched. This variable is also added to scope interface in order to be monitored in graphical representation.

The screenshot shows the FreeMASTER interface for a project named 'S12ZVM'. The interface includes a menu bar (File, Edit, View, Scope, Item, Project, Tools, Help), a toolbar, and a main workspace. On the left, a 'New Project' panel shows a tree view with 'time\_scope' selected. The main workspace is divided into two sections: an oscilloscope window and a watch window. The oscilloscope window displays a graph of the variable 'time' over time, with the y-axis labeled 'I/Axis' ranging from 0 to 60000 and the x-axis labeled 'Time [sec]' ranging from 12 to 22. The graph shows a sawtooth pattern with a sharp drop at approximately 14.5 seconds and 18.5 seconds. The watch window below the graph shows a table with the following data:

Name	Value	Unit	Period
time	63562	DEC	0

At the bottom of the interface, the status bar shows 'Done', 'RS232;COM4;speed=9600', and 'Scope Running'. Annotations with arrows point to various parts of the interface:

- 'Start/Stop serial communication with target' points to the 'STOP' button in the toolbar.
- 'Scope selector' points to the 'time\_scope' item in the project tree.
- 'Visualization of variable "time" in scope' points to the oscilloscope graph.
- 'Variable "time" in watch window' points to the 'time' entry in the watch window table.



# FreeMASTER Interface

Start/Stop serial communication with target

Scope selector

Visualization of variable in scope

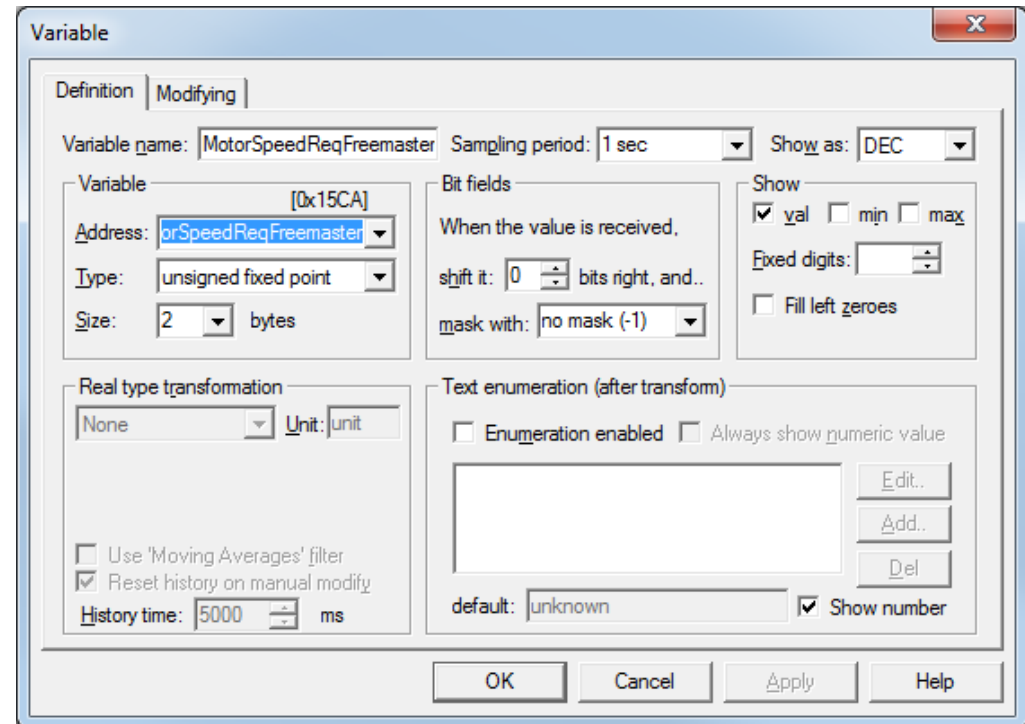
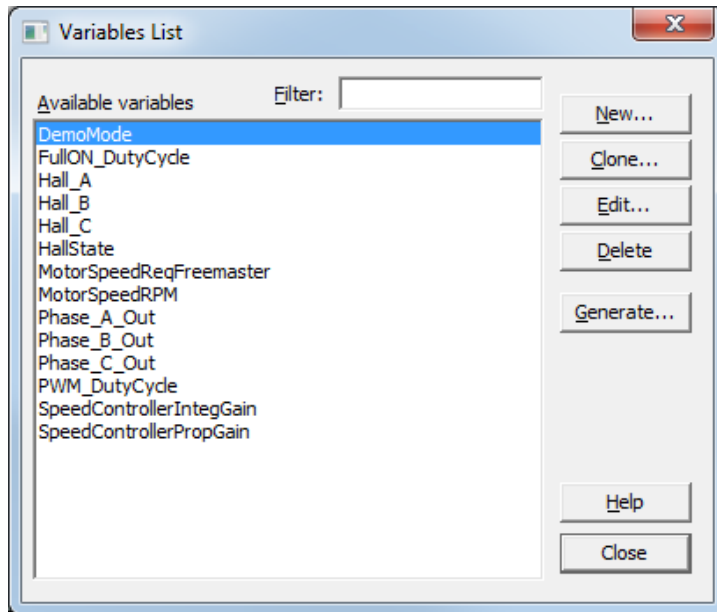
Variable in watch window

The screenshot displays the FreeMASTER software interface. The main window is titled "S12ZVM\_Lab2\_FRMSTR.pmp - FreeMASTER". It features a menu bar (File, Edit, View, Scope, Item, Project, Tools, Help) and a toolbar with various icons. On the left, there is a "Scope selector" panel with options for "New Project" and "Potentiometer". The central area shows an oscilloscope plot of the variable "pot\_value" over time. The y-axis is labeled "IAxis" and ranges from 0 to 5000. The x-axis is labeled "Time [sec]" and ranges from 20 to 30. The plot shows a green line that starts at approximately 2500, drops to a minimum of about 200 at 25.5 seconds, and then rises to a final value of about 4000 at 30 seconds. Below the plot is a table for the "algorithm block description" with the "oscilloscope" block selected. The table has columns for Name, Value, Unit, and Period. The "pot\_value" variable is listed with a value of 4093 and a period of 100. At the bottom of the window, there are status indicators: "Ready", "Not connected", and "Scope Running".

Name	Value	Unit	Period
pot_value	4093	DEC	100

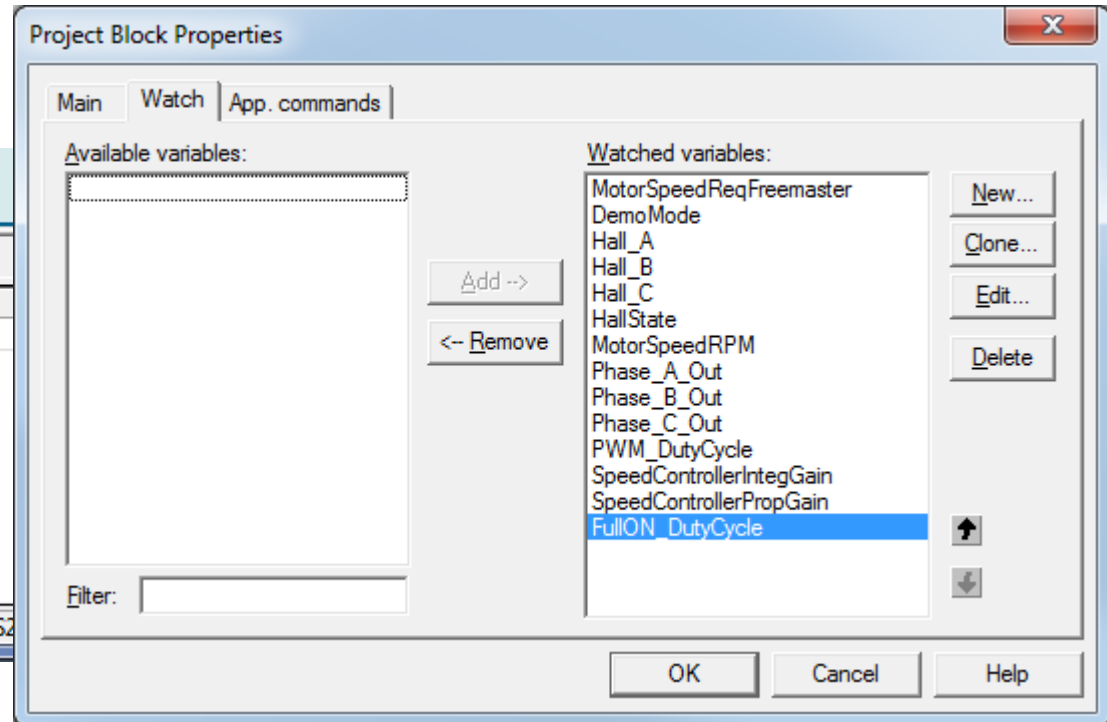
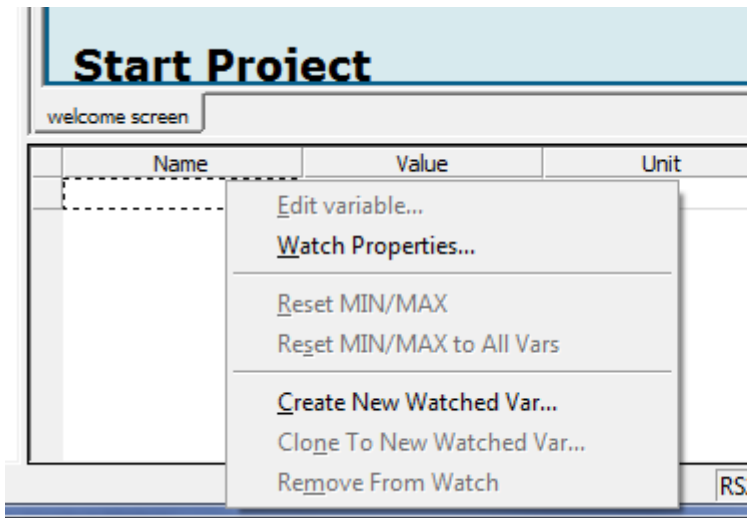
# Adding Variables

- On the menu bar, go to Project > Variables
- When the window appears, select “New”
- Type the variable name in the “Address” field
- Click on “OK” and Repeat until finished with all needed variables.
- Close “Variables list” window



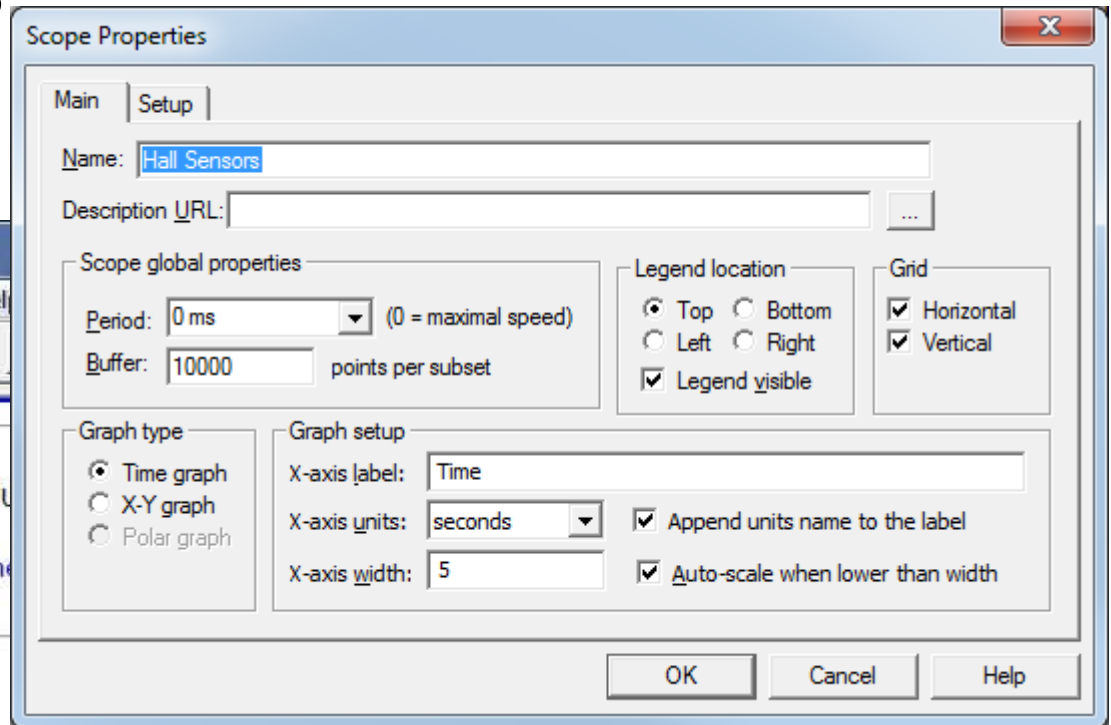
# Adding variables to the Watch List

- Right click into “watch” area and select “Watch Properties”
- Switch to tab “Watch” in Project Block Properties
- Select the variables to watch and click on “Add”



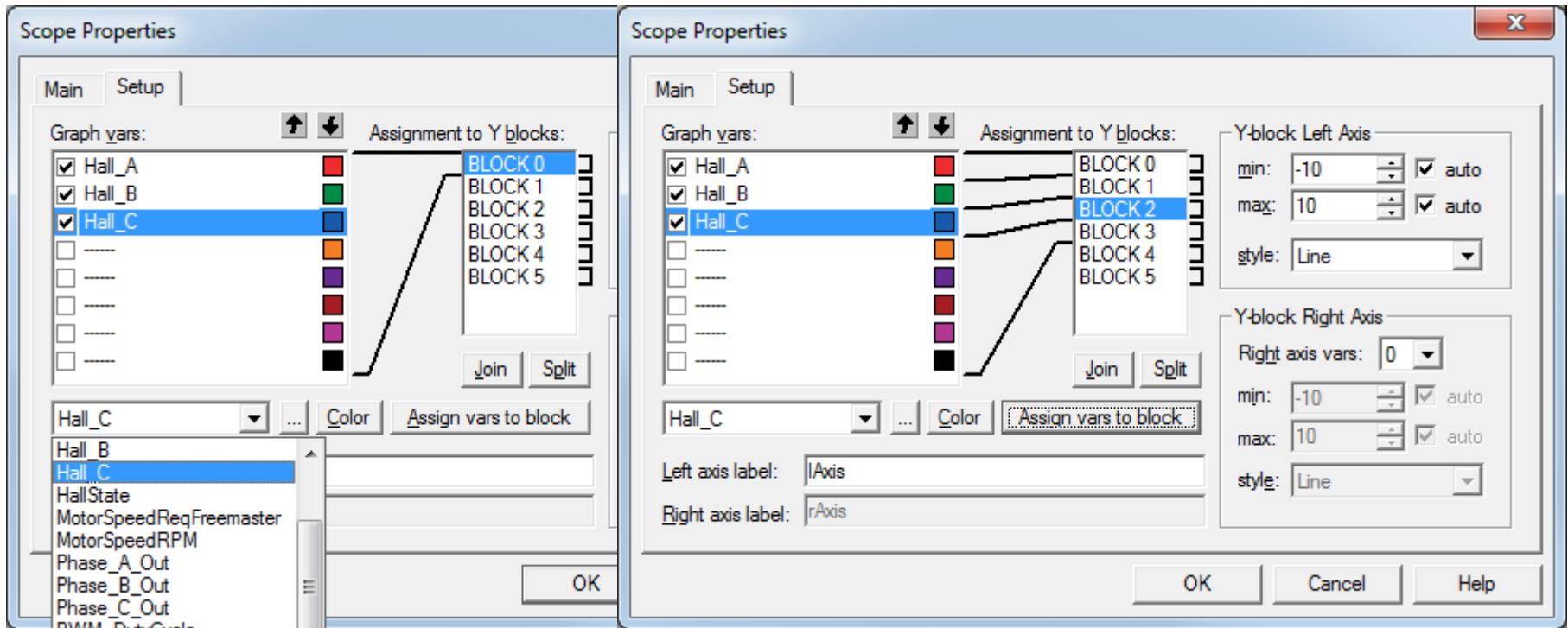
# Adding a Scope

- Right-click on New Project and select the option “Create Scope”
- Define a name for the scope
- Change Period to 0ms, and Buffer to 10000 points per subset
- Change X-axis width to 5



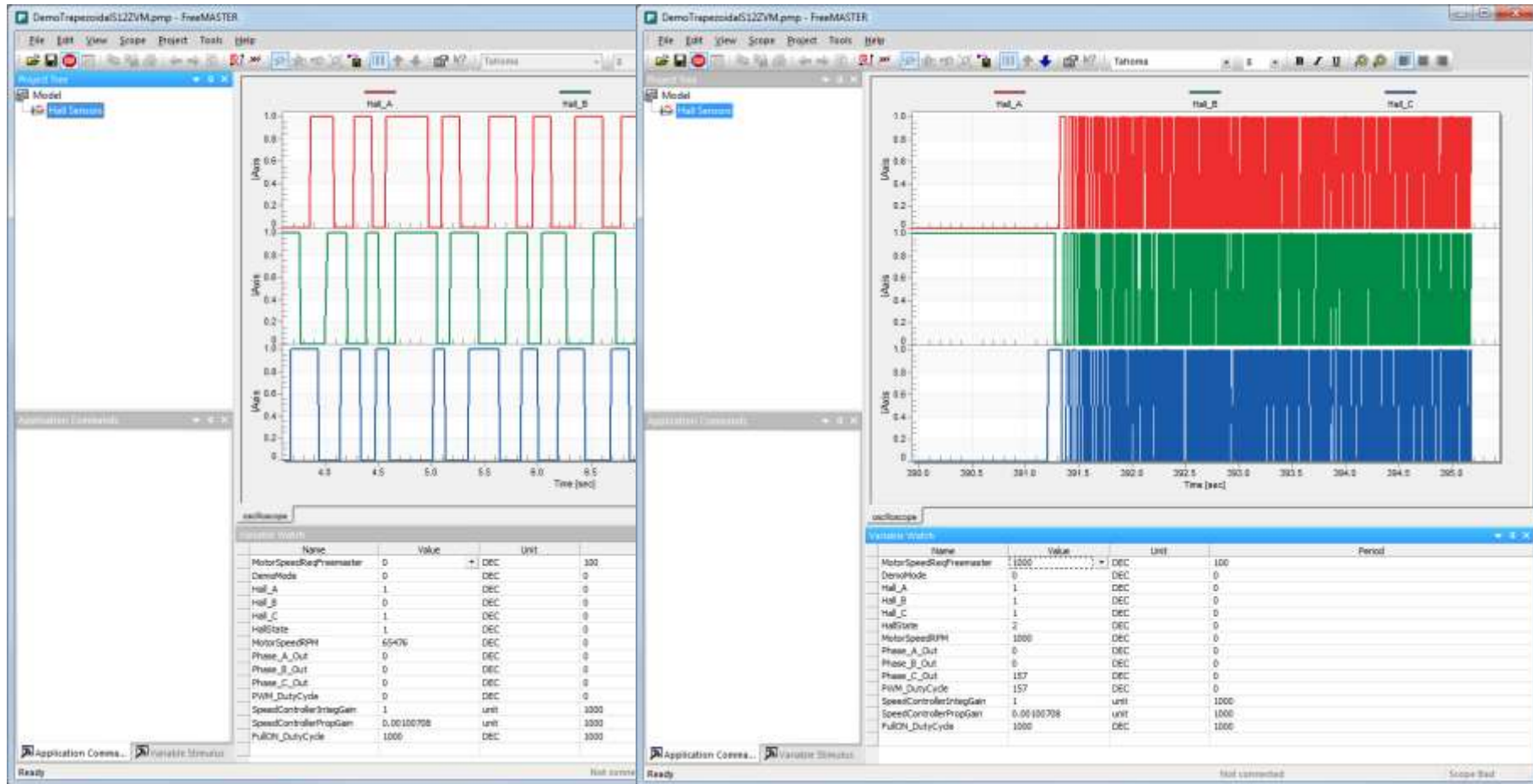
# Adding a Scope (Hall Sensors)

- Add Hall\_A BLOCK 0, Hall\_B to BLOCK 1, & Hall\_C to BLOCK 2
- Assign a variable(s) to BLOCK by selecting variables and clicking “Assign vars to block” button.
- Click “OK”



# Adding a Scope (Hall Sensors)

- Spin the motor by hand or set the speed to 1000 RPM



# Adding a Scope (MotorSpeedRPM)

The image displays two overlapping screenshots of the 'Scope Properties' dialog box in a software application.

**Top Screenshot (Main Tab):**

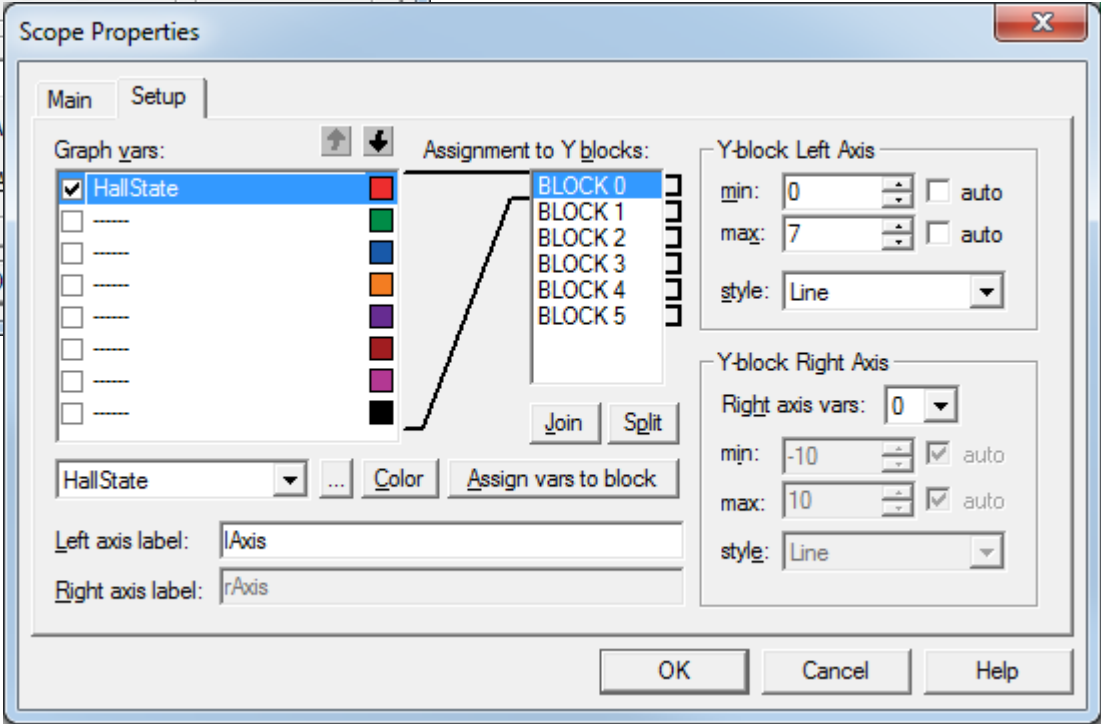
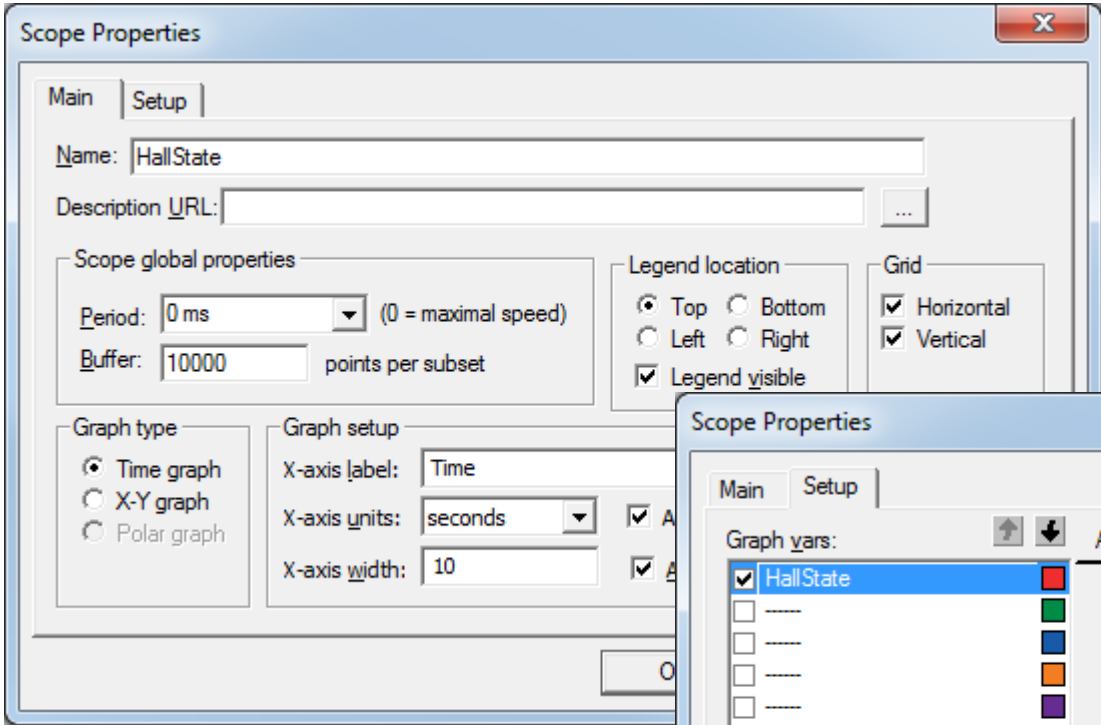
- Name:** MotorSpeedRPM
- Description URL:** [Empty field]
- Scope global properties:**
  - Period:** 0 ms (0 = maximal speed)
  - Buffer:** 10000 points per subset
- Legend location:**
  - Top  Bottom
  - Left  Right
  - Legend visible
- Grid:**
  - Horizontal
  - Vertical
- Graph type:**
  - Time graph
  - X-Y graph
  - Polar graph
- Graph setup:**
  - X-axis label:** Time
  - X-axis units:** seconds
  - X-axis width:** 10

**Bottom Screenshot (Setup Tab):**

- Graph vars:**
  - MotorSpeedRPM (Red)
  - MotorSpeedReqFreemaster (Green)
  - PWM\_DutyCycle (Blue)
  - [Empty]
  - [Empty]
  - [Empty]
  - [Empty]
  - [Empty]
  - [Empty]
- Assignment to Y blocks:**
  - BLOCK 0 (Selected)
  - BLOCK 1
  - BLOCK 2
  - BLOCK 3
  - BLOCK 4
  - BLOCK 5
- Y-block Left Axis:**
  - min: -4000 (auto)
  - max: 4000 (auto)
  - style: Line
- Y-block Right Axis:**
  - Right axis vars: 0
  - min: -10 (auto)
  - max: 10 (auto)
  - style: Line
- Buttons:** Join, Split, Color, Assign vars to block
- Axis Labels:** Left axis label: lAxis, Right axis label: rAxis
- Buttons:** OK, Cancel, Help



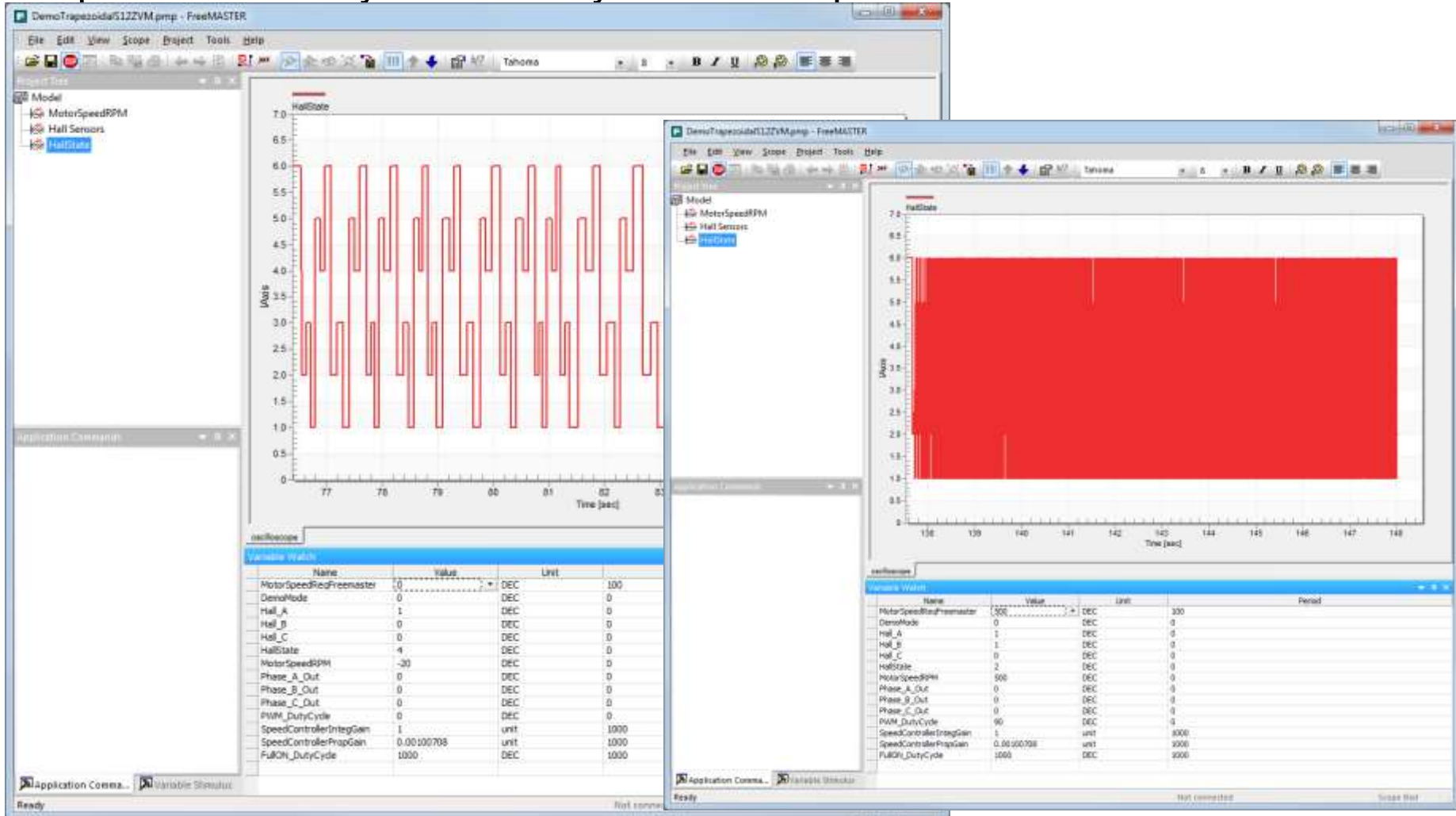
# Adding a Scope (HallState)





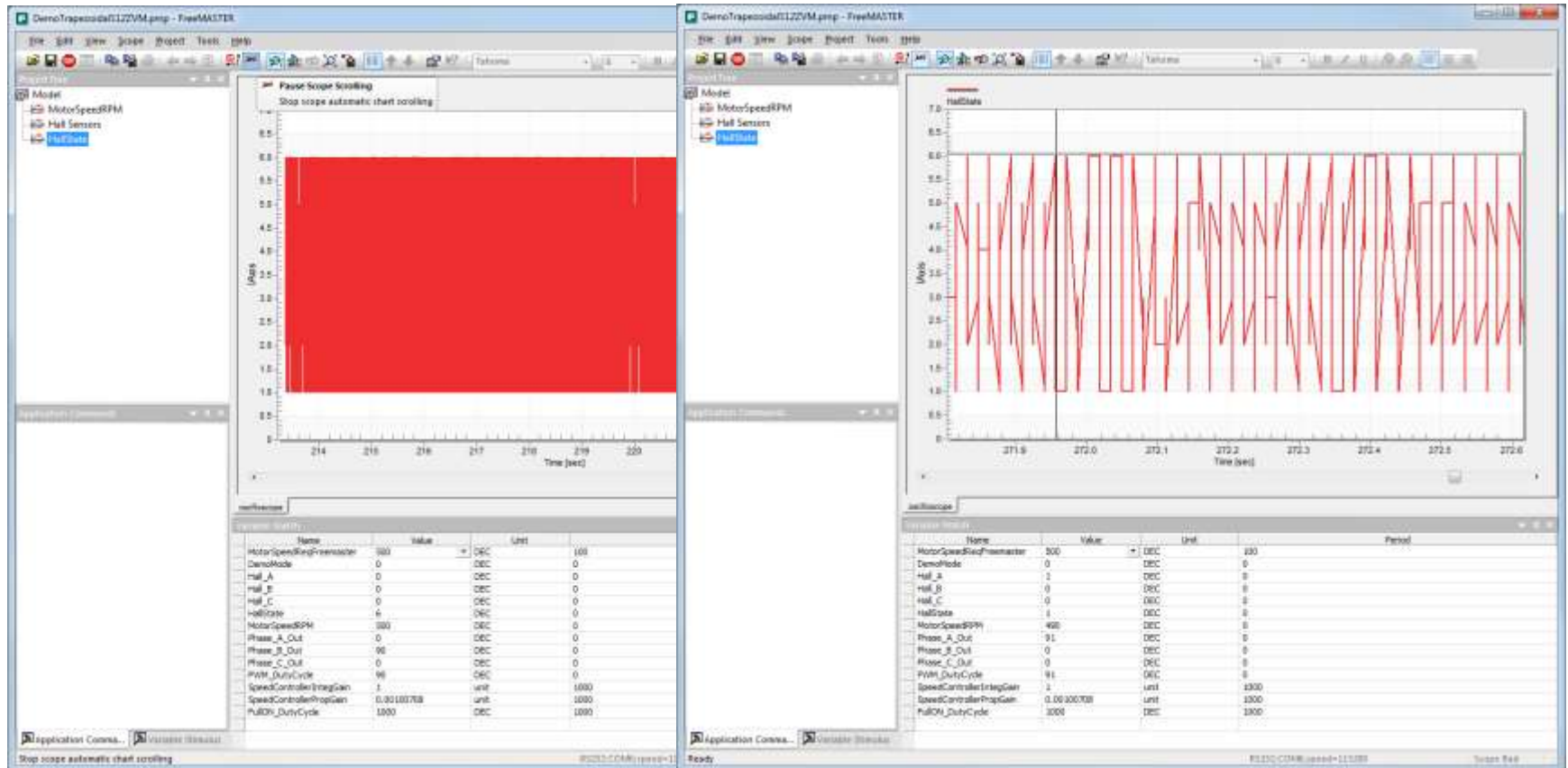
# Adding a Scope (HallState)

- Spin the motor by hand slowly or set the speed to 500 RPM



# Adding a Scope (HallState)

- For the 500 RPM rate you can zoom it to get more detail. Pause scope recording and use the zoom. You can see that because the signal is changing too fast so that polling aliases the signal.



# Need to integrate FreeMASTER Data Recording into our Application to avoid aliasing (changes to freemaster\_cfg.h)

```

/*****
 * Recorder support
 *****/
#define FMSTR_USE_RECORDER      1          /* enable/disable recorder support */
#define FMSTR_MAX_REC_VARS     8          /* max. number of recorder variables (2..8) */
#define FMSTR_REC_OWNBUFF     0          /* use user-allocated rec. buffer (1=yes) */

/* built-in recorder buffer (use when FMSTR_REC_OWNBUFF is 0) */
#define FMSTR_REC_BUFF_SIZE    1024      /* built-in buffer size */

```

# Need to integrate FreeMASTER Data Recording into our Application to avoid aliasing (changes to DemoTrapezoidalS12ZVM.c)

```
/* Output and update for function-call system: '<Root>/HallSensorInputs' */
void DemoTrapezoida_HallSensorInputs(void)
{
    uint8_T rtb_Gain1;
    uint8_T rtb_Gain2;

    /* S-Function (gpio_s12zvm_input): '<S3>/Digital_Input' */
    Hall_A = PTIT_PTIT1;          /* GPI Pin Data Input Registers */

    /* Gain: '<S3>/Gain1' incorporates:
     * DataTypeConversion: '<S3>/Data Type Conversion5'
     */
    rtb_Gain1 = (uint8_T)(Hall_A << 2);

    /* S-Function (gpio_s12zvm_input): '<S3>/Digital_Input1' */
    Hall_B = PTIT_PTIT2;          /* GPI Pin Data Input Registers */

    /* Gain: '<S3>/Gain2' incorporates:
     * DataTypeConversion: '<S3>/Data Type Conversion2'
     */
    rtb_Gain2 = (uint8_T)(Hall_B << 1);

    /* S-Function (gpio_s12zvm_input): '<S3>/Digital_Input2' */
    Hall_C = PTIT_PTIT3;          /* GPI Pin Data Input Registers */

    /* Sum: '<S3>/Sum' incorporates:
     * DataTypeConversion: '<S3>/Data Type Conversion6'
     */
    HallState = (uint8_T)((((uint16_T)rtb_Gain1 + rtb_Gain2) + Hall_C);

    /* S-Function (fcncallgen): '<S3>/Function-Call Generator' incorporates:
     * SubSystem: '<Root>/TrapezoidalControl'
     */
    DemoTrapezoi_TrapezoidalControl();

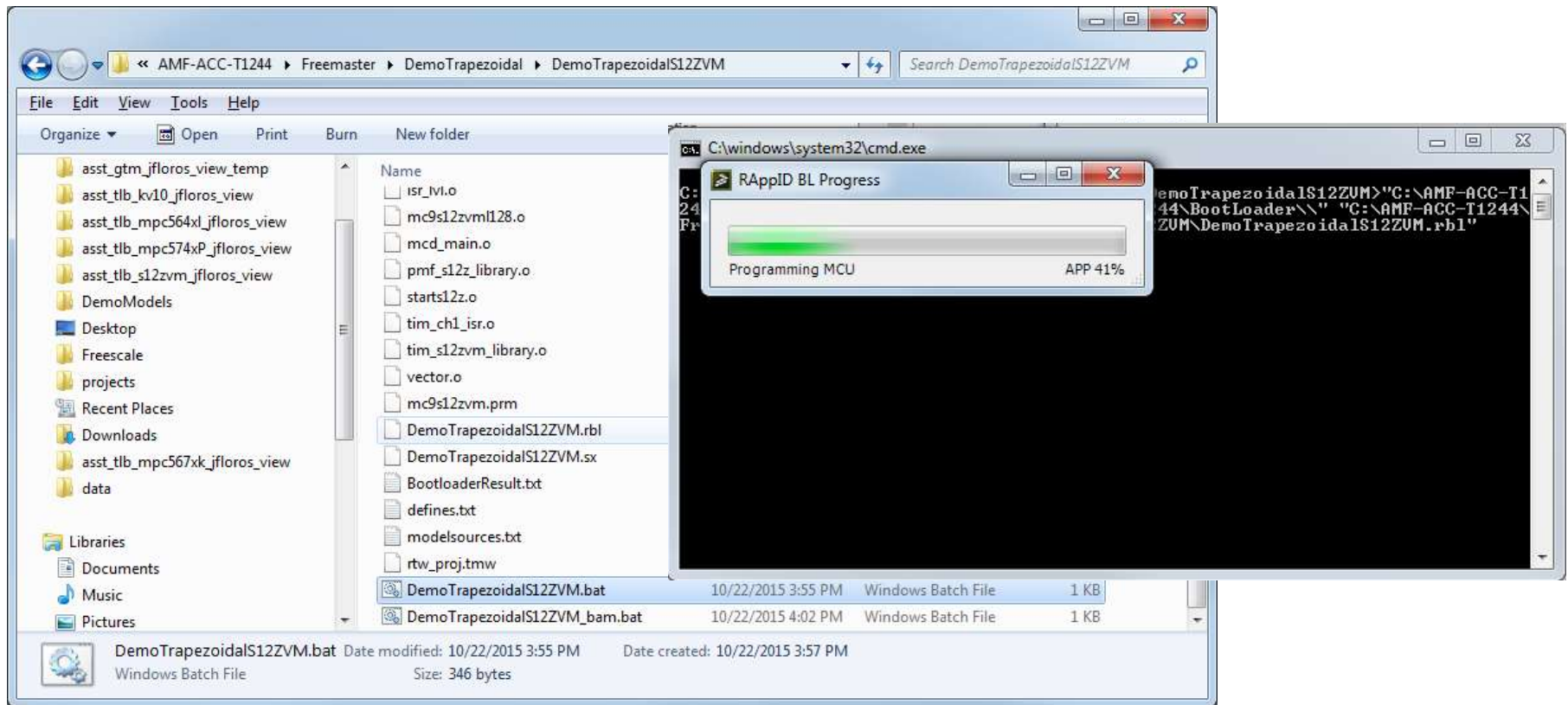
    /* S-Function (gpio_s12zvm_output): '<S3>/Digital_Output' */
    PTS_PTSS = Hall_A;           /* Pin Data Output Signal Update PCR[8] */

    /* S-Function (gpio_s12zvm_output): '<S3>/Digital_Output3' */
    PTP_PTP0 = TRUE;             /* Pin Data Output Signal Update PCR[61] */

    /* FreeMaster_Data_Recorder */
    FMSTR_Recorder();
}
```

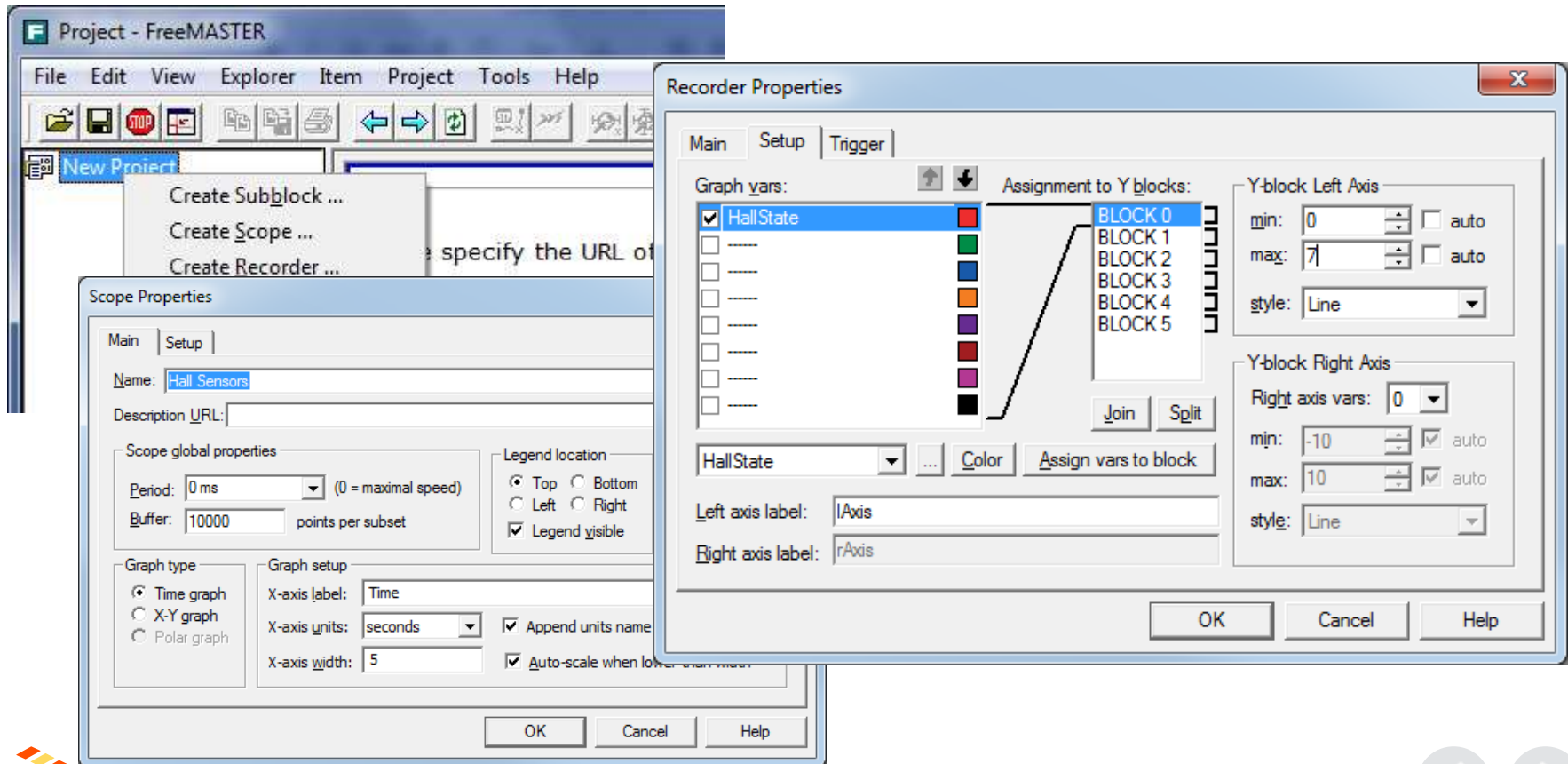
# Rebuild our Application code with the changes we made and reload software

1. Run DemoTrapezoidalS12ZVM.bat.
2. When build complete run DemoTrapezoidalS12ZVM\_bam.bat to download the code to the MCU.



# Adding a Recorder

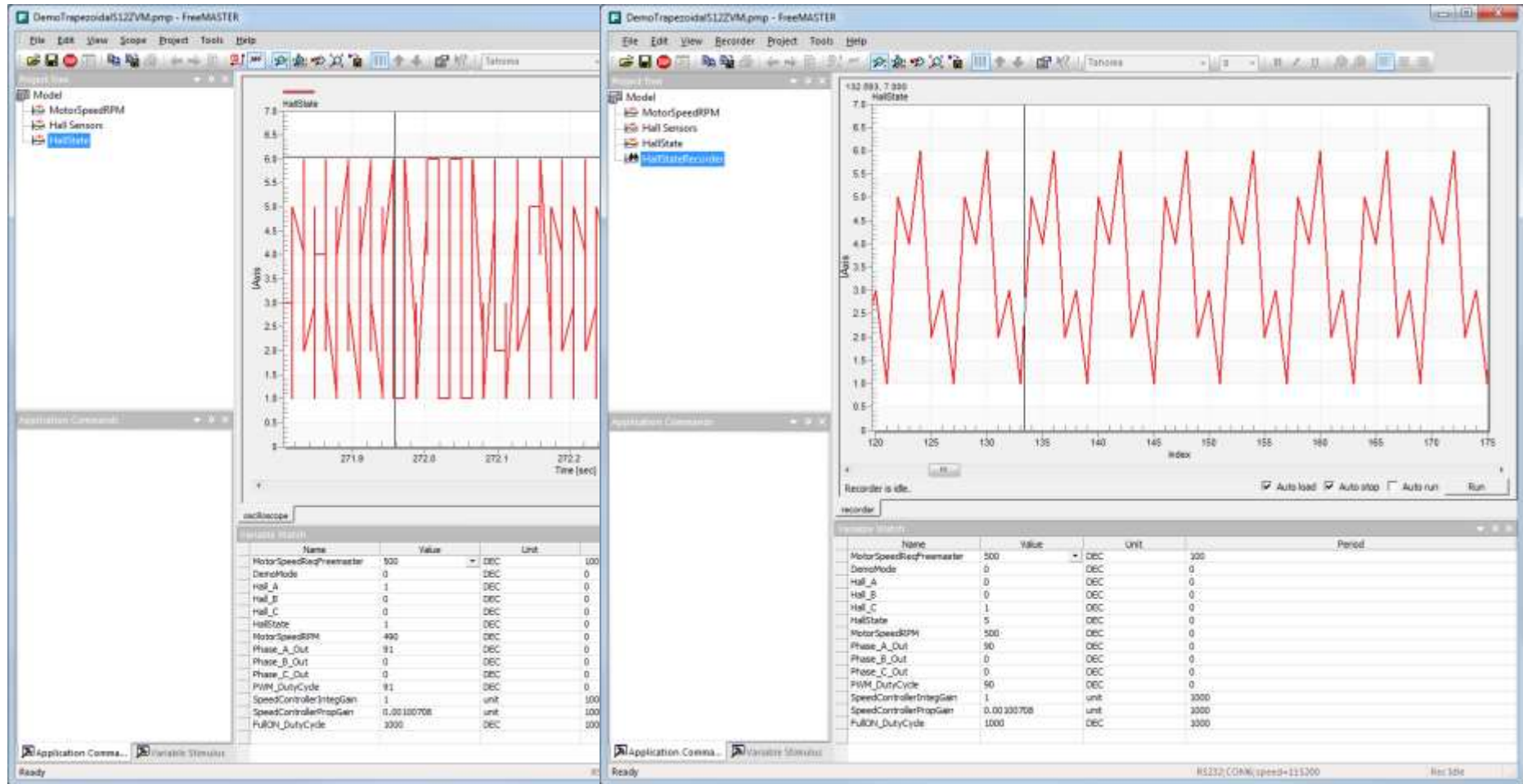
- Right-click on New Project and select the option “Create Recorder”
- Define a name for the Recorder
- Change Recorded samples to 1000 points





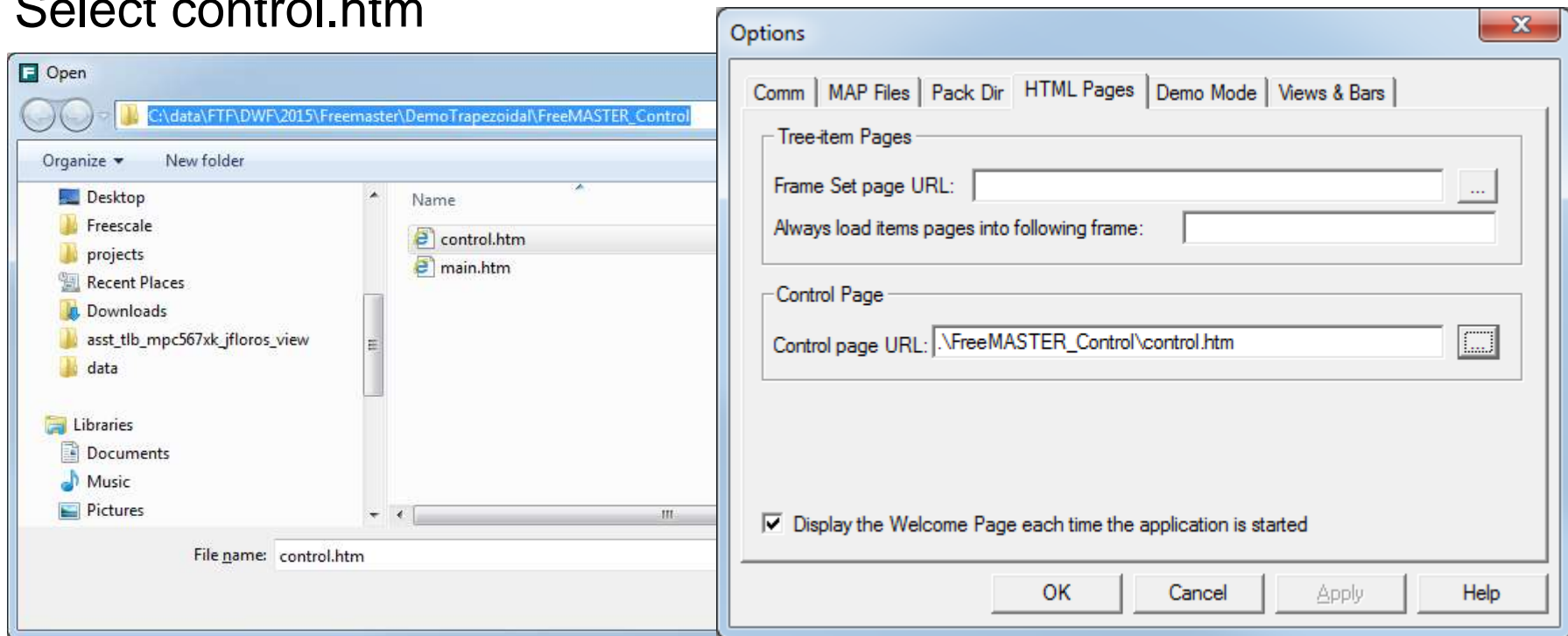
# Adding a Scope (HallState)

- Set 500 RPM and select HallStateRecorder and wait for data. No longer aliasing the data.



# Adding to Freemaster Project Setup Control Page URL

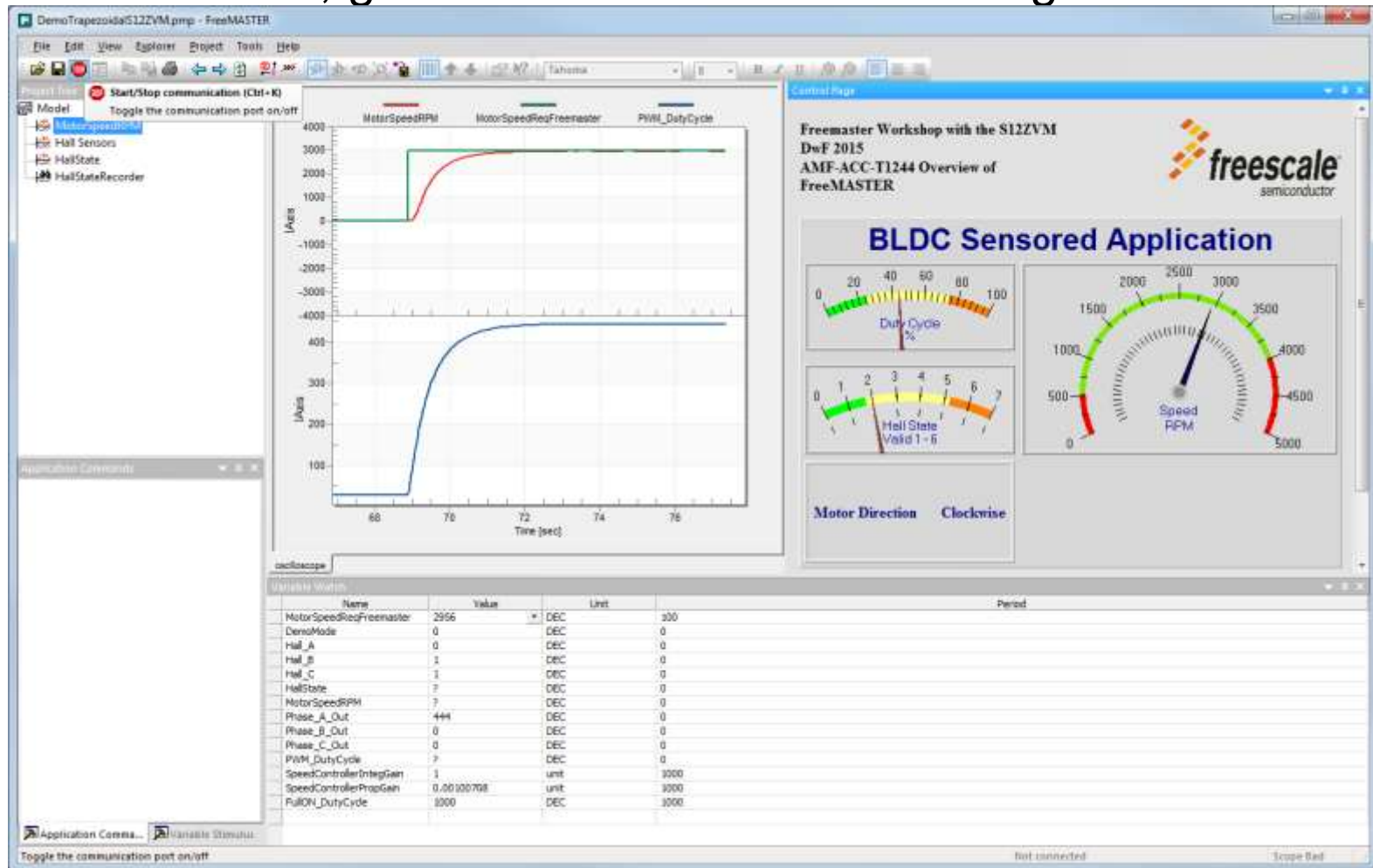
- On the menu bar, go to Project > Options
- When the window appears, select “HTML Pages”
- Browse to FreeMASTER\_Control Directory
- Select control.htm





# Adding to Freemaster Project Setup Control Page URL

- On the menu bar, go to View > Show Control Page as Bar



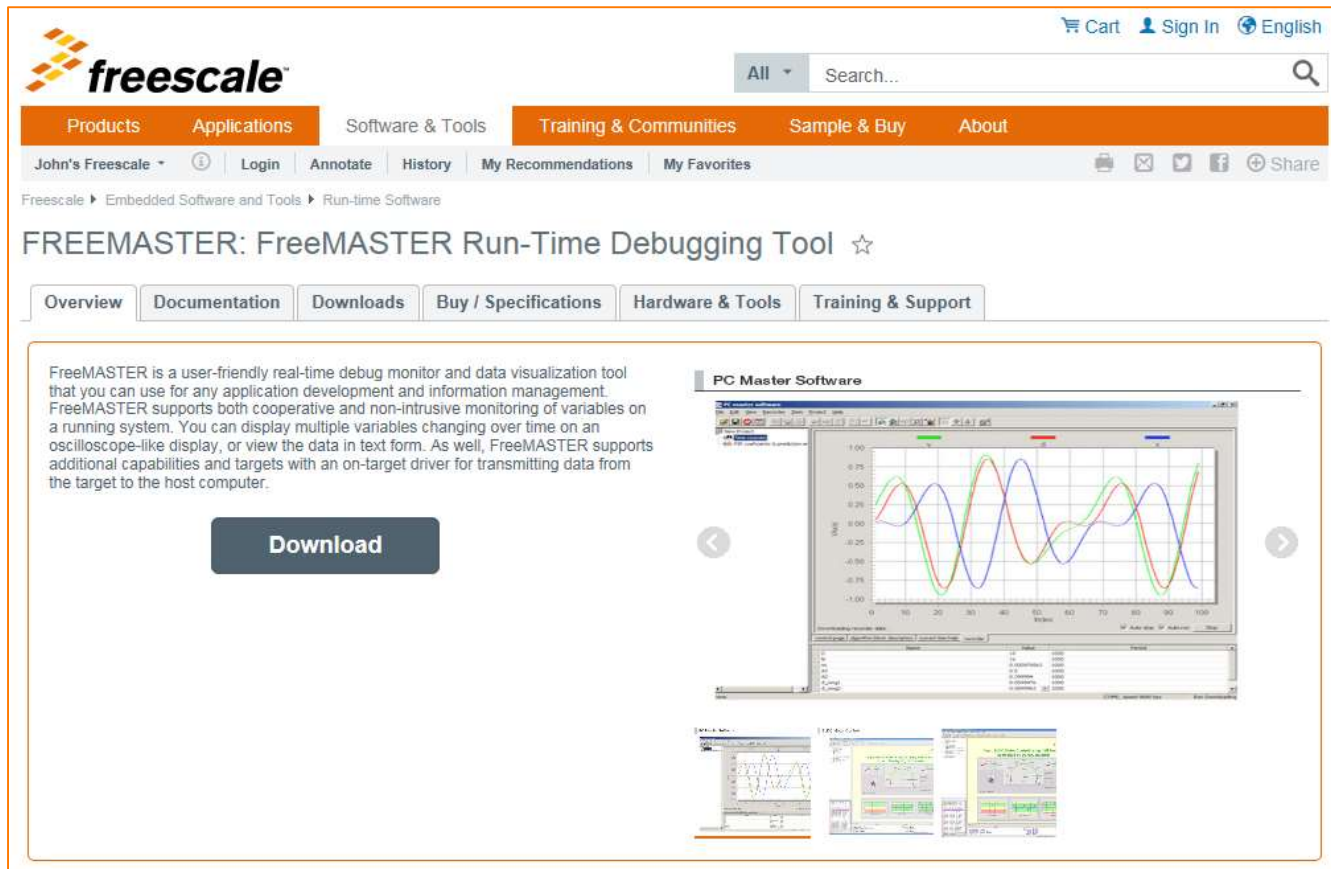
# Any Questions?



# Summary: More Information

For FreeMASTER, visit the Freescale website:

[www.freescale.com/Freemaster](http://www.freescale.com/Freemaster)



The screenshot displays the Freescale website's product page for FreeMASTER. At the top, the Freescale logo is on the left, and navigation links for 'Cart', 'Sign In', and 'English' are on the right. A search bar is positioned below the logo. The main navigation menu includes 'Products', 'Applications', 'Software & Tools', 'Training & Communities', 'Sample & Buy', and 'About'. Below this, a secondary menu shows 'John's Freescale', 'Login', 'Annotate', 'History', 'My Recommendations', and 'My Favorites'. The breadcrumb trail reads 'Freescale > Embedded Software and Tools > Run-time Software'. The main heading is 'FREEMASTER: FreeMASTER Run-Time Debugging Tool' with a star icon. Below the heading are tabs for 'Overview', 'Documentation', 'Downloads', 'Buy / Specifications', 'Hardware & Tools', and 'Training & Support'. The 'Overview' tab is active, showing a text description of FreeMASTER as a real-time debug monitor and data visualization tool. A prominent 'Download' button is located below the text. To the right, a section titled 'PC Master Software' features a large image of the software's oscilloscope-like interface, which displays multiple colored waveforms (red, green, blue) over time. Below this main image are three smaller thumbnail images showing different views of the software interface.



# Summary

- Any Questions?
- Please Fill Out Your Surveys
- Thank you for your time





[www.Freescale.com](http://www.Freescale.com)