

FreeMASTER Quick Start: Development and Debug Made Easy

Peter Pinewski

Senior Automotive Applications Engineer

October 2018 | AMF- AUT-T3393

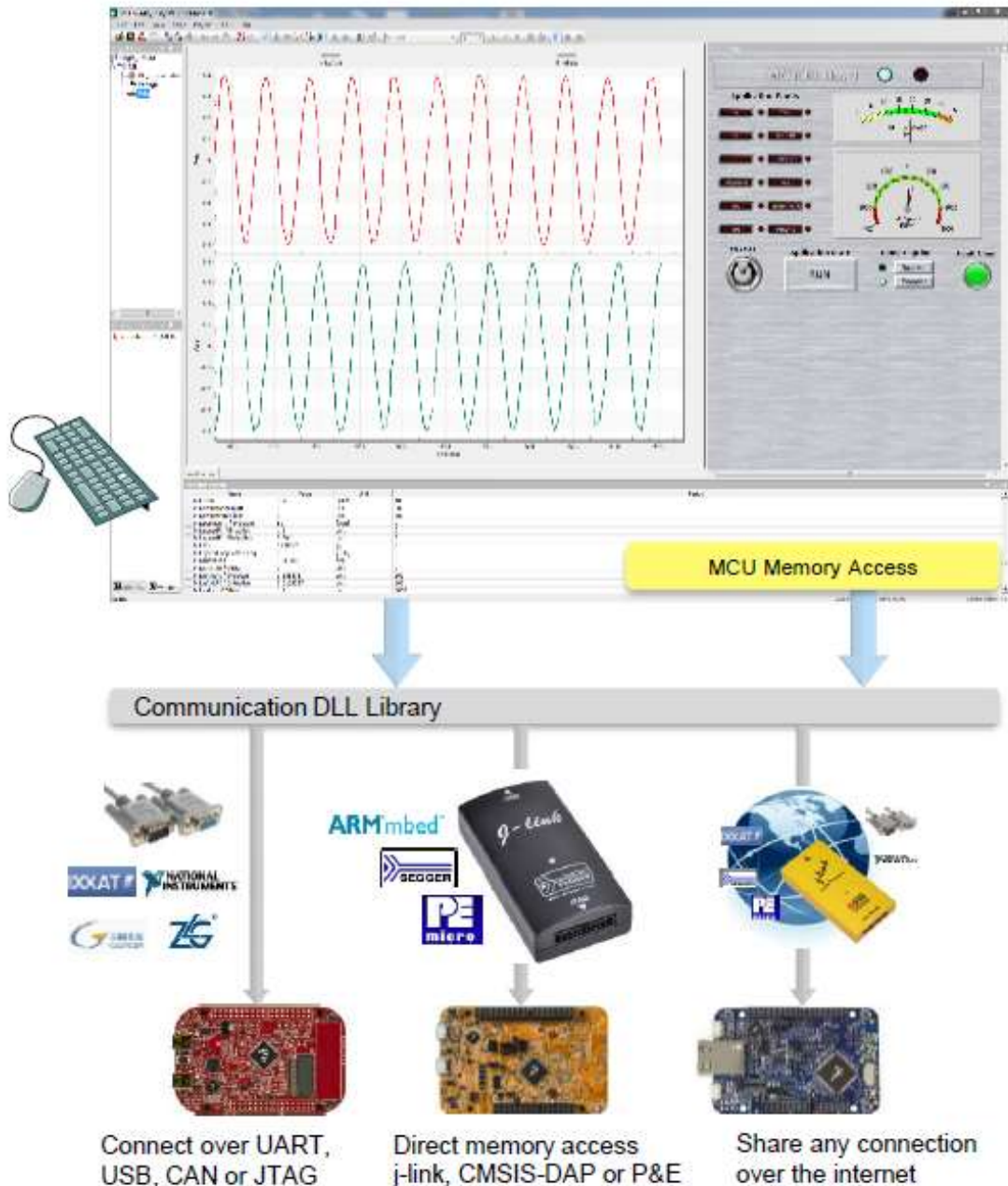


SECURE CONNECTIONS
FOR A SMARTER WORLD

Company External – NXP, the NXP logo, and NXP secure connections for a smarter world are trademarks of NXP B.V. All other product or service names are the property of their respective owners. © 2018 NXP B.V.

Agenda

- What is FreeMASTER
- How to get FreeMASTER
- FreeMASTER features
 - As a Real-Time Monitor
 - As a Control GUI
 - vs. a Debugger
- Setting up FreeMASTER – no Driver
- Configuring a Project
- Setting up a Project App with Freemaster Driver
- Summary



What is FreeMASTER?

FreeMASTER is a user-friendly **real-time debug monitor and data visualization** tool that can be used for application development and information management.

- Supports **non-intrusive monitoring of variables** on a running system.
- Display multiple variables changing over time on an **oscilloscope-like display**, or view **data in text form**.
- Supports **additional capabilities** and targets **with an on-target driver** for transmitting data from the target to the host computer.

What do we do with FreeMASTER?

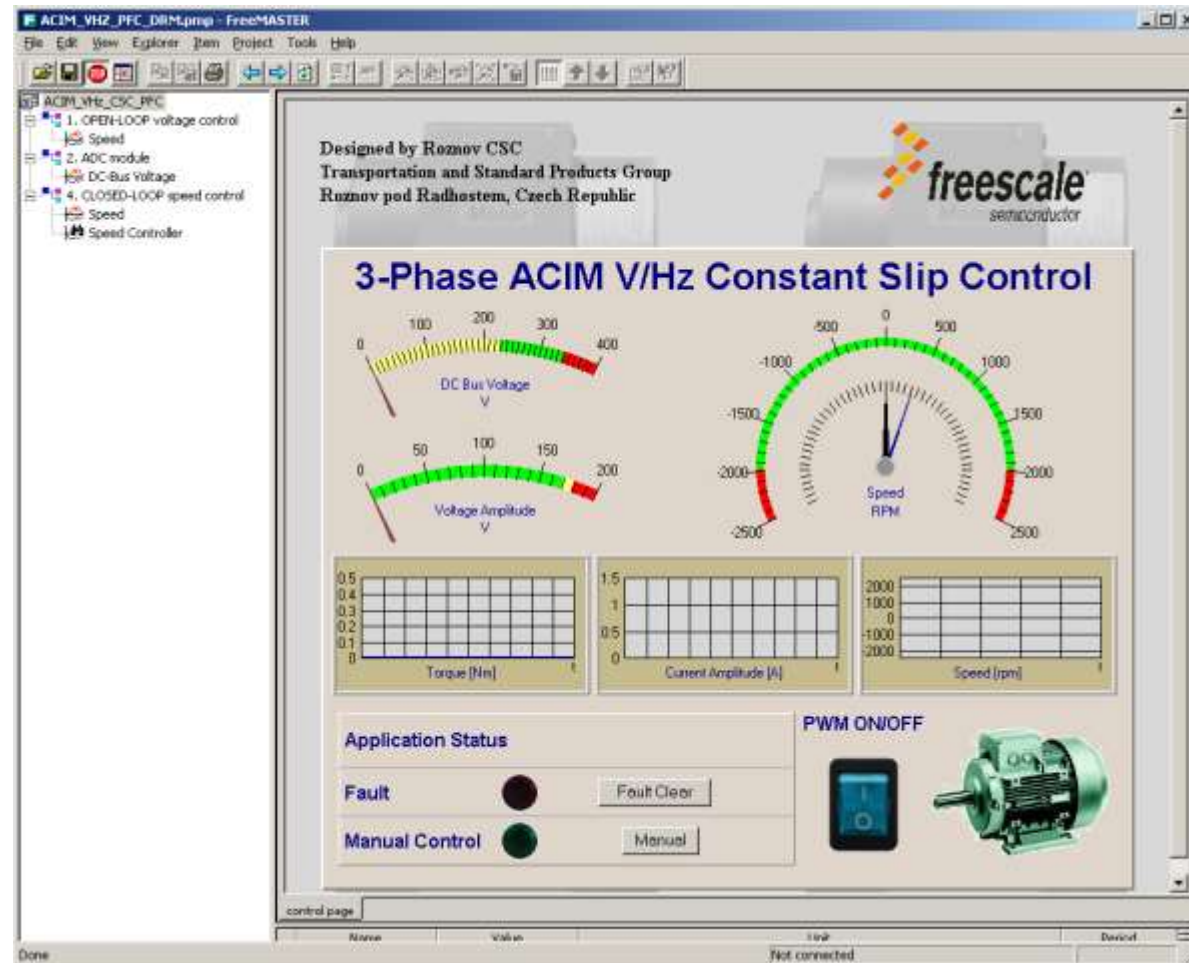
- **Connect**: to target MCU over UART, CAN, BDM, JTAG etc
- **Monitor**: read & show variables in run-time
- **Control**: set variables, send commands
- **Share**: enable Excel, Matlab or a script engine to add hardware to the control loop

What is FreeMASTER?

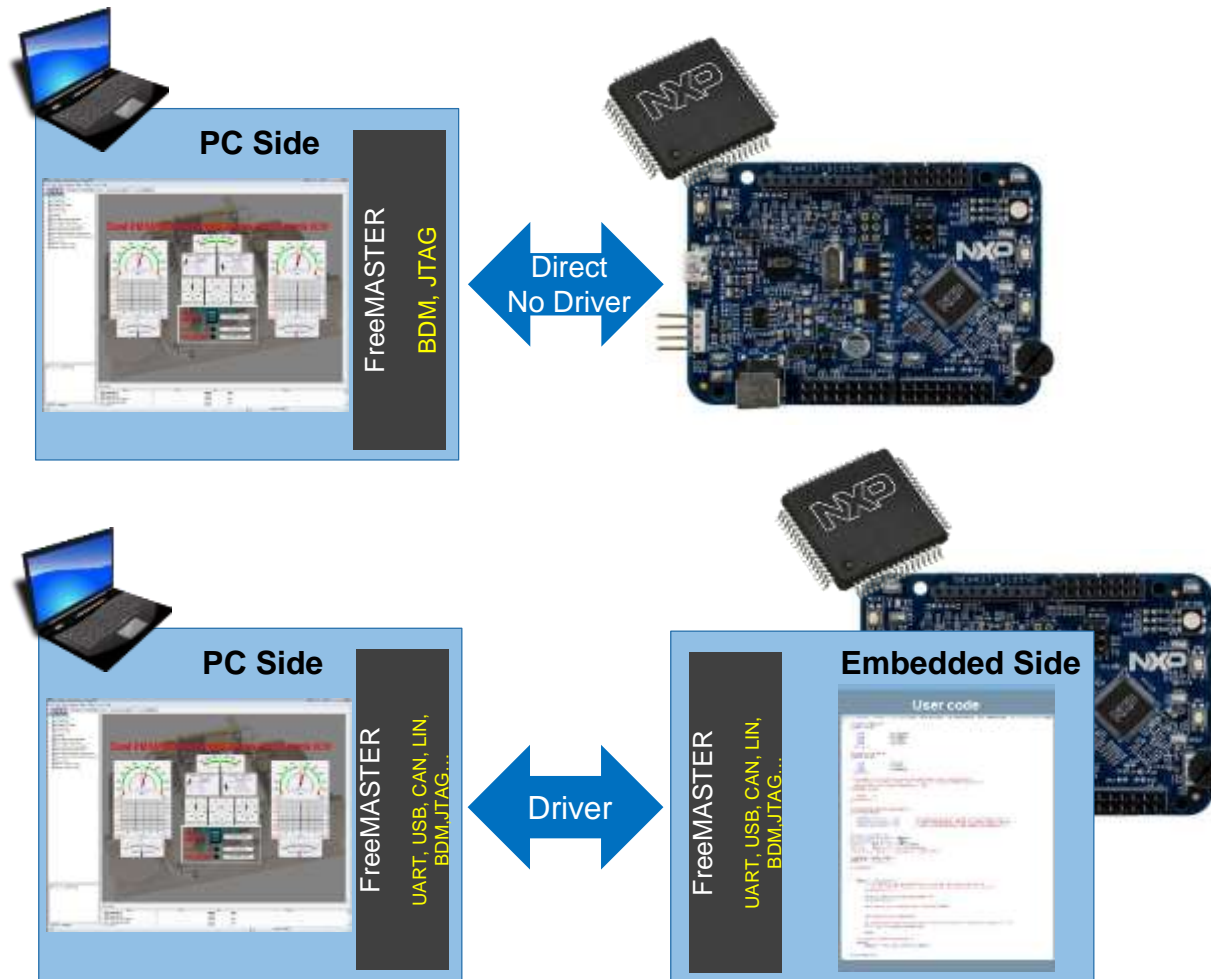
Application control
and monitor

Live graphs,
variable watches,
and graphical
control page

Real-time
operation monitor



FreeMASTER Connection Options



Supported Devices

- S08
- DSC
- ARM Cortex-M (Kinetis/**S32K**)
- S12/S12X/S12Z(MagniV),
- MPC56xx, MPC57xx
- ColdFire V1/V2

Supported Interfaces

- BDM
- JTAG/SWD (Segger,PE,CMSIS DAP,etc)
- Serial
- CAN
- LIN
- USB

FreeMASTER Supported Devices and Interfaces

MCU Families	No Driver	Target Driver Required						
	BDM /JTAG	Packet-Driven BDM	Serial	CAN	LIN	USB	MQX IO	eOnce /JTAG
S12 MagniV® Mixed Signal, S12 and S12X MCUs	✓	✓	✓	✓	✓			
S32 MCUs based on ARM Cortex-M	✓	✓	✓	✓				
MPC56xx MCUs based on Power Architecture	✓	✓	✓	✓				
MPC57xx based on Power Architecture	✓	✓	✓	✓				
Kinetis MCUs based on ARM Cortex-M	✓	✓	✓	✓		✓		
S08 MCUs	✓	✓	✓	✓		✓		
DSC			✓	✓				✓
ColdFire MCUs	✓	✓	✓	✓		✓	✓	

NOTE:

If it is desired to run the debugger and **FreeMASTER** concurrently, the target driver option is required!

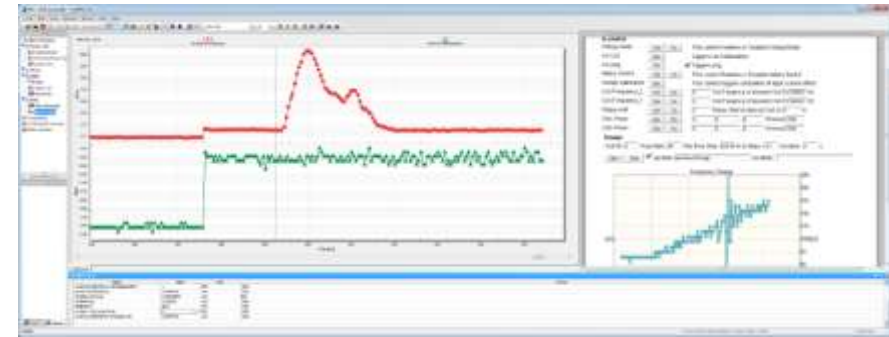
System Requirements

- Host side operating system: Windows XP to through Windows 10 32/64bit
- Required software: Internet Explorer 8 or higher installed beforehand.
- Hard drive space: 50 MB
- Other hardware requirements: Serial RS-232 port for local control or USB-to Serial converter.

FreeMASTER Features and Usage

Real Time Monitor

- Watching on-board variables or memory locations in various formats
- Text (name, value, min, max, enumerated labels...)
- Real-time waveform (real-time oscilloscope)
- High-speed recorded data (on-board memory oscilloscope)
- User-defined dashboard for data visualization



Control Panel

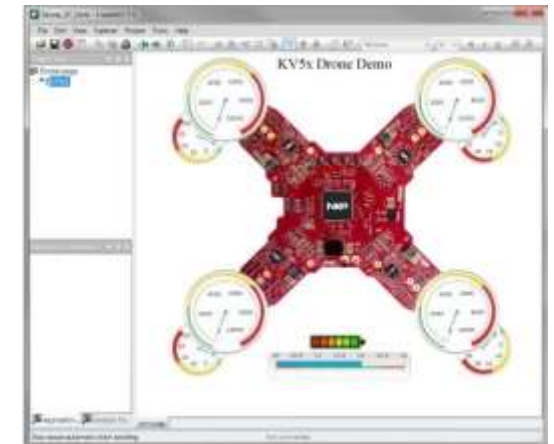
- Direct setting of the variable value from the variable watch
- Time-table stimulation of the variable value
- User command/message control
- Visual Basic script or JScript-powered HTML Forms (with push buttons, indicators and sliders) or custom HTML5 gauges
- By external application like Excel, Matlab or other which support ActiveX embedding

Demonstration Platform

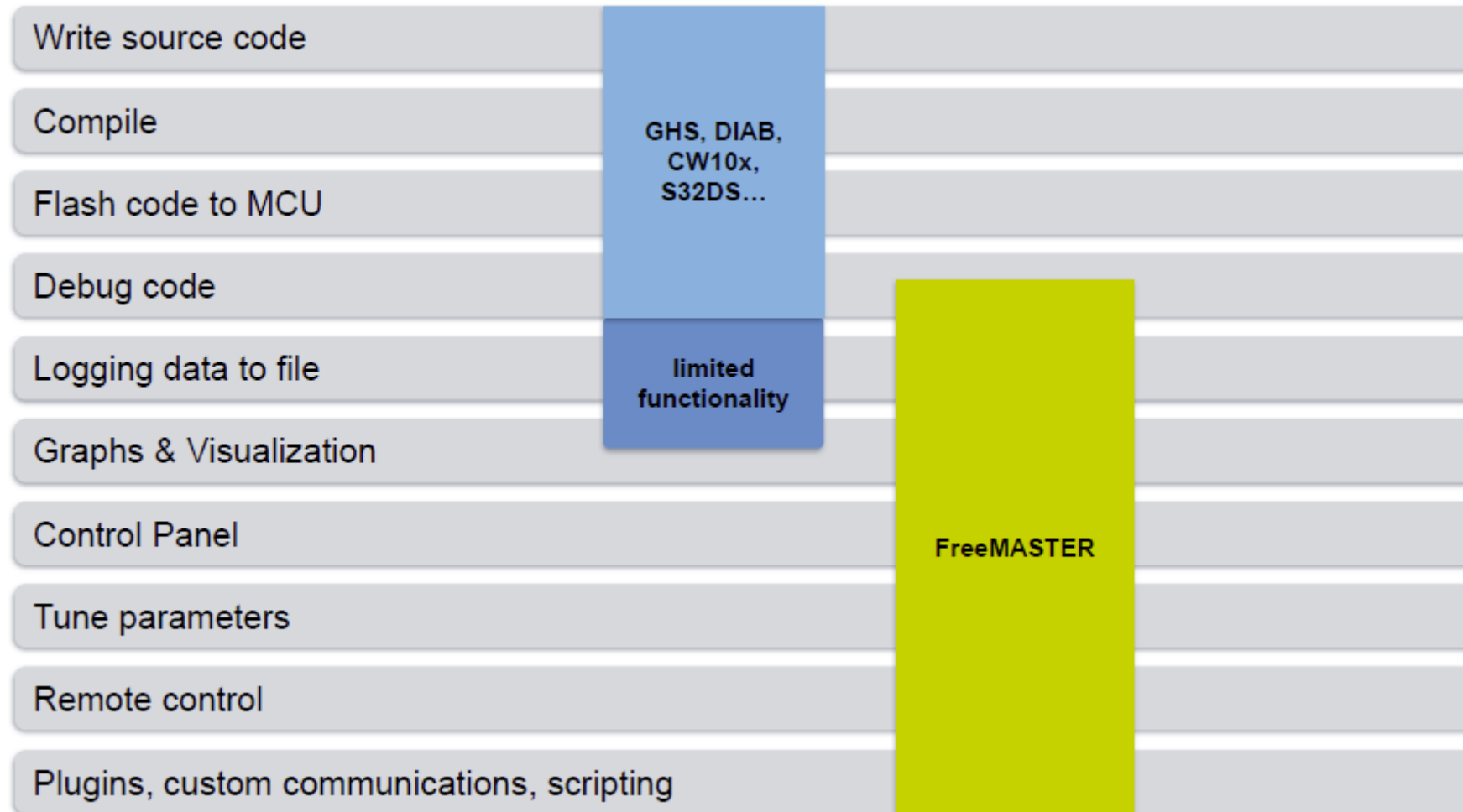
- You can both describe and demonstrate your embedded application by HTML pages that contain pictures, sounds, video sequences, links or any web content
- Display simultaneous real-time data monitoring
- Browse through the functional blocks of the embedded application

Easy Project Deployment

- Entire project saved to a single file
- All resources/files packed in the project file
- "Demo mode" with password protection available
- New in v2.0: Project files embedded in target MCU Flash memory



FreeMASTER vs. IDE/Debugger



Tuning Application Constants With FreeMASTER

- The most challenging task for the developer is the setting of the application constants, sometimes trial-error method must be used when the system (drive) parameters are difficult to identify:
 - P and I constants of the regulators
 - Filter constants
 - Constants of the position estimation algorithms
 - Tuning the merging process when switching from the open loop start-up to full sensorless mode

FreeMASTER Highlights

- FreeMASTER helps developers to debug or tune their applications
- Replaces debugger in situations when the processor core can not be simply stopped (e.g. motor control)
- Recorder may be used to visualize transitions in near 10-us resolution
- No EXTRA code is required on the embedded side to interface to FreeMASTER via BDM/OpenSDA/OSJTAG plug in modules.

How to Get FreeMASTER

FreeMASTER is a **FREE** download from www.nxp.com/freemaster

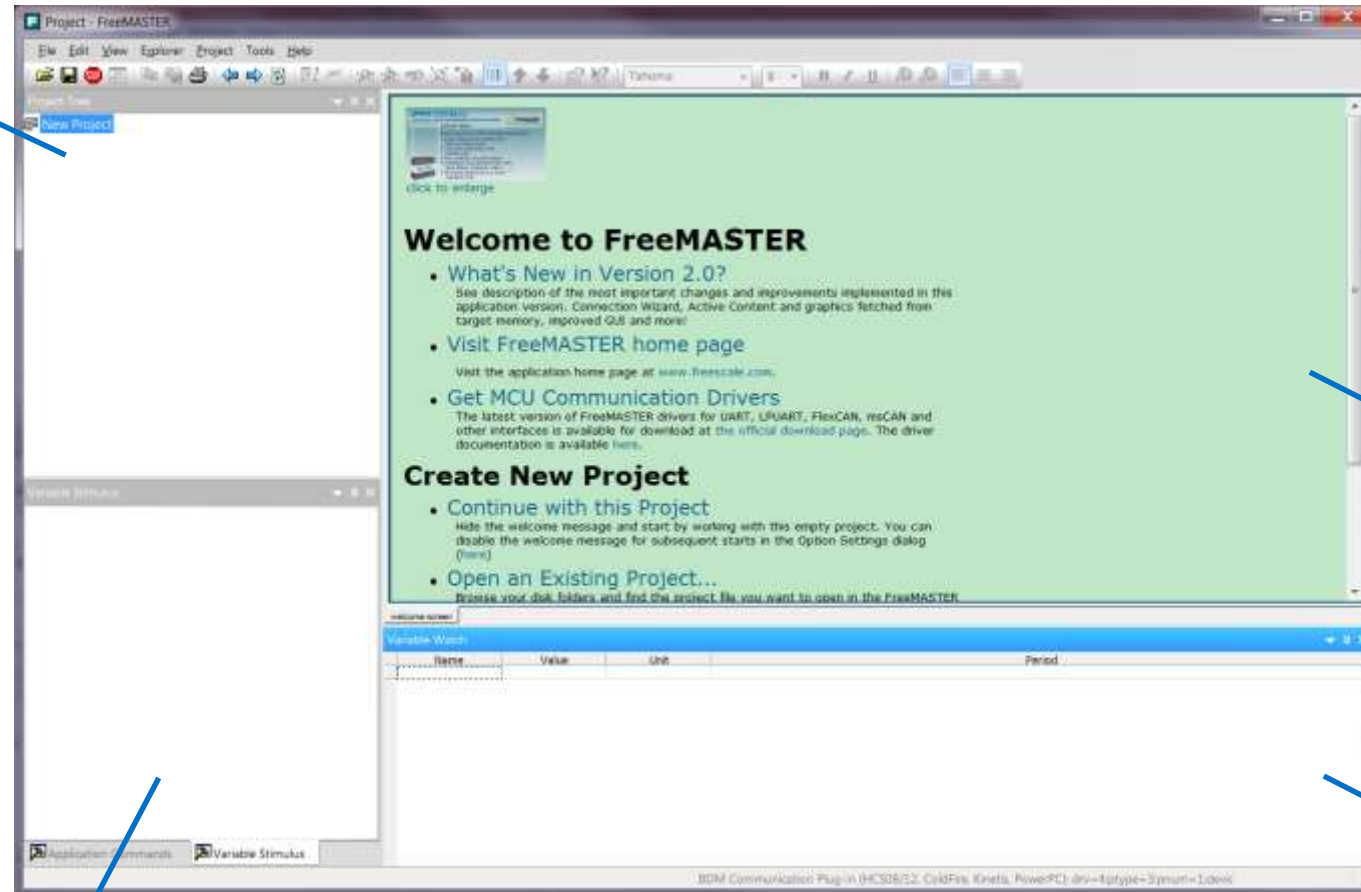
The screenshot shows the NXP website's product page for the FreeMASTER Run-Time Debugging Tool. The page features a navigation bar with links to PRODUCTS, SOLUTIONS, SUPPORT, and ABOUT. A search bar is located on the right. Below the navigation bar, the page title is "FREEMASTER: FreeMASTER Run-Time Debugging Tool". A tabbed interface shows "OVERVIEW", "BUY/PARAMETRICS", "DOCUMENTATION", "DOWNLOADS", and "HARDWARE & TOOLS". The "DOWNLOADS" tab is active. On the left, there is a "Filter By" section with a "Show All" link and a search bar. Below this, a list of categories is shown: "Recommended Software & Tools (1)", "Run-time Software (3)", "Application Specific - Reference Applications (1)", "Middleware - Device Drivers (1)", "Middleware - Libraries (3)", "Software Development Tools (1)", "IDE - Debug, Compile and Build Tools (1)", "Initialization/Boot/Device Driver Code Generation (1)", and "Software Support (1)". The main content area displays "Recommended Software & Tools (1)" with a download button for "FreeMASTER 2.0 Application Installation (REV 2.0.5)". Below this, "Reference Applications (1)" is shown with a link to "Motor Control Application Tuning (MCAT) Tool". The "Device Drivers (1)" section shows a download button for "FreeMASTER Communication Driver (REV 2.0)". Finally, the "Libraries (3)" section is visible at the bottom.

FreeMASTER Window Description



FreeMASTER Application Windows

Project Tree



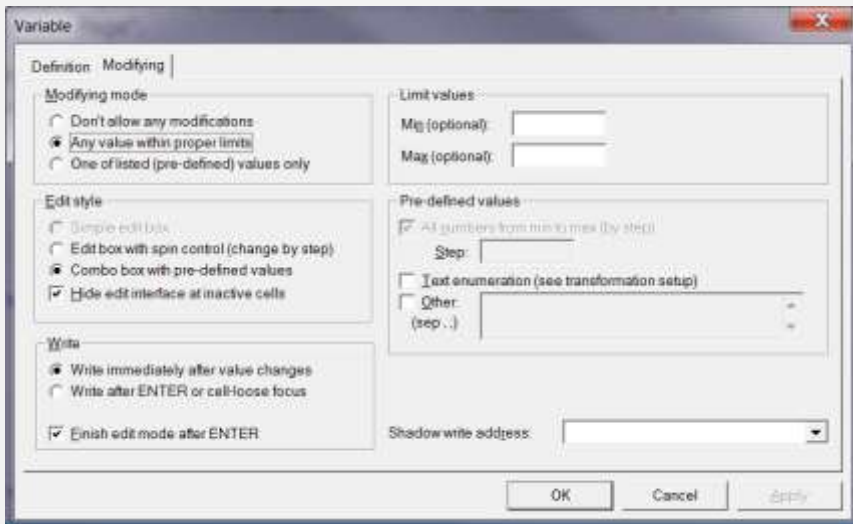
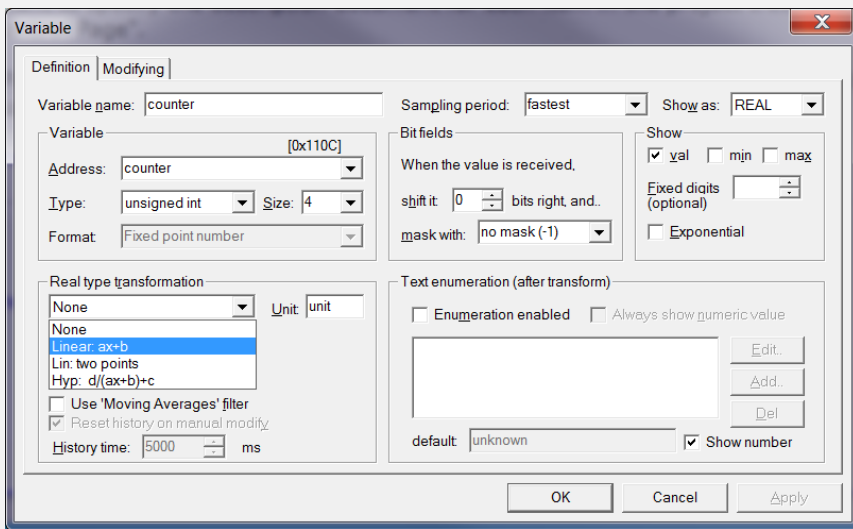
Detail View

- Scope / Recorder
- Description / Control

Variable Stimulus / App Commands

Variable Watch

- Watch
- Control



FreeMASTER Variable Watch

Variable Transformations

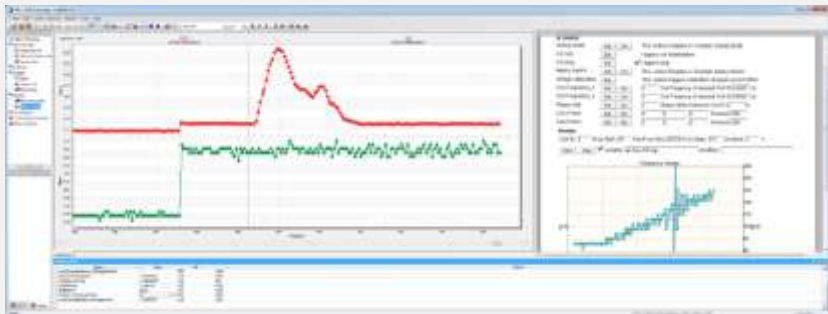
- Variable value can be transformed to a custom unit
- Variable transformations may reference other variable values
- Values are transformed back when writing a new value to the variable

Ability to protect memory regions

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

Variable update rate

- Choose rate of update based on type of variable
- Rate selectable from 100ms to 10s or as fast as possible (limited by communication interface)



FreeMASTER Explanation of Variables

The screenshot shows the 'Variable' dialog box in FreeMASTER, which is used to configure variables for monitoring and control. The dialog is divided into two tabs: 'Definition' and 'Modifying'. The 'Modifying' tab is currently selected. The dialog contains several sections for configuring the variable's behavior and display.

Variable Name: The 'Variable name' field is set to 'counter'.

Variable Address/Type/Size and Format: The 'Variable' section shows the address as 'counter' (with a memory address of [0x110C]), the type as 'unsigned int', the size as 4, and the format as 'Fixed point number'.

Variable Update Rate: The 'Sampling period' is set to 'fastest'.

Display Format: The 'Show as' dropdown is set to 'REAL'.

Display Value and optionally display min/max values: The 'Show' section has checkboxes for 'val' (checked), 'min', and 'max'. There is also a 'Fixed digits (optional)' field.

Bit Field Manipulation: The 'Bit fields' section includes options for 'shift it' (0 bits right), 'mask with' (no mask (-1)), and a 'Show' section with checkboxes for 'val', 'min', and 'max'.

Variable Transformation if Show as: is REAL: The 'Real type transformation' section shows a dropdown menu with options: 'None', 'Linear: ax+b', 'Lin: two points', and 'Hyp: d/(ax+b)+c'. The 'Unit' is set to 'unit'.

Enumerate values to text: The 'Text enumeration (after transform)' section has checkboxes for 'Enumeration enabled' and 'Always show numeric value'. There is a text area for defining enumerations and buttons for 'Edit..', 'Add..', and 'Del'.

Other options: There are checkboxes for 'Use 'Moving Averages' filter' and 'Reset history on manual modify'. The 'History time' is set to 5000 ms.

Buttons: The dialog has 'OK', 'Cancel', and 'Apply' buttons at the bottom.

FreeMASTER Explanation of Variables

The screenshot shows the 'Variable' dialog box in FreeMASTER, which is used to configure how a variable is displayed and edited in the software. The dialog is divided into two tabs: 'Definition' and 'Modifying'. The 'Modifying' tab is currently selected.

Annotations point to the following sections:

- Variable Protection:** Points to the 'Modifying mode' section, which contains three radio buttons: 'Don't allow any modifications', 'Any value within proper limits' (selected), and 'One of listed (pre-defined) values only'.
- Define appearance of variable editing:** Points to the 'Edit style' section, which contains four radio buttons: 'Simple edit box', 'Edit box with spin control (change by step)', 'Combo box with pre-defined values' (selected), and 'Hide edit interface at inactive cells' (checked).
- Define when variable is actually written:** Points to the 'Write' section, which contains two radio buttons: 'Write immediately after value changes' (selected) and 'Write after ENTER or cell-loose focus', and a checked checkbox 'Finish edit mode after ENTER'.
- Limit variable modification as Min/Max:** Points to the 'Limit values' section, which contains two text input fields: 'Min (optional):' and 'Max (optional):'.
- Limit variable by pre-defined values:** Points to the 'Pre-defined values' section, which contains a checked checkbox 'All numbers from min to max (by step)', a 'Step:' text input field, and two unchecked checkboxes: 'Text enumeration (see transformation setup)' and 'Other: (sep .:)'.

At the bottom of the dialog, there is a 'Shadow write address:' label followed by a dropdown menu. The 'OK', 'Cancel', and 'Apply' buttons are located at the bottom right.

FreeMASTER Variable Watch Customization

Variable Color and Size

Variable Stimulus

Display Order

Variable Watch

Name	Value	Unit	Period
UARTbaud_act	9600	Baudrate	10000
counter	12351	DEC	0
RollOverCounter	9774	DEC	1000
70000	DEC	1000	
TRUE [1]	ENUM	1000	
0	DEC	500	
0	DEC	500	
0	DEC	1000	

Project Block Properties

Main: Watch App.commands

Available variables:

Watched variables:

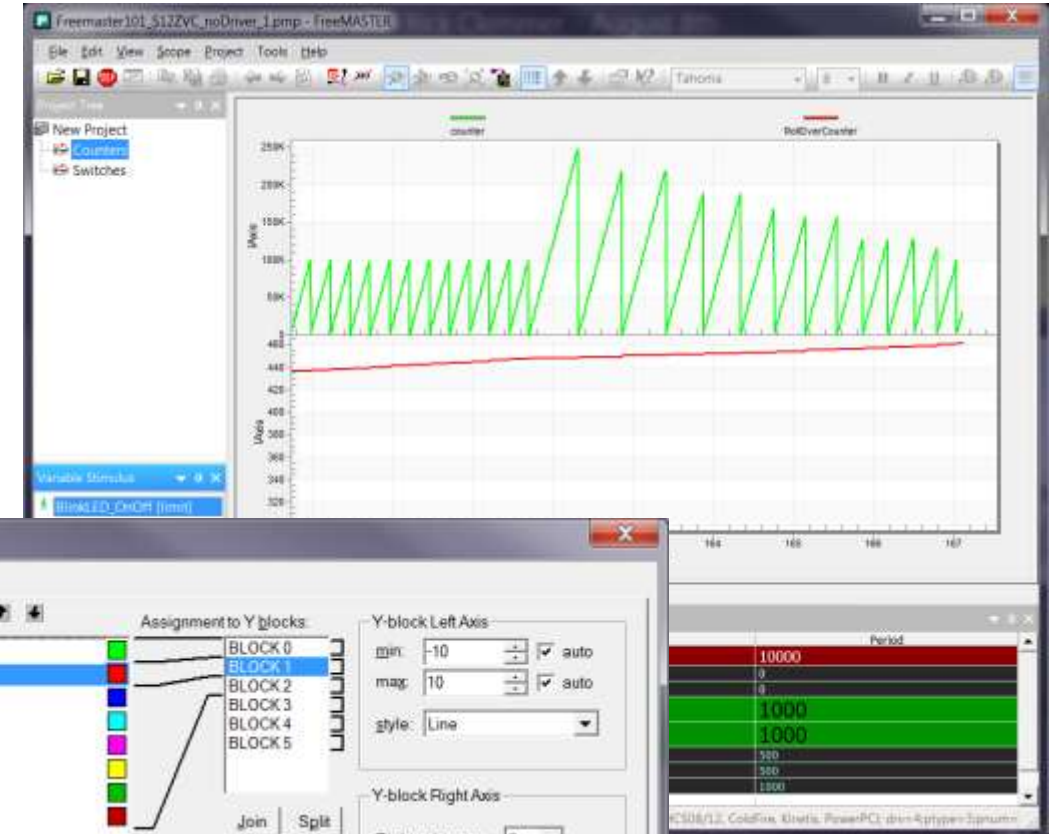
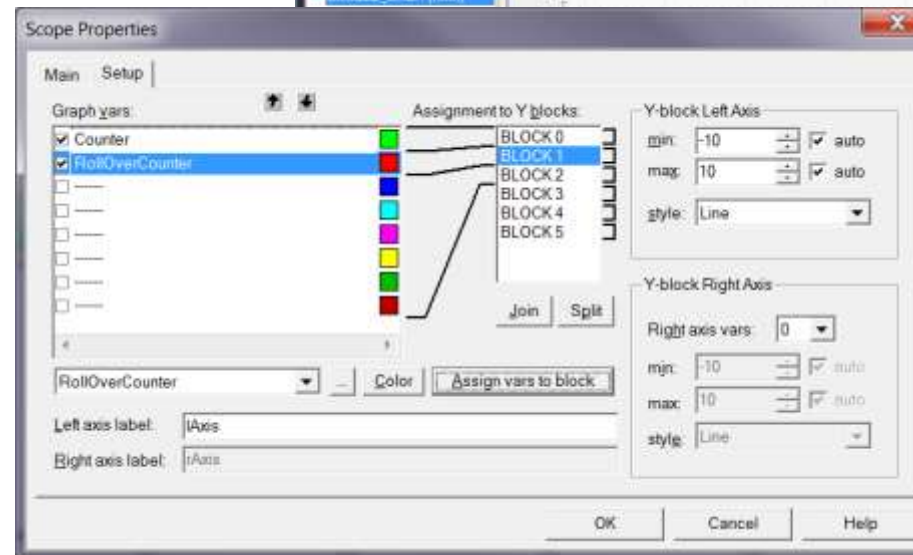
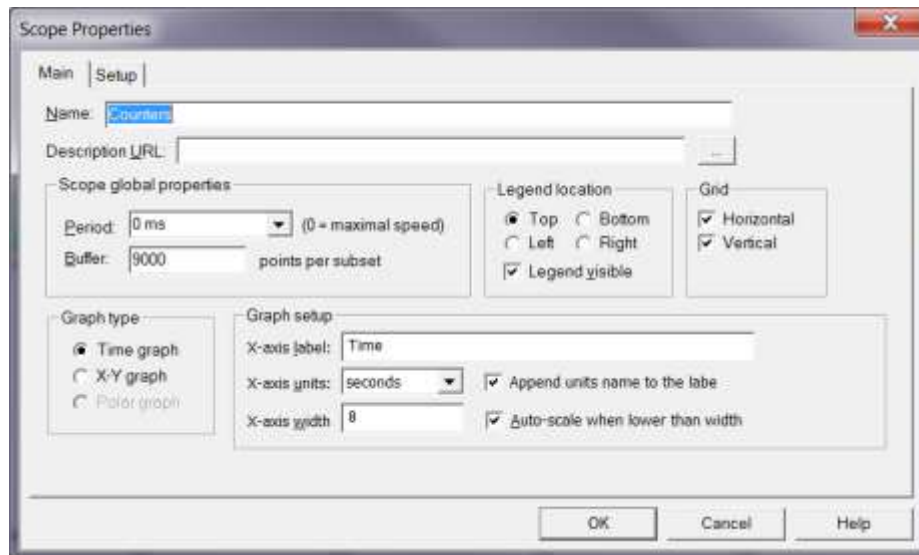
- UARTbaud_act
- counter
- RollOverCounter
- limit
- BlinkLed
- switch2
- switch3
- potentiometer

OK Cancel Help

FreeMASTER Scope and Recorder Views

Multiple Scope or Recorders can be configured

- Select between different Scope views
- Can do time graphing or X-Y graphing
- Scope can display multiple variables and/or multiple axis
- Can set up a left and right y-axis on same graph



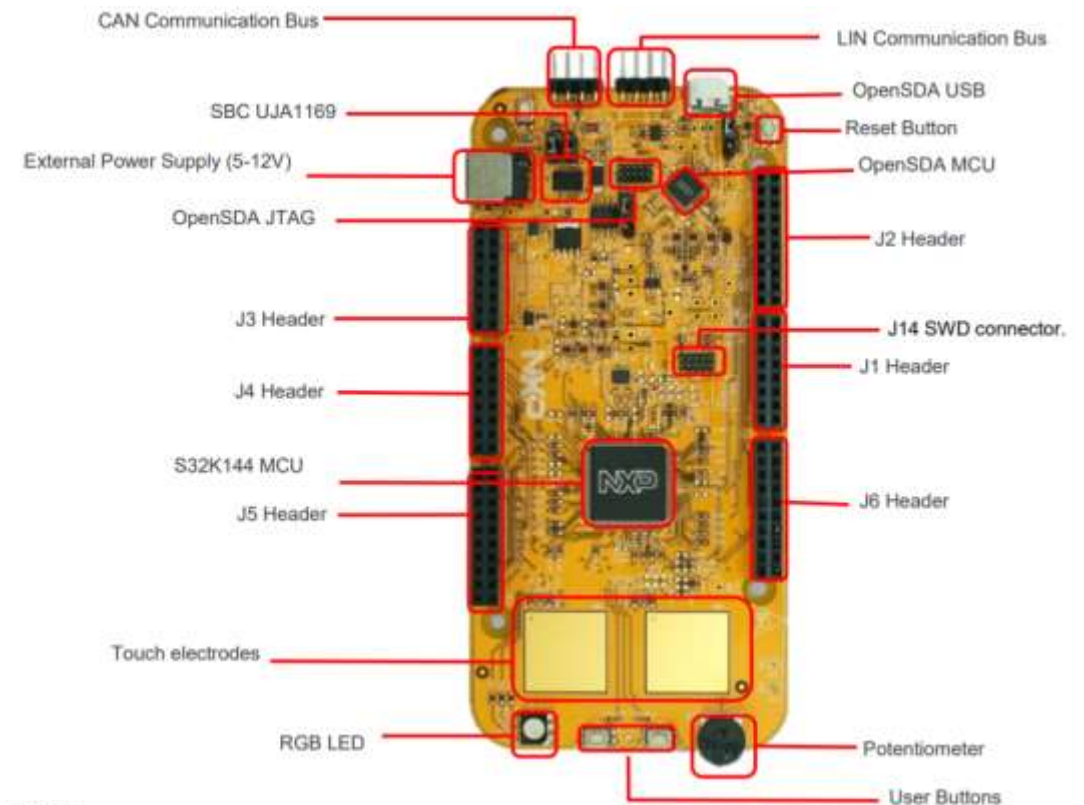
S32K144-EVB



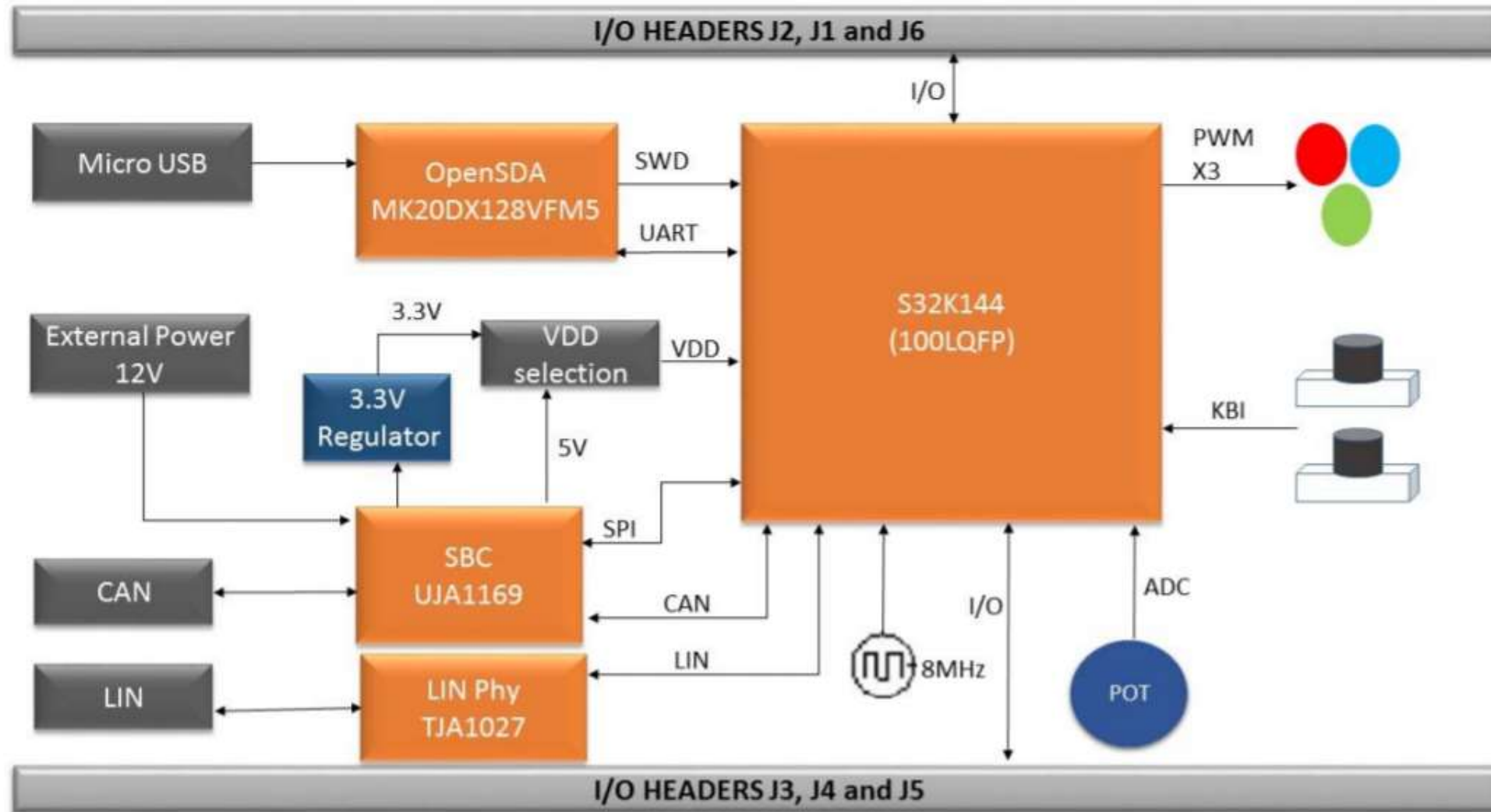
Get To Know the S32K144-EVB

The **S32K144-EVB** is a low-cost development platform for the S32K144 MCU.

- Supports S32K144 100LQFP
- Small form factor size supports up to 6" x 4"
- Arduino™ UNO footprint-compatible with expansion "shield" support
- Integrated open-standard serial and debug adapter (OpenSDA) with support for several industry-standard debug interfaces
- Easy access to the MCU I/O header pins for prototyping
- On-chip connectivity for CAN, LIN, UART/SCI.
- SBC UJA1169 and LIN phy TJA1027
- Potentiometer for precise voltage and analog measurement
- RGB LED
- Two push-button switches (SW2 and SW3) and two touch electrodes
- Flexible power supply options
 - microUSB or
 - external 12V power supply



S32K144-EVB Programming Interface and Peripherals



S32K144-EVB Connections and Peripherals

Header/Pinout Mapping for S32K144

PIN	PORT	FUNCTION	J3	PIN	PORT	FUNCTION
J3-02	PTB6*	GPIO		J3-01		VIN
J3-04	PTB7*	GPIO		J3-03		IOREF
J3-06	PTE0	GPIO		J3-05	PTA5	RESET
J3-08	PTE9	GPIO		J3-07		3V3
J3-10	PTC5	GPIO		J3-09		5V
J3-12	PTC4	GPIO		J3-11		GND
J3-14	PTA10	GPIO		J3-13		GND
J3-16	PTA4	GPIO		J3-15		VIN

PIN	PORT	FUNCTION	J4	PIN	PORT	FUNCTION
J4-02	PTC7	GPIO		J4-01	PTD4	ADC0
J4-04	PTC6	GPIO		J4-03	PTB12	ADC1
J4-06	PTB17	GPIO		J4-05	PTB0	ADC2
J4-08	PTB14	GPIO		J4-07	PTB1	ADC3
J4-10	PTB15	GPIO		J4-09	PTA6/PTE11/PTA2	ADC4
J4-12	PTB16	GPIO		J4-11	PTC0/PTE10/PTA3	ADC5
J4-14	PTC14	GPIO		J4-13	PTE2	ADC6
J4-16	PTC3	GPIO		J4-15	PTE6	ADC7

PIN	PORT	FUNCTION	J5	PIN	PORT	FUNCTION
J5-02	PTE16	GPIO		J5-01	PTA15/PTD11	ADC8
J5-04	PTE15	GPIO		J5-03	PTA16/PTD10	ADC9
J5-06	PTE14	GPIO		J5-05	PTA1	ADC10
J5-08	PTE13	GPIO		J5-07	PTA0	ADC11
J5-10		VDD		J5-09	PTA7	ADC12
J5-12		GND		J5-11	PTB13	ADC13
J5-14	PTE1	GPIO		J5-13	PTC1	ADC14
J5-16	PTD7	GPIO		J5-15	PTC2	ADC15
J5-18	PTD6	GPIO		J5-17		GPIO
J5-20	PTC15	GPIO		J5-19	NC	N/A

PIN	PORT	FUNCTION	J2	PIN	PORT	FUNCTION
J2-19	PTE10/PTA3	D15/I2C_CLK		J2-20	NC	GPIO
J2-17	PTE11/PTA2	D14/I2C_SDA		J2-18	NC	GPIO
J2-15		ANALOGUE REF		J2-16	PTA14	GPIO
J2-13		GND		J2-14	PTE7	GPIO
J2-11	PTB2	D13/SPI_SCK		J2-12	PTC13	GPIO
J2-09	PTB3	D12/SPI_SIN		J2-10	PTC12	GPIO
J2-07	PTB4	D11/SPI_SOUT		J2-08	PTE8	GPIO
J2-05	PTB5	D10/SPI_CS		J2-06	PTD0	GPIO
J2-03	PTD14	D9/PWM		J2-04	PTD16	GPIO
J2-01	PTD13	D8/PWM		J2-02	PTD15	GPIO

PIN	PORT	FUNCTION	J1	PIN	PORT	FUNCTION
J1-15	PTC11/PTE8	D7		J1-16	PTE3	GPIO
J1-13	PTC10/PTC3	D6		J1-14	PTD3	GPIO
J1-11	PTB11	D5		J1-12	PTD5	GPIO
J1-09	PTB10	D4		J1-10	PTD12	GPIO
J1-07	PTB9	D3		J1-08	PTD11	GPIO
J1-05	PTB8	D2		J1-06	PTD10	GPIO
J1-03	PTA3	D1		J1-04	PTA17	GPIO
J1-01	PTA2	D0		J1-02	PTA11	GPIO

PIN	PORT	FUNCTION	J6	PIN	PORT	FUNCTION
J6-19	PTA9	D14		J6-20	PTE4	GPIO
J6-17	PTA8	D15		J6-18	PTE5	GPIO
J6-15	PTC12	D16		J6-16	PTA12	GPIO
J6-13	PTD17	D17		J6-14	PTA13	GPIO
J6-11	PTC9	D18		J6-12		GND
J6-09	PTC8	D19		J6-10		VDD
J6-07	PTD8	D20		J6-08	PTC16	GPIO
J6-05	PTD9	D21		J6-06	PTC17	GPIO
J6-03	PTD2	GPIO		J6-04	PTD3	GPIO
J6-01	PTD0	GPIO		J6-02	PTD1	GPIO

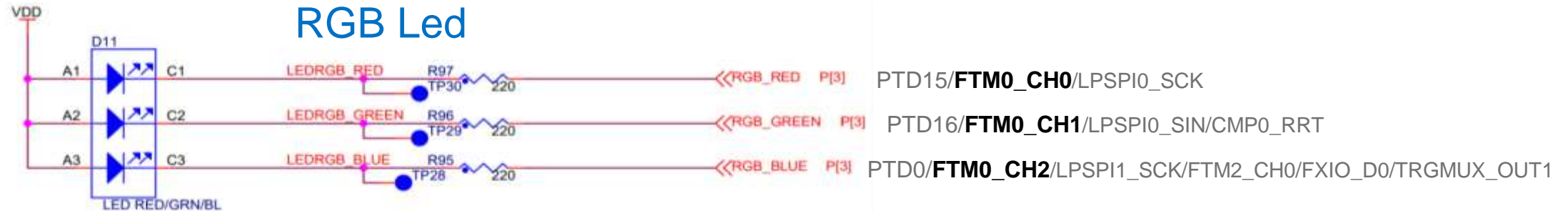


■ Arduino compatible pins
■ NXP pins

*0ohm resistor is not connected

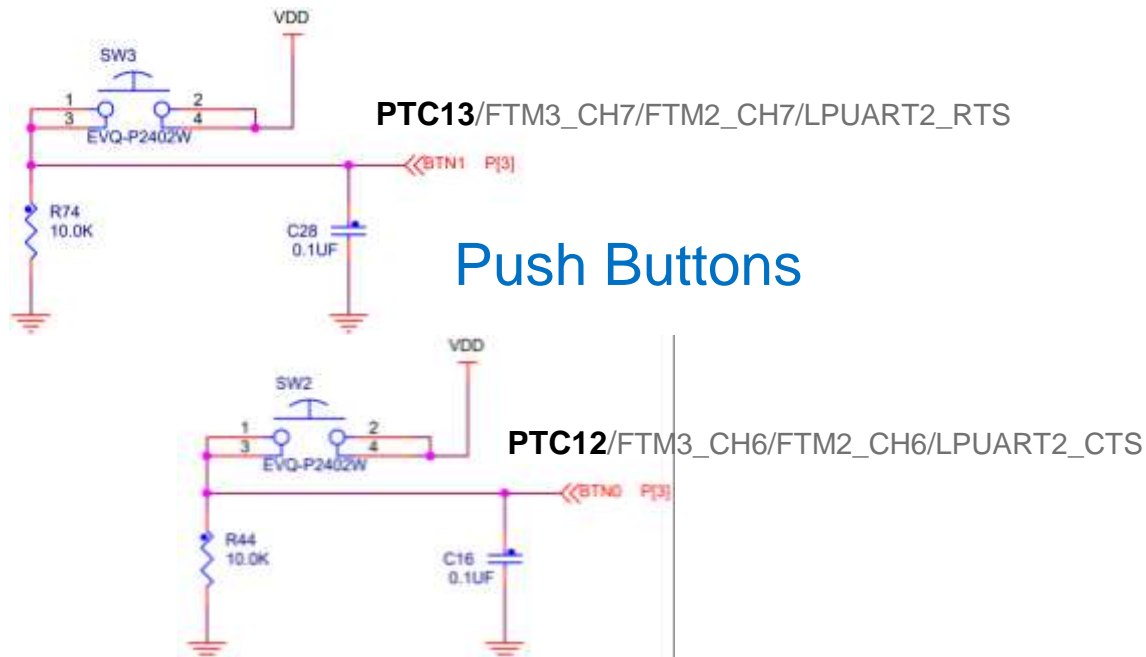
S32K144-EVB Interface Connections

RGB Led



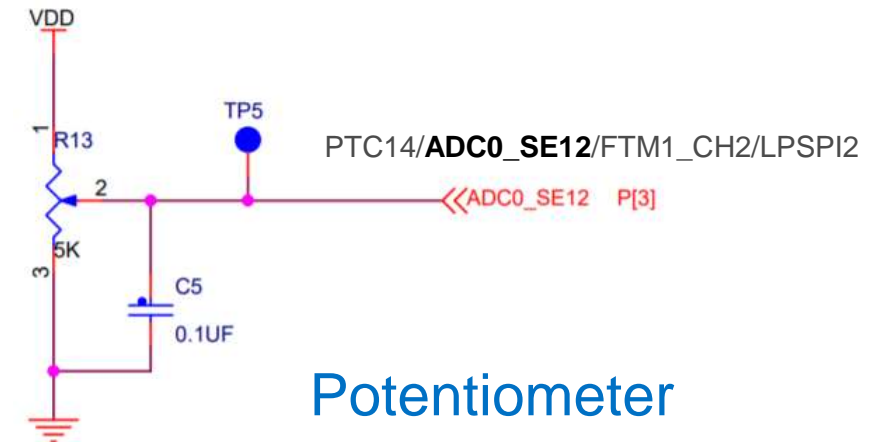
PTC13/FTM3_CH7/FTM2_CH7/LPUART2_RTS

Push Buttons



PTC14/ADC0_SE12/FTM1_CH2/LPSP12

Potentiometer



Exercise #1: Starting with FREEMASTER – No Driver



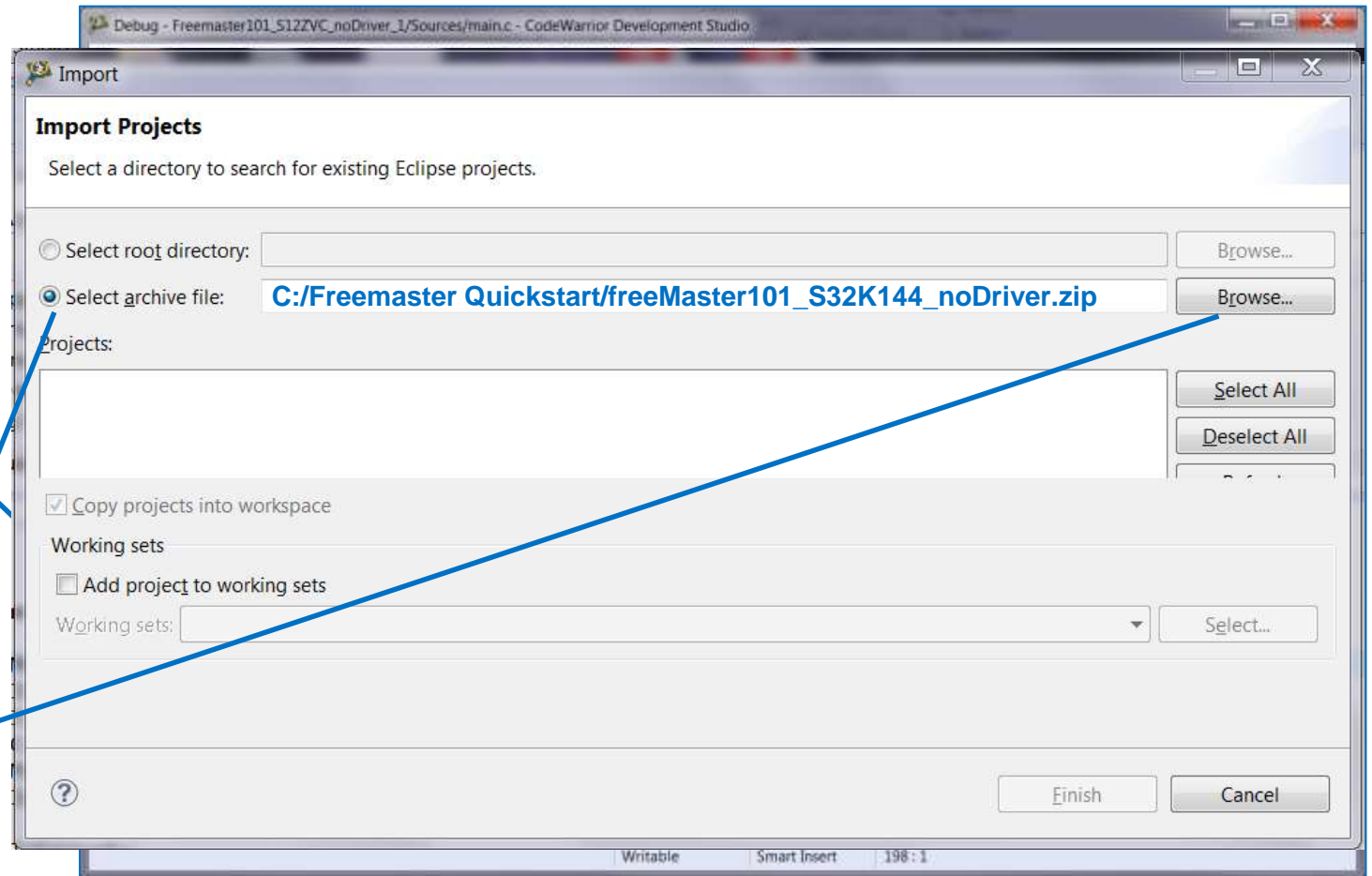
Exercise #1 – Programming the Target

- Open S32DS for ARM
- Import the test project

1 – Select Import Project

2 – Check Select archive file:

3 – Browse to project file



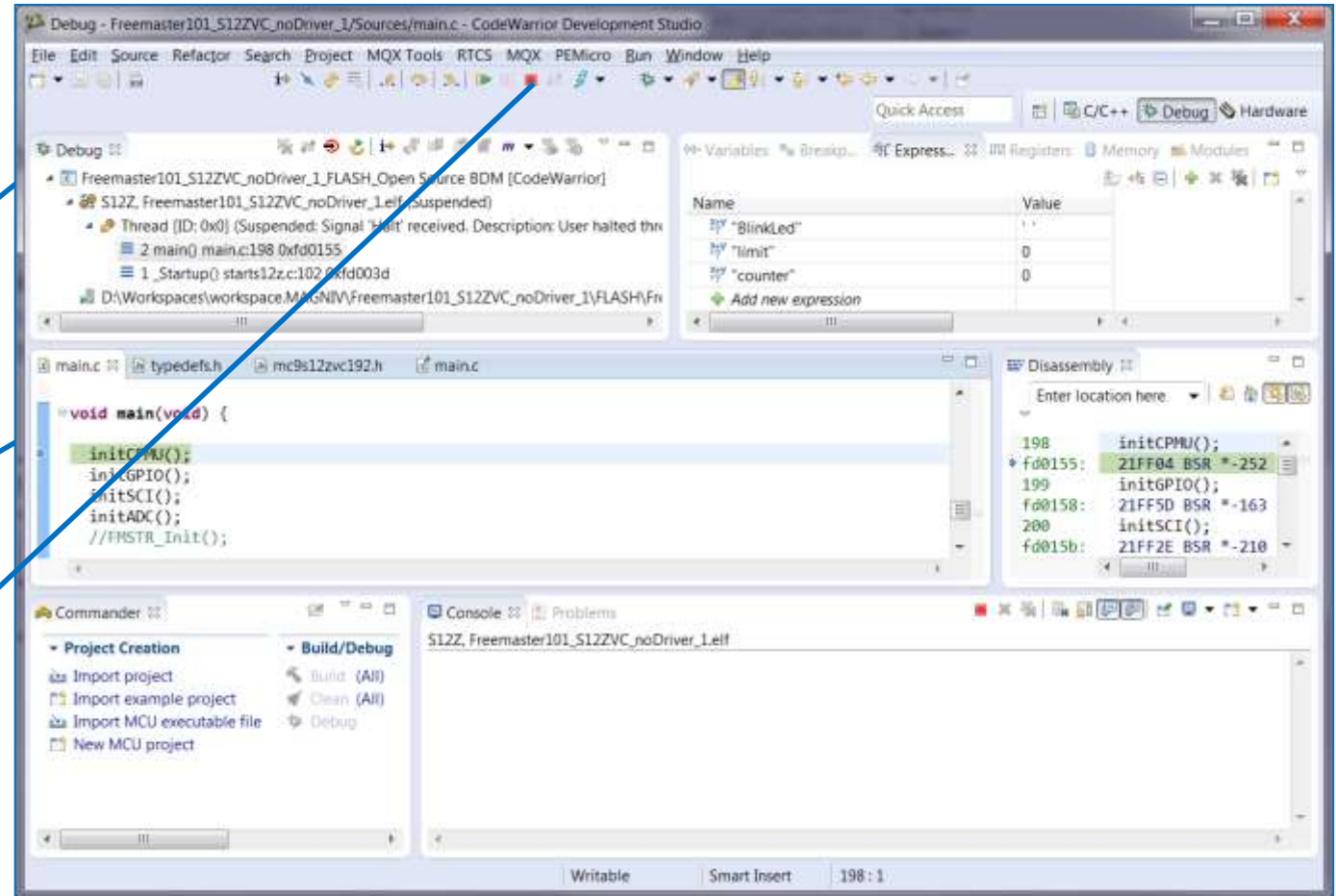
Exercise #1 – Programming the Target

- Open S32DS for ARM
- Select the test project
[Freemaster101_s32k144_noDriver](#)
- Build and download the project

1 - Build Project

2 - Debug Project

3 – Terminate Connection

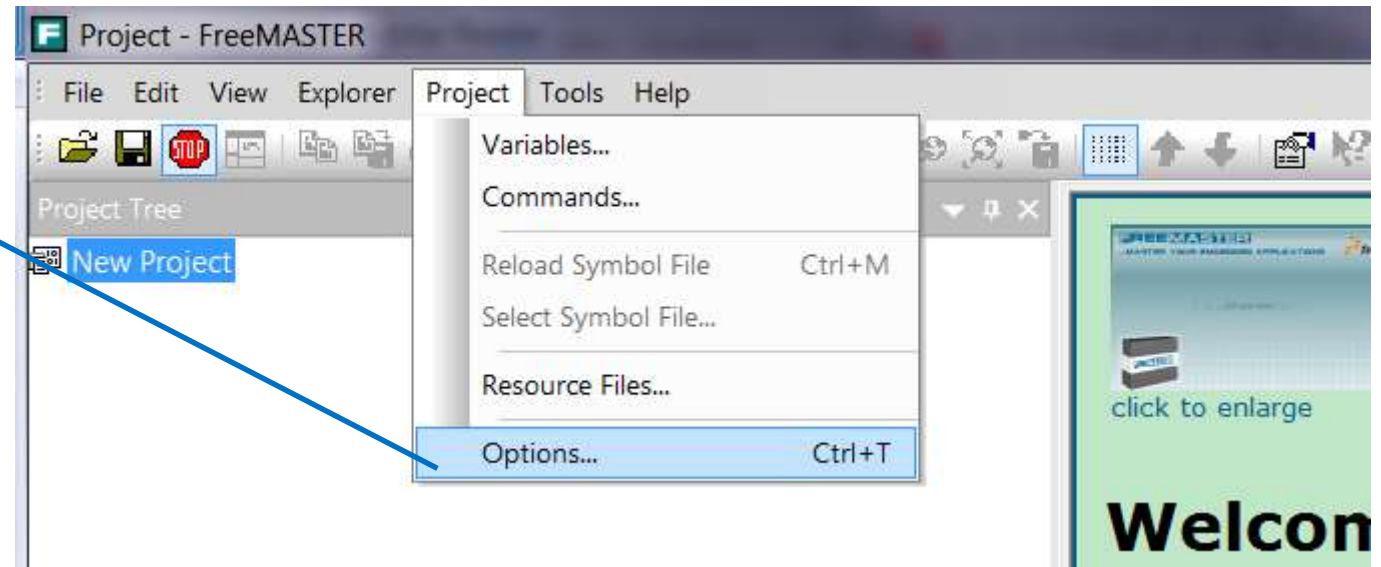


Note: Freemaster and Debugger cannot access debug port at the same time

Exercise #1 – No Driver Configuration

- Open FREEMASTER (v2.0)
- Select Project Options

1. Select Project / Options



FREEMASTER

Exercise #1 – No Driver Configuration

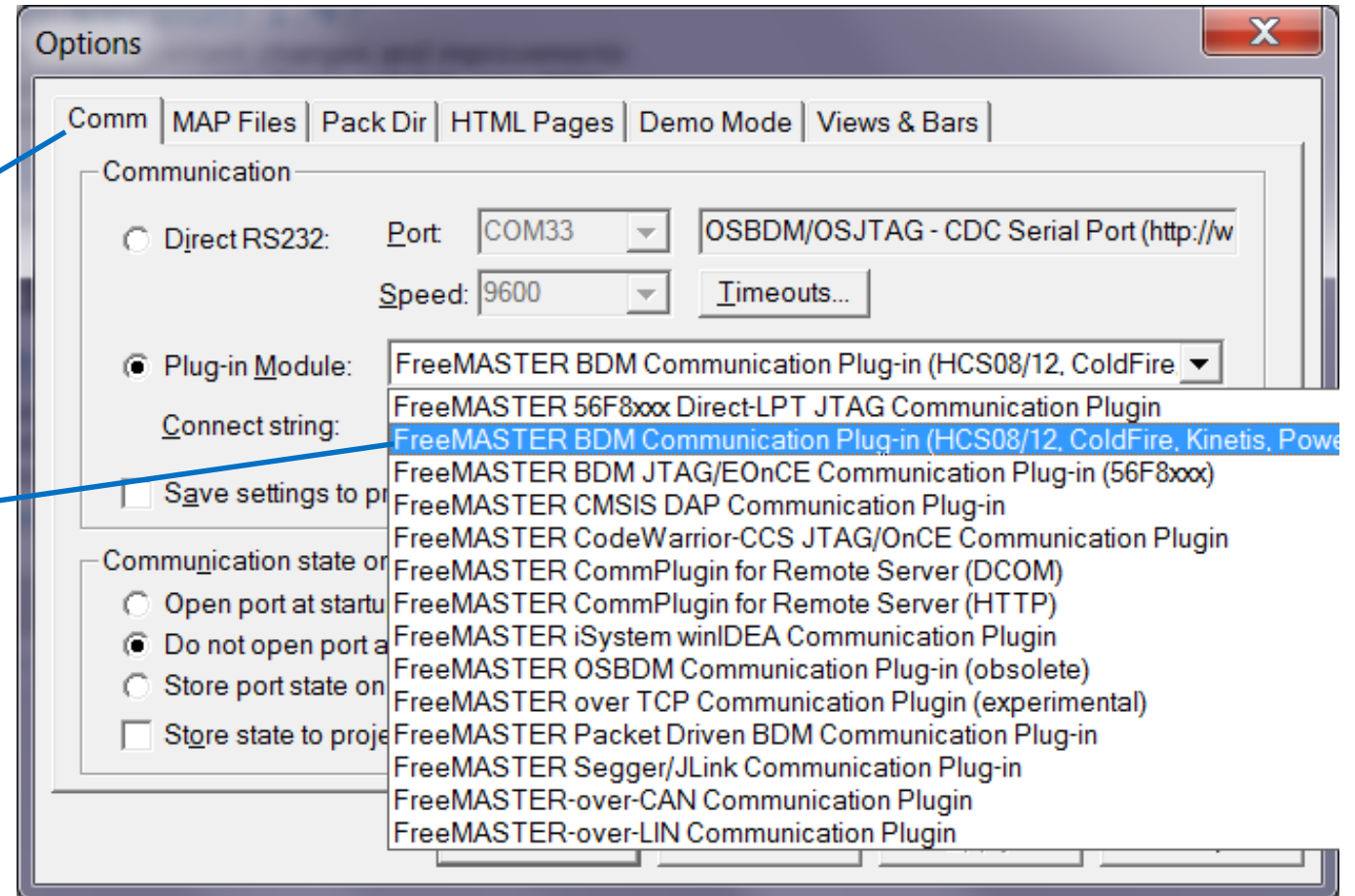
- Configure Freemaster for target board and project

Set up connection (Comm Tab)

2. Select **Comm** Tab

3. Select Connection Interface
BDM Communication Plug-in(HCS08/12...)

FREEMASTER

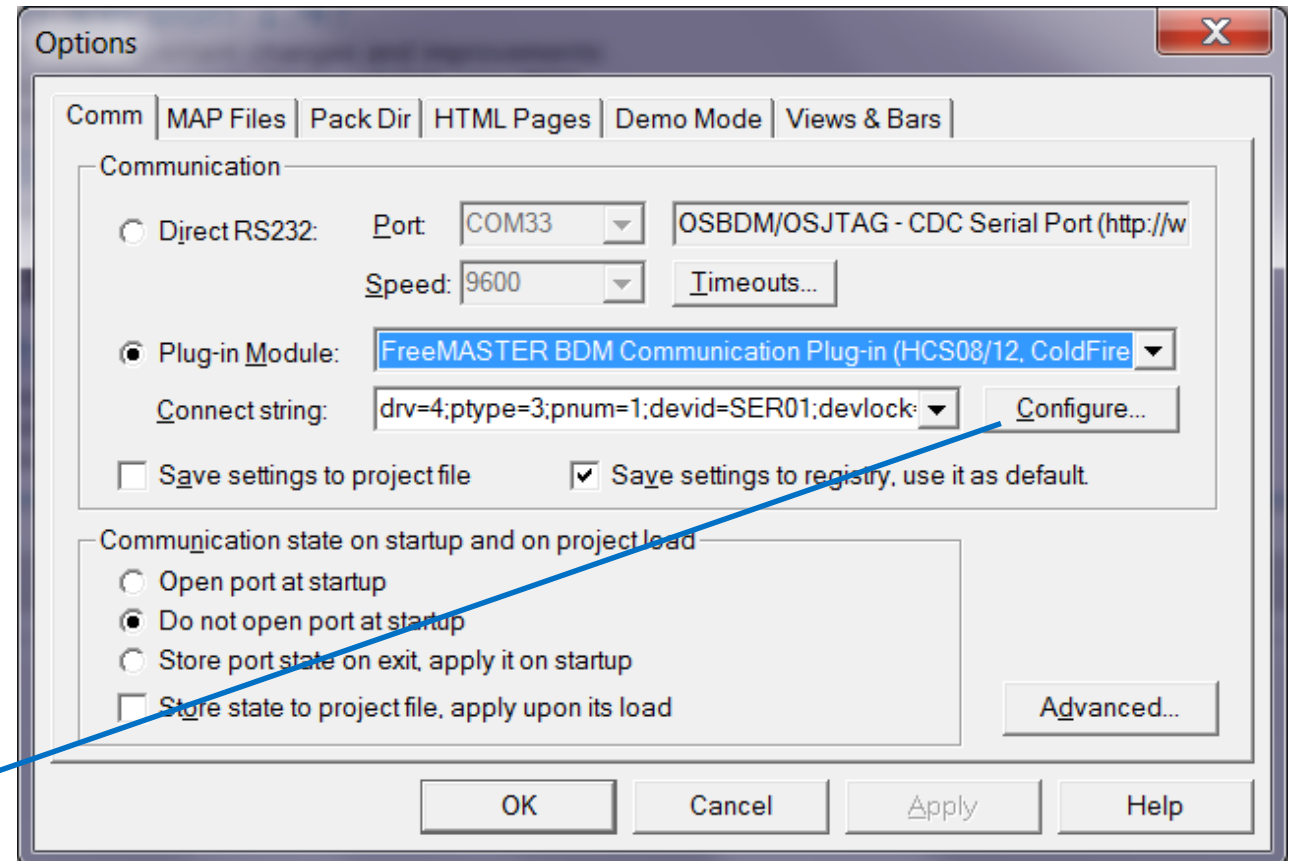


Exercise #1 – No Driver Configuration

- Configure Freemaster for target board and project

Set up connection (Comm Tab)

4. Configure the Connection



FREEMASTER

Exercise #1 – No Driver Configuration

- Configure Freemaster for target board and project

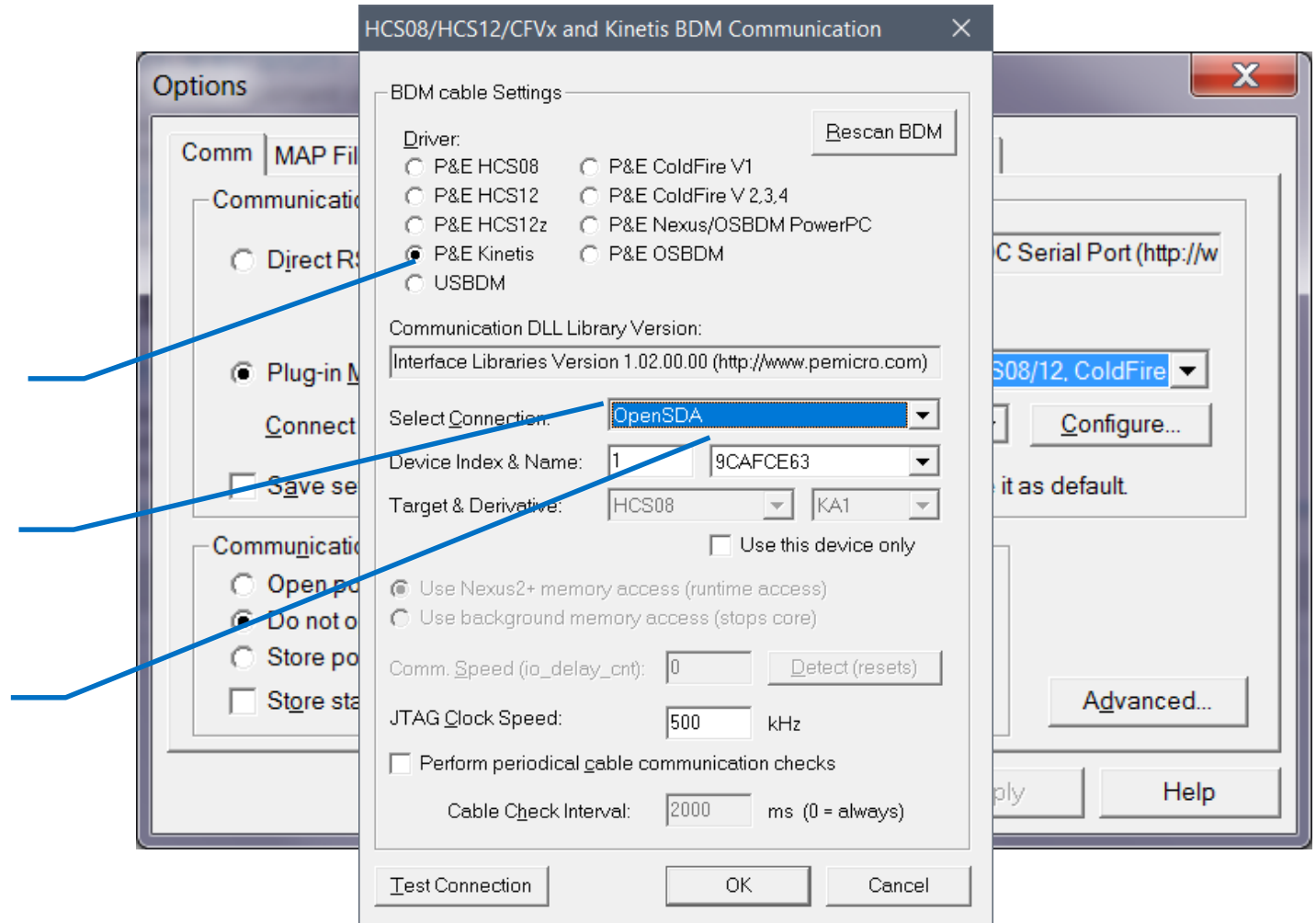
Set up connection (Comm Tab)

5. Select Connection Driver P&E Kinetis

6. Select Connection Interface USB Multilink

7. PE OpenSDA Device number should be found

FREEMASTER



Exercise #1 – Testing the Connection

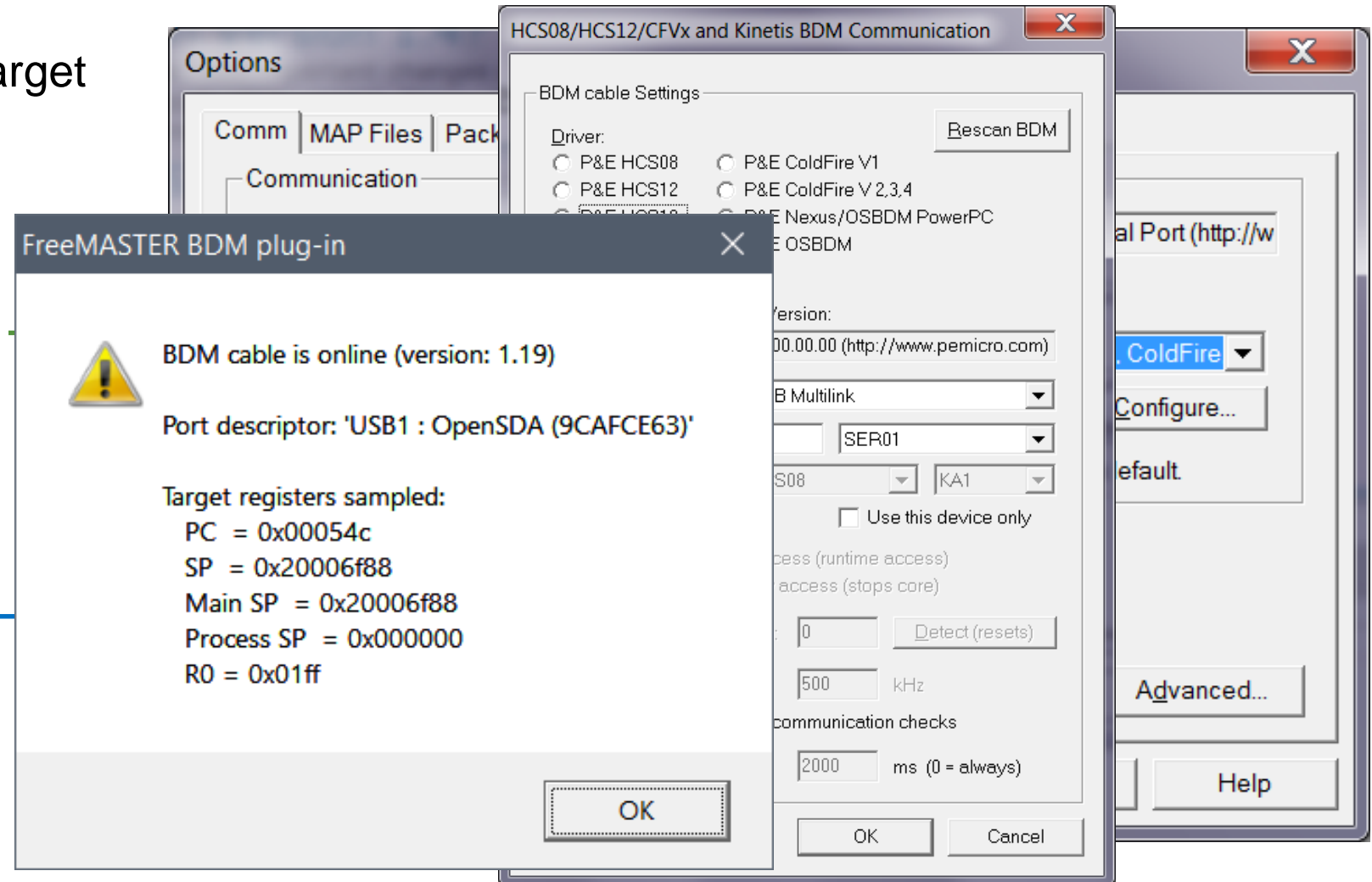
- Configure Freemaster for target board and project

Test the connection

If everything is good
...communications will be
indicated by the sampled
target registers

8. Click Test Connection

FREEMASTER



Exercise #1 – Set Up Symbol File

- Configure Freemaster for target board and project

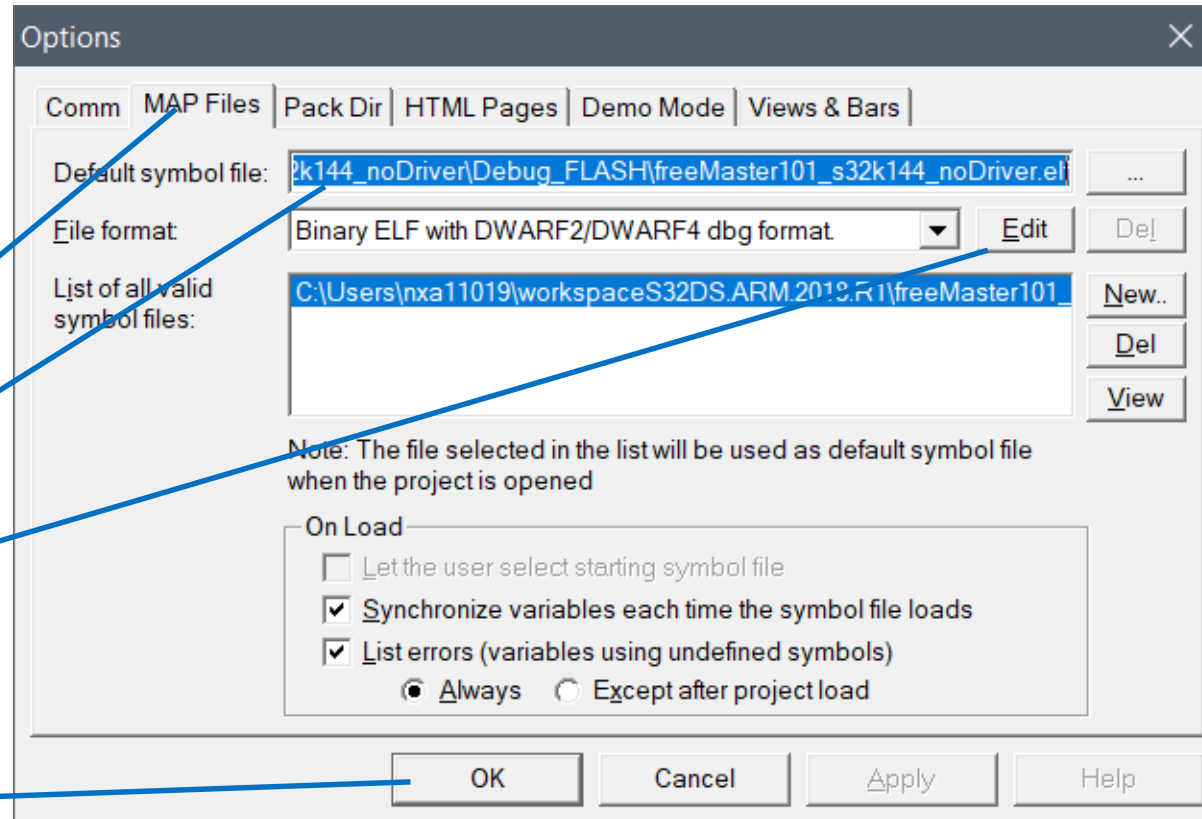
Set up symbol file (MAP files tab)

1 - Select MAP Files Tab

2 - Find symbol file from IDE

3 - Select File Format

Click OK



FREEMASTER

Exercise #1 – Start the Connection

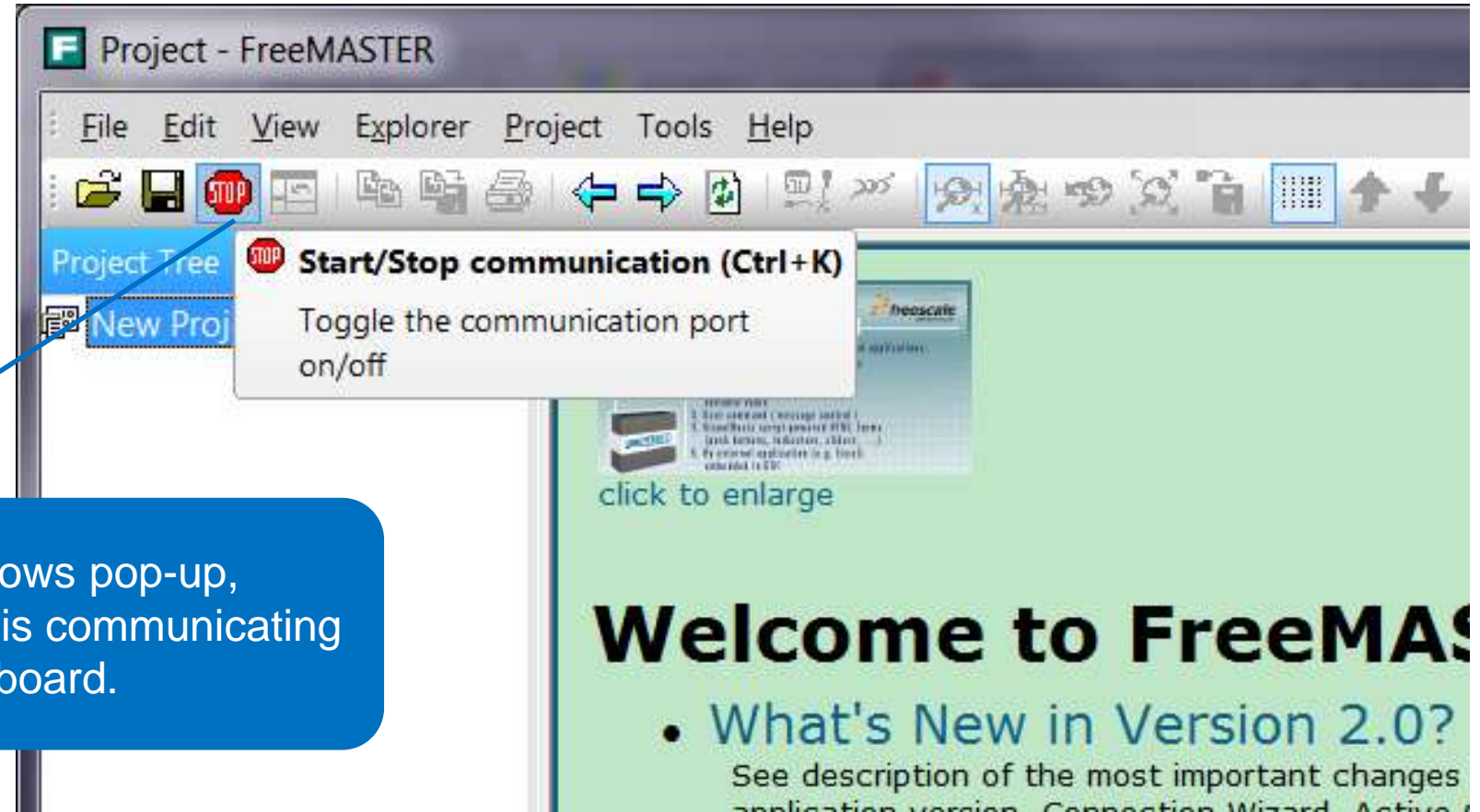
- Configure FreeMASTER for target board and project

Connect to the target board

1 – Start communication

If no error windows pop-up,
FREEMASTER is communicating
with the target board.

FREEMASTER



Exercise #2: Customizing the Project



Exercise #2 – Configuring Watch Variables

- counter2, counter2max – generic counter and limit
- sw2 – state of switch2 (PTC12)
- sw3 – state of switch3 (PTC13)
- ftmCounter - increments every FTM rollover
- ftmCounterLimit – sets time to execute motor algorithm
- freq_test – enables PWM testing
- freq – changes frequency during pwm testing.
- voltage – changes voltage during pwm testing.
- duty0enable – enables PWM connected to RED led
- duty0updown – duty cycle increments/decrements or increments
- BlueLED – connected to motor PWM
- GreeLED – connected to motor PWM
- adcRawValue – raw adc (potentiometer) reading
- adcValue – adc value in voltage (floating poin)

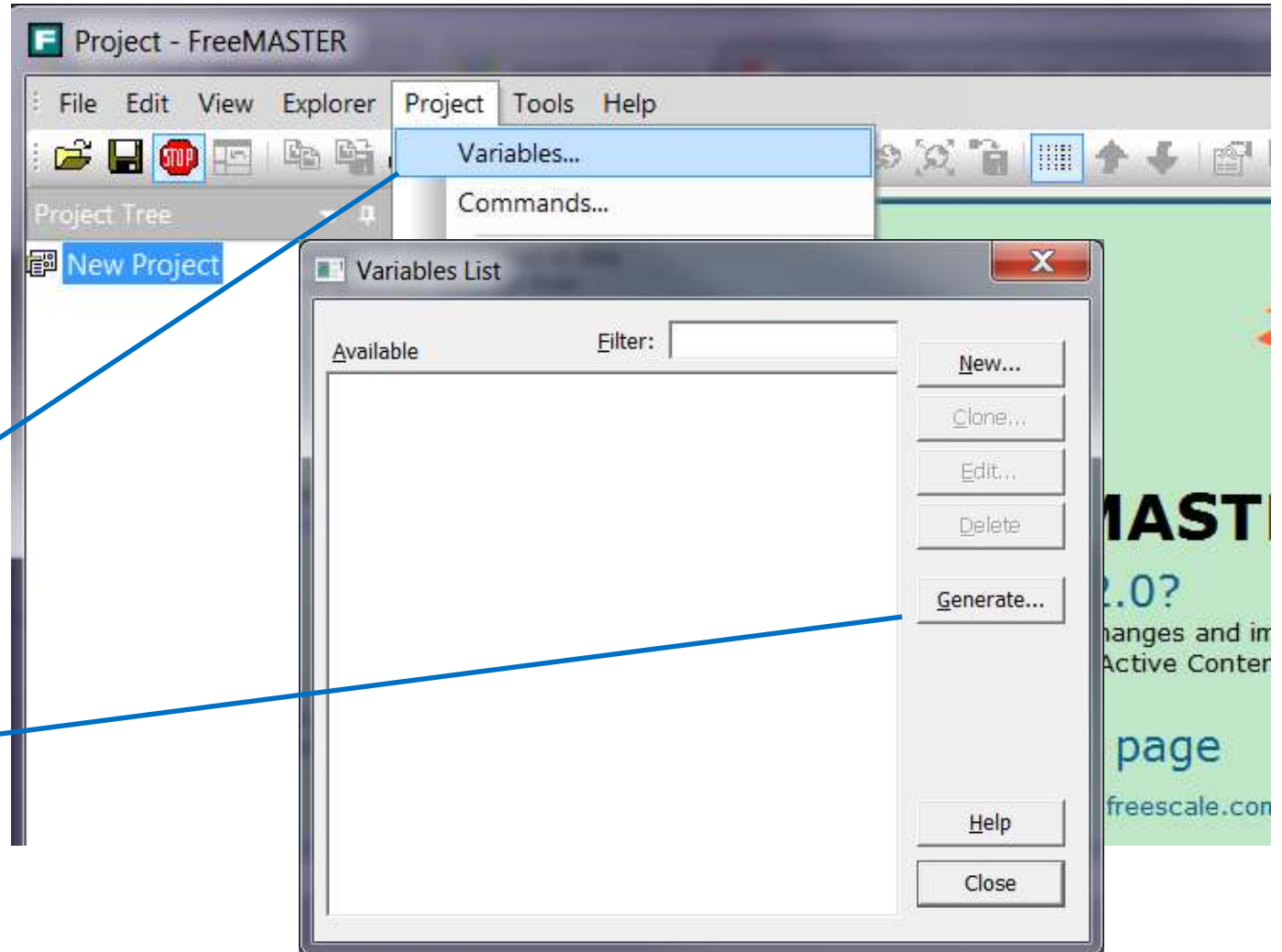
Exercise #2 – Configuring Watch Variables

- Generate links to project variables

1 – Select Project/Variables

2 – Select Generate

FreeMASTER



Exercise #2 – Configuring Watch Variables

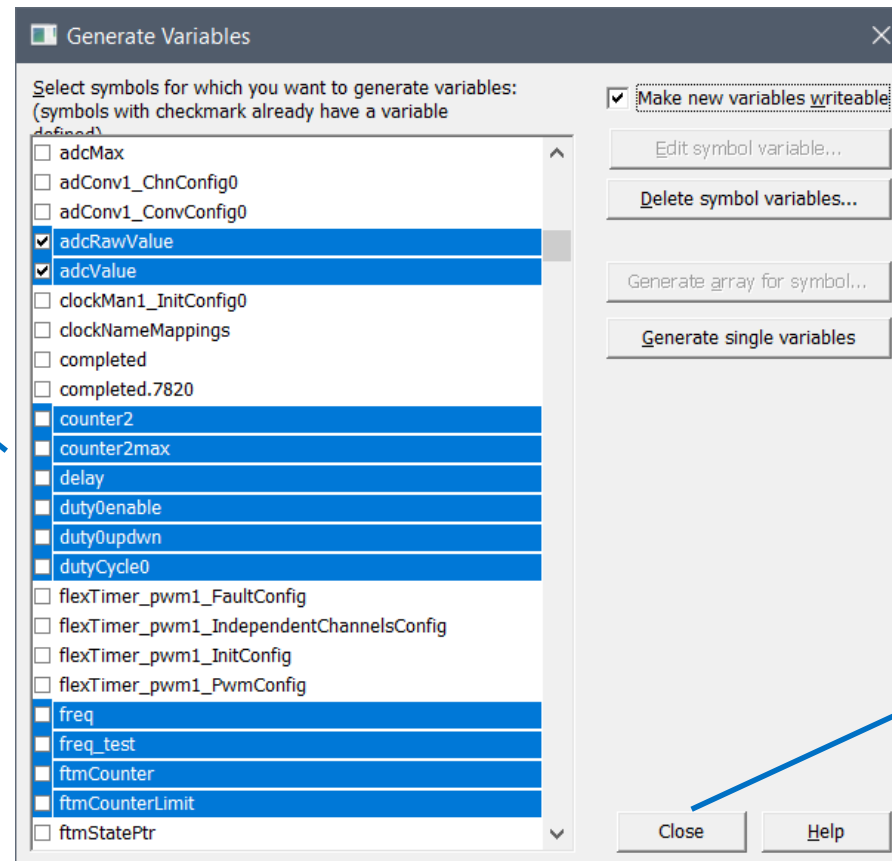
- Generate links to project variables

1. Select Desired Variables

2. Make sure they are writeable

3. Select Generate single variables

Close

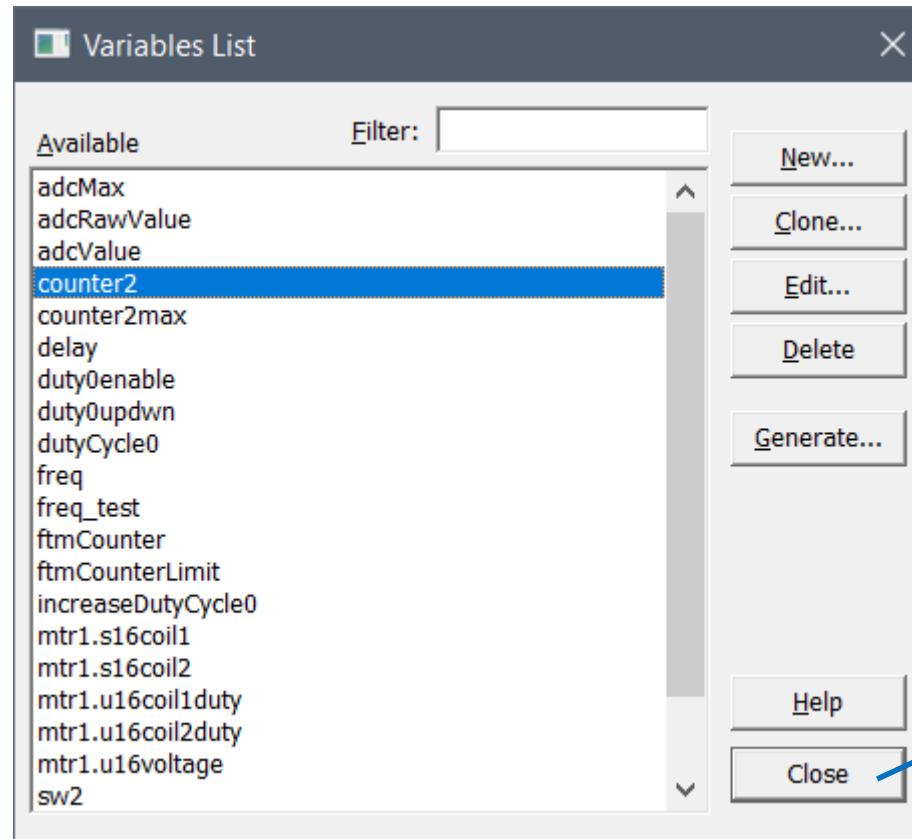


FREEMASTER

Exercise #2 – Configuring Watch Variables

- Generate links to project variables

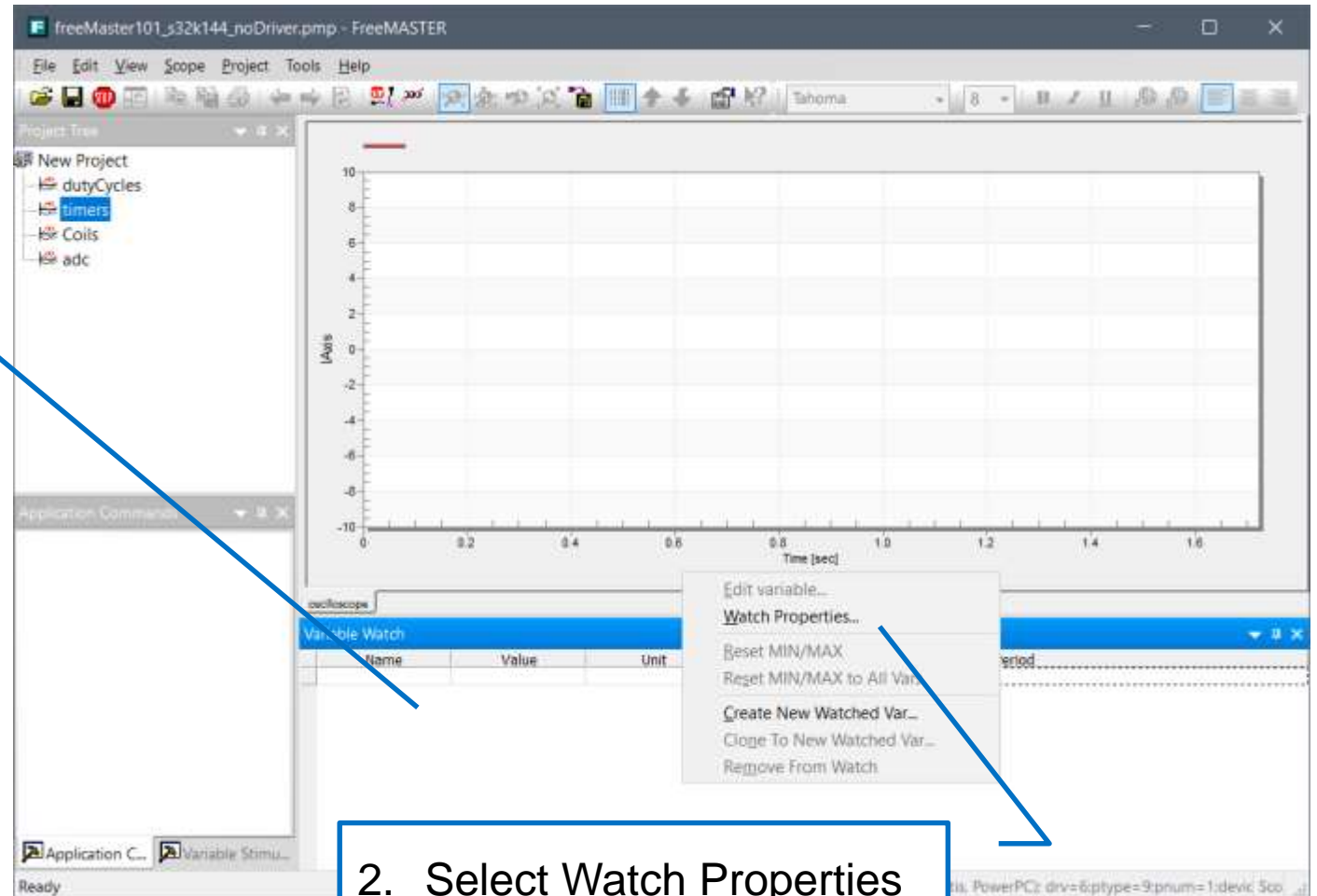
Available variables
are now listed



Exercise #2 – View Watch Variables

- Select which variables to view

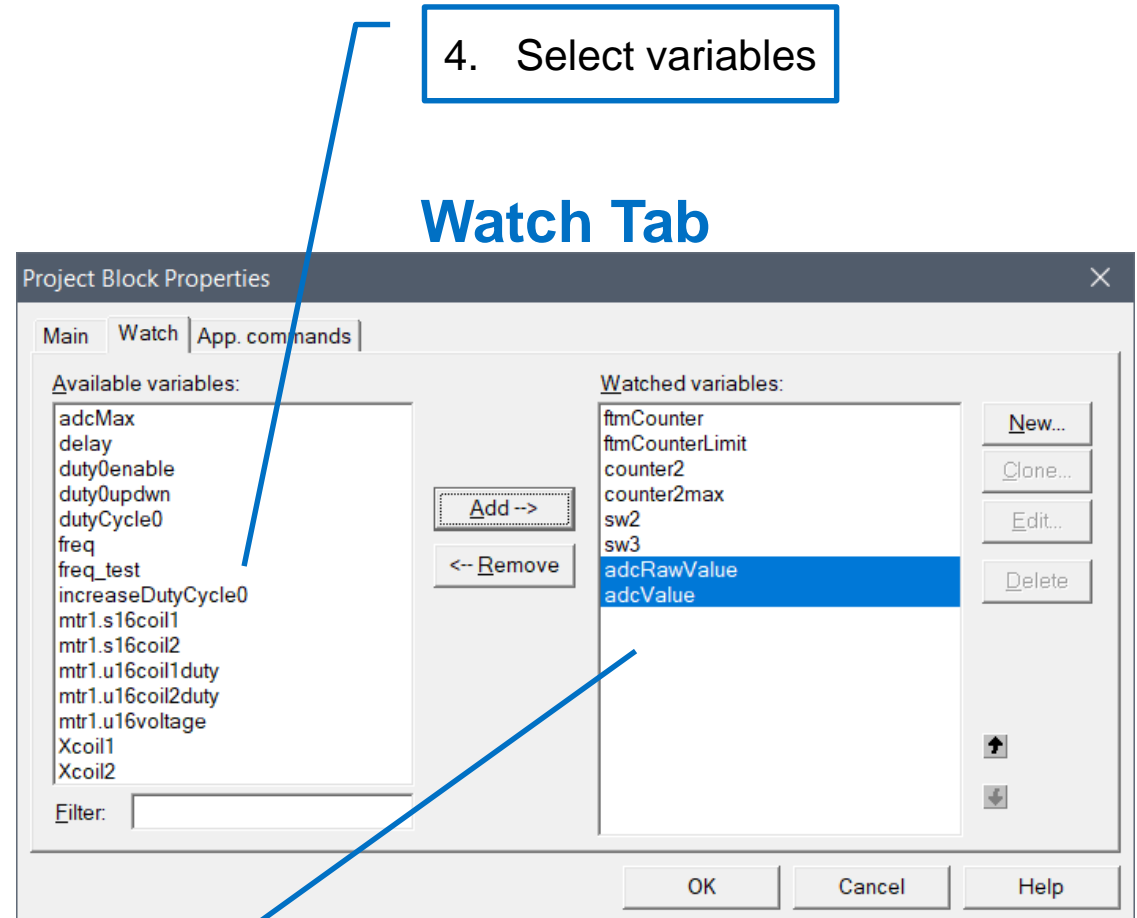
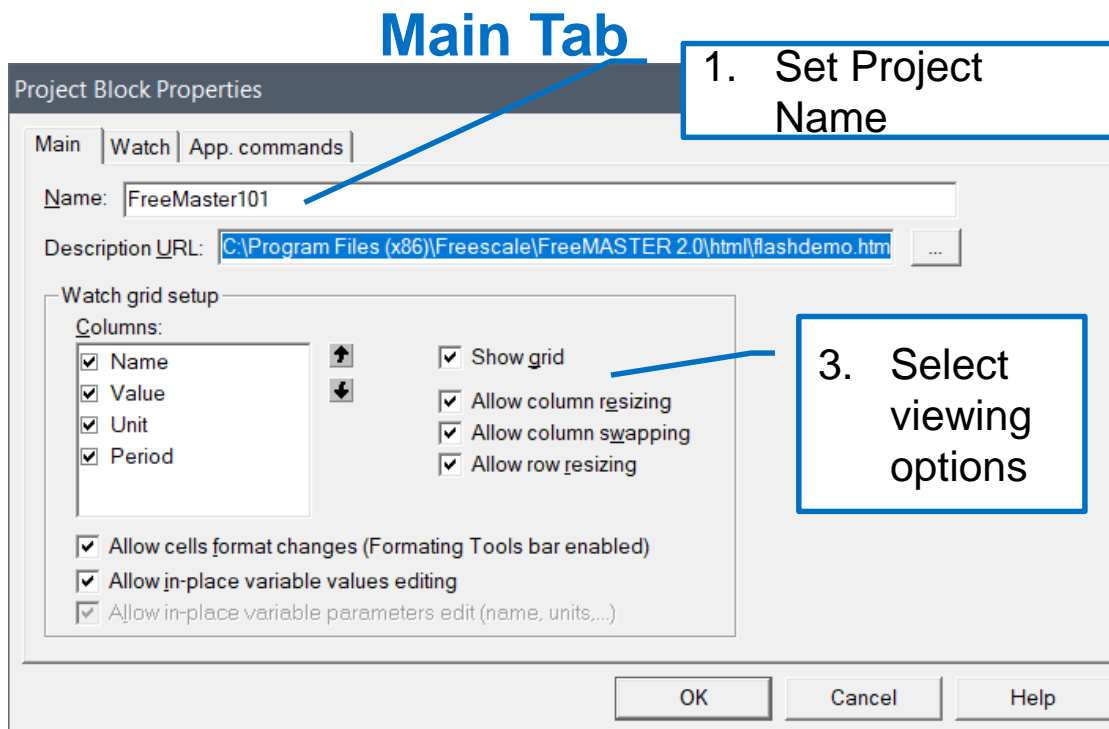
1. Right click in Variable Watch Pane



2. Select Watch Properties

Exercise #2 – View Watch Variables

- Select which variables to view



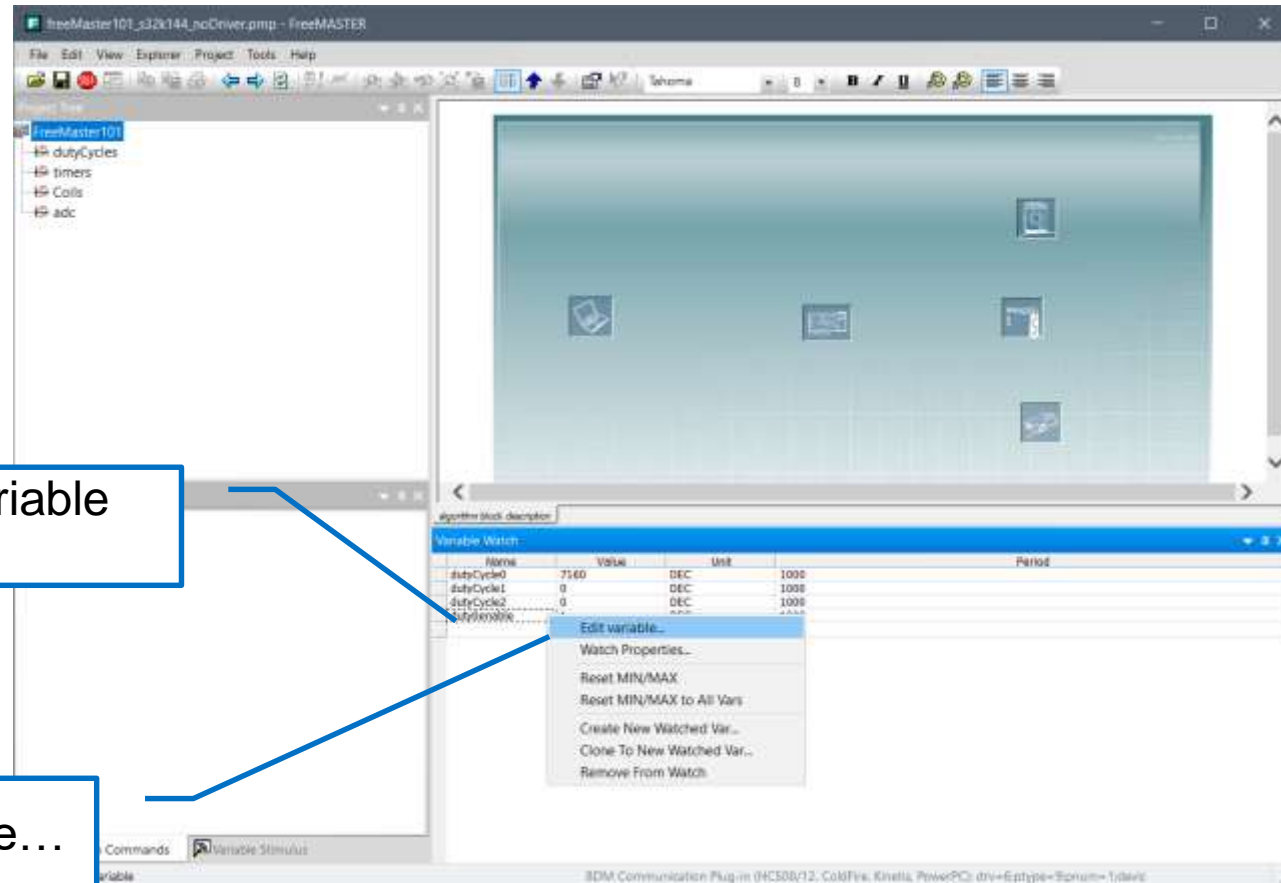
FreeMASTER

Exercise #2 – View Watch Variables

- Modify variable characteristics
- modify duty0enable

1. Right click variable to modify

2. Select Edit variable...



FREEMASTER

Exercise #2 – View Watch Variables

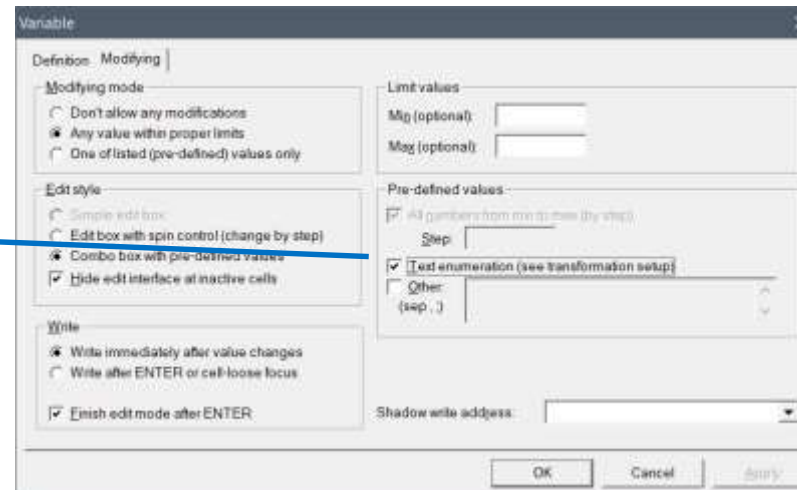
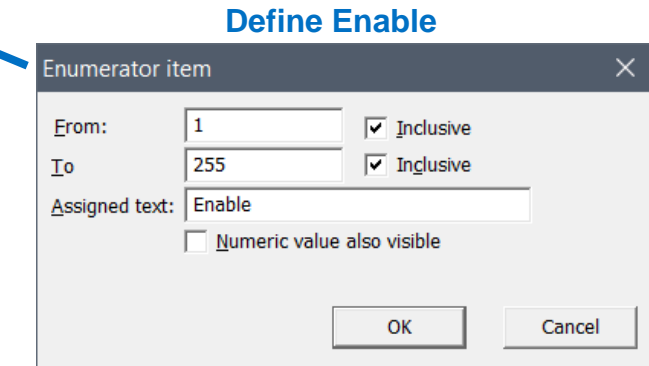
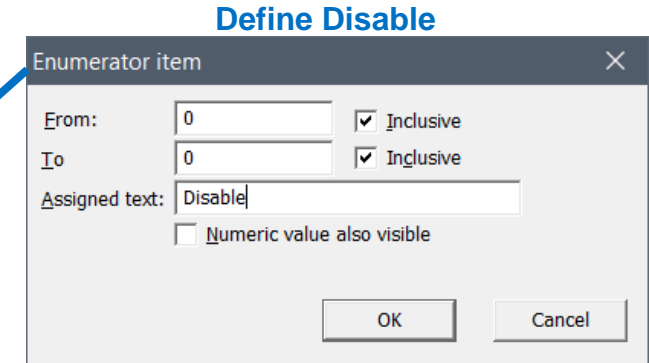
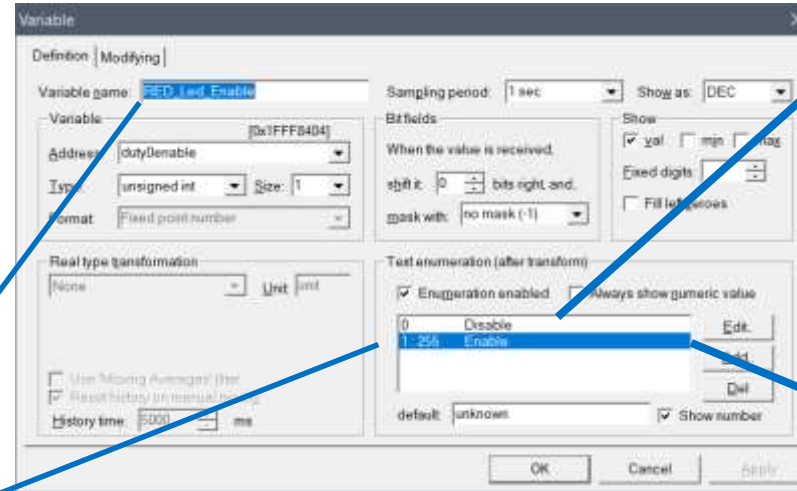
- Modify duty0enable
 - Change variable name
 - Set text enumeration

1. Change variable name

2. Change variable name
3. Add Enumeration Definitions

4. In Modifying Tab, Select Text enumerations

freemaster



Exercise #2 – View Watch Variables

- Try adding other variables
- Try changing colors, font and size of displayed variables
- Try variable transformations

(dutyCycle0: 0 = 0%, 32767 = 100%)

- counter2, counter2max – generic counter and limit
- sw2 – state of switch2 (PTC12)
- sw3 – state of switch3 (PTC13)
- ftmCounter - increments every FTM rollover
- ftmCounterLimit – sets time to execute motor algorithm
- freq_test – enables PWM testing
- freq – changes frequency during pwm testing.
- voltage – changes voltage during pwm testing.
- duty0enable – enables PWM connected to RED led
- duty0updown – duty cycle increments/decrements or increments
- BlueLED – connected to motor PWM
- GreeLED – connected to motor PWM
- adcRawValue – raw adc (potentiometer) reading
- adcValue – adc value in voltage (floating point)

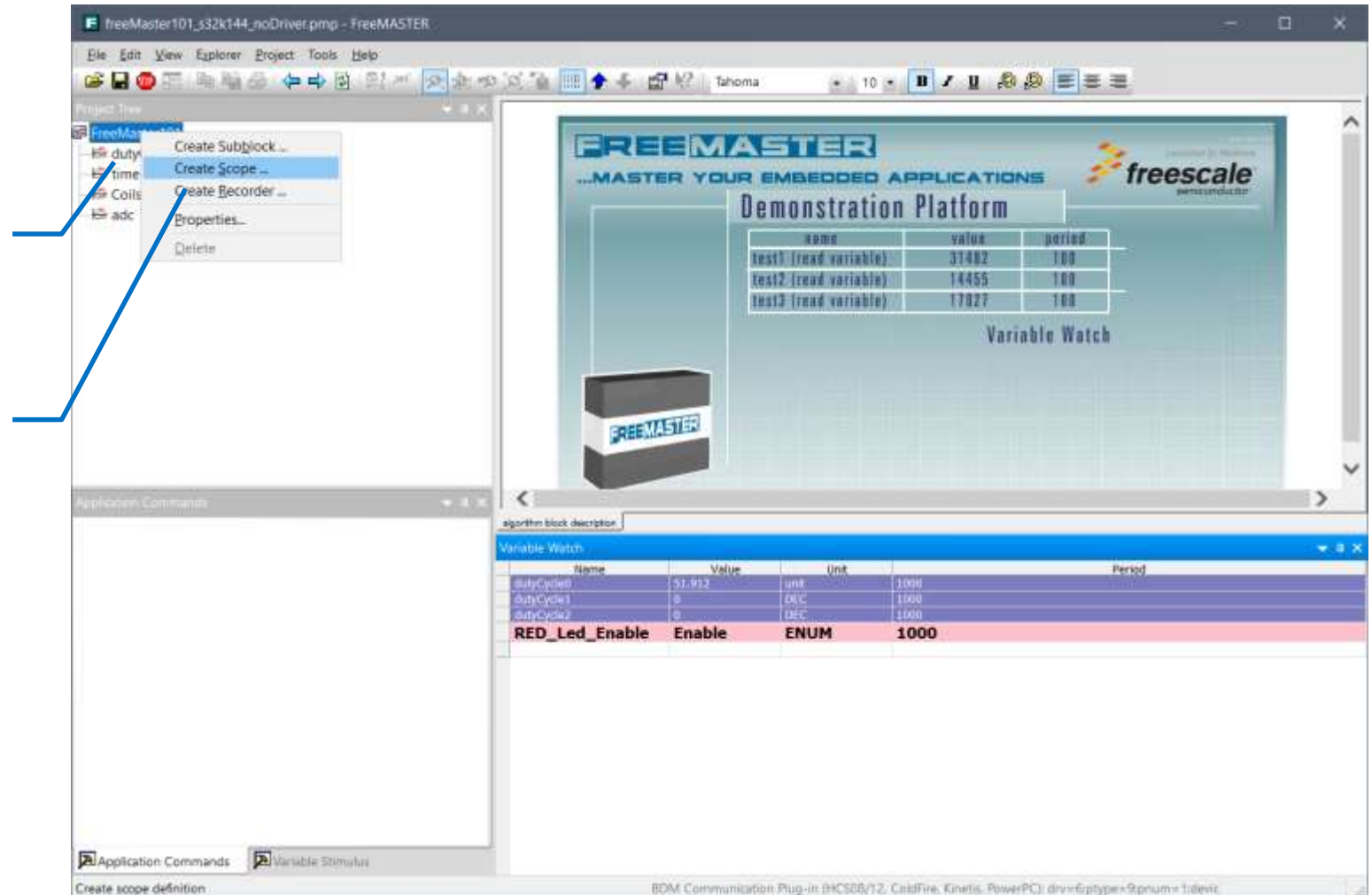


Exercise #2 – Configuring a Scope

- Create Scope

1. Right-click project name

2 – Select Create Scope



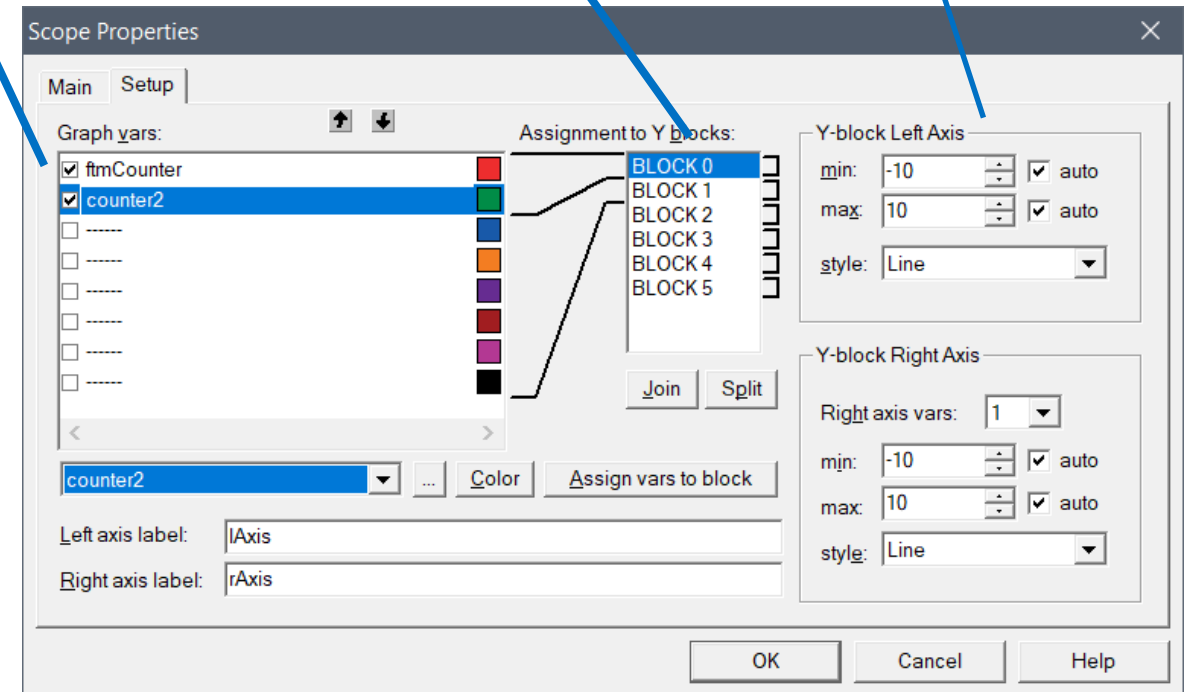
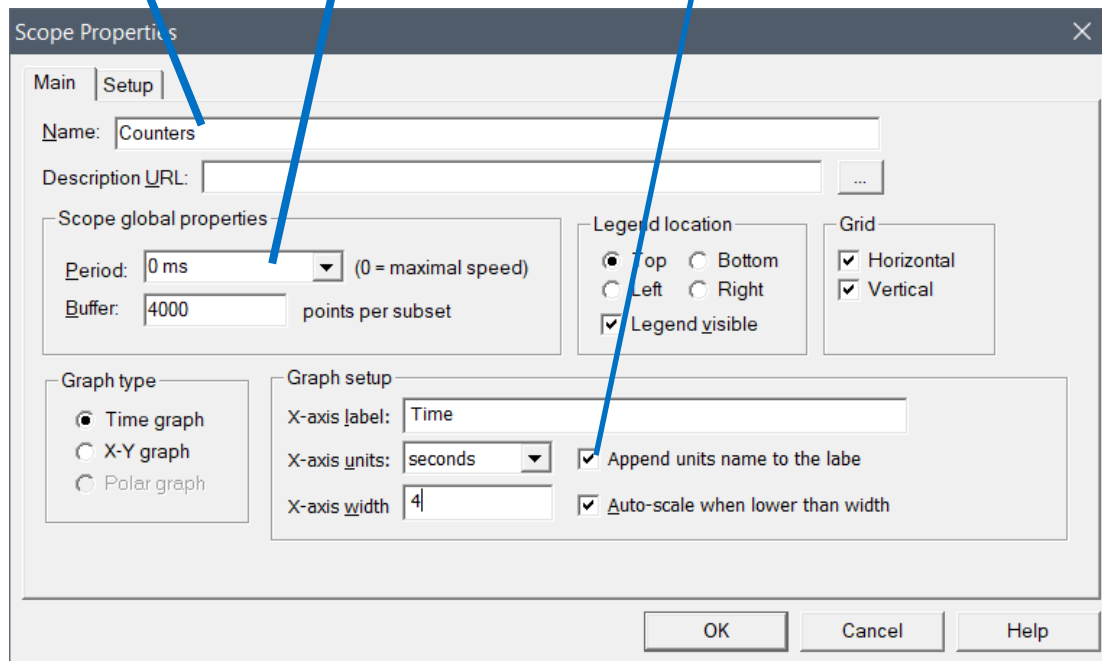
Example #2 – Configuring the Scope

In Main Tab:

1. Create scope name
2. Define sample speed
3. Define Axis width

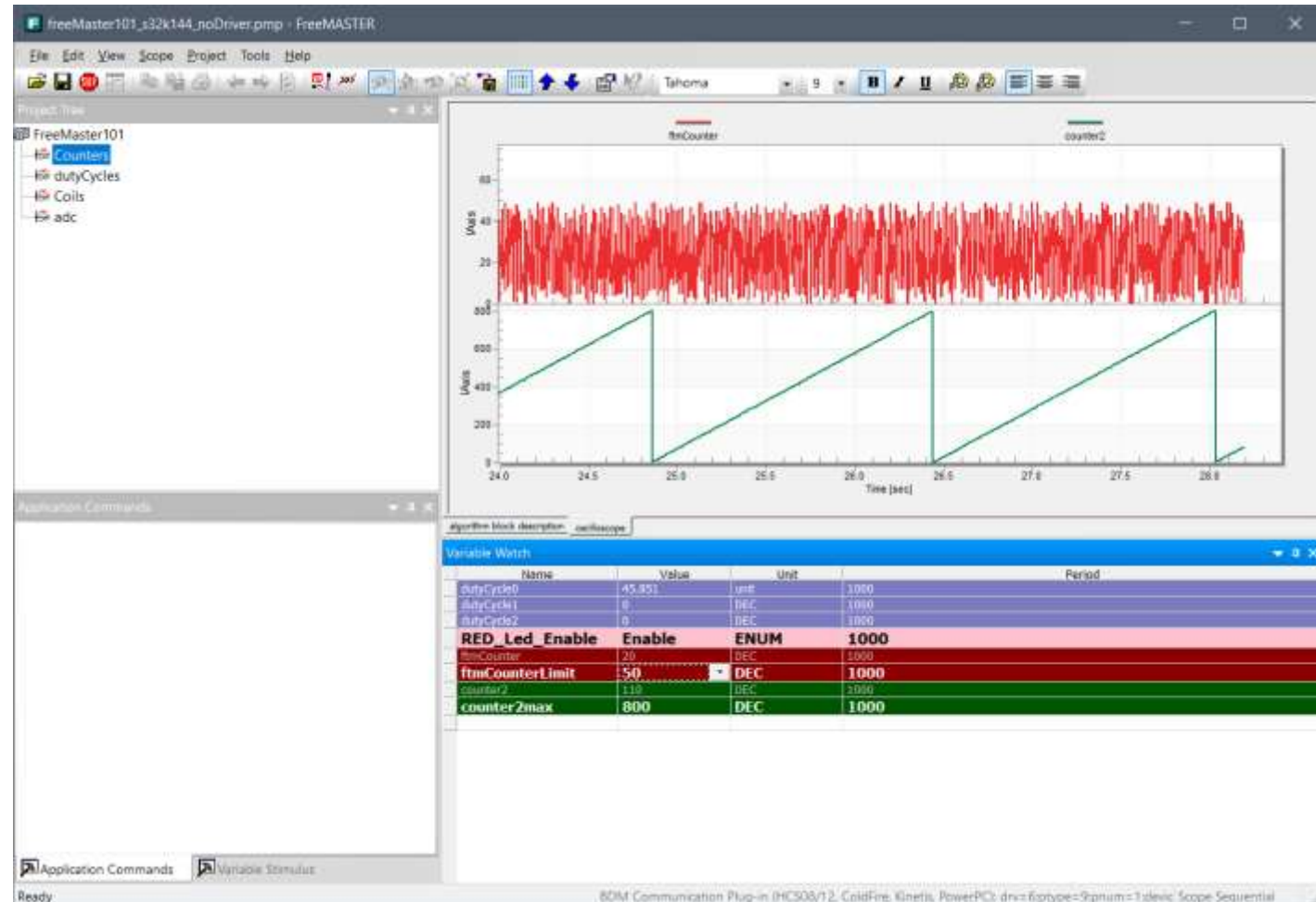
In Setup Tab:

1. Select Variables to graph
2. Define Blocks
3. Define Axis



Example #2 – Configuring the Scope

- Try changing scope properties and blocks
- Try changing `ftmCounterLimit` and `counter2max`
- Try setting right hand axis
- Try adding another scope

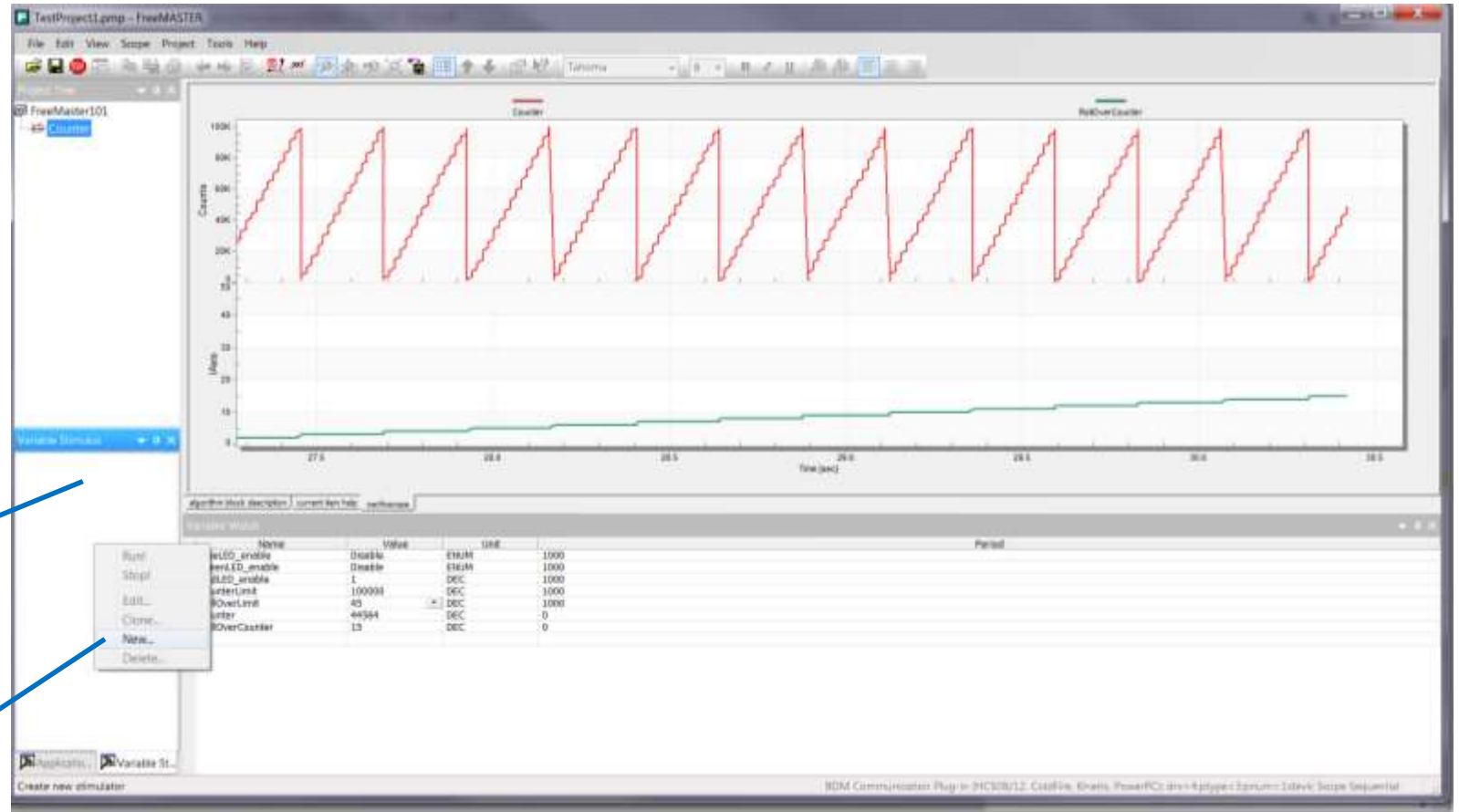


Example #2 – Set Up Variable Stimulus

- Add automatic control of a variable(s)

1. Right click in **Variable Stimulus** Pane

2. Select **New...**



Example #2 – Set Up Variable Stimulus

- Add automatic control of a variable(s)

1. Select **Stimulus** name

2. Select Variable to stimulate

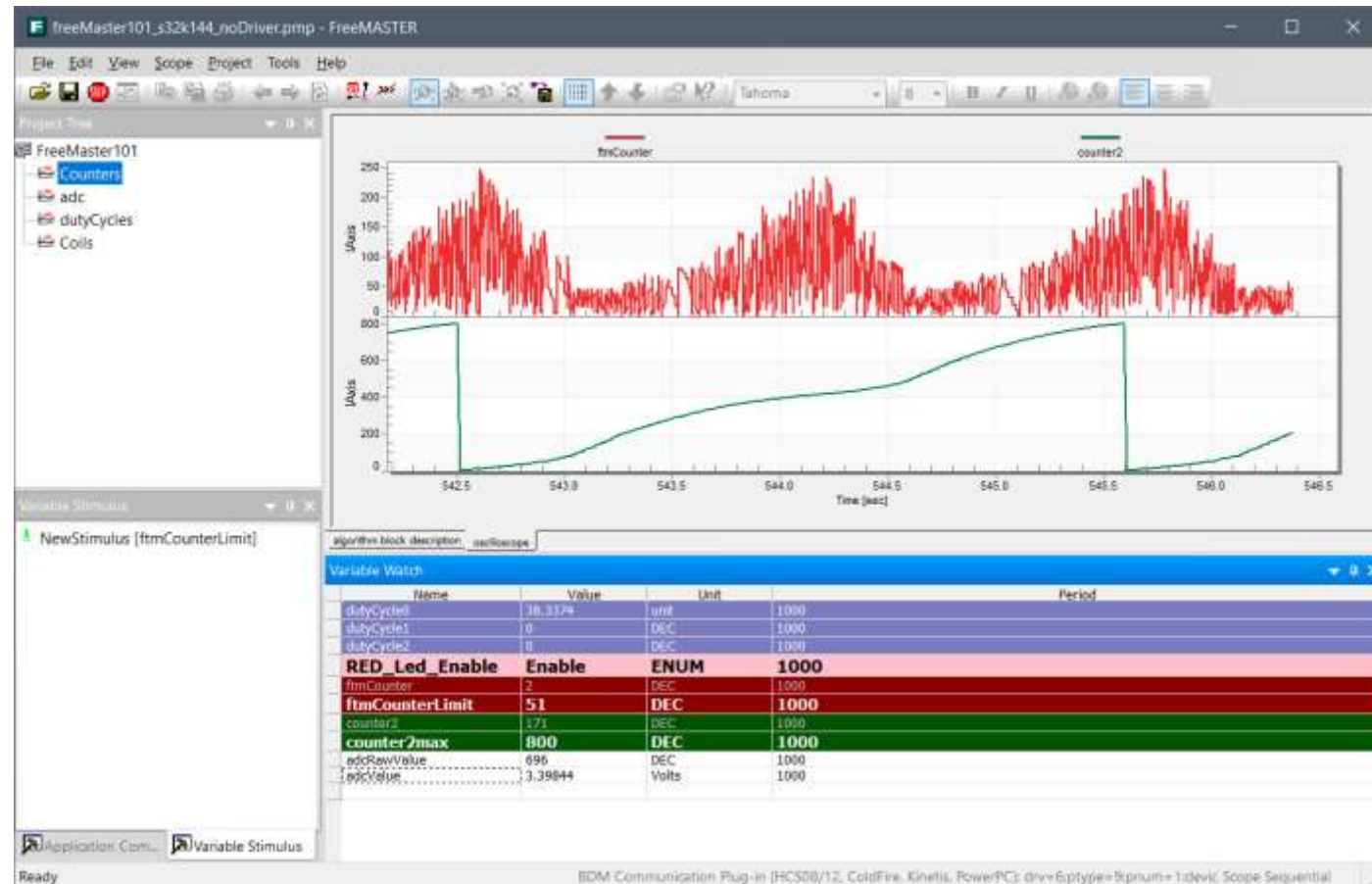
3. Set time points and value

The screenshot shows the 'Stimulator' dialog box. Annotations with blue lines point to specific fields: 'Name: NewStimulus' (labeled 1), 'Stimulated' dropdown set to 'ftmCounterLimit' (labeled 2), and the 'Run in the loop' checkbox (labeled 3). The 'Approximate time' is set to '100 ms'. The 'Tabular' section contains a table with time points and values. The 'Time' column has values '0', '500 millisec', '1 seconds', and '1 seconds 500 millisec'. The 'Value' column has values '50', '100', '250', and '25'. The last row is highlighted in blue. At the bottom, the 'Time' dropdown is set to '1 s 500 ms' and the 'Value' dropdown is set to '25'.

Time	Value
0	50
500 millisec	100
1 seconds	250
1 seconds 500 millisec	25

Example #2 – Set Up Variable Stimulus

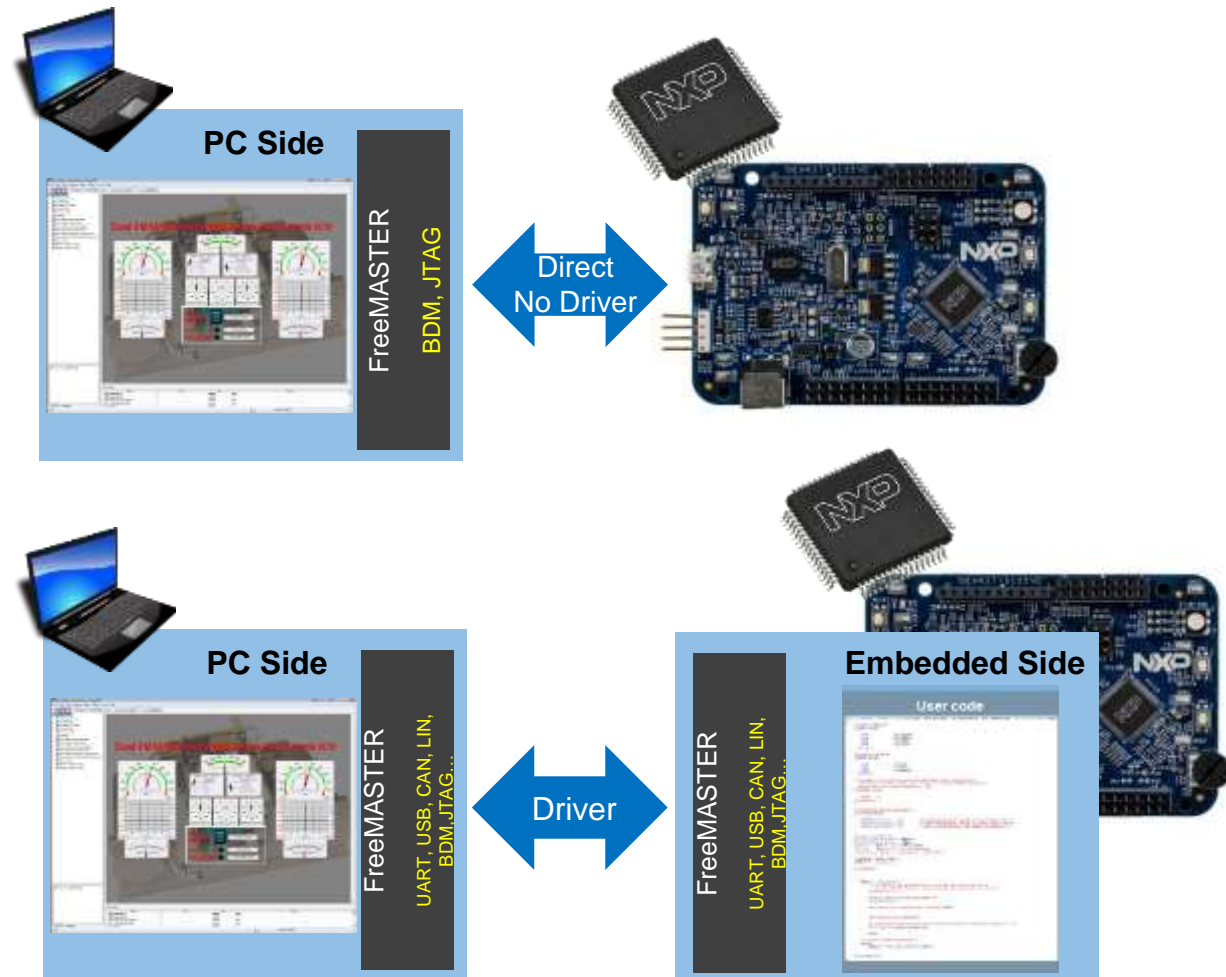
- Run the Stimulus!
- Try changing the stimulus times and iteration
- Try changing the stimulus variable



Exercises #1 and #2 – Complete!

Summary:

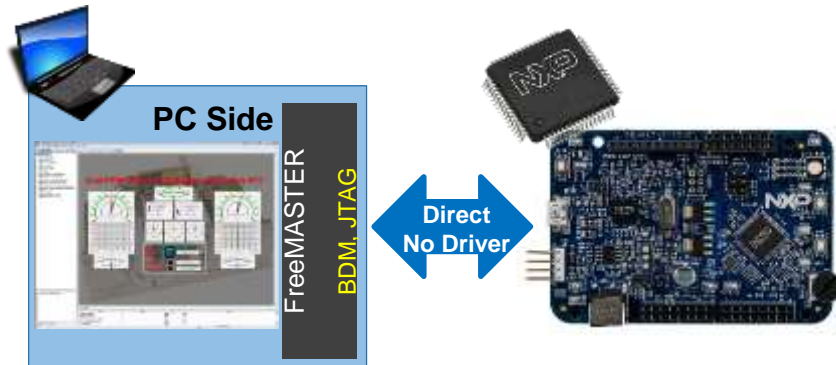
- FreeMASTER can be used without any SW modification (No Driver Option)
- FreeMASTER is a powerful debug and demonstration tool.
- FreeMASTER cannot be connected at the same time as a debugger unless the communication driver option is used.
- More features can be used (i.e. Application Commands, Recorder) if an embedded driver is used.



FreeMaster With Communication Driver



Why Add Freemaster Communication Driver to a Project?

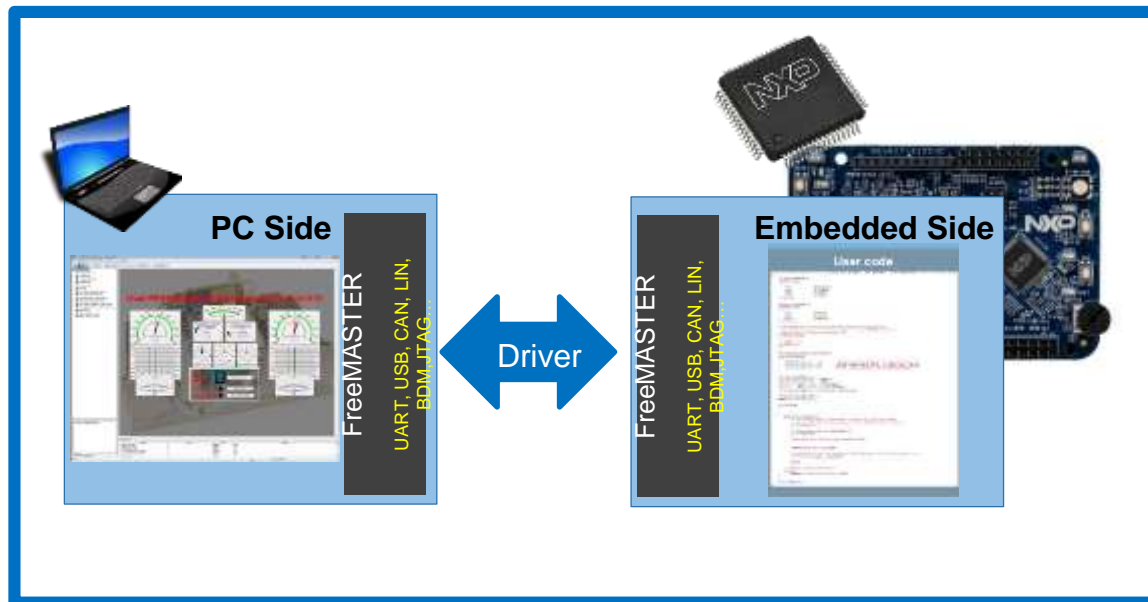


Application Commands

- Command codes and parameters are delivered to an application for arbitrary processing
- After command is processed, a response code can be sent back

Recorder

- Provides monitoring/visualization of application variables that are changing at a rate faster than the sampling rate of the oscilloscope.
- Created in SW on the Target board and stores changes of variables in real-time.
- Can define list of variables which will be recorded by the embedded side periodic service routine
- After the requested number of variable samples are stored, data is transferred to the FreeMASTER Recorder pane



Run debugger and Freemaster simultaneously!

- Use FreeMASTER to view or control variables real-time
- Use Debugger to Start/Stop code, single step and edit.
- Eliminates need to open and close connections!

Exercise #3: Adding the Comm Driver



Exercise #3 – Install the Freemaster Communication Drivers

- Go to:
www.nxp.com/Freemaster
- Select the **DOWNLOADS** tab

Download
FreeMASTER Communication Driver

Recommended Software & Tools (1)

FreeMASTER 2.0 Application Installation (REV 2.0.7)
IDE - Debug, Compile and Build Tools
EXE: 40.4 MB | FMASTERSW
2018-01-01 00:00:00
Download

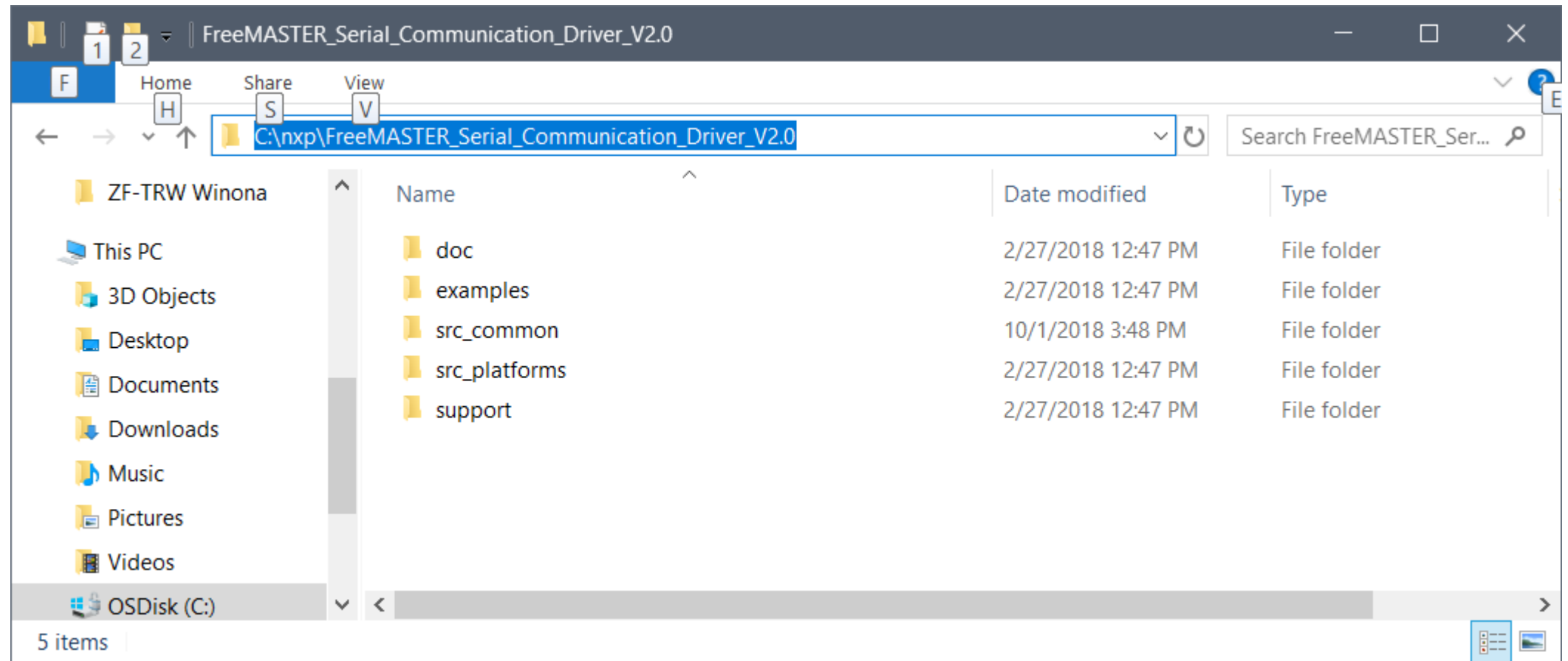
Device Drivers (1)

FreeMASTER Communication Driver (REV 2.0)
FreeMASTER Communication Driver for S06, HCS12, S12 Magniv®, 96F800E, MPC55xx/MPC56xx/MPC57xx, ColdFire® V1/V2, Kinlet®, S32
EXE: 5.3 MB | FMASTERGCDRV
09/20/2016
Download

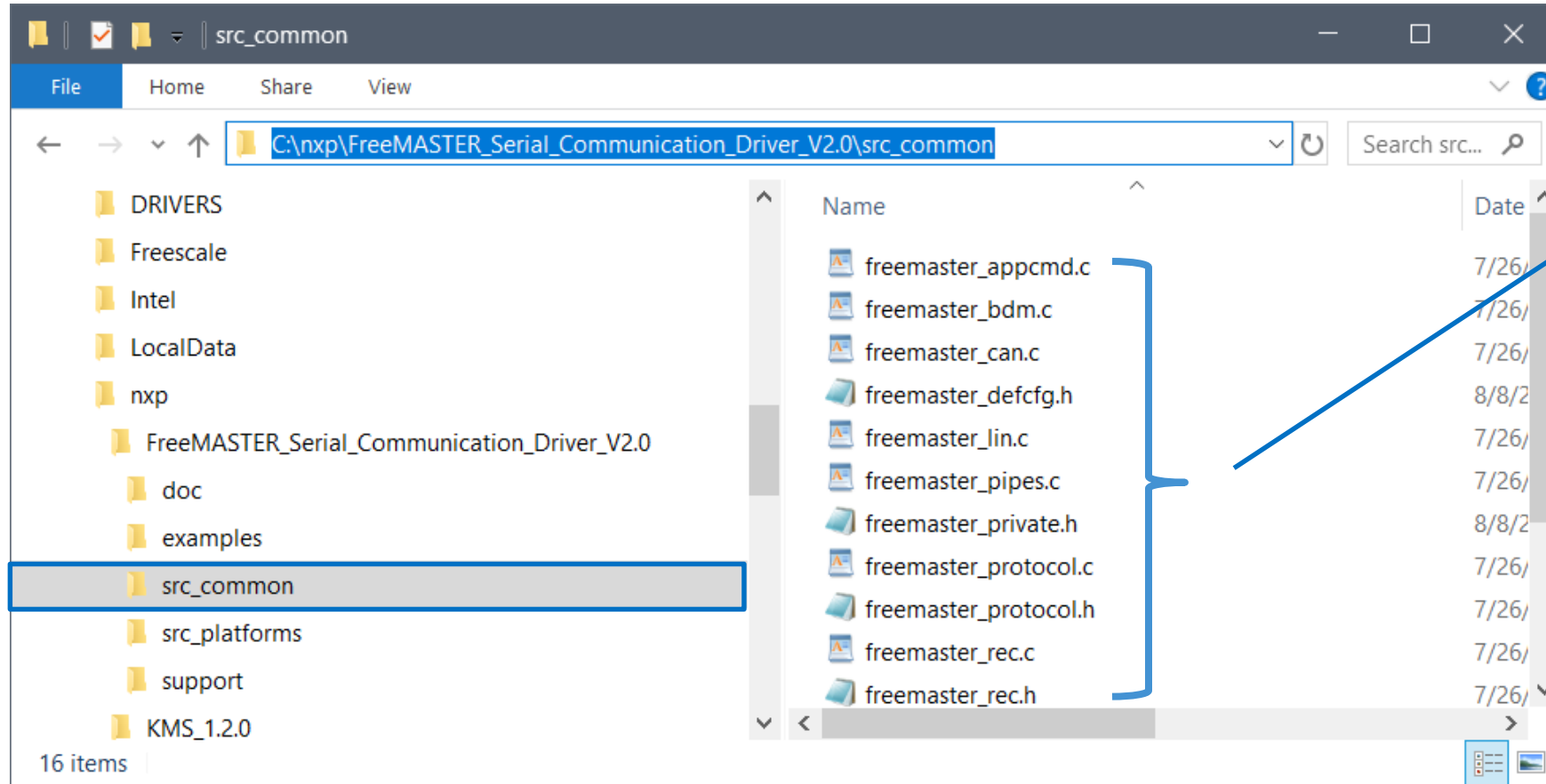
Exercise #3 – Install the Freemaster Communication Drivers

- After installing the drivers will be at:

C:\nxp\FreeMASTER_Serial_Communication_Driver_V2.0

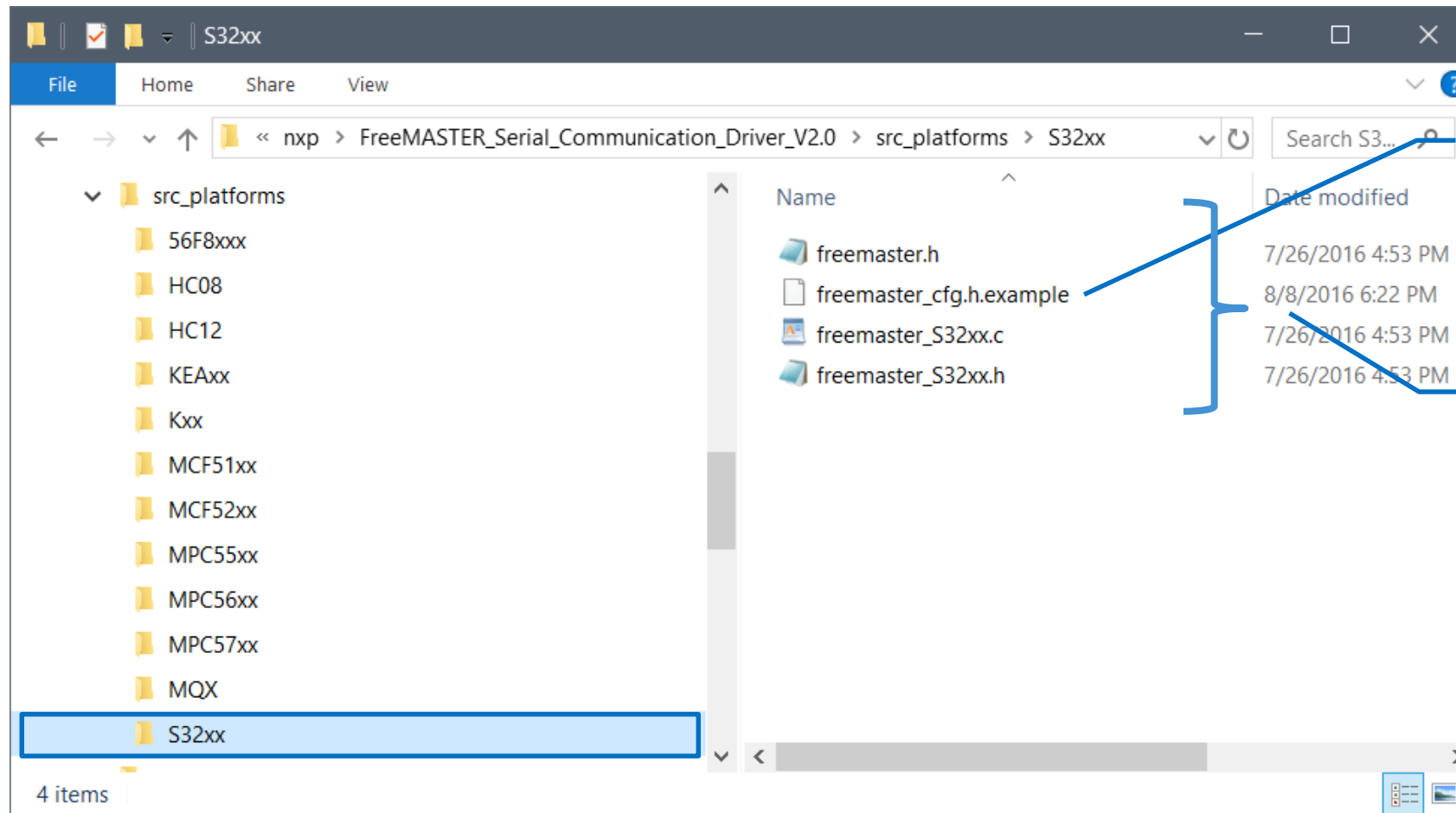


Exercise #3 – Copy Driver Files to Your Project



Copy all files from
src_common

Exercise #3 – Copy Driver Files to Your Project



Rename
freemaster_cfg.h.example
to
freemaster_cfg.h

Copy all files from
src_platforms/S32xx

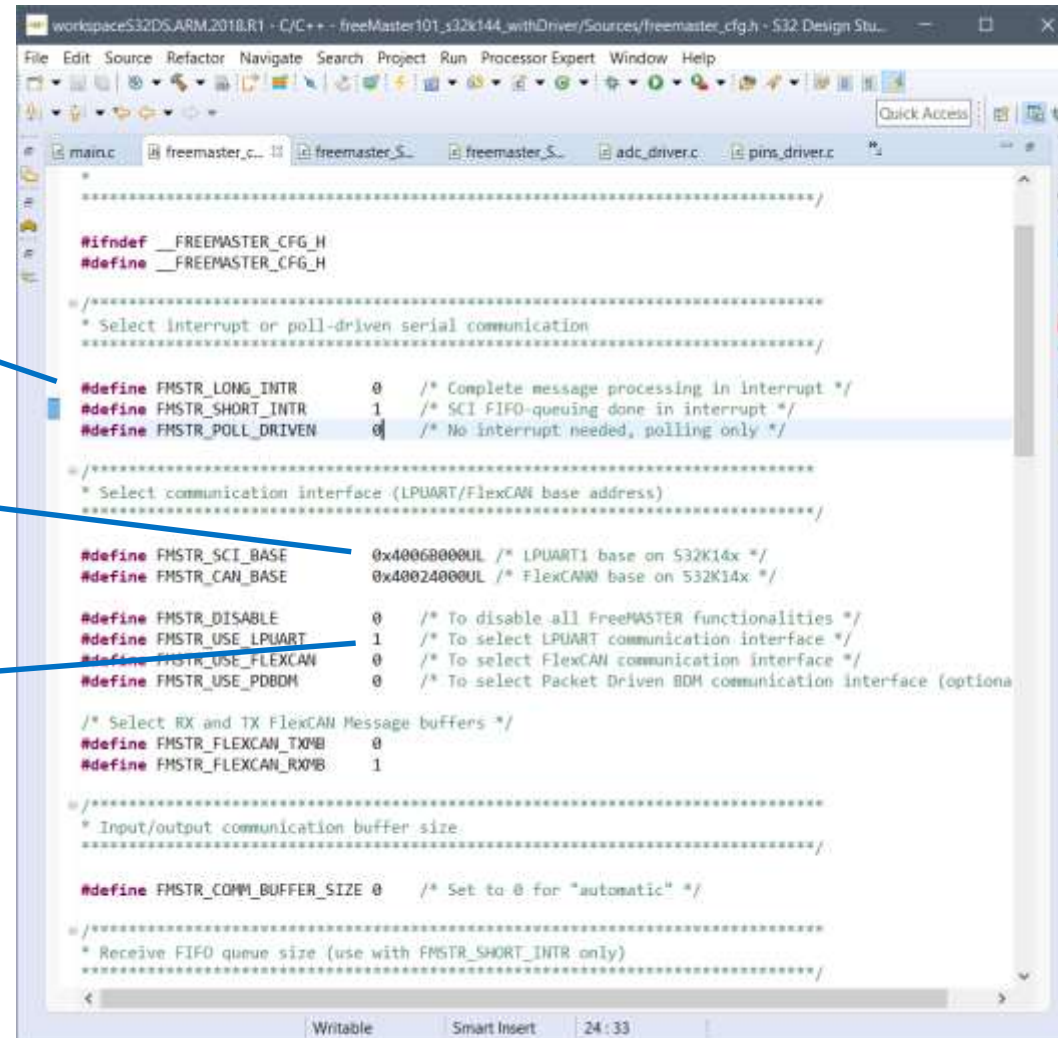
Exercise #3 – Configure the Driver – freemaster_cfg.h

- Edit freemaster_cfg.h

Define as
FMSTR_SHORT_INTR

Set UART address
based on device

Set to use UART



```

/*=====*/
#ifndef __FREEMASTER_CFG_H
#define __FREEMASTER_CFG_H

/*=====*/
/* Select interrupt or poll-driven serial communication
/*=====*/
#define FMSTR_LONG_INTR    0    /* Complete message processing in interrupt */
#define FMSTR_SHORT_INTR   1    /* SCI FIFO-queuing done in interrupt */
#define FMSTR_POLL_DRIVEN  0    /* No interrupt needed, polling only */

/*=====*/
/* Select communication interface (LPUART/FlexCAN base address)
/*=====*/
#define FMSTR_SCI_BASE      0x4006B000UL /* LPUART1 base on S32K14x */
#define FMSTR_CAN_BASE     0x40024000UL /* FlexCAN base on S32K14x */

#define FMSTR_DISABLE       0    /* To disable all FreeMASTER functionalities */
#define FMSTR_USE_LPUART    1    /* To select LPUART communication interface */
#define FMSTR_USE_FLEXCAN   0    /* To select FlexCAN communication interface */
#define FMSTR_USE_PDBDM     0    /* To select Packet Driven BDM communication interface (optional)

/* Select RX and TX FlexCAN Message buffers */
#define FMSTR_FLEXCAN_TXMB  0
#define FMSTR_FLEXCAN_RXMB  1

/*=====*/
/* Input/output communication buffer size
/*=====*/
#define FMSTR_COMM_BUFFER_SIZE 0    /* Set to 0 for "automatic" */

/*=====*/
/* Receive FIFO queue size (use with FMSTR_SHORT_INTR only)
/*=====*/

```

The screenshot shows the code editor for `freemaster_cfg.h`. Three blue boxes on the left contain instructions, with blue lines pointing to specific lines in the code: 1. 'Define as FMSTR_SHORT_INTR' points to the line `#define FMSTR_SHORT_INTR 1`. 2. 'Set UART address based on device' points to the line `#define FMSTR_SCI_BASE 0x4006B000UL`. 3. 'Set to use UART' points to the line `#define FMSTR_USE_LPUART 1`. The code includes various preprocessor directives for configuring the FreeMASTER driver, such as selecting the communication interface (LPUART or FlexCAN) and buffer sizes.

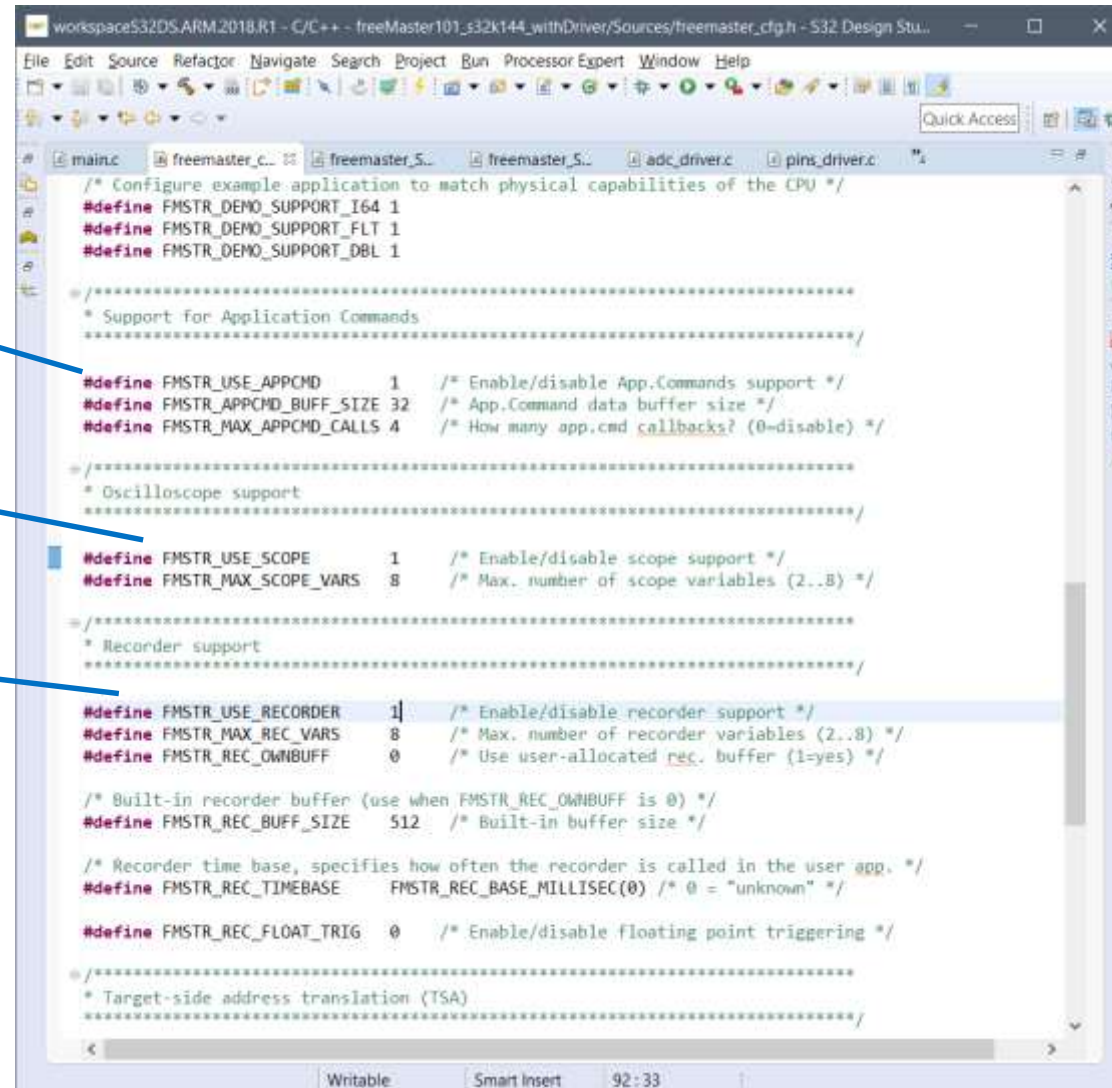
Exercise #3 – Configure the Driver – freemaster_cfg.h

- Edit freemaster_cfg.h

Enable APPCMDs

Enable SCOPE usage

Enable Recorder



The screenshot shows the `freemaster_cfg.h` file in a code editor. Three blue boxes on the left contain the text "Enable APPCMDs", "Enable SCOPE usage", and "Enable Recorder". Blue lines point from these boxes to the corresponding configuration lines in the code. The code defines various macros for configuring the driver, including support for application commands, scope usage, and recorder support.

```
/* Configure example application to match physical capabilities of the CPU */
#define FMSTR_DEMO_SUPPORT_I64 1
#define FMSTR_DEMO_SUPPORT_FLT 1
#define FMSTR_DEMO_SUPPORT_DBL 1

/* =====
 * Support for Application Commands
 * ===== */

#define FMSTR_USE_APPCMD 1 /* Enable/disable App.Commands support */
#define FMSTR_APPCMD_BUFF_SIZE 32 /* App.Command data buffer size */
#define FMSTR_MAX_APPCMD_CALLS 4 /* How many app.cmd callbacks? (0=disable) */

/* =====
 * Oscilloscope support
 * ===== */

#define FMSTR_USE_SCOPE 1 /* Enable/disable scope support */
#define FMSTR_MAX_SCOPE_VARS 8 /* Max. number of scope variables (2..8) */

/* =====
 * 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 512 /* Built-in buffer size */

/* Recorder time base, specifies how often the recorder is called in the user app. */
#define FMSTR_REC_TIMEBASE FMSTR_REC_BASE_MILLISEC(0) /* 0 = "unknown" */

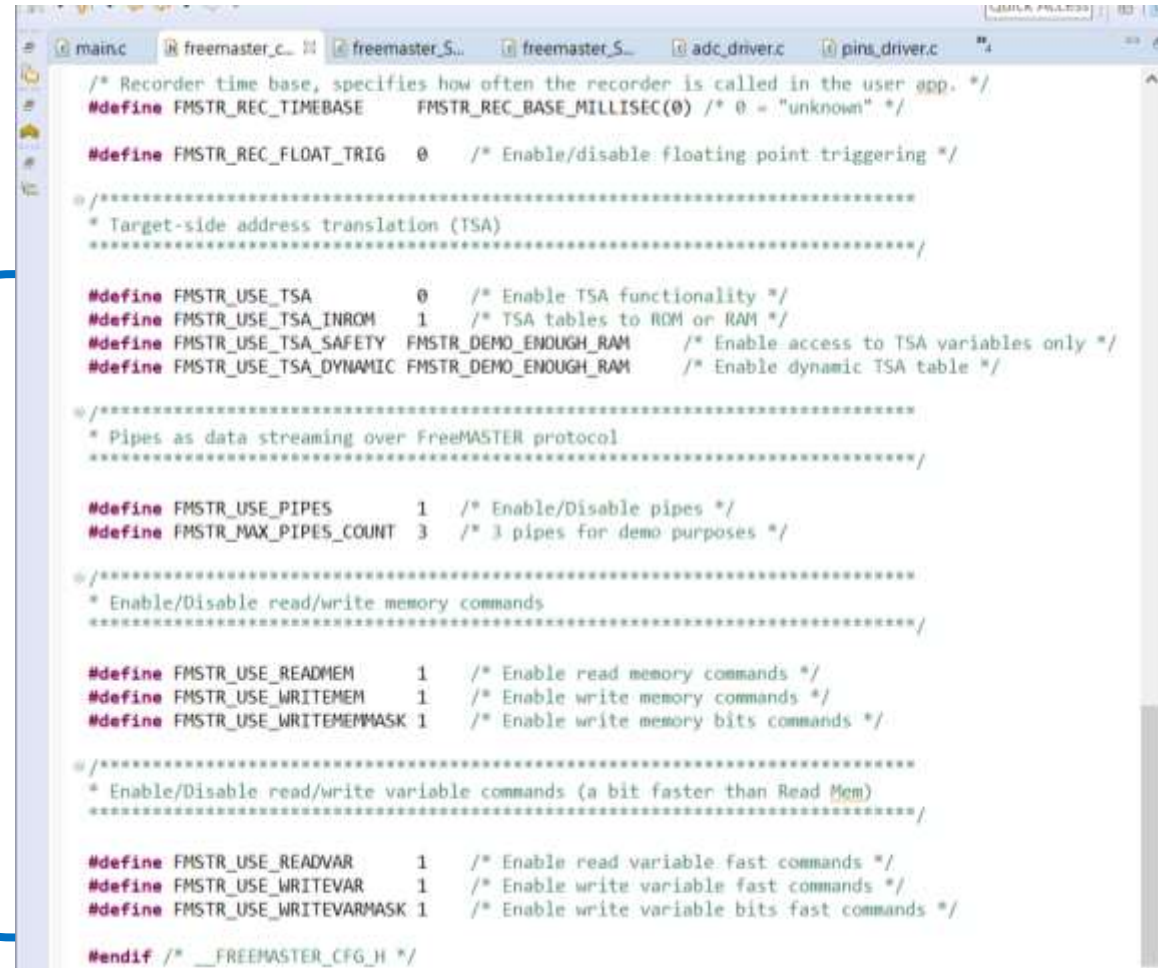
#define FMSTR_REC_FLOAT_TRIG 0 /* Enable/disable floating point triggering */

/* =====
 * Target-side address translation (TSA)
 * ===== */
```

Exercise #3 – Configure the Driver – freemaster_cfg.h

- Edit freemaster_cfg.h

Enable Memory and
Variable access



```
/* Recorder time base, specifies how often the recorder is called in the user app. */
#define FMSTR_REC_TIMEBASE FMSTR_REC_BASE_MILLISEC(0) /* 0 = "unknown" */

#define FMSTR_REC_FLOAT_TRIG 0 /* Enable/disable floating point triggering */

/* =====
 * Target-side address translation (TSA)
 * ===== */

#define FMSTR_USE_TSA 0 /* Enable TSA functionality */
#define FMSTR_USE_TSA_INROM 1 /* TSA tables to ROM or RAM */
#define FMSTR_USE_TSA_SAFETY FMSTR_DEMO_ENOUGH_RAM /* Enable access to TSA variables only */
#define FMSTR_USE_TSA_DYNAMIC FMSTR_DEMO_ENOUGH_RAM /* Enable dynamic TSA table */

/* =====
 * Pipes as data streaming over FreeMASTER protocol
 * ===== */

#define FMSTR_USE_PIPES 1 /* Enable/Disable pipes */
#define FMSTR_MAX_PIPES_COUNT 3 /* 3 pipes for demo purposes */

/* =====
 * Enable/Disable read/write memory commands
 * ===== */

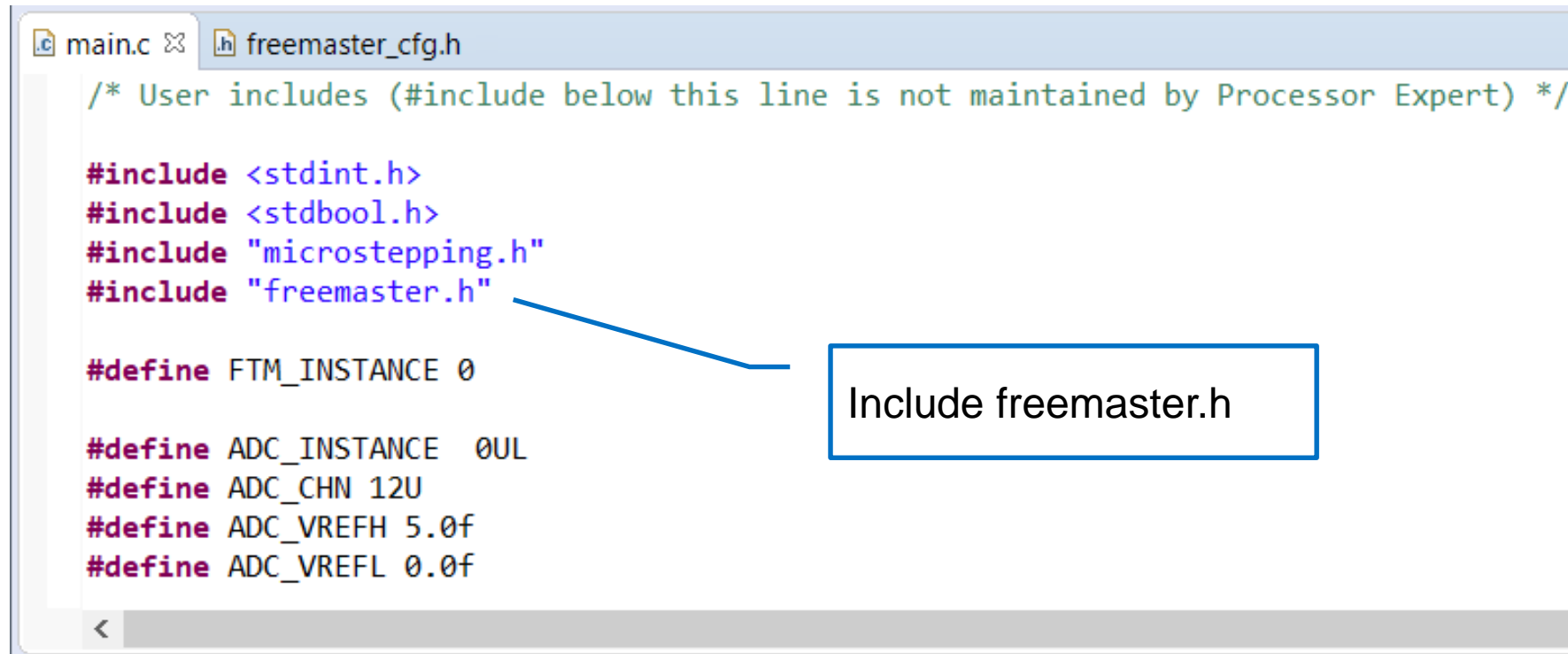
#define FMSTR_USE_READMEM 1 /* Enable read memory commands */
#define FMSTR_USE_WRITEVAR 1 /* Enable write memory commands */
#define FMSTR_USE_WRITEVARMASK 1 /* Enable write memory bits commands */

/* =====
 * Enable/Disable read/write variable commands (a bit faster than Read Mem)
 * ===== */

#define FMSTR_USE_READVAR 1 /* Enable read variable fast commands */
#define FMSTR_USE_WRITEVAR 1 /* Enable write variable fast commands */
#define FMSTR_USE_WRITEVARMASK 1 /* Enable write variable bits fast commands */

#endif /* __FREEMASTER_CFG_H */
```

Exercise #3 – Include freemaster.h



```
main.c x freemaster_cfg.h
/* User includes (#include below this line is not maintained by Processor Expert) */

#include <stdint.h>
#include <stdbool.h>
#include "microstepping.h"
#include "freemaster.h"

#define FTM_INSTANCE 0

#define ADC_INSTANCE 0UL
#define ADC_CHN 12U
#define ADC_VREFH 5.0f
#define ADC_VREFL 0.0f
```

Include freemaster.h

Exercise #3 – Configure the UART in Your Project

- Initialize the LPUART for Freemaster usage
- Enable UART interrupts

- Set UART interrupt callback to **FMSTR_Isr**

```
*main.c  freemaster_cfg.h

PINS_DRV_Init(NUM_OF_CONFIGURED_PINS, g_pin_mux_initConfig);

/* Get ADC max value from the resolution */
if (adConv1_ConvConfig0.resolution == ADC_RESOLUTION_8BIT)
    adcMax = (uint16_t) (1 << 8);
else if (adConv1_ConvConfig0.resolution == ADC_RESOLUTION_10BIT)
    adcMax = (uint16_t) (1 << 10);
else
    adcMax = (uint16_t) (1 << 12);
ADC_DRV_ConfigConverter(ADC_INSTANCE, &adConv1_ConvConfig0);
DEV_ASSERT(adConv1_ChnConfig0.channel == ADC_CHN);

/* Initialize LPUART1 for use with FreeMASTER
 * PTC6 and PTC7 are Rx and Tx on S32K144 EVB
 * Set up LPUART1 for interrupts on RxTx
 */
LPUART_DRV_Init(INST_LPUART1, &lpuart1_State, &lpuart1_InitConfig0);
INT_SYS_InstallHandler(LPUART1_RxTx_IRQn, FMSTR_Isr, NULL);
INT_SYS_EnableIRQ(LPUART1_RxTx_IRQn);

/* Initialize FTM PWM channels 0 and 1, PTD15 and PTD16 for EVB
 * - See ftm component for more info
 */
FTM_DRV_Init(FTM_INSTANCE, &flexTimer_pwm1_InitConfig, &ftmStateStruct);
/* Initialize FTM PWM combined channels 1 and 0 */
FTM_DRV_InitPwm(FTM_INSTANCE, &flexTimer_pwm1_PwmConfig);

INT_SYS_InstallHandler(FTM0_Ovf_Reload_IRQn, &ftmTimerISR, (isr_t*) 0U);
INT_SYS_EnableIRQ(FTM0_Ovf_Reload_IRQn);
```

Exercise #3 – Call the FMSTR Functions

- Call **FMSTR_Init()**

- Call **FMSTR_Poll()** at regular rate or in idle task

NOTE:

if the Recorder function is desired,
FMSTR_Recorder() also needs to be called.

This is normally called in a **periodic routine** where the variables of interest are updated or sampled.

```
*main.c  freemaster_cfg.h
...
FTM_DRV_Init(FTM_INSTANCE, &flexTimer_pwm1_InitConfig, &ftmStateStruct);
/* Initialize FTM PWM combined channels 1 and 0 */
FTM_DRV_InitPwm(FTM_INSTANCE, &flexTimer_pwm1_PwmConfig);

INT_SYS_InstallHandler(FTM0_Ovf_Reload_IRQn, &ftmTimerISR, (isr_t*) 0U);
INT_SYS_EnableIRQ(FTM0_Ovf_Reload_IRQn);
MotorInit(&mtr1);
mtr1.u16voltage = 0xffff;
mtr1.u16pwmperiod = 1000;

FMSTR_Init();
/* Infinite loop
 * - increment or decrement duty cycle
 * - Update channel duty cycle
 * - Wait for a number of cycles to make
 *   the change visible
 */
while(1)
{
    FMSTR_Poll();
    if (duty0updown == true)
    {
        if (increaseDutyCycle0 == false)
        {
            dutyCycle0--;
            if (dutyCycle0 < 10)
                increaseDutyCycle0 = true;
        }
    }
}
```

Exercise #3 – Setting Up the Communication Driver

Driver setup should be complete!

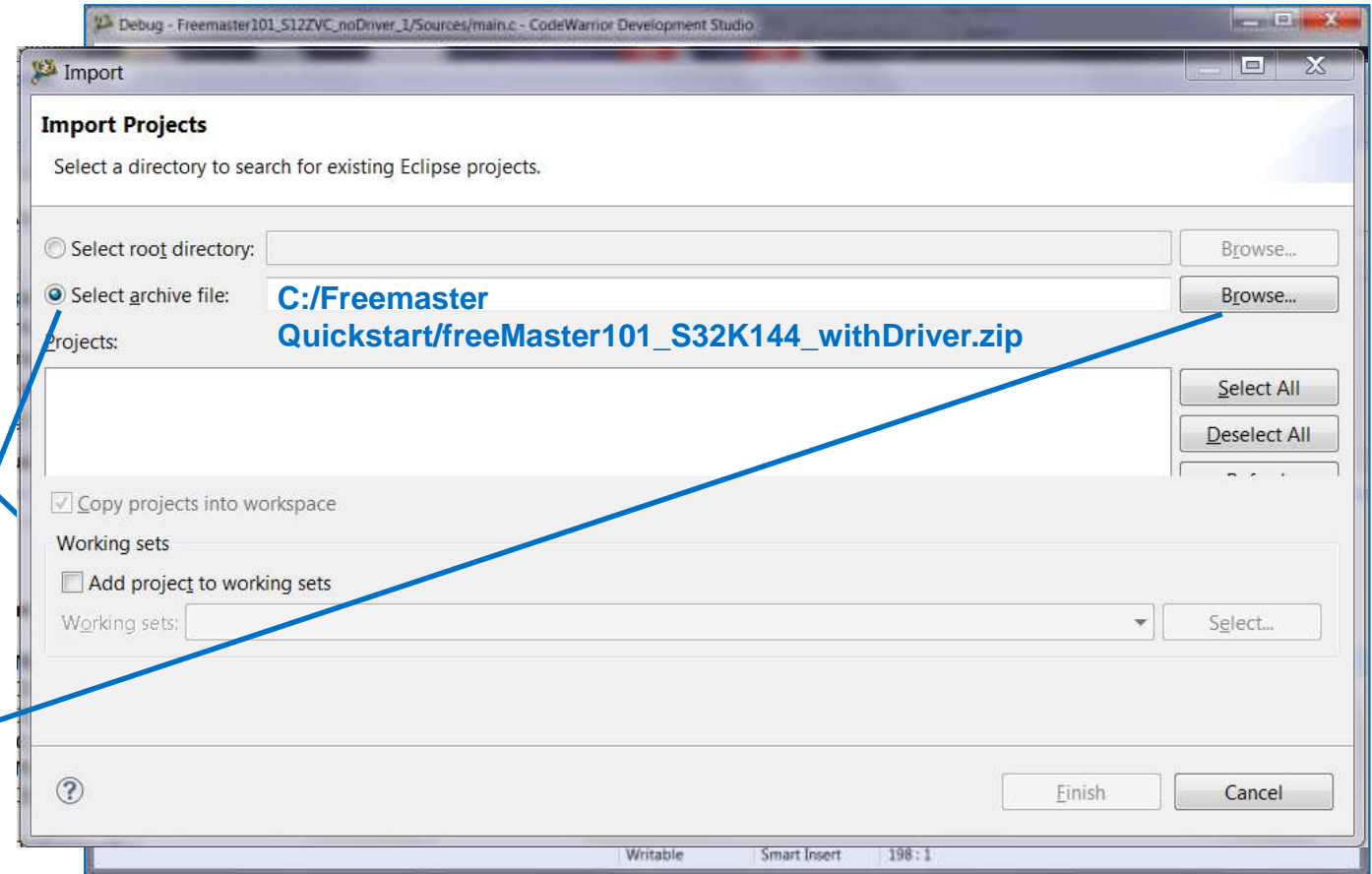
Exercise #3 – Program example project with driver

- Open S32DS for ARM
- Import the test project
freeMaster101_S32K144_withDriver.zip

1 – Select Import Project

2 – Check **Select archive file:**

3 – Browse to project file

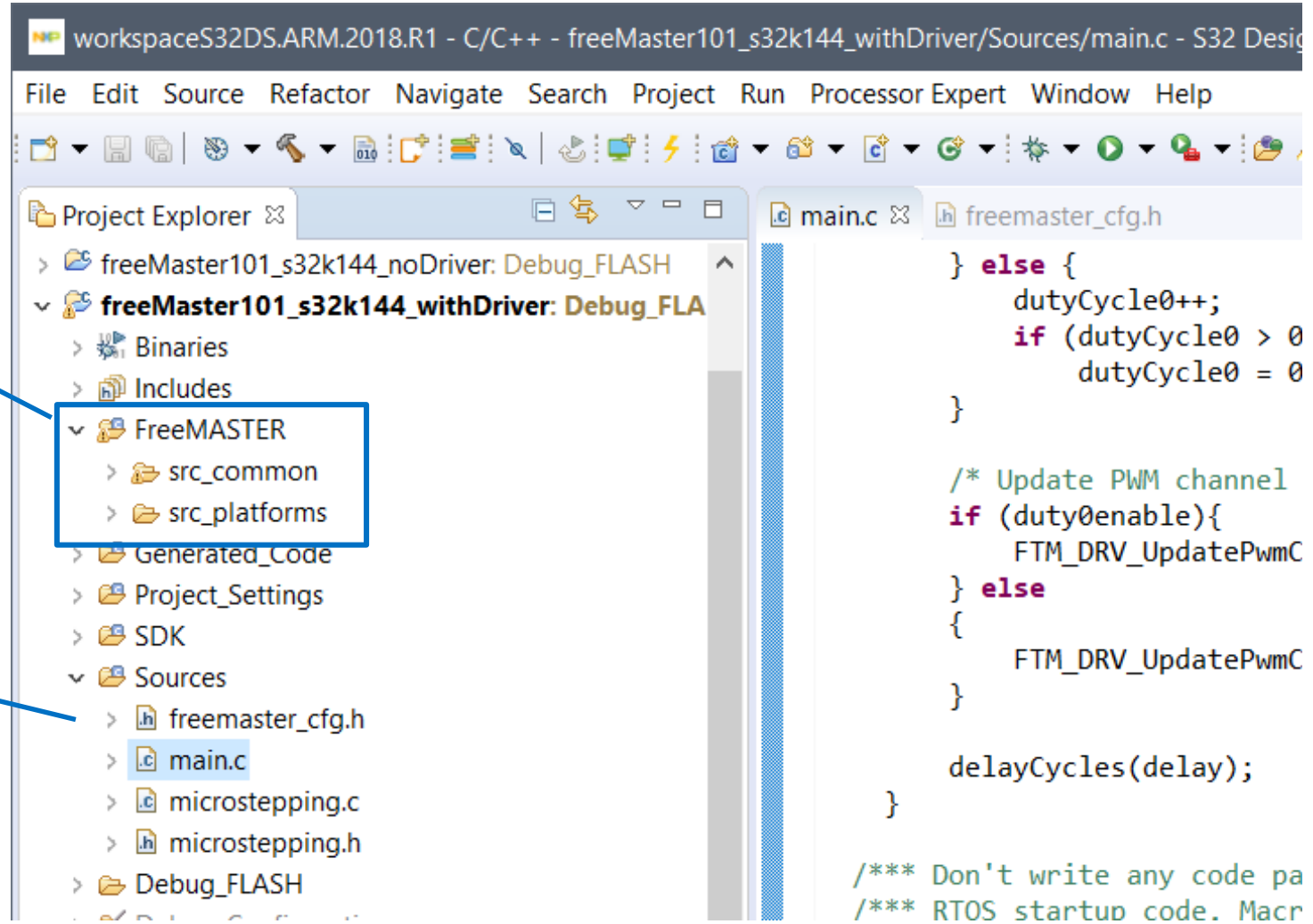


Exercise #3 – Examine the Code

Note Freemaster files

Freemaster drivers

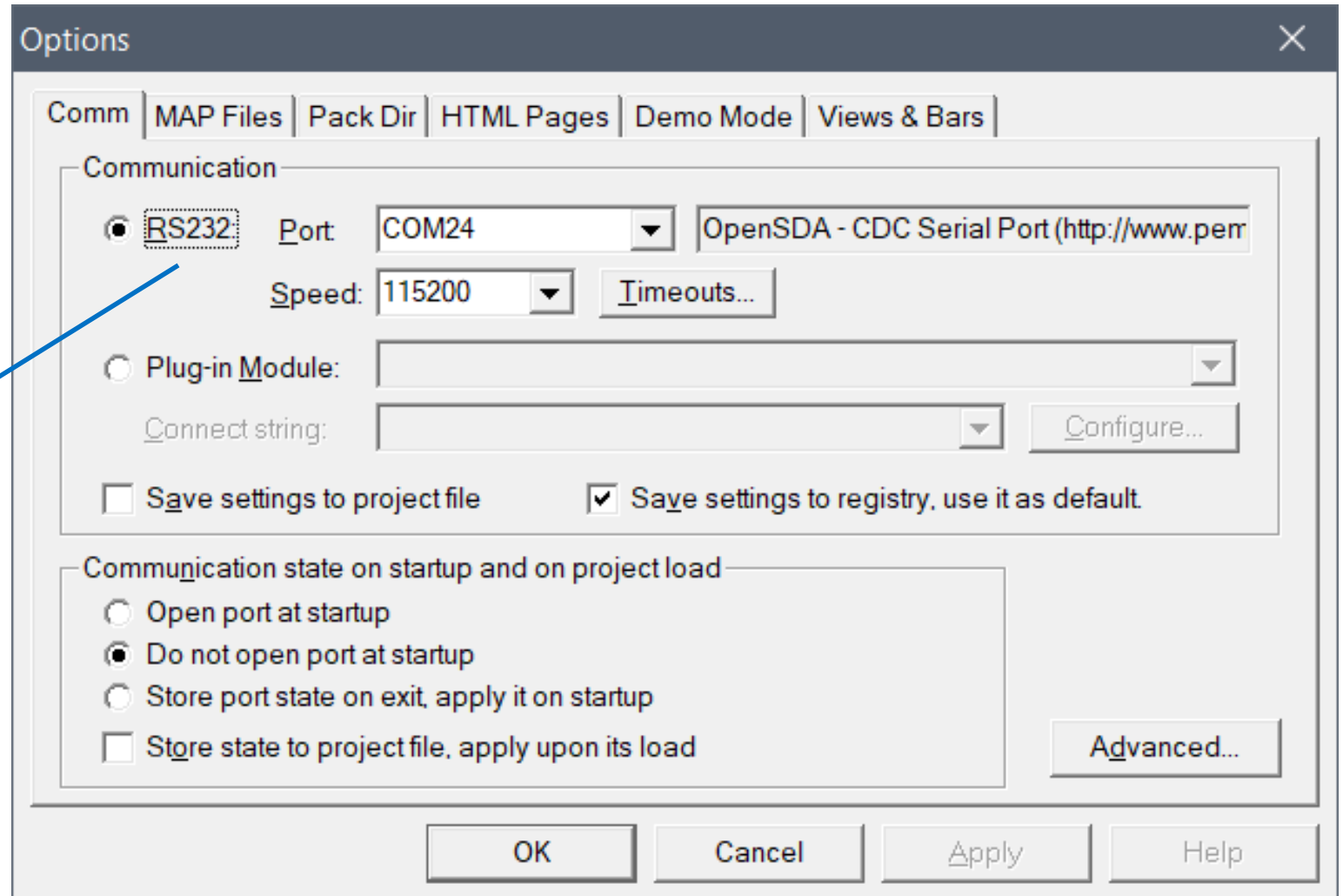
Freemaster config file



Exercise #3 – Change Freemaster Communication Option

Change Communication
for UART usage

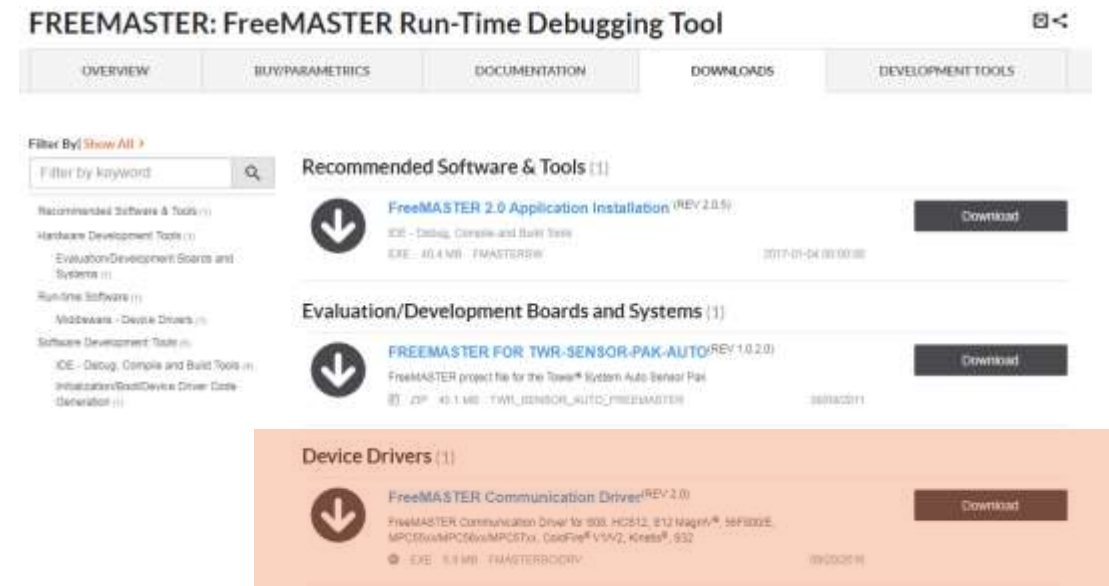
Note: We can now leave the S32DS debugger active and connected while also using Freemaster for visualization.



Exercise #3 – Summary – Adding the FreeMASTER Driver

Adding the FreeMASTER driver:

1. Download communication drivers from www.nxp.com/freemaster and click on **Downloads** Tab
2. Add files to project from:
[C:\nxp\FreeMASTER_Serial_Communication_Driver_V2.0\src_common](#)
3. Add files to project from:
[C:\nxp\FreeMASTER_Serial_Communication_Driver_V2.0\src_platforms\S32xx](#)
4. Edit freemaster_cfg.h
5. Add **#define freemaster.h** to project C file
6. Add **FMSTR_Init()** to initialization of C file
7. Add **FMSTR_Poll()** (or other) to loop in project
8. Optionally add **FMSTR_Recorder()** in periodic interrupt
9. Change FreeMASTER connection
10. Continue with FreeMASTER debugging



Exercise #4: Adding a Recorder



Exercise #4 – Adding a Recorder

- Call **FMSTR_Recorder()** in pwm interrupt

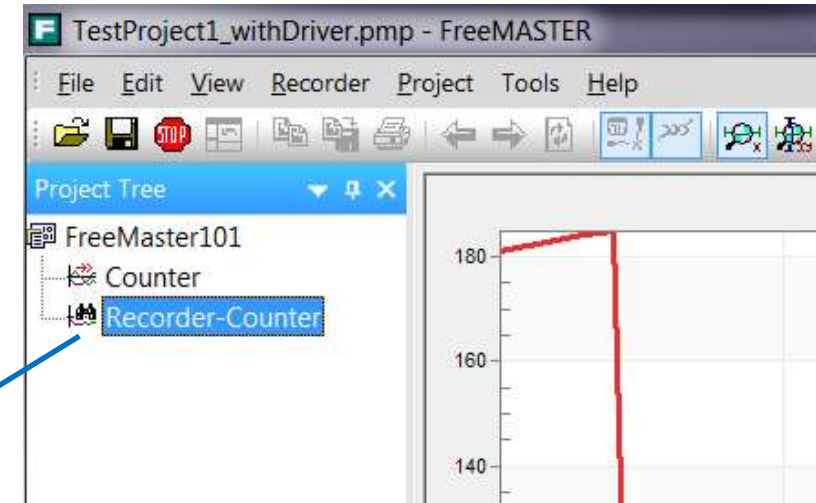
```
main.c x freemaster_cfg.h
/* instance
*/
void ftmTimerISR(void)
{
    FMSTR_Recorder();
    ADC_DRV_ConfigChan(ADC_INSTANCE, 0U, &adConv1_ChnConfig0);
    ADC_DRV_WaitConvDone(ADC_INSTANCE); /* Wait for the conversion to be done */
    ADC_DRV_GetChanResult(ADC_INSTANCE, 0U, &adcRawValue); /* Store the channel result into a local
    adcValue = ((float) adcRawValue / adcMax) * (ADC_VREFH - ADC_VREFL);

    ftmCounter++;
    if (ftmCounter >= ftmCounterLimit)
    {
        ftmCounter = 0;
        counter2++
    }
}
```


Exercise #4 – Adding a Recorder

In FreeMASTER ...

1. Right-Click on Freemaster project
2. Select **Create Recorder ...**



Exercise #4 – Adding a Recorder

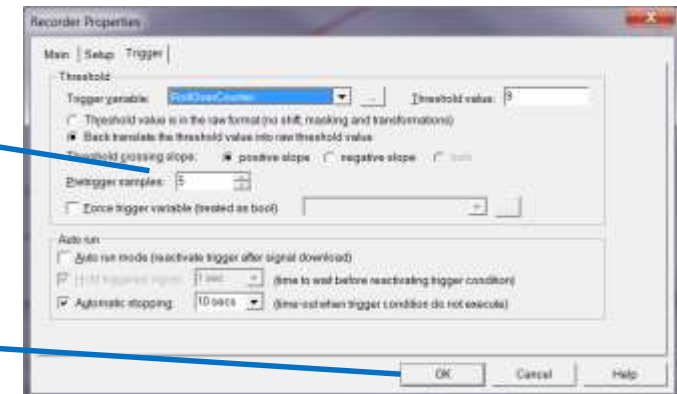
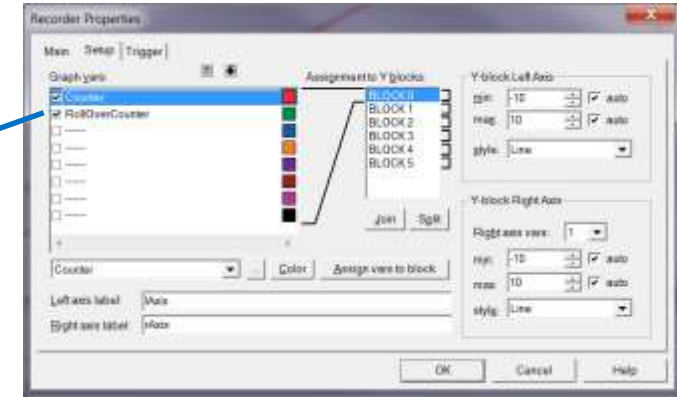
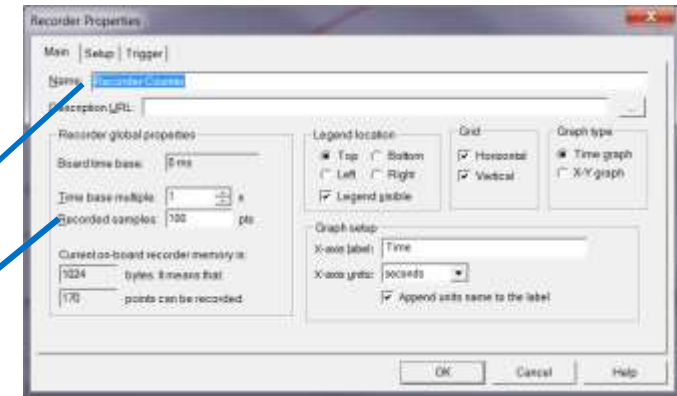
In FreeMASTER ...

1. In **Main** tab, enter name, sample points, time base

2. In **Setup** tab, select variables of interest

2. In **Trigger** tab, select #pre trigger samples, trigger variable and trigger threshold
(i.e RollOverCounter, Threshold 9)

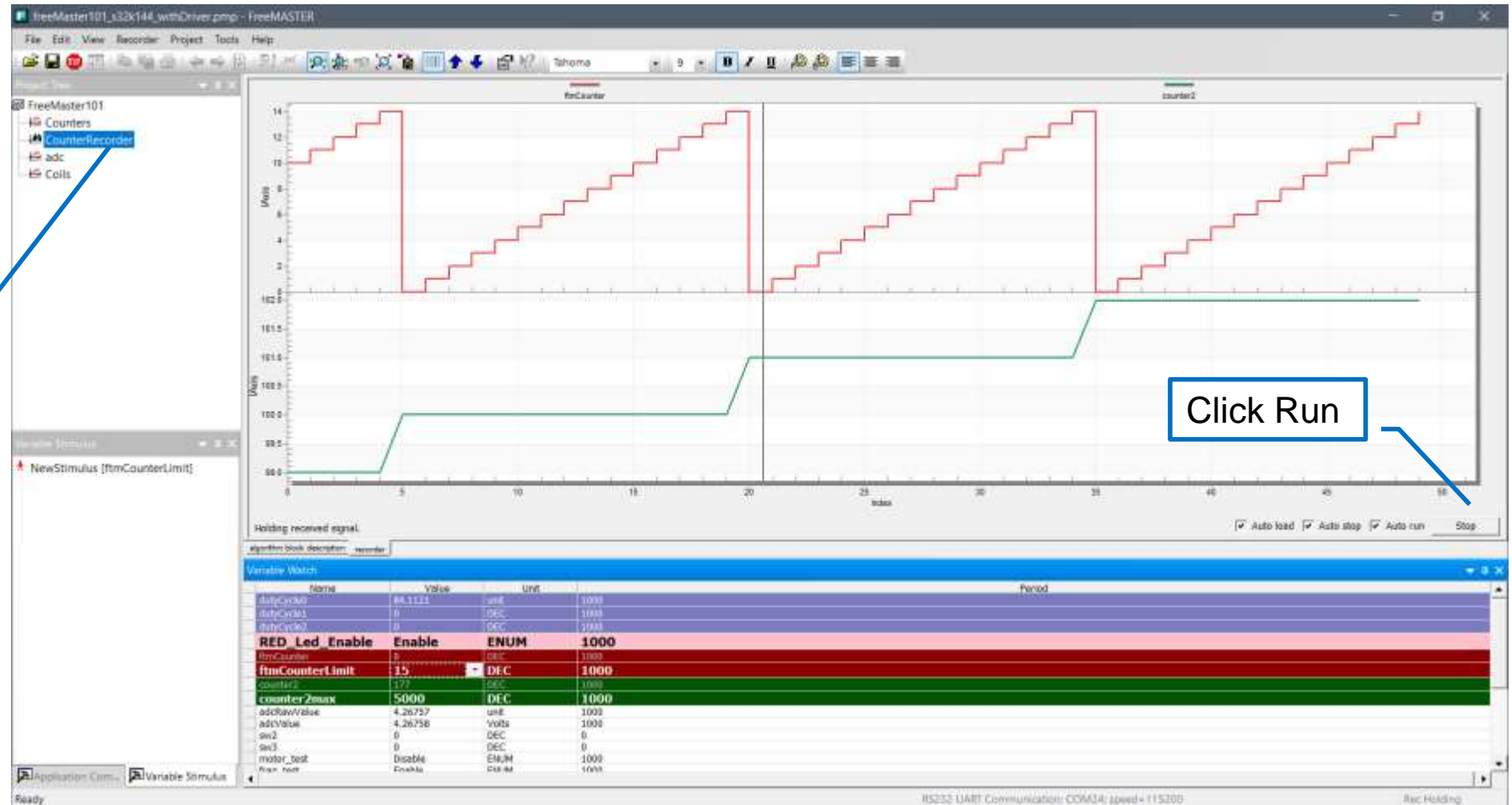
Click OK



Exercise #4 – Adding a Recorder

In FreeMASTER ...

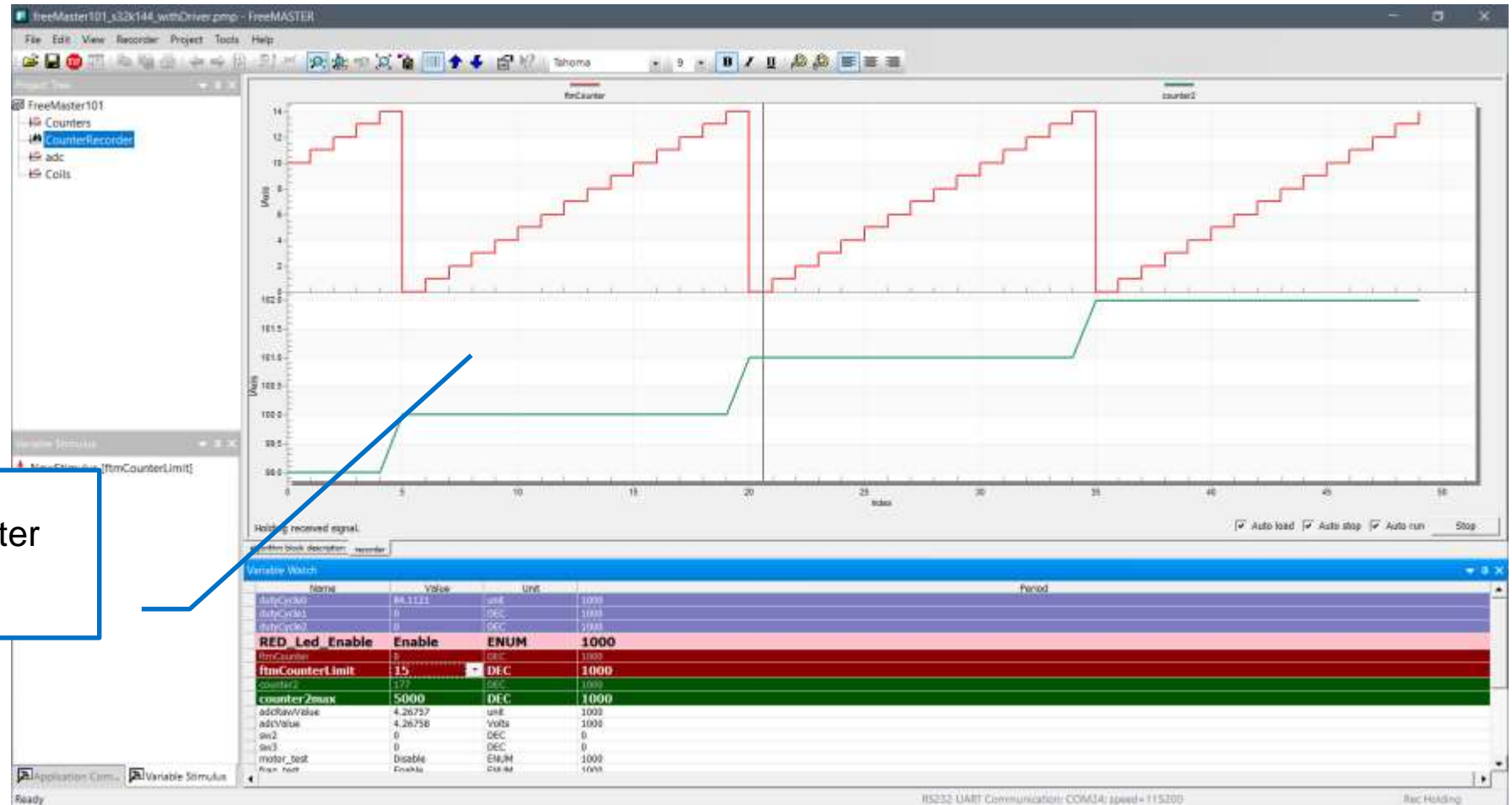
Select the new Recorder



Exercise #4 – Adding a Recorder

In FreeMASTER ...

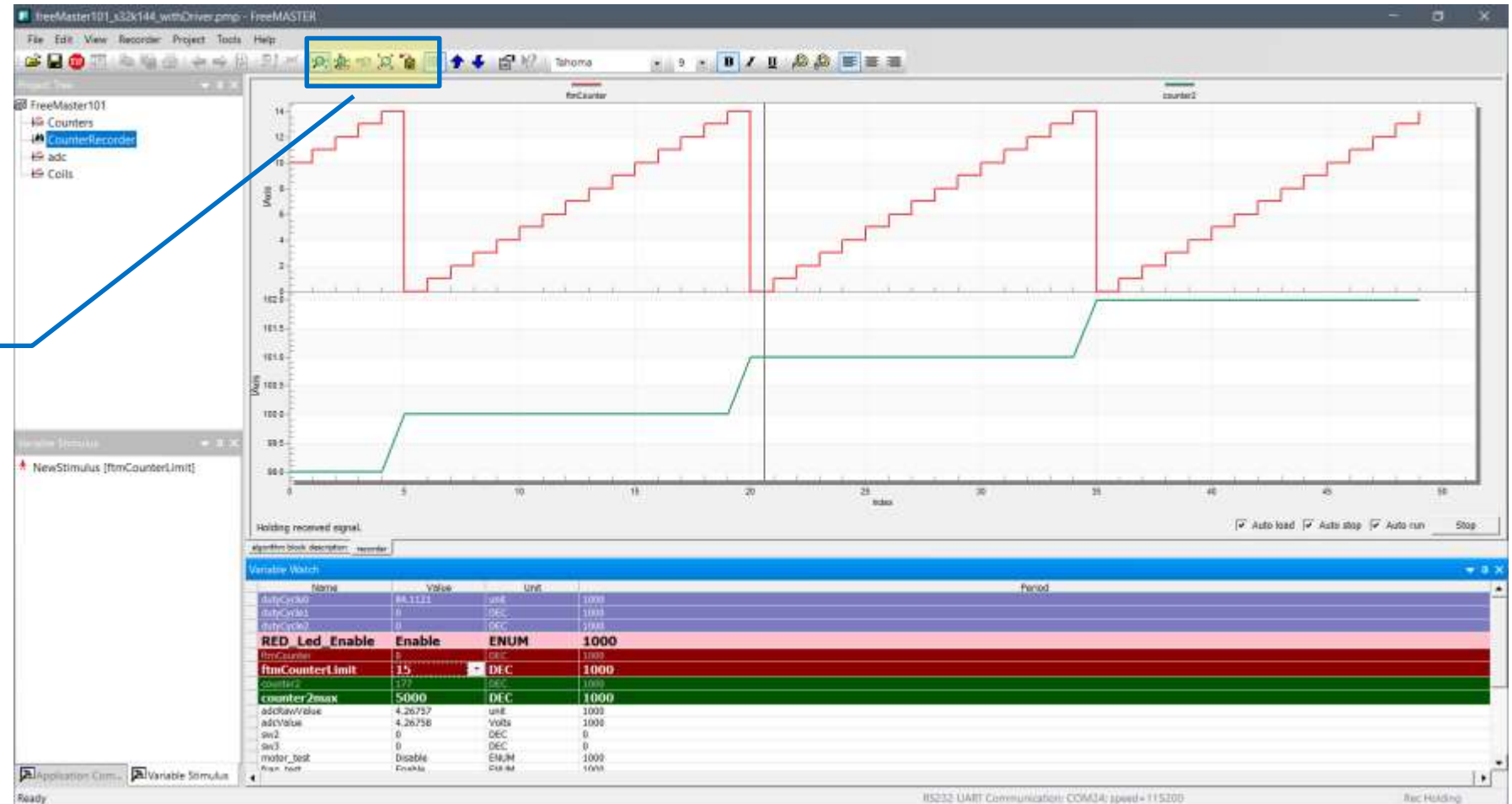
We can now see the counter with better resolution



Exercise #4 – Adding a Recorder

In FreeMASTER ...

Try zooming in on the data



Summary



Summary

- At this point:
- We should all be able to use and configure FreeMASTER
 - Watch variables
 - Control variables
 - Create a scope
 - Create a Recorder
- We should be able to configure a project to use the driver connection
- We should be able to use the debugger and use FreeMASTER simultaneously!



SECURE CONNECTIONS
FOR A SMARTER WORLD

www.nxp.com

NXP, the NXP logo, and NXP secure connections for a smarter world are trademarks of NXP B.V. All other product or service names are the property of their respective owners. © 2018 NXP B.V.