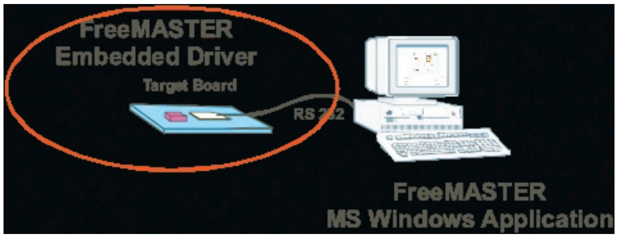
Lab

FreeMASTER Serial Communication Driver

Introduction

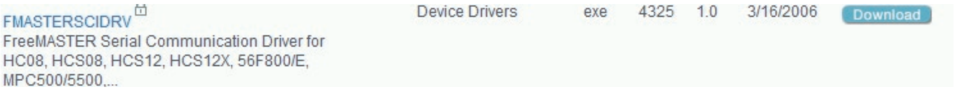
FreeMASTER Serial Communication Driver is an embedded-side software driver which implements the serial communication protocol between the embedded application and the host PC over USB.



The USB CDC bridge enables replacing RS232 with USB. (See MC56F8006_serial_lab)

Installation of FreeMASTER Serial Communication Driver

- Download the driver from freescale.com by searching for FMASTERSCIDRV.
  - You must accept the terms of a licence agreement before installation.
- Run the downloaded file “FMASTERSCIDRV.exe” to install the driver on your system



FMASTERSCIDRV FreeMASTER Serial Communication Driver for HC08, HCS08, HCS12, HCS12X, 56F800/E, MPC500/5500,...	Device Drivers	exe	4325	1.0	3/16/2006	Download

  - The default installation location is “C:\Program Files\Freescale\FreeMASTER Serial Communication”

Hint: The latest FreeMASTER application is also available from the same download page

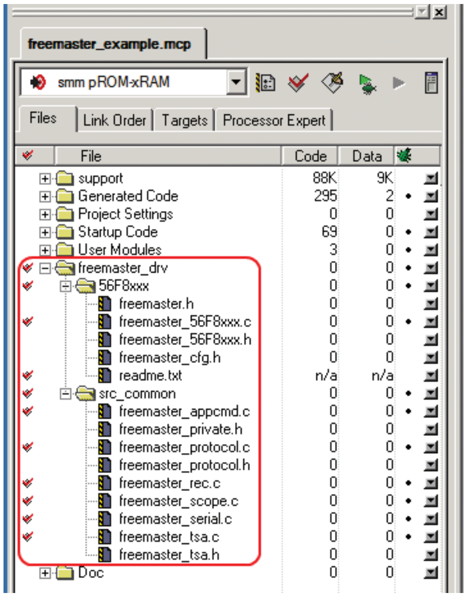
Copy FreeMASTER Serial Communication Driver Files into Your Project

- Create a new folder “freemaster_src” in a root of your Code Warrior project
- Copy driver files into your project:
  - Open “c:\Program Files\Freescale\FreeMASTER Serial Communication folder”
  - Copy folder “src_common“ to your CW project “{Project}\freemaster_src”
  - Open “c:\Program Files\Freescale\FreeMASTER Serial Communication\src_platforms” folder
  - Copy folder “56F8xxx” to your CW project “{Project}\freemaster_src”
  - Rename file “freemaster_cfg.h.example” in “56F8xxx” folder to “freemaster_cfg.h”

Hint: Make sure you also have installed the latest Freemaster application on you PC before continuing

Create “freemaster_drv” Group in Your Project

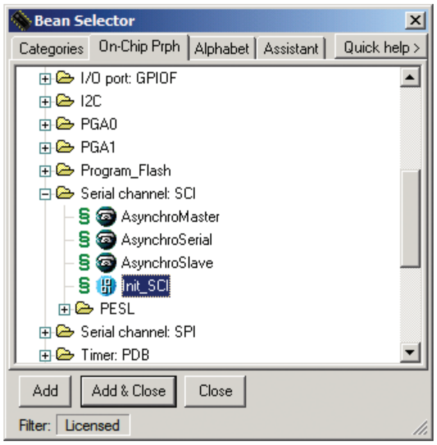
- Create/Open MC56F8006 project
- Create a new group “freemaster_drv” in your project “Files tree”
- Add folders “src_common“ and “56F8xxx” to the new group



✓	File	Code	Data	
	support	88K	9K	
	Generated Code	295	2	•
	Project Settings	0	0	
	Startup Code	69	0	•
	User Modules	3	0	•
✓	freemaster_drv	0	0	•
✓	56F8xxx	0	0	•
	freemaster.h	0	0	
✓	freemaster_56F8xxx.c	0	0	•
	freemaster_56F8xxx.h	0	0	
	freemaster_cfg.h	0	0	
✓	readme.txt	n/a	n/a	
✓	src_common	0	0	•
✓	freemaster_appcmd.c	0	0	•
	freemaster_private.h	0	0	
✓	freemaster_protocol.c	0	0	•
	freemaster_protocol.h	0	0	
✓	freemaster_rec.c	0	0	•
✓	freemaster_scope.c	0	0	•
✓	freemaster_serial.c	0	0	•
✓	freemaster_tsa.c	0	0	•
	freemaster_tsa.h	0	0	
	Doc	0	0	

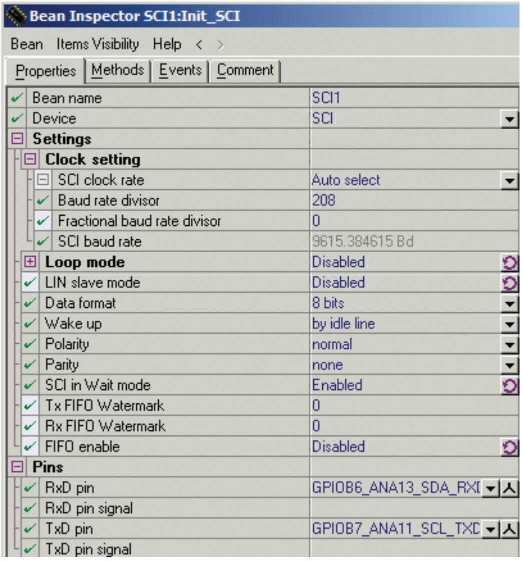
Add SCI Peripheral Bean into Your Project

- Go to Processor Expert pane
- Right-click on “Beans” in a list and select “Add Bean(s)…” from the context menu. Bean selector window will open
- Select “On-Chip Prph” pane and expand 56F8006_32_LQFP processor group
- Locate “Serial channel:SCI” in the expanded list and click on Init_SCI peripheral bean



Configure SCI Bean

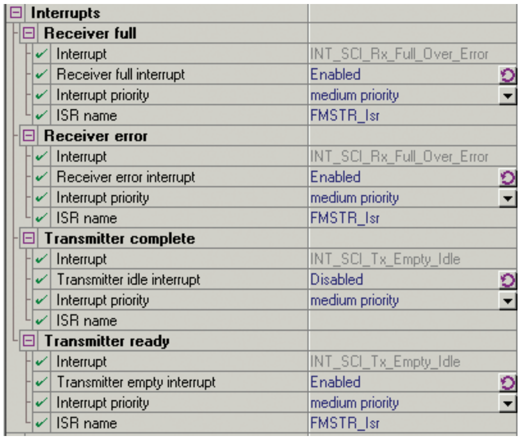
- Configure Init_SCI Bean as follows:
    - Baud rate divisor = 208 ~ 9600Bd
    - RxD pin = GPIOB6
    - TxD pin = GPIOB7
    - Enable peripheral clock = yes
    - Enable SCI Transmitter = yes
    - Enable SCI receiver = yes


Bean Inspector SCI1:Init_SCI
Bean Items Visibility Help < >
Properties Methods Events Comment

Bean name	SCI1
Device	SCI
Settings	
Clock setting	
SCI clock rate	Auto select
Baud rate divisor	208
Fractional baud rate divisor	0
SCI baud rate	9615.384615 Bd
Loop mode	Disabled
LIN slave mode	Disabled
Data format	8 bits
Wake up	by idle line
Polarity	normal
Parity	none
SCI in Wait mode	Enabled
Tx FIFO Watermark	0
Rx FIFO Watermark	0
FIFO enable	Disabled
Pins	
RxD pin	GPIOB6_ANA13_SDA_RXI
RxD pin signal	
TxD pin	GPIOB7_ANA11_SCL_TXD
TxD pin signal	

Configure SCI Interrupts

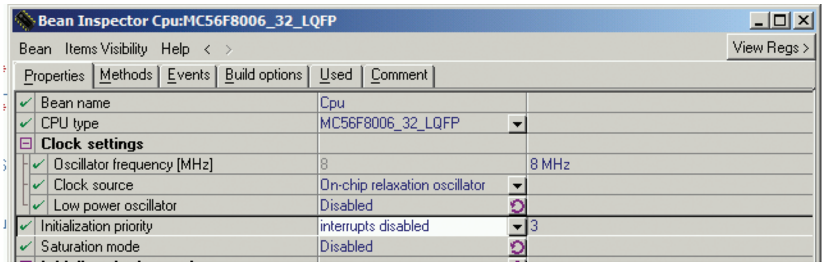
- Switch to ProcessorExpert pane
- Open SCI peripheral bean “Init_SCI”
- In interrupts section, configure module interrupts as follows:
    - Receiver full: Enabled
    - ISR name: FMSTR_Isr
    - Receiver error: Enabled
    - ISR name: FMSTR_Isr
    - Transmitter idle: Disabled
    - ISR name: N/A
    - Transmitter empty: Enabled
    - ISR name: FMSTR_Isr



Interrupts	
Receiver full	
Interrupt	INT_SCI_Rx_Full_Over_Error
Receiver full interrupt	Enabled
Interrupt priority	medium priority
ISR name	FMSTR_Isr
Receiver error	
Interrupt	INT_SCI_Rx_Full_Over_Error
Receiver error interrupt	Enabled
Interrupt priority	medium priority
ISR name	FMSTR_Isr
Transmitter complete	
Interrupt	INT_SCI_Tx_Empty_Idle
Transmitter idle interrupt	Disabled
Interrupt priority	medium priority
ISR name	
Transmitter ready	
Interrupt	INT_SCI_Tx_Empty_Idle
Transmitter empty interrupt	Enabled
Interrupt priority	medium priority
ISR name	FMSTR_Isr

Configure CPU Interrupts

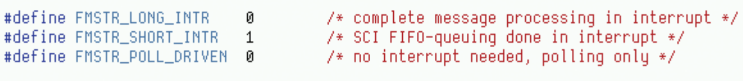
- Switch to ProcessorExpert pane
- Open CPU:MC56F8006 Bean
- Set Initialization priority to “interrupts disabled”
- Interrupts must be disabled before FreeMASTER driver initialization is finished


Bean Inspector Cpu:MC56F8006_32_LQFP
Bean Items Visibility Help < >
View Regs >
Properties Methods Events Build options Used Comment

Bean name	Cpu	
CPU type	MC56F8006_32_LQFP	
Clock settings		
Oscillator frequency [MHz]	8	8 MHz
Clock source	On-chip relaxation oscillator	
Low power oscillator	Disabled	
Initialization priority	interrupts disabled	3
Saturation mode	Disabled	

Configure FreeMASTER Serial Driver, Part 1

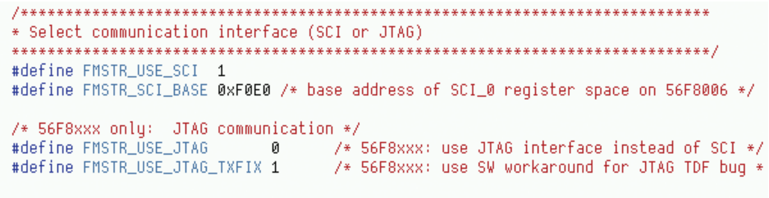
- Open freemaster_cfg.h file.
- Select short interrupt-based communication #define FMSTR_SHORT_INTR 1



```
#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 */
```

Configure SCI Interrupts

- Configure Communication Interface (SCI or JTAG)
  - Important: For SCI add the following line into the SCI config section #define FMSTR_USE_SCI 1
  - Set correct SCI base address (for MC56F8006 = 0xF0E0) #define FMSTR_SCI_BASE 0xF0E0

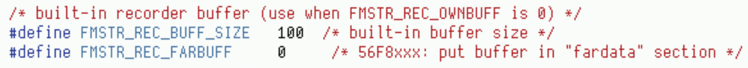


```
/*******************************************************************************
* Select communication interface (SCI or JTAG)
********************************************************************************/
#define FMSTR_USE_SCI  1
#define FMSTR_SCI_BASE 0xF0E0 /* base address of SCI_0 register space on 56F8006 */

/* 56F8xxx only:  JTAG communication */
#define FMSTR_USE_JTAG       0      /* 56F8xxx: use JTAG interface instead of SCI */
#define FMSTR_USE_JTAG_TXFIX 1      /* 56F8xxx: use SW workaround for JTAG TDF bug *
```

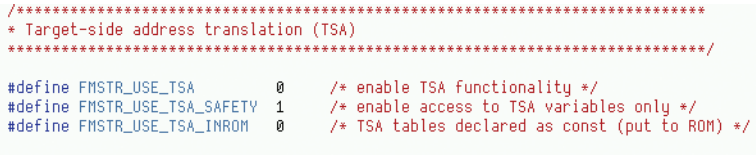
Configure FreeMASTER Serial Driver, Part 2

- Decrease recorder buffer size to 100 bytes
  - #define FMSTR_REC_BUFF_SIZE 100 /* built-in buffer size */
- Disable "put buffer in "fardata" section"
  - #define FMSTR_REC_FARBUFF 0



```
/* built-in recorder buffer (use when FMSTR_REC_OWNBUFF is 0) */
#define FMSTR_REC_BUFF_SIZE   100  /* built-in buffer size */
#define FMSTR_REC_FARBUFF     0     /* 56F8xxx: put buffer in "fardata" section */
```

- Disable target-side addressing feature
  - #define FMSTR_USE_TSA 0 /* enable TSA functionality */



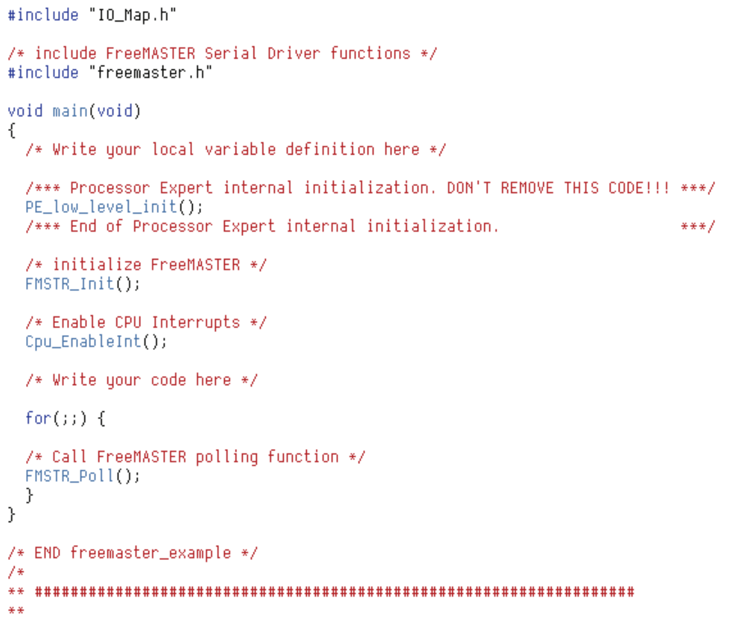
```
/*******************************************************************************
* Target-side address translation (TSA)
********************************************************************************/

#define FMSTR_USE_TSA         0     /* enable TSA functionality */
#define FMSTR_USE_TSA_SAFETY  1     /* enable access to TSA variables only */
#define FMSTR_USE_TSA_INROM   0     /* TSA tables declared as const (put to ROM) */
```

Add FreeMASTER Driver Initialization to a User Code

- To finish FreeMASTER serial driver, it must be initialized in a user code
- Add following lines to the user main:
  - Add #include "freemaster.h“ to an include section of your main application c-file
  - Call FMSTR_Init() routine. The routine must be called before CPU interrupts are enabled
  - Enable CPU interrupts. Call Cpu_EnableInt() CPU PE method
  - Add FreeMASTER polling function FMSTR_Poll() to a periodically called section of your code (background/idle loop)

FreeMASTER Driver Initialization Example



```
#include "IO_Map.h"

/* include FreeMASTER Serial Driver functions */
#include "freemaster.h"

void main(void)
{
  /* Write your local variable definition here */

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  /* initialize FreeMASTER */
  FMSTR_Init();

  /* Enable CPU Interrupts */
  Cpu_EnableInt();

  /* Write your code here */

  for(;;) {

  /* Call FreeMASTER polling function */
  FMSTR_Poll();
  }
}

/* END freemaster_example */
/*
** ###################################################################
**
```

FreeMASTER Communication Polling Mode vs. Interrupts Only

- FreeMASTER driver can run in three different modes:
  - Completely Interrupt-Driven (FMSTR_LONG_INTR = 1) - Both the SCI communication and the FreeMASTER protocol decoding and execution is done in the FMSTR_Isr interrupt service routine.
  - Mixed Interrupt and Polling Modes (FMSTR_SHORT_INTR = 1) - The raw SCI communication is handled by the FMSTR_Isr interrupt service routine, while the protocol decoding and execution is handled in the FMSTR_Poll routine. The user typically calls the FMSTR_Poll during the idle time in the application “main loop.”
  - Completely Poll-Driven (FMSTR_POLL_DRIVEN = 1) - Both the SCI communication and the FreeMASTER protocol execution is done in the FMSTR_Poll routine. No interrupts are needed, the FMSTR_Isr code compiles to an empty function.
- It is recommended to use either a Mixed Interrupt, Polling mode or Completely Poll-Driven mode of communication
- The above modes of communication apply to both SCI and JTAG interfaces

FreeMASTER Serial Communication Driver User’s Manual

- For more details on the FreeMASTER Serial Communication Driver, please refer to “FreeMASTER Serial Communication Driver User’s Manual,” which is copied to your PC at the driver install process
- It can be located here in MS Windows Start Menu:
  - Start->Programs->Freescale->FreeMASTER Serial Communication->User Manual

How to Run the FreeMASTER demo on MC56F8006DEMO

If you have installed the CDC bridge:

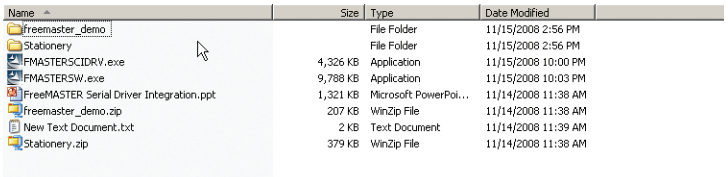
- You have a com port dedicated to the SCI on the MC56F8006DEMO card when you configure RX_EN and TX_EN with jumpers connecting Berg pins 1 and 2
- When you run PCMASTER, configure it for this com port, then make sure the stop button is not depressed

To keep a different image in the S08JM60

- In this case the COM port will not be generally available to your PC, but only to particular applications
- In this case, to use the serial communications ability of FreeMASTER, populate the RS232 circuits on the MC56F8006DEMO card and use the RS232 COM port on your PC if so equipped
- Otherwise there is a JTAG-based communication method using polling that will function on the board documented in the freeMASTER documentation

Freemaster Demo

- Open and run the project in the FreeMASTER demo using CW for DSC 8.2.3. Once the code is loaded, you can kill thread, close project, close CodeWarrior and power cycle MC56F8006DEMO.

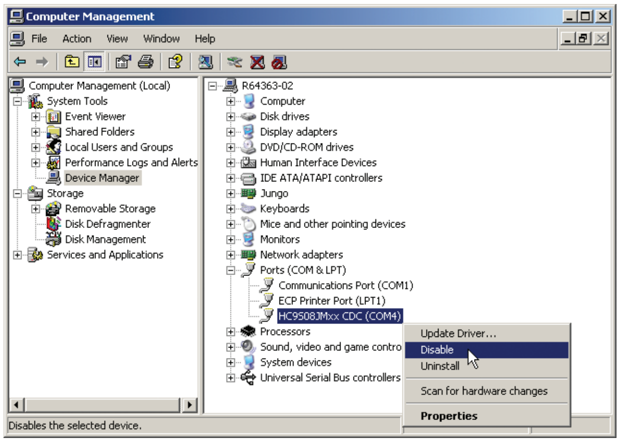


Name	Size	Type	Date Modified
freemaster_demo		File Folder	11/15/2008 2:56 PM
Stationery		File Folder	11/15/2008 2:56 PM
FMASTERSCIDRV.exe	4,326 KB	Application	11/15/2008 10:00 PM
FMASTERSW.exe	9,788 KB	Application	11/15/2008 10:03 PM
FreeMASTER Serial Driver Integration.ppt	1,321 KB	Microsoft PowerPoi...	11/14/2008 11:38 AM
freemaster_demo.zip	207 KB	WinZip File	11/14/2008 11:38 AM
New Text Document.txt	2 KB	Text Document	11/14/2008 11:39 AM
Stationery.zip	379 KB	WinZip File	11/14/2008 11:38 AM

Run the FreeMASTER Application on Your PC

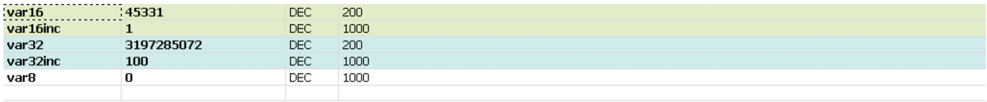
- You will find a file with a .pmp extension in the CodeWarrior project directory just visited
- Open this file with the FreeMASTER application
- Configure it for the correct com port
- Make sure a connection is made from the board to your PC for the com port (USB or RS232)
- If the Stop button in the FreeMASTER application window is depressed, click it to turn off the "Stop" state
- Observe the updating variables

If USB com port seems to not work, disable/enable it and try again



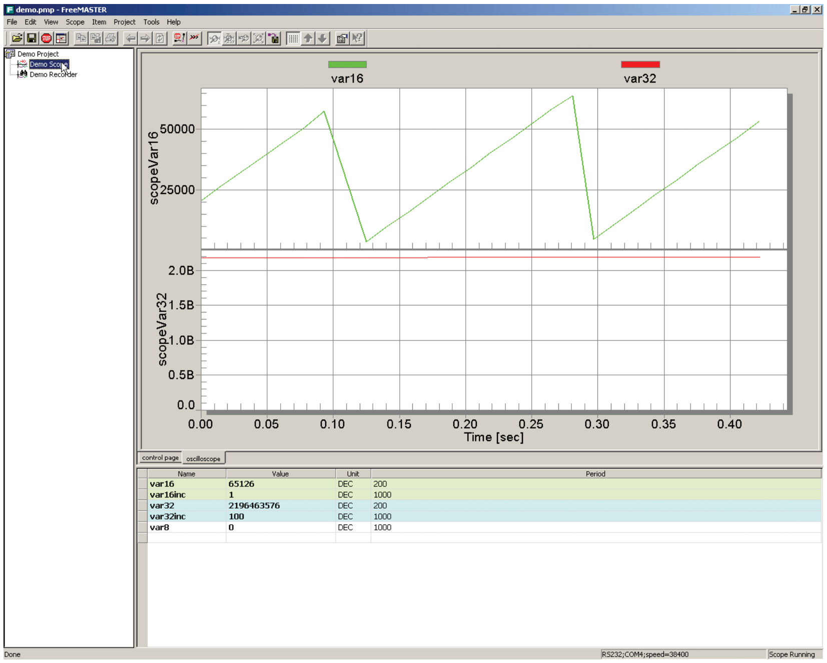
Disables the selected device.

The updating variable will be visible in real time



var16	45331	DEC	200
var16inc	1	DEC	1000
var32	3197285072	DEC	200
var32inc	100	DEC	1000
var8	0	DEC	1000

Click on Demo Scope to see the real-time scope



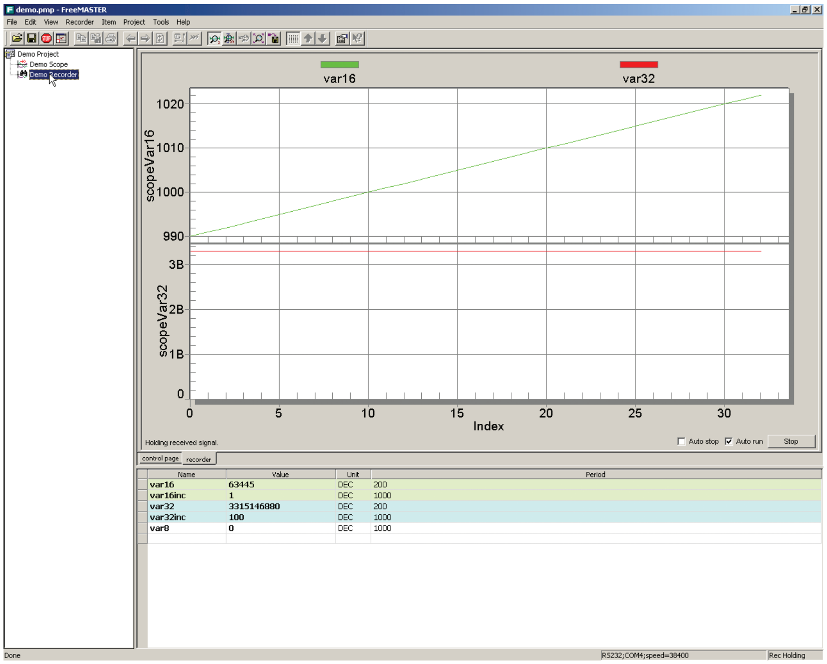
Name	Value	Unit	Period
var16	65126	DEC	200
var16inc	1	DEC	1000
var32	2196463576	DEC	200
var32inc	100	DEC	1000
var8	0	DEC	1000

Done

RS232;COM4;speed=38400

Scope Running

Click on Demo Recorder to see Exacting Recorder



Holding received signal.

Name	Value	Unit	Period
var16	63445	DEC	200
var16inc	1	DEC	1000
var32	3315146880	DEC	200
var32inc	100	DEC	1000
var8	0	DEC	1000

Done

RS232;COM4;speed=38400

Rec Holding

Recorder vs. Scope

- The recorder captures more detail but is only intended to be occasionally sent to a host PC This allows more time resolution in the images that are sent.
- The scope captures real-time data, sending it all to the host PC, but to avoid swamping the com link, only sends coarse time resolution data.
- For power conversion debug, both can be useful at different times.