C Compilers • Real-Time OS • Simulators • Education • Evaluation Boards

Implementing µVision DLL's for Advanced Generic Simulator Interface

Application Note 154
Rev. 4
Extended by Infineon Technologies

Contents

Introduction

The µVision Debugger supports a simulator interface for implementing user-defined peripherals. This interface is called **A**dvanced **G**eneric **S**imulator **I**nterface (AGSI). The AGSI introduces a flexible and easy way for adding new user defined peripherals directly to µVision. It provides functions that are necessary to simulate the peripheral's behavior as well as functions to display peripheral dialogs.

To ease the development of a user-defined peripheral, the AGSI and a configuration framework is provided in two example projects. Only µVision version 2.21 or later supports all functions that are described in this document.

AGSI Revision 3 is for µVision3 (version 3.00 or higher) and adds the following features:

- AgsiEntry has new function codes (nCODE= AGSI_PRE_RESET, AGSI_CMDOUT)
- The CALLBACK function for AgsiSetWatchOnSFR, AgsiSetWatchOnVTR, and AgsiSetWatchOnMemory gets now the address and access reason when it is called.

AGSI DLL's that have been developed for µVision2 can still be used with µVision3.

SPeriDLL

SPeriDLL, is a synonym for 'Sample Peripheral DLL'. It is a ready to run peripheral DLL which implements a 'A/D Converter from Analog Devices ADuC812' as a sample peripheral. It uses most of the AGSI functions to implement this peripheral. The project consists of a MS Visual-C++ (6.0) project file and the following source files:

AGSI.h:	prototypes for the AGSI functions (do not modify!)
SPeriDLL.h:	main header file with various prototypes and definitions
SPeriDLL.cpp:	main file (created by AppWizard) contains setup code and simulation
PeriDialog.h:	header file (created by Class Wizard) for a modeless peripheral dialog
PeriDialog.cpp:	implementation file for a modeless peripheral dialog

Also a simple µVision test project 'Single A/D conversion with ADuC812' is included in the file S812ADC.zip which shows how to include and test the implemented peripheral.

STimerDLL

STimerDLL, is a synonym for 'Sample Timer DLL'. It is a ready to run peripheral DLL which implements a 'Timer 3' as a sample peripheral. As regards functionality, 'Timer 3' is identical to a standard 8051 Timer 1 but has different SFR addresses so that it can be loaded in addition to a 'Timer 1'. It uses most of the AGSI functions to implement this peripheral. The project consists of a MS Visual-C++ (6.0) project file and the following source files:

AGSI.h:	prototypes for the AGSI functions (do not modify!)
Common.h:	header file with various prototypes and definitions
Common.cpp:	common support functions for AGSI and dialog functions
STimerDLL.h:	header file for STimerDLL class
STimerDLL.cpp:	main file provides peripheral setup code and simulation
PeriDialog.h:	header file (created by Class Wizard) for a modeless peripheral dialog
PeriDialog.cpp:	implementation file for a modeless peripheral dialog

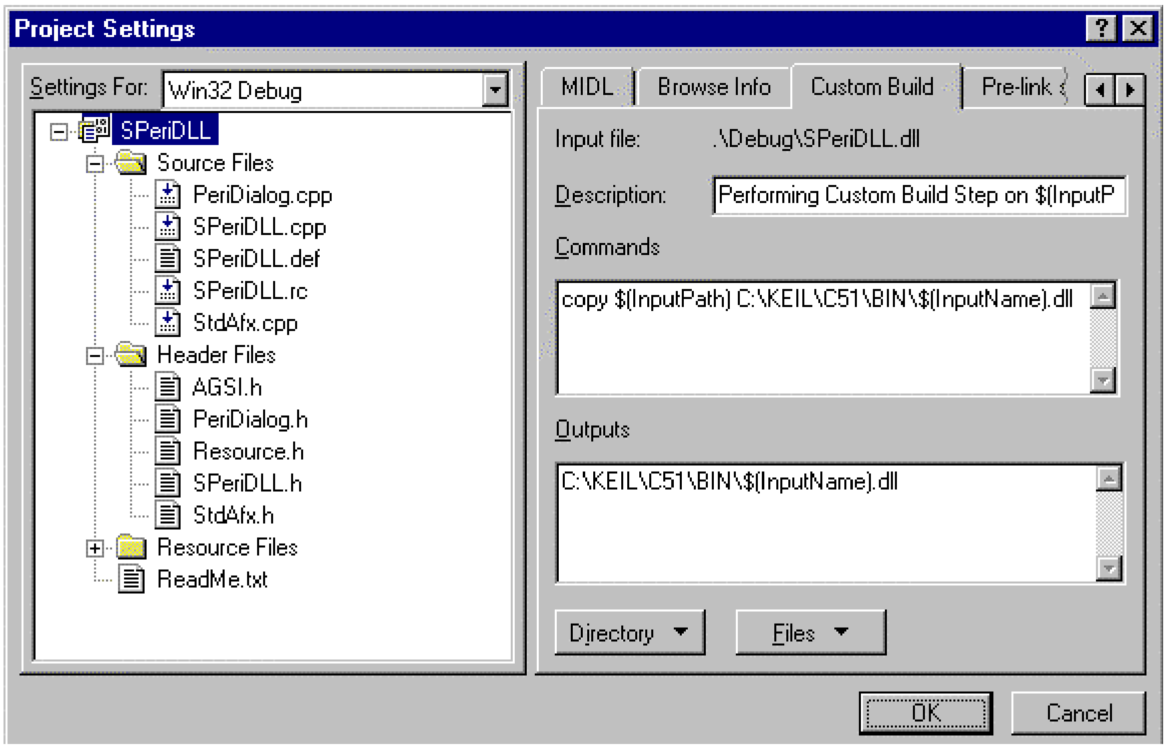
Also a simple µVision test project is included in the file Timer3.zip which shows how to include and test the implemented peripheral.

In order to develop a peripheral, knowledge about C/C++ programming and the MS Visual-C++ 6.00 Programming Environment is required.

How to use a Sample Peripheral DLL

In order to use one of the Sample Peripheral DLL's, you must perform the steps below. The following steps are described for the **SPeriDLL** but the same applies for the **STimerDLL**:

- Install µVision and the C51 Compiler on your PC.
- Create a folder such as D:\Src32\SPeriDLL\
- Unzip the file **SPeriDLL.zip** into the folder. Make sure that the 'use folder names' checkbox is checked since SPeriDLL uses some subfolders.
- Create a folder such as C:\Keil\C51\Examples\S812ADC\
- Unzip the file **S812ADC.zip** into the folder.
- Start Visual-C, select the 'SPeriDLL.dsw' project file.
- Select 'Project – Settings'. Click at the 'Debug' tab. Browse for the 'Executable for Debug session'. You need to select the file Uv2.Exe. It is normally in C:\Keil\Uv2 but this depends on where you have installed µVision.
- Then click at the 'Custom Build' tab and write in the 'Commands' window the command 'copy $(InputPath) C:\KEIL\C51\BIN\$(InputName).dll' and write in the 'Outputs' window the output file 'C:\KEIL\C51\BIN\$(InputName).dll'. This step is required to automatically copy the created DLL after building it in the BIN subfolder of µVision which is normally C:\KEIL\C51\BIN but depends on where you have installed µVision. If everything is right, then the dialog should look like this:


Project Settings

Settings For: Win32 Debug

- SPeriDLL
  - Source Files
    - PeriDialog.cpp
    - SPeriDLL.cpp
    - SPeriDLL.def
    - SPeriDLL.rc
    - StdAfx.cpp
  - Header Files
    - AGSI.h
    - PeriDialog.h
    - Resource.h
    - SPeriDLL.h
    - StdAfx.h
  - Resource Files
  - ReadMe.txt

MIDL | Browse Info | Custom Build | Pre-link

Input file: .\Debug\SPeriDLL.dll

Description: Performing Custom Build Step on $(InputP

Commands

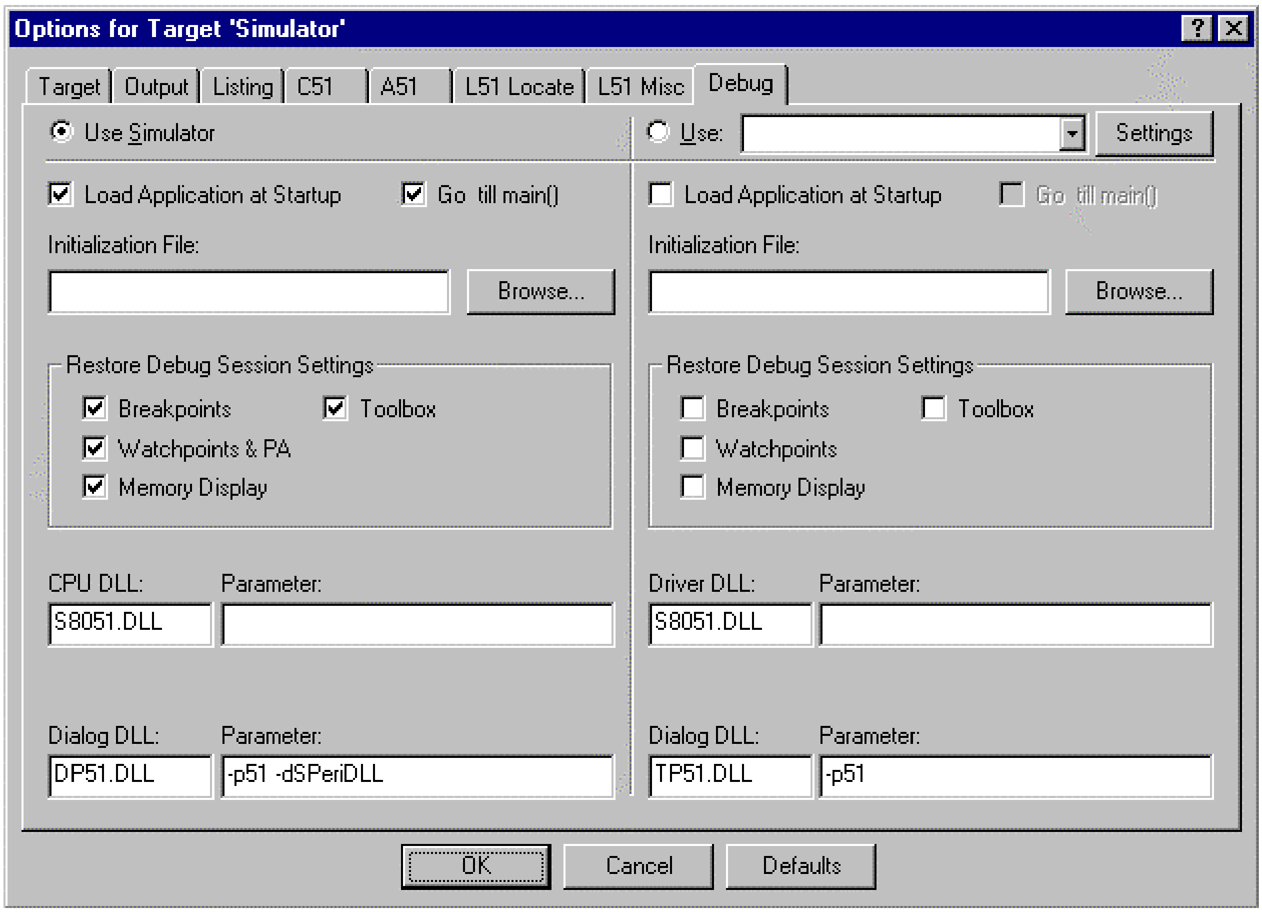
```
copy $(InputPath) C:\KEIL\C51\BIN\$(InputName).dll
```

Outputs

```
C:\KEIL\C51\BIN\$(InputName).dll
```

- After that, close the dialog.

- Select ‘Build – Set active configuration’, choose the SPeriDLL Win32 Debug configuration.
- Select ‘Build – Rebuild All’ to create the DLL.
- Run µVision by pressing the F5 key. Select ‘Project – Open Project’, the Select Project dialog comes up. Select the ‘S812ADC.uv2’ project. It can be found in the folder that you have created and copied the project files into (normally C:\Keil\C51\Examples\S812ADC).
- Select ‘Rebuild all target files’ to build the project.
- Select ‘Options for Target – Debug’. Enable loading of the SPeriDLL peripheral DLL by simply adding the parameter ‘-dSPeriDLL’ to the parameter list of the ‘Dialog DLL’. The parameter format for peripheral DLL’s is ‘-dDLLName’ (DLL name without extension). Make sure that the ‘Use Simulator’ radio button is checked. If everything is right, then the dialog should look like this:


Options for Target 'Simulator'

Target | Output | Listing | C51 | A51 | L51 Locate | L51 Misc | Debug

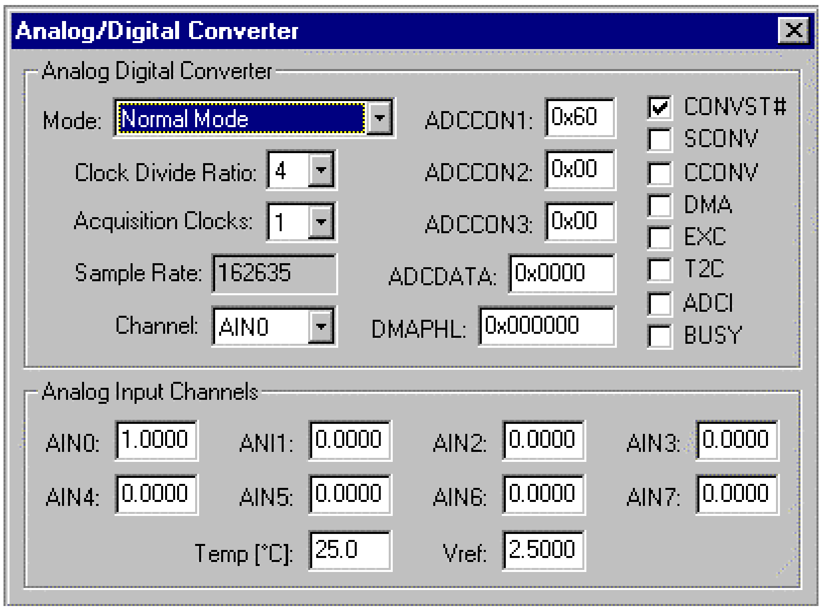
[● Use Simulator] ○ Use:

[✓] Load Application at Startup [✓] Go till main()
Initialization File:
Restore Debug Session Settings: [✓] Breakpoints [✓] Toolbox [✓] Watchpoints & PA [✓] Memory Display
CPU DLL: S8051.DLL Parameter:
Dialog DLL: DP51.DLL Parameter: -p51 -dSPeriDLL

[ ] Load Application at Startup [ ] Go till main()
Initialization File:
Restore Debug Session Settings: [ ] Breakpoints [ ] Toolbox [ ] Watchpoints [ ] Memory Display
Driver DLL: S8051.DLL Parameter:
Dialog DLL: TP51.DLL Parameter: -p51

- Close the dialog.

- Select ‘Debug – Start/Stop Debug Session’. This will start the µVision Debugger. It initializes and loads also our SPeriDLL.dll. In the ‘Peripherals’ menu a new item should be present with the label ‘A/D Converter’. Click on this item to open the peripheral dialog which looks like this:


Analog/Digital Converter

Analog Digital Converter

Mode: Normal Mode
Clock Divide Ratio: 4
Acquisition Clocks: 1
Sample Rate: 162635
Channel: AIN0

ADCCON1: 0x60
ADCCON2: 0x00
ADCCON3: 0x00
ADCDATA: 0x0000
DMAPHL: 0x000000

[✓] CONVST#
[ ] SCONV
[ ] CCONV
[ ] DMA
[ ] EXC
[ ] T2C
[ ] ADCI
[ ] BUSY

Analog Input Channels

AIN0:	1.0000	ANI1:	0.0000	AIN2:	0.0000	AIN3:	0.0000
AIN4:	0.0000	AIN5:	0.0000	AIN6:	0.0000	AIN7:	0.0000
		Temp [°C]:	25.0	Vref:	2.5000		

- Now you can single step through the code of the ‘Single A/D conversion with ADuC812’ sample and observe the behavior of the ‘A/D Converter’ peripheral and also other peripherals like ‘Port 0’ , ‘Port 2’, ‘Port 3’ and ‘Interrupt’.

Note that this sample program demonstrates functionality of only a small part of the A/D Converter.

Implementing own peripheral DLLs: Required Steps

In order to develop a peripheral DLL you should perform the following steps:

- Start MS Visual-C++ and create a new project ‘MFC AppWizard (dll)’.
- Add prototypes and definitions for the AGSI and SFR’s (**S**pecial **F**unctions **R**egisters) and VTR’s (**V**ir**t**ual **R**egisters) definitions to the main header file (*.h).
- Write code for the peripheral initialization into the main file (*.cpp). This includes AGSI setup (GetFunctionPointers), declaration of peripheral menu entries and associated dialogs (DefineAllMenuEntries), declaration of SFR’s (DefineAllSFR), VTR’s (DefineAllVTREG), Watches (DefineAllWatches) and Interrupts (DefineAllInterrupts). Write also code for peripheral reset - SFR’s reset values (ResetPripheral). All this functions are called from the function AgsiEntry() which must be exported by this peripheral DLL.
- Write functions for simulation of the peripheral into the main file (these functions are triggered by the defined watches). Include also prototypes of these functions.
- Create a peripheral dialog with the Resource Editor (if the dialog is required) and the associated header file (*.h) and implementation file (*.cpp) using the ‘MFC ClassWizard’. Don’t forget to set the ‘Visible’ property of the dialog and include the default buttons ‘OK’ and ‘Cancel’ and make them invisible (required for the behavior of the ESC and Enter keys). Change the default constructor for the dialog and add functions PeriDisp() – displays dialog, PeriUpdate() – updates display contents which calls function Update() and PeriKill() – closes the dialog. Add also a menu definition (AGSIMENU) and a dialog definition (AGSIDLGD).
- Write the code for updating the display contents into the Update() function in the dialog implementation file. This function is called automatically when an update is requested and is used to reflect the current state of the peripheral.
- Add functions for dialog control item’s messages by using ‘MFC ClassWizard’. Most frequently used messages are: ON_BN_CLICKED for Buttons, ON_EN_KILLFOCUS for Edit Boxes, ON_CBN_SELCHANGE for Combo Boxes …
  Include also functions for the two invisible buttons ‘OK’ and ‘Cancel’.
- Select ‘Project – Settings’. Click at the ‘Debug’ tab. Browse for the ‘Executable for Debug session’ and select the file Uv2.Exe. It is normally in C:\Keil\Uv2 but this depends on where you have installed µVision.
- Rebuild your peripheral DLL. Then copy the DLL file to the BIN subfolder of µVision which is normally C:\KEIL\C51\BIN but depends on where you have installed µVision or use the ‘Custom Build’ within MS Visual-C++ and write the command that automatically copies the DLL after rebuild (see previous description in the ‘Sample Peripheral DLL’).
- Test your peripheral DLL by running µVision (press the F5 key). Select a test project and enable loading of the implemented peripheral DLL by simply adding the parameter ‘−dDLLName’ (DLL name without extension) to the parameter list of the peripheral DLL (see previous description in the ‘Sample Peripheral DLL’).
- If the implemented peripheral is running, switch into Release Mode and rebuild it. Then test the peripheral DLL again (don’t forget to copy the ‘Release DLL’ file to the BIN subfolder of µVision).

How simulation basically works

If every simulated peripheral would be updated with every simulated CPU instruction, the performance of the simulator would be extremely low. That's why µVision simulator uses a event driven simulation instead. Events (also called watches) are read or write accesses to special function registers (SFR), virtual register (VTR) or memory areas and when a software timer expires. The following two examples explain this in detail:

Analog Digital Converter (see SPeriDLL):

Let's assume that an A/D converter has configuration register (SFR's ADCCONx), data register (SFR's ADCDATAx), 8 analog inputs (VTR's AIN0-7) and one external pin (VTR CONVST) to start a conversion. The A/D converter does nothing until it is started so no functions are called to simulate it and no simulation time is consumed at this time. In order to 'see' when the A/D converter is configured and started, so called access watches (AgsiSetWatchOnSFR and AgsiSetWatchOnVTR) need to be set on the configuration register and on the external start pin. This is done in the function 'DefineAllWatches'. Whenever a new value is written into the ADCCONx register or into the external start pin, the function 'AdcConv' is called. This function has to check the configuration, reference voltages and the analog inputs in order to calculate the digital value. The digital result cannot be written into the data register at this time. A real A/D converter needs some time to sample and convert an analog voltage. In order to simulate this behavior, a software timer is set (AgsiSetTimer) which calls the function 'AdcCompleted' after the specified number of states. This function writes the digital value into the data register, clears the busy flag, and sets the interrupt request bit.

16 Bit Timer (see STimerDLL):

Even a timer does not need to update (recalculate) its values with every simulated instruction. Typically, the timer values need to be updated when the configuration changes (start/stop, prescaler value) and when the actual timer value is read. Therefore, a write access watch must be set on the configuration register and read access watches must be set on the timer register. The timer calculation function stores the time (states) in a static variable whenever it is called. With the time difference (actual states – last states) the actual timer value can be calculated. With this method, the timer values can be calculated at any time with a minimum of calculation overhead. Only the interrupt on a timer overflow cannot be handled that way. With an additional software timer (AgsiSetTimer) set to this event, the timer is recalculated with every overflow. When a overflow is detected, the interrupt request flag is set and the timer is reloaded with 0 or a specific reload value.

Address representation

Depending on the microcontroller family, µVision maps the different memory areas (XDATA / DATA / CODE) into **one** linear address range. These different memory areas are represented with the following values in the most significant byte of a 32 bit address:

80166 Microcontroller: This microcontroller has a 16 Mbyte linear address space. The valid address range therefore is from 0 to 0x00FFFFFF. No different memory types are needed.

8051 Microcontroller:

Define	Memory type	Address Range
amXDATA	XDATA	0x0000 – 0xFFFF
amPDATA	PDATA	0x0000 – 0x00FF (one page of XDATA)
amDATA	DATA	0x0000 – 0x00FF
amIDATA	IDATA	0x0000 – 0x00FF (0x00 – 0x7F = DATA)
amCODE	CODE	0x0000 – 0xFFFF
amBANK0	Bank 0	0x0000 – 0xFFFF
amBANK0 + n	Bank n	0x0000 – 0xFFFF
amBANK31	Bank 31	0x0000 – 0xFFFF

80251 Microcontroller: Following types can be used in addition to the memory types of the 8051 Family

Define	Memory type	Address Range
amEDATA	EDATA	0x0000 – 0xFFFF
amECODE	ECODE	0x0000 – 0xFFFFFF
amHDATA	HDATA	0x0000 – 0xFFFFFF
amHCONS	HCONST	0x0000 – 0xFFFFFF
amCONST	CONST	0x0000 – 0xFFFF

SLE66 Microcontroller: Following types can be used in addition to the memory types of the 8051 Family

Define	Memory type	Address Range
amPHYS	physical	0x0000 – 0xFFFFFF

Example:

```
BYTE buffer[10];

AgsiReadMemory(0x1000|(amCODE<<24),10,buffer);// read 10 bytes to (CODE) address 0x1000

AgsiReadMemory(0x1000|(amCODE<<16),10,buffer);// the same function but for SLE66 processor
```

AGSI Function Description

AgsiEntry is the only function of a peripheral DLL that is called directly from the µVision simulator. All other functions described below are in the µVision simulator and can be called from the peripheral DLL.

Functions to define SFR's, VTR's, interrupts, timer, menus, dialogs and access watches. These functions can only be called during the initialization.

AgsiDefineSFR	AgsiDefineVTR
AgsiDeclareInterrupt	AgsiSetWatchOnSFR
AgsiSetWatchOnVTR	AgsiSetWatchOnMemory
AgsiCreateTimer	AgsiDefineMenuItem
AgsiRegisterExecCallBack	

Functions to read and write memory, SFR's and VTR's:

AgsiWriteSFR	AgsiReadSFR
AgsiWriteVTR	AgsiReadVTR
AgsiWriteMemory	AgsiReadMemory

Functions to retrieve simulator status information:

AgsiGetStates	AgsiGetProgramCounter
AgsiIsInInterrupt	AgsiIsSleeping
AgsiIsSimulatorAccess	AgsiIsSyncInstruction
AgsiGetSyncCount	AgsiGetExternalClockRate
AgsiGetInternalClockRate	AgsiGetClockFactor
AgsiGetLastMemoryAddress	

Functions to control the simulator:

AgsiSetTimer	AgsiSetSFRReadValue
AgsiStopSimulator	AgsiWakeUp
AgsiTriggerReset	AgsiContinue
AgsiRequestInterrupt	AgsiSetSyncCount
AgsiSetSyncDelay	AgsiSetExternalClockRate
AgsiUpdateWindows	AgsiHandleFocus
AgsiMessage	AgsiExecuteCommand

Functions to store and retrieve configuration information:

AgsiSetTargetKey	AgsiGetTargetKey

Functions to retrieve symbol values or symbol names:

AgsiGetSymbolByName	AgsiGetSymbolByValue

AgsiEntry

Summary:

```
extern "C" DWORD AGSIEXPORT AgsiEntry (DWORD nCode, void *vp)
```

Parameter:

`nCode`	Function selector. All possible values are listed below.
`vp`	Pointer to various objects depending on `nCode`. In order to use this pointer, it must be casted to the required data type.

Return Value:

The function should return TRUE(1) if completed successfully or FALSE(0) if an error occurred.

Description:

AgsiEntry is the only function of a peripheral DLL that has to be exported. It is called from µVision when a debug session is started to initialize the peripheral simulation as well as during the debugging session to notify events. The meaning of `vp` depends on the value of `nCode`. `AgsiEntry` can be called with the following `nCode` values:

Value of `nCode`	Value of `vp`	Function
AGSI_CHECK	8051 or 80166	Check CPU Type
AGSI_INIT	Pointer to AGSICONFIG	Initialize DLL
AGSI_TERMINATE	Not used	Terminate
AGSI_RESET	Not used	Reset
AGSI_PREPLL	Not used	CPU clock is about to be changed
AGSI_POSTPLL	Not used	CPU clock was changed
AGSI_PRERESET	Not used	Called before Reset, but CPU cycles still valid
AGSI_CMDOUT	Pointer to Text	Command output of 'exec' commands
AGSI_ONINTERRUPT	Pointer to Interrupt number	Interrupt from a device
AGSI_ONRETI	Pointer to Interrupt number	Interrupt acknowledged
AGSI_ENTERSLEEP	Not used	Power down mode entered
AGSI_EXITSLEEP	Not used	Power down mode exited

AGSI_CHECK

The first call to `AgsiEntry` is done with `nCode=AGSI_CHECK`. The pointer `vp` points to a DWORD which contains either the value 8051 or 80166, or 7 (for ARM) depending on the microcontroller family that is selected in the current project. This call checks if the DLL can be used for the specified microcontroller family. The function should return TRUE(1) if the DLL supports this microcontroller family or FALSE(0) if not.

AGSI_INIT

The second call to `AgsiEntry` is done with `nCode=AGSI_INIT`. The pointer `vp` points to the structure `AGSICONFIG` which contains information about the project and the parameters for this DLL. This information can be used to configure the peripheral DLL. Additional parameters (format: –option) for the DLL can be entered in the µVision dialog 'Options for Target -> Debug - > Dialog DLL Parameter'. The DLL can analyze the 'm_pszConfiguration' string to extract the information. The 'm_pszProjectPath' can be used to store log files or additional configuration files for the current project.

```
typedef struct {

  HINSTANCE   m_hInstance;           // Instance handle to retrieve the function addresses

  const char* m_pszProjectPath;      // Path to application e.g. C:\KEIL\C51\EXAMPLES\HELLO
  const char* m_pszDevice;           // Simulated Device e.g. 52. This string is extracted
                                     // out of the -p option.
  const char* m_pszConfiguration;    // Complete dialog DLL options e.g. -p52 -dmydll ...
  const char* m_pszAppFile;          // Name of loaded OMF file including path e.g.
                                     // C:\KEIL\C51\EXAMPLES\HELLO\HELLO
  HWND        m_hwnd;                // Mainframe parent window
} AGSICONFIG;
```

When `AgsiEntry` is called with `AGSI_INIT`, all special function register (SFR), virtual register (VTR), interrupts, watch points and dialogs that need to be simulated must be defined. The function should return FALSE(0) if an error occurs or TRUE(1) if the function has been executed successfully.

AGSI_TERMINATE

`AgsiEntry` is called with `nCode=AGSI_TERMINATE` when the µVision debugger is closed. The pointer `vp` is not used in this case. When files have been opened during initialization they must be closed and if memory has been allocated, it must be freed.

AGSI_RESET

`AgsiEntry` is called with `nCode=AGSI_RESET` when the simulated CPU is reset. All peripherals (SFR's) must be set to their reset state. There are several situations where a CPU reset is executed:

- When the simulator is started (after AGSI_INIT).
- After an application is loaded.
- When RESET is entered in the command line or when the reset button is pressed in the toolbar.
- When a watchdog timer overflow occurs.

The pointer `vp` is not used in this case.

AGSI_PREPLL, AGSI_POSTPLL

`AgsiEntry` is called with `nCode=AGSI_PREPLL` or `AGSI_POSTPLL` before and after the CPU clock frequency changes. Some CPU's have a clock prescaler that can be reprogrammed to save power. In case a peripheral is not connected to the same clock as the CPU, the values for `AgsiSetTimer` probably need to be recalculated when the CPU clock is modified. This function call notifies a peripheral before and after the CPU clock frequency has changed so that the timer values can be corrected for the new clock. The pointer `vp` is not used in this case.

AgsiDefineSFR

Summary:

```
BOOL AgsiDefineSFR(const char* pszSfrName, AGSIADDR dwAddress,
                   AGSITYPE eType, BYTE bBitPos);
```

Parameter:

`pszSfrName`	Pointer to name of the SFR
`dwAddress`	Address of the SFR. Following address ranges are possible: 8051/251: 0x80 – 0xFF 8051Mx: 0x80 – 0xFF and 0x180 – 0x1FF 80166: 0xF000 – 0xF1FE and 0xFE00 – 0xFFFE (even address)
`eType`	Type of the SFR. 8051: AGSIBYTE or AGSIBIT 80166: AGSIWORD or AGSIBIT With AGSIBIT, `dwAddress` must point to a bit addressable area: 8051: 0x80, 0x88, … , 0xF0, 0xF8 every 8th byte 80251: 0x80 - 0xFF every byte 8051Mx: 0x80 - 0xF8 and 0x180 – 0x1F8 every 8th byte 80166: 0xF100 – 0xF1FE and 0xFF00 – 0xFFFE (even address)
`bBitPos`	Bit position within SFR (only for `eType`=AGSIBIT). 8051/251/Mx: 0 – 7 80166: 0 – 15

Return Value:

TRUE if successful, FALSE in case of wrong address or too many definitions (at least 300 for all loaded DLL's).

Description:

This function is used to define a SFR (**S**pecial **F**unction **R**egister) or a SFR bit. These definitions can be listed in the symbol window and can be used in the watch window and command line.

Note:

This function may only be called during the initialization process.

Example:

```
AgsiDefineSFR("IE", 0xA8, AGSIBYTE, 0);     // 8051: IE
AgsiDefineSFR("EA", 0xA8, AGSIBIT, 7);      // 8051: EA bit in IE
AgsiDefineSFR("PSW", 0xFF10, AGSIWORD, 0); // 80166: PSW
AgsiDefineSFR("IEN", 0xFF10, AGSIBIT, 11); // 80166: IEN bit in PSW
```

AgsiDefineVTR

Summary:

```
AGSIVTR AgsiDefineVTR(const char* pszVtrName, AGSITYPE eType,
                      DWORD dwValue);
```

Parameter:

`pszVtrName`	Name of the VTR
`eType`	Type of the VTR (AGSIVTRCHAR, AGSIVTRWORD, AGSIVTRLONG or AGSIVTRFLOAT)
`dwValue`	Initial Value of the VTR. Initializing float values is a little difficult since `dwValue` is defined as DWORD. In this case, the float value can be converted to a DWORD using a union.

Return Value:

VTR handle if successful otherwise NULL.

Description:

This function is used to declare a VTR (**V**ir**t**ual **R**egister). VTR's are used to display or to set values that are usually set by hardware. With this function, new VTR's can be defined or the handle of already defined VTR's can be retrieved.

Note:

This function may only be called during the initialization process.

Example:

```
hXTAL = AgsiDefineVTR("XTAL", AGSIVTRLONG, 0x00B71B00);        // 12MHz
hVREF = AgsiDefineVTR("VREF", AGSIVTRFLOAT, 0x40200000);       // 2.5V
hVREF = AgsiDefineVTR("MYPORT", AGSIVTRCHAR, 0x000000FF);      // all pins high
```

AgsiDeclareInterrupt

Summary:

```
BOOL AgsiDeclareInterrupt(AGSIINTERRUPT *pInterrupt);
```

Parameter:

`PInterrupt`	Pointer to an AGSIINTERRUPT structure (see below).

Return Value:

TRUE if successful otherwise FALSE.

Description:

This function is used to define an interrupt source of an on-chip peripheral. All the information about this interrupt source is passed to the function with a structure. It defines the interrupt vector address as well as all request, enable and priority flags. This information automatically adds another line in the Interrupt Dialog. The structure is different for every microcontroller family and is described in the following AGSIINTERRUPT structure:

```
// 8051/251/8051Mx

typedef struct {
  AGSIADDR         vec;       // interrupt vector address
  char           *mess;       // interrupt name (will be shown in interrupt dialog)
                              // The mode bit is only shown in the interrupt dialog.
                              // It has no influence on interrupt processing.
  AGSIADDR        msfr;       // interrupt mode sfr.
  WORD           mmask;       // interrupt mode bit mask (only one bit may be set)
  const char    *mname;       // name of interrupt mode bit
  AGSIADDR        rsfr;       // interrupt request sfr
  WORD           rmask;       // interrupt request bit mask (only one bit may be set)
  const char    *rname;       // name of interrupt request bit
  AGSIADDR        esfr;       // interrupt enable sfr
  WORD           emask;       // interrupt enable bit mask (only one bit may be set)
  const char    *ename;       // name of interrupt enable bit
  AGSIADDR       p0sfr;       // interrupt priority 0 sfr
  WORD          p0mask;       // interrupt priority 0 bit mask (only one bit may be set)
  const char    *pname;       // name of interrupt priority bit
  AGSIADDR       p1sfr;       // interrupt priority 1 sfr. =0 if CPU only supports 2 levels
  WORD          p1mask;       // interrupt priority 1 bit mask (only one bit may be set)
  WORD             pwl;       // priority within level (1 - lowest priority)
  WORD      auto_reset;       // reset interrupt request flag on interrupt entry
} AGSIINTERRUPT;


// 80166
typedef struct {
  AGSIADDR         vec;       // interrupt vector address (must be a even address)
  char           *mess;       // interrupt name (will be shown in interrupt dialog)
  AGSIADDR         sfr;       // interrupt control sfr which contains ILVL, GLVL, IR and IE
} AGSIINTERRUPT;
```

```
// SLE66
typedef struct {
   AGSIADDR        vec;
   char           *mess;       // Interrupt name
   const char   *rname;        // name of interrupt request bit
   const char   *ename;        // name of interrupt enable bit
   const char   *pname;        // name of interrupt priority bit
   DWORD           num;        // Interrupt Number
   DWORD           pwl;        // priority within level
} AGSIINTERRUPT;
```

Note:

This function may only be called during the initialization process.

Example:

```
// 8051/251/8051Mx
#define TCON 0x88
#define IE 0xA8
#define IP 0xB8
#define IPH 0xB7

AGSIINTERRUPT ExtInt0 = { // External Interrupt 0
0x0003, "P3.2/Int0", TCON, 0x01, "IT0", TCON, 0x02, "IE0", IE, 0x01, "EX0", IP,
0x01, "Pri", IPH, 0x01, 8, 1
};
AGSIINTERRUPT Timer0Int = { // Timer 0 Interrupt
0x000B, "Timer 0", 0, 0, "", TCON, 0x20, "TF0", IE, 0x02, "ET0", IP,
0x02, "Pri", IPH, 0x02, 6, 1
};

AgsiDeclareInterrupt(&Timer0Int);
AgsiDeclareInterrupt(&ExtInt0);

// 80166
#define S0TIC 0xFF6C

AGSIINTERRUPT SerTransmitInt = { // Serial Transmit Interrupt
0x00A8, "S0TINT", S0TIC
}
AgsiDeclareInterrupt(&SerTransmitInt);
```

AgsiSetWatchOnSFR

Summary:

```
BOOL AgsiSetWatchOnSFR(AGSIADDR SFRAddress, AGSICALLBACKA
                       pfnReadWrite,AGSIACCESS eAccess);
```

Parameter:

`SFRAddress`	Address of the SFR
`pfnReadWrite`	Pointer to a function that is called on SFR access. The function gets as argument the address and the access reason to the memory and does not have a return value (void function(DWORD adr, AGSICB_REASON r)).
`eAccess`	Access type (AGSIREAD, AGSIWRITE, AGSIREADWRITE)

Return Value:

TRUE if successful otherwise FALSE.

Description:

This function is used to set a watch on SFR access. Whenever the specified SFR is accessed, the specified function is called.

Example:

```
#define TCON 0x88

#define TL1 0x8B
#define TH1 0x8D

static void timer1(DWORD adr, AGSICB_REASON r) {
// watch function implementation
}

AgsiSetWatchOnSFR(TH1, timer1, AGSIREADWRITE); //Call 'timer1' when TH1 is written or read
AgsiSetWatchOnSFR(TL1, timer1, AGSIREADWRITE); //Call 'timer1' when TL1 is written or read
AgsiSetWatchOnSFR(TCON, timer1, AGSIWRITE);    //Call 'timer1' when TCON is written
```

AgsiSetWatchOnVTR

Summary:

```
BOOL AgsiSetWatchOnVTR(AGSIVTR hVTR, AGSICALLBACKA pfnReadWrite,
                       AGSIACCESS eAccess);
```

Parameter:

`hVTR`	Handle of previously defined VTR
`pfnReadWrite`	Pointer to a function that is called on VTR access. The function gets as argument the address and the access reason to the memory and does not have a return value (void function(DWORD adr, AGSICB_REASON r)).
`eAccess`	Access type (AGSIREAD, AGSIWRITE, AGSIREADWRITE)

Return Value:

TRUE if successful otherwise FALSE.

Description:

This function is used to set a watch on virtual register (VTR) access. Whenever the specified VTR is accessed, the specified function is called.

Example:

```
static void timer1(DWORD adr, AGSICB_REASON r) {
// watch function implementation
}

hPORT3 = AgsiDefineVTR("PORT3", AGSIVTRCHAR, 0xFF);     // Port 3 pins

AgsiSetWatchOnVTR(hPORT3, timer1, AGSIWRITE);
```

AgsiSetWatchOnMemory

Summary:

```
BOOL AgsiSetWatchOnMemory(AGSIADDR StartAddress, AGSIADDR EndAddress,
                          AGSICALLBACKA pfnReadWrite, AGSIACCESS eAccess);
```

Parameter:

`StartAddress`	Start Address of Memory range. See chapter ‘Address representation’!
`EndAddress`	End Address of Memory range. See chapter ‘Address representation’!
`pfnReadWrite`	Pointer to a function that is called on Memory range access.
`eAccess`	Access type (AGSIREAD, AGSIWRITE, AGSIREADWRITE).

Return Value:

TRUE if successful otherwise FALSE.

Description:

This function is used to set a watch on memory range access. Whenever the specified memory area is accessed, the specified function is called. Please make sure that the specified memory area is mapped before a watch is set on it. If the memory area is not mapped, an ‘access violation’ would be reported in the output window of µVision. With the ‘AgsiExecuteCommand’ a MAP command (see µVision manual) can be executed.

The `Startaddress` and `Endaddress` usually refer to the XDATA memory for 8051/251 and 8051MX architectures when values between 0 and 0xFFFF are used. For other memory areas, please see chapter ‘Address Representation’.

Example:

```
static void eeprom(DWORD adr, AGSICB_REASON r) {
// watch function implementation
}

AgsiSetWatchOnMemory(0x0200, 0x02FF, eeprom, AGSIWRITE); // Watch on Write to Memory range
```

AgsiCreateTimer

Summary:

```
AGSITIMER AgsiCreateTimer(AGSICALLBACK pfnTimer);
```

Parameter:

`pfnTimer`	Pointer to a function that is called when timer expires

Return Value:

Timer handle if successful otherwise NULL.

Description:

This function is used to create a software timer which is associated with the specified function. Whenever the timer expires (see AgsiSetTimer function) the specified function is called.

Note:

This function may only be called during the initialization process.

Example:

```
static void AdcCompleted(void) {
// Timer function implementation
}

AGSITIMER Timer;
Timer = AgsiCreateTimer(AdcCompleted);

AgsiSetTimer(Timer, 10);      // set timer (with AdcCompleted handle) to 10 cycles
```

AgsiSetTimer

Summary:

```
BOOL AgsiSetTimer(AGSITIMER hTimer, DWORD dwClock);
```

Parameter:

`hTimer`	Timer handle
`dwClock`	Number of machine cycles before the timer watch function is called. A value of –1 (0xffffffff) disables the timer.

Return Value:

TRUE if successful otherwise FALSE.

Description:

This function is used to set the timer expiration time in states. When the specified number of states is executed, the function that is associated with the timer handle (see AgsiCreateTimer) is called. This function must set a new timer value with AgsiSetTimer, either a new value or –1 to disable the timer. A timer does not automatically reload the last value, it must be set every time it expires or before.

Example:

```
static void AdcCompleted(void) {
// Timer function implementation
}

AGSITIMER Timer;
Timer = AgsiCreateTimer(AdcCompleted);

AgsiSetTimer(Timer, 10);      // set timer (with AdcCompleted handle) to 10 cycles
```

AgsiDefineMenuItem

Summary:

```
BOOL AgsiDefineMenuItem(AGSIMENU *pDym);
```

Parameter:

`pDym`	Pointer to an AGSIMENU structure (see below).

Return Value:

TRUE if successful otherwise FALSE.

Description:

This function is used to define a new menu item in the 'Peripherals' pull-down menu of µVision and the associated dialog. The menu item is described in the following AGSIMENU structure:

```
#define AGSIMENU struct AgsiDynaM
struct AgsiDynaM {                    // Menu item data structure
  int             nDelim;             // Menu template delimiter
  char           *szText;             // Menu item text
  void   (*fp) (AGSIMENU *pM);        // create/bring DlgtoTop function
  DWORD               nID;            // uv2 assigned ID_xxxx
  DWORD           nDlgId;             // Dialog ID
  AGSIDLGD          *pDlg;            // link to dialog attributes
};
```

`nDelim`:	1: Standard Menu entry 2: Popup-Entry (nested submenu) -2: End of Popup-Group-List
`szText`:	The text that appears in the pull-down menu.
`fp`:	Function that will be called on menu-selection.
`nID`:	Variable used for internal purposes. Do not modify!
`NDlgId`:	Dialog identifier.
`PDlg`:	Pointer to AGSIDLGD structure which contains the dialog properties (see below).

The dialog is described in the following AGSIDLGD structure:

```
#define AGSIDLGD struct AgsiDlgDat
struct AgsiDlgDat {                   // every dialog has it's own structure
  DWORD            iOpen;             // auto reopen dialog (pos := 'rc')
  HWND                hw;             // Hwnd of Dialog
  BOOL (CALLBACK *wp) (HWND hw, UINT msg, WPARAM wp, LPARAM lp);
  RECT                rc;             // Position rectangle
  void   (*Update) (void);            // Update dialog content
  void (*Kill) (AGSIDLGD *pM);        // Kill dialog
  void                *vp;            // reserved for C++ Dialogs (Dlg *this)
};
```

`iOpen`:	This member of the structure can be used to store the status of the dialog (open/close) when the µVision debugger is closed.
`hw`:	Window handle of dialog when it is open.
`wp`:	C dialog function that gets windows messages and notifications when no Microsoft Foundation Class (MFC) is used. Set to NULL when MFC is used.
rc:	Dialog position coordinates can be stored here in order to reopen the dialog at the same location. These coordinates can be written into the project file with AgsiSetKey when the debugger is closed.

`Update`: Pointer to dialog update function. This function is called whenever the screen needs to be updated, for example after a single step.

`Kill`: Pointer to function that closes the dialog. This function is called when debugger is closed.

Note:

This function may only be called during the initialization process.

Example:

```
// Prototypes for forward references
static void PeriUpdate (void);
static void PeriKill (AGSIDLGD *pM);
static void PeriDisp (AGSIMENU *pM);

// Peripheral Dialog
AGSIDLGD PeriDlg = { 0, NULL, NULL, { -1, -1, -1, -1, }, PeriUpdate, PeriKill };

// Peripheral Menu Item
AGSIMENU PeriMenu = { 1, "&A/D Converter" , PeriDisp, 0, IDD_ADCON, &PeriDlg };

void Init(void) {
  AgsiDefineMenuItem(&PeriMenu);
}
```

AgsiRegisterExecCallBack

Summary:

```
BOOL AgsiRegisterExecCallBack(void (*fp) (DWORD pc, DWORD Ypc));
```

Parameter:

`fp`	Pointer to callback function which receives the parameters DWORD `pc` and DWORD `Ypc`. `pc` contains the logical address of the current instruction, `Ypc` contains the physical address.

Return Value:

TRUE if successful otherwise FALSE.

Description:

This function registers a callback function which will be called before the execution of each instruction.

Note:

This function may only be called during the initialization process.

Example:

```
static void Execution(DWORD pc, DWORD Ypc) {
// will be called at each instruction execution
}

AgsiRegisterExecCallBack(Execution);
```

AgsiWriteSFR

Summary:

```
BOOL AgsiWriteSFR(AGSIADDR SFRAddress, DWORD dwValue, DWORD dwMask);
```

Parameter:

`SFRAddress`	Address of the SFR
`dwValue`	Value to write into the SFR
`dwMask`	Mask to use for writing

Return Value:

TRUE if successful otherwise FALSE.

Description:

This function is used to write a new value into an SFR. The mask specifies which SFR bits are overwritten with the new value. A mask bit of 0 does not modify the old value.

Example:

```
AgsiWriteSFR(0xA8, 0x80, 0xFF);      // Write 0x80 to the SFR at Address 0xA8
AgsiWriteSFR(0xA8, 0x80, 0x80);      // Set the MSB bit of the SFR at Address 0xA8
                                     // (other bits unchanged)
```

AgsiReadSFR

Summary:

```
BOOL AgsiReadSFR (AGSIADDR SFRAddress, DWORD* pdwCurrentValue,
                  DWORD* pdwPreviousValue, DWORD dwMask);
```

Parameter:

`SFRAddress`	Address of the SFR
`pdwCurrentValue`	Pointer to current Value of the SFR which will be read
`pdwPreviousValue`	Pointer to previous Value of the SFR which will be read
`dwMask`	Mask to use for reading

Return Value:

TRUE if successful otherwise FALSE.

Description:

This function is used to read the value from the SFR with specified mask. The value of the SFR is ANDed with `dwMask` before it is written into the `pdwCurrentValue` or `pdwPreviousValue`. Of course, the value of the SFR remains unchanged. `pdwCurrentValue` and `pdwPreviousValue` is only different when AgsiReadSFR is called within a watch function that was triggered by a write access to the same SFR. This is used to detect transitions of bits in SFRs (e.g. start bit).

Example:

```
#define IE 0xA8
DWORD cIE, pIE;
AgsiReadSFR(IE, cIE, pIE, 0xFF);
```

AgsiSetSFRReadValue

Summary:

```
BOOL AgsiSetSFRReadValue(DWORD dwValue);
```

Parameter:

`dwValue`	SFR Read Value

Return Value:

TRUE if successful otherwise FALSE.

Description:

This function is used to override the value that was read from a SFR with the current instruction. The SFR value itself is not modified. This is needed for I/O ports and maybe also other SFR's. When a I/O port (configured as input) is read with an instruction like 'MOV A,P0' the CPU reads the value of the port pins instead of the content of the P0 register. In order to simulate this behavior, the AGSISetSFRReadValue function has to pass the value of the port pins to the instruction simulator whenever the port is read. The AGSISetSFRReadValue function must be called from a function that is called with a read access to the respective SFR.

Example:

```
#define P0 0x80

void init(void) {
  AgsiSetWatchOnSFR(P0, P0Read, AGSIREAD); //Call 'P0READ' when P0 is read
}

static void P0Read(void) {     // function called with every read access of P0
  ...
  AgsiSetSFRReadValue(P0 & PORT0);   // P0 (Port0 SFR Value), PORT0 (Port0 pins)
}
```

AgsiWriteVTR

Summary:

```
BOOL AgsiWriteVTR(AGSIVTR hVTR, DWORD dwValue);
```

Parameter:

`hVTR`	VTR handle
`dwValue`	Value to write into the VTR

Return Value:

TRUE if successful otherwise FALSE.

Description:

This function is used to write a value into the VTR.

Example:

```
DWORD port1;

union fv {       // float value union
  float f;
  DWORD DW;
} vref;

port1 = 0x01;
AgsiWriteVTR(hPORT1, port1);    // Write port1 to VTR with hPORT1 handle (char VTR)

vref.f = 2.5;
AgsiWriteVTR(hVREF, vref.DW);   // Write vref to VTR with hVREF handle (float VTR)
```

AgsiReadVTR

Summary:

```
BOOL AgsiReadVTR (AGSIVTR hVTR, DWORD* pdwCurrentValue);
```

Parameter:

`hVTR`	VTR handle
`pdwCurrentValue`	Pointer to current Value of the VTR which will be read

Return Value:

TRUE if successful otherwise FALSE.

Description:

This function is used to read the value from the VTR.

Example:

```
DWORD port1;

union fv {       // float value union
  float f;
  DWORD DW;
} vref;

AgsiReadVTR(hPORT1, &port1);    // Read VTR value with hPORT1 handle (char VTR) into port1
AgsiReadVTR(hVREF, &vref.DW);   // Read VTR value with hVREF handle (float VTR) into vref
```

AgsiWriteMemory

Summary:

```
BOOL AgsiWriteMemory(AGSIADDR Address, DWORD dwCount, BYTE* pbValue);
```

Parameter:

`Address`	Start Address of Memory. See chapter ‘Address representation’!
`dwCount`	Number of bytes to write.
`pBValue`	Pointer to buffer which data will be written.

Return Value:

TRUE if successful otherwise FALSE.

Description:

This function is used to write data into memory. The Address usually refers to the XDATA memory for 8051/251 and 8051MX architectures when values between 0 and 0xFFFF are used. For other memory areas, please see chapter ‘Address Representation’.

Please make sure that the specified memory area is mapped before writing to it. If the memory area is not mapped, an ‘access violation’ would be reported in the output window of µVision. With the ‘AgsiExecuteCommand’ a MAP command (see µVision manual) can be executed.

Example:

```
BYTE buffer[10];
AgsiWriteMemory(0x1000, 10, buffer);            // write 10 bytes to (XDATA) address 0x1000
AgsiWriteMemory(0x1000|(amCODE<<24),10,buffer); // write 10 bytes to (CODE) address 0x1000
```

AgsiReadMemory

Summary:

```
BOOL AgsiReadMemory(AGSIADDR Address, DWORD dwCount, BYTE* pbValue);
```

Parameter:

`Address`	Start Address of Memory
`dwCount`	Number of bytes to read
`pBValue`	Pointer to buffer in which data will be read

Return Value:

TRUE if successful otherwise FALSE.

Description:

This function is used to read data from memory. The Address usually refers to the XDATA memory for 8051/251 and 8051MX architectures when values between 0 and 0xFFFF are used. For other memory areas, please see chapter 'Address Representation'.

Please make sure that the specified memory area is mapped before reading from it. If the memory area is not mapped, an 'access violation' would be reported in the output window of µVision. With the 'AgsiExecuteCommand' a MAP command (see µVision manual) can be executed.

Example:

```
BYTE buffer[10];
AgsiReadMemory(0x1000, 10, buffer);           // read 10 bytes from (XDATA) address 0x1000
AgsiReadMemory(0x1000|(amCODE<<24),10,buffer); // read 10 bytes to (CODE) address 0x1000
```

AgsiGetStates

Summary:

```
UINT64 AgsiGetStates(void);
```

Parameter:

None

Return Value:

Number of machine states executed.

Description:

This function is used to get the number of states executed. This number is also visible in the register window.

Example:

```
UINT64 States;
States = AgsiGetStates();
```

AgsiGetProgramCounter

Summary:

```
AGSIADDR AgsiGetProgramCounter(void);
```

Parameter:

None

Return Value:

Value of program counter, see chapter ‘Address representation’.

Description:

This function is used to get the current program counter value (PC). This number is also visible in the register window.

Example:

```
AGSIADDR pc;
pc = AgsiGetProgramCounter();
```

AgsiIsInInterrupt

Summary:

```
DWORD AgsiIsInInterrupt(void);
```

Parameter:

None

Return Value:

The following values or the sum of several values are possible:

0: No interrupt is pending or in progress.

1: An interrupt with priority 0 is in progress.

2: An interrupt with priority 1 is in progress.

4: An interrupt with priority 2 is in progress.

8: An interrupt with priority 3 is in progress.

A value of 10 means that an interrupt with priority 3 has interrupted another interrupt with priority 1. The values are OR'ed if interrupts are nested.

Description:

This function is used to examine if and which interrupt level is in progress. It also indicates the number of interrupts that are currently nested.

Note:

This function is only implemented for 8051/251 and 8051Mx derivatives, not for 80C166 derivatives. In 80C166 derivatives, the PSW register contains the current interrupt level.

Example:

```
if (AgsiIsInInterrupt()) {
    // interrupt in progress
} else {
    // normal program execution
}
```

AgsiIsSleeping

Summary:

```
BOOL AgsiIsSleeping(void);
```

Parameter:

None

Return Value:

TRUE if CPU is in sleep mode, FALSE if CPU is running.

Description:

This function is used to examine if the CPU is in sleep mode (power save mode).

Example:

```
if (AgsiIsSleeping()) {
    // CPU is in sleep mode.
} else {
    // Normal CPU mode
}
```

AgsiIsSimulatorAccess

Summary:

```
BOOL AgsiIsSimulatorAccess(void);
```

Parameter:

None

Return Value:

TRUE if called from the simulated program, FALSE if called for simulator internal use.

Description:

This function is used to distinguish between a running simulation and a call of a callback function to retrieve data for µVision internal use. This includes, beside others, the display of values in the memory window. It can be used to return the value of a SFR without doing any other simulation overhead or changes.

Example:

```
#define TCON 0x88

static void timer1(void) {
  if (AgsiIsSimulatorAccess()) {
    // do all necessary operations to simulate reading this SFR
  } else {
    AgsiSetSFRReadValue(currentValue);     // simply set the current SFR value
  }
}

AgsiSetWatchOnSFR(TCON, timer1, AGSIREAD); // call 'timer1' when TCON is read
```

AgsiIsSyncInstruction

Summary:

```
BOOL AgsiIsSyncInstruction(void);
```

Parameter:

None

Return Value:

TRUE if called the currently executed instruction is a SYNC (SLE66: MOVB), FALSE otherwise.

Description:

This function is used to test for the execution of a SLE66 MOVB instruction.

Example:

```
#define TCON 0x88

static void timer1(void) {
  if (AgsiIsSyncInstruction()) {
    AgsiSetSyncDelay(1);      // delay this instruction, ‘timer1’ will be called again
  }
}

AgsiSetWatchOnSFR(TCON, timer1, AGSIREAD); // call ‘timer1’ when TCON is read
```

AgsiGetSyncCount

Summary:

`int AgsiGetSyncCount(void);`

Parameter:

None

Return Value:

Access count of a MOVB instruction.

Description:

This function returns the access count of a SLE66 MOVB instruction.

Example:

```
#define TCON 0x88

static void timer1(void) {
  if (AgsiIsSyncInstruction()) {
    int count = AgsiGetSyncCount();  // current access count
    AgsiSetSyncCount(count+1);       // set new access count
  }
}

AgsiSetWatchOnSFR(TCON, timer1, AGSIREAD); // call ‘timer1’ when TCON is read
```

AgsiSetSyncCount

Summary:

```
void AgsiSetSyncCount(int nCount);
```

Parameter:

`nCount`	New access count of a MOVB instruction.

Return Value:

None.

Description:

This function sets the access count of a SLE66 MOVB instruction.

Example:

```
#define TCON 0x88

static void timer1(void) {
  if (AgsiIsSyncInstruction()) {
    int count = AgsiGetSyncCount();  // current access count
    AgsiSetSyncCount(count+1);       // set new access count
  }
}

AgsiSetWatchOnSFR(TCON, timer1, AGSIREAD); // call ‘timer1’ when TCON is read
```

AgsiSetSyncDelay

Summary:

```
BOOL AgsiSetSyncDelay(DWORD dwClock);
```

Parameter:

`dwClock`	0 if no delay, otherwise the MOVB instruction will be repeated.

Return Value:

TRUE if successful otherwise FALSE.

Description:

This function requests a delay for the current SFR access, if `dwClock` is not 0 the access will be repeated.

Note:

This function may only be called in a SFR watch function.

Example:

```
#define TCON 0x88

static void timer1(void) {
  if (AgsiIsSyncInstruction()) {
    AgsiSetSyncDelay(1);      // delay this instruction, 'timer1' will be called again
  }
}

AgsiSetWatchOnSFR(TCON, timer1, AGSIREAD); // call 'timer1' when TCON is read
```

AgsiStopSimulator

Summary:

```
void AgsiStopSimulator(void);
```

Parameter:

None.

Return Value:

None.

Description:

This function is used to stop the simulation. This is useful when the simulated peripheral or the application causes a serious conflict. An error message should be printed into the command window (see AgsiMessage) or a message box should be opened in this case to notify the user.

Example:

```
if (critical_error) {
  AgsiStopSimulator()          // stop simulation
}
```

AgsiWakeUp

Summary:

`void AgsiWakeUp(WORD nSleepNumber);`

Parameter:

`nSleepNumber`	currently not in use, must be 0

Return Value:

None.

Description:

Exit the sleep mode of the processor. All peripheral DLLs will be notified with AGSI_EXITSLEEP.

Example:

```
static void AdcCompleted(void) {
  AgsiWakeUp(0);              // exit sleep mode
}

AGSITIMER Timer;
Timer = AgsiCreateTimer(AdcCompleted);

AgsiSetTimer(Timer, 10);      // set timer (with AdcCompleted handle) to 10 cycles
```

AgsiTriggerReset

Summary:

```
void AgsiTriggerReset(void);
```

Parameter:

None.

Return Value:

None.

Description:

This function is used to trigger a CPU reset and stop the simulation. It can be used when a watchdog timer should be simulated or an external device causes a reset.

Example:

```
if (WatchdogOverflow) {
  AgsiTriggerReset()          // reset the CPU
}
```

AgsiContinue

Summary:

```
void AgsiContinue(void);
```

Parameter:

None.

Return Value:

None.

Description:

Continue execution after a stop and exit the sleep mode of the processor. All peripheral DLLs will be notified with AGSI_EXITSLEEP. To be used for example after an AgsiTriggerReset().

Example:

```
if (WatchdogOverflow) {
  AgsiTriggerReset()           // reset the CPU
  AgsiContinue()               // continue the execution at the reset vector
}
```

AgsiRequestInterrupt

Summary:

```
BOOL AgsiRequestInterrupt(WORD nNumber);
```

Parameter:

`nNumber`	Interrupt number

Return Value:

TRUE if successful otherwise FALSE.

Description:

This function sets the corresponding interrupt request flag. The Interrupt number must be in the range of possible interrupts.

Example:

```
if (WatchdogOverflow) {
  AgsiRequestInterrupt(7);     // request a NMI
}
```

AgsiSetSyncCount

Summary:

```
void AgsiSetSyncCount(int nCount);
```

Parameter:

`nCount`	Access count of a MOVB instruction

Return Value:

None.

Description:

This function sets the access count of a SLE66 MOVB instruction.

Example:

```
#define TCON 0x88

static void timer1(void) {
  if (AgsiIsSyncInstruction()) {
    int count = AgsiGetSyncCount();  // current access count
    AgsiSetSyncCount(count+1);       // set new access count
  }
}

AgsiSetWatchOnSFR(TCON, timer1, AGSIREAD); // call ‘timer1’ when TCON is read
```

AgsiUpdateWindows

Summary:

```
void AgsiUpdateWindows(void);
```

Parameter:

None.

Return Value:

None.

Description:

This function is used to force µVision to update all windows. This function is necessary to keep dialogs, watch windows or memory windows up to date when a new SFR value has been entered in a different dialog. Do not use this function from a function that is called because of a time or access watch! Calling this function frequently would slow down the simulator performance immense.

Example:

```
void CPeriDialog::OnPort0Input() {
  AgsiWriteVTR(hPORT0, port0);
  AgsiUpdateWindows();        // update all windows to display the new Port0 value
}
```

AgsiHandleFocus

Summary:

```
void AgsiHandleFocus (HWND hwndDialog);
```

Parameter:

hwndDialog	Dialog handle if dialog gets focus or NULL if dialog looses focus.

Return Value:

None.

Description:

This function is needed to forward accelerator keys such as TAB to the dialog message handler. Whenever a dialog receives a WM_ACTIVATE message, it has to update its status with this function.

Example:

```
void CPeriDialog::OnActivate(UINT nState, CWnd* pWndOther, BOOL bMinimized) {
  CDialog::OnActivate(nState, pWndOther, bMinimized);
  switch (nState) {
    case WA_INACTIVE:
      AgsiHandleFocus(NULL);         // Clear Modeless Handle
      break;
    case WA_ACTIVE:
    case WA_CLICKACTIVE:
      AgsiHandleFocus(m_hWnd);       // Set Modeless Handle
      break;
  }
}
```

AgsiGetExternalClockRate

Summary:

```
DWORD AgsiGetExternalClockRate(void)
```

Parameter:

None.

Return Value:

External clock frequency in Hz.

Description:

This function is used to retrieve the external clock rate. The virtual register XTAL contains the same value. The external clock frequency can be set in µVision under ‘Options for Target’ -> ‘Target Clock’ (e.g. a standard 8051 microcontroller runs at 12 MHz).

Example:

```
If (AgsiGetExternalClockRate() > 20000000) { // ext. clock > 20MHz
  AgsiMessage(“CAN controller cannot work with the specified CPU clock\n”);
}
```

AgsiSetExternalClockRate

Summary:

```
BOOL AgsiSetExternalClockRate(DWORD dwRate)
```

Parameter:

`dwRate`	External clock frequency in Hz.

Return Value:

TRUE if successful otherwise FALSE.

Description:

This function is used to set the external clock rate. The virtual register XTAL contains the same value. The external clock frequency can also be set in µVision under ‘Options for Target’ -> ‘Target Clock’ (e.g. a standard 8051 microcontroller runs at 12 MHz).

Example:

```
If (bRequestLowPower) {
  AgsiSetExternalClockRate(10000)    // set ext. clock to a slow rate for low power
}
```

AgsiGetInternalClockRate

Summary:

```
DWORD AgsiGetInternalClockRate(void)
```

Parameter:

None.

Return Value:

Internal clock frequency in Hz.

Description:

This function is used to retrieve the internal CPU clock rate. The virtual register CLOCK contains the same value. The internal clock frequency is calculated out of the external clock frequency divided by a clock prescaler. This prescaler is programmable in some derivatives in order to save power. The internal clock frequency of a standard 8051 microcontroller is typically 1 MHz (external clock /12).

Example:

```
DWORD Clock;
Clock = AgsiGetInternalClockRate();         // get internal clock rate
Baudrate = Clock / BaudratePrescaler;       // calculate baudrate
```

AgsiGetClockFactor

Summary:

```
double AgsiGetClockFactor(void)
```

Parameter:

None.

Return Value:

External clock to internal clock ratio.

Description:

This function is used to retrieve the external clock to internal clock ratio. This value is needed whenever a simulated peripheral is driven with the external clock or if it has its own independent timing. Since time watches are always based on the internal clock rate, this factor is needed to calculate the time for such peripherals. This factor may change during program execution if the simulated microcontroller has a programmable clock prescaler in order to save power. On the other hand, the value of this factor may be between 0 and 1 when the CPU has a clock multiplier (PLL). Before and after this factor changes, all peripherals are notified (see AgsiEntry AGSI_PREPLL and AGSI_POSTPLL) so that time watches can be recalculated.

Example:

```
double prescaler;
prescaler = AgsiGetClockFactor();
AgsiSetTimer(mytimer, (DWORD)(1000.0 / prescaler)); // set timer to 1000 external cycles
```

AgsiMessage

Summary:

```
void AgsiMessage(const char* pszFormat, ...)
```

Parameter:

printf compatible.

Return Value:

None.

Description:

This function prints a string into the command window of µVision. The parameters are compatible to a C printf function. With this function it is possible to output warnings or debug messages.

Note:

The content of the command window can be logged to file by using the 'LOG' command.

Example:

```
AgsiMessage ("Timer was started at address %d \n", AgsiGetProgramCounter());
```

AgsiSetTargetKey

Summary:

```
BOOL AgsiSetTargetKey(const char* pszKey, const char *pszString)
```

Parameter:

`pszKey`	String that specifies the key name.
`pszString`	String that contains the information to be stored.

Return Value:

None.

Description:

This function stores a text string in the project file (*.OPT). It can be used to store configuration information such as dialog positions so that a dialog opens at the same place. These settings are stored for each target of a project separately.

Example:

```
// store dialog position in the project file
AgsiSetTargetKey ("MYKEY", "Dialog1 XPOS=%d YPOS=%d", rc.x, rc.y);
```

AgsiGetTargetKey

Summary:

```
const char * AgsiGetTargetKey(const char* pszKey)
```

Parameter:

`pszKey`	String that specifies the key name.
`pszString`	String that contains the information to be stored.

Return Value:

Pointer to string that was stored in the project file or NULL if the key was not found.

Description:

This function retrieves a text string that was written into the project file (*.OPT) with AgsiSetTargetKey before.

Example:

```
pMyConfiguration = AgsiGetTargetKey (“MYKEY”);    // retrieves the string
sscanf(pMyConfiguration, “%d”, &rc.x);            // convert string into values…
```

AgsiExecuteCommand

Summary:

```
void AgsiExecuteCommand(const char* pszCommand)
```

Parameter:

`PszCommand`	Pointer to string that contains a valid µVision debugger command.

Return Value:

None.

Description:

This function copies the specified string into the µVision command line and executes it. The string must contain a valid µVision debugger command. The command and its output are visible in the command window. It can be used e.g. to map memory or to open a log file.

Example:

```
AgsiExecuteCommand(“MAP X:0x1000, X:0x1FFF read write”);              // for an 8051
AgsiExecuteCommand(“MAP 0x100000, 0x10FFFF read write execute”);      // for an 80166
AgsiExecuteCommand(“LOG >C:\MYLOGFILE.LOG”);
```

AgsiGetLastMemoryAddress

Summary:

```
DWORD AgsiGetLastMemoryAddress(void)
```

Parameter:

None.

Return Value:

Address of last memory access. See chapter ‘Address representation’!

Description:

This function is used to determine which read or write watch caused the function call when more than one access watch is set.

Example:

```
void updatetimer(void) {
  DWORD LastAddress;
  LastAddress = AgsiGetLastMemoryAddress();
  if (LastAddress == TCFG) {           // updatetimer was called because of TCFG access
  } else if (LastAddress == THIGH) {   // updatetimer was called because of THIGH access
  }
}
```

AgsiGetSymbolByName

Summary:

```
DWORD AgsiGetSymbolByName (AGSISYMDSC *pSymbol)
```

Parameter:

`pSymbol`	Pointer to AGSISYMDSC structure.

Return Value:

TRUE if successful otherwise FALSE.

Description:

This function is used to determine the value of a symbol. The function parameter points to a structure that holds the name of the symbol to be searched. The search result (val, type and Ok) is written into the same structure.

```
typedef struct {               // Search for Sym by Name or Value.
  AGSISYMMASK nMask;           // search mask (AGSI_SYM_LOC | ...)
  char szName [256];           // search/found name (zero-terminated
  UINT64        val;           // search/found Adr/Value
  AGSISYMTYPE  type;           // type of found symbol (AGSI_TP_???)
  DWORD          Ok;           // 1:=Ok, else find failed.
} AGSISYMDSC;
```

`nMask`:	Not used with this function call.
`szName`:	Name of symbol to be searched for. Must be set before AgsiGetSymbolByName is called.
`val`:	Address or value of found symbol.
`type`:	Type of found symbol. See AGSITYPE description on the next page.
`Ok`:	Same as function return value. True if search has been successful or FALSE if not.

Note:

The user application is not yet loaded when the peripheral DLL is initialized. Searching for symbols at this time makes no sense. Since a reset is executed after a load command, the reset function is a good place to request symbol values.

Example:

```
AGSISYMDSC MainSymbol;
DWORD      Found;
AGSIADDR   MainAddress;

strcpy(MySymbol.szName, “main”);
Found = AgsiGetSymbolByName(&MainSymbol);
if (Found && (MainSymbol.type == AGSI_TP_FUNC)) {        // found ‘main’ function?
  MainAddress = (AGSIADDR) MainSymbol.val;               // address of ‘main’ function
}
```

AgsiGetSymbolByValue

Summary:

`DWORD AgsiGetSymbolByValue (AGSISYMDSC *pSymbol)`

Parameter:

`pSymbol`	Pointer to AGSISYMDSC structure.

Return Value:

TRUE if successful otherwise FALSE.

Description:

This function is used to determine a symbol name from its value. The function parameter points to a structure that holds the value and mask of the symbol to be searched. The search result (name, type and Ok) is written into the same structure.

```
typedef struct {                // Search for Sym by Name or Value.
  AGSISYMMASK nMask;            // search mask (AGSI_SYM_LOC | ...)
  char szName [256];            // search/found name (zero-terminated
  UINT64        val;            // search/found Adr/Value
  AGSISYMTYPE  type;            // type of found symbol (AGSI_TP_???)
  DWORD          Ok;            // 1:=Ok, else find failed.
} AGSISYMDSC;
```

`nMask`: Specifies the symbol type to search for. This parameter must be set before AgsiGetSymbolByValue is called. Different types can be combined with '|'. Possible values are:
AGSI_SYM_VAR search for non-bit Variables
AGSI_SYM_CON search for named Constants
AGSI_SYM_BIT search for Bit in Memory
AGSI_SYM_LOC search for Function/Label
AGSI_SYM_SFR search for SFR name.
szName: Name of found symbol.

`val`: Address or value of symbol to search for. This must be set before AgsiGetSymbolByValue is called.

`type`: Type of found symbol. See AGSITYPE below for all possible symbol types.
AGSI_TP_VOID number without specific type
AGSI_TP_BIT bit
AGSI_TP_CHAR signed char (8 bit)
AGSI_TP_UCHAR unsigned char (8 bit)
AGSI_TP_INT signed integer (16 bit)
AGSI_TP_UINT unsigned integer (16 bit)
AGSI_TP_SHORT signed integer (16 bit)
AGSI_TP_USHORT unsigned integer (16 bit)
AGSI_TP_LONG signed long (32 bit)
AGSI_TP_ULONG unsigned long (32 bit)
AGSI_TP_FLOAT floating point number (32 bit)
AGSI_TP_DOUBLE double precision floating point number (64 bit)
AGSI_TP_PTR pointer
AGSI_TP_UNION union
AGSI_TP_STRUCT structure
AGSI_TP_FUNC function
AGSI_TP_STRING char array
AGSI_TP_ENUM enumeration
AGSI_TP_FIELD array

`Ok`: Same as function return value. True if search has been successful or FALSE if not.

Note:

The user application is not yet loaded when the peripheral DLL is initialized. Searching for symbols at this time makes no sense. Since a reset is executed after a load command, the reset function is a good place to request symbol values.

Example:

```
AGSISYMDSC MySymbol;
DWORD      Found;
AGSIADDR   MainAddress;

MySymbol.val = 0xFF;          // search for SFR at address 0xFF
MySymbol.nMask = AGSI_SYM_SFR;
Found = AgsiGetSymbolByValue(&MainSymbol);
if (Found) {                  // found SFR?
  MainSymbol.szName;          // szName contains the SFR name at address 0xFF
}
```

Extensions by Infineon Technologies

The following AGSI functions are modified by Infineon:

AgsiEntry	AgsiDeclareInterrupt

The following AGSI functions are added by Infineon:

AgsiRegisterExecCallBack	AgsiIsSimulatorAccess
AgsiIsSyncInstruction	AgsiGetSyncCount
AgsiGetClockFactor	AgsiWakeUp
AgsiContinue	AgsiRequestInterrupt
AgsiSetSyncCount	AgsiSetSyncDelay
AgsiSetExternalClockRate	

Index