AVIX Real Time Operating System

AVIX for PIC32MX Microchip MPLAB Port Guide

Product Version 5.0.0

AVIX-RT

AVIX
Advanced
Virtual
Integrated
eXecutive

AVIX-RT
Maïsveld 84
5236 VC ‘s-Hertogenbosch
The Netherlands

phone +31(0)6 15 28 51 77
e-mail info@avix-rt.com
www www.avix-rt.com

Table of Contents

Table of Figures

Table of Tables

Table of Code Samples

1 Introduction

This document describes how to use AVIX with a controller belonging to the PIC32MX family of microcontrollers using the MPLAB development environment combined with the C32/XC32 compiler suite.

This document forms an extension to the AVIX User Guide and Reference Guide and should be considered an indivisible part of it. Terms and abbreviations defined in the AVIX User Guide and Reference Guide are also applicable to this Port Guide.

► *The full name of the AVIX port described in this document is AVIX for PIC32MX. Whenever using the name AVIX throughout this document, this refers to AVIX for PIC32MX.*

1.1 Reader Guidance

To find the information you are looking for as efficient as possible, here an overview is given of the different chapters this document contains.

The different chapters can be divided in three categories:

1. **Chapter 2** contains background information on the compiler toolsuite AVIX can be used with.

2. **Chapter 3** contains a compact overview how to create an AVIX based application. When familiar with AVIX, this is all needed to get up and running.

3. **Chapters 4, 5 and 6** contain information how to install AVIX, make required changes to the project settings and configure AVIX. The information in these chapters is not pure background but essential to obtain a working project.

4. The remaining chapters can be considered to contain background information which is not essential to get up and running but do contain information about the way AVIX interacts with the application and with the underlying hardware. In more detail these are:

   - **Chapter 7, Stack usage and Interrupt Service Routines,** provides a description of how to declare ISR's to make use of the AVIX system stack.

   - **Chapter 8, Extended Memory**, provides information on extended RAM, a type of memory present in other AVIX ports and which compatibility mechanisms AVIX provides to allow AVIX based source code to be ported from other ports to AVIX for PIC32MX.

   - **Chapter 9, Power Management**, provides information how to use the power management capabilities offered by the controller and how this is supported by AVIX.

   - **Chapter 10, Resource Usage**, description of the hardware resources used by AVIX and the rules to obey when these or related resources are required by the application.

2 Compiler selection and compatibility

AVIX can be used with applications build using either the C32 or the newer XC32 compiler suite. Both compiler suites can be selected to be used from within the legacy MPLAB8x development environment or from the new MPLABX development environment.

Throughout this document a number of dialogs are shown as they are used by the MPLAB development environment to make specific project settings. These dialogs are based on the C32 compiler suite. When using the XC32 compiler suite, similar dialogs are presented by MPLAB in essence offering the exact same entry fields. Some of the content of the XC32 based dialogs may be a little different from those of the C32 based dialogs.

► *In case of using MPLAB8x, When switching between compiler toolsuites (either from C32 to XC32 or from XC32 to C32), a number of project settings are restored to their default values. Since AVIX depends on a number of custom project settings, after switching between compiler toolsuites, it is essential for these settings to be restored. For details on the applicable settings, see chapter 5.1****Error! Reference source not found.***

When using MPLABX, switching between compilers does not restore any other setting!

When referring to the compiler toolsuite in this document, use is made of the phrase C32/XC32 compiler suite to denote both compiler toolsuites may be used.

3 Getting Started

This chapter contains a quick guide to get up and running with AVIX and the steps needed to create an AVIX based project.

AVIX can be used with the legacy Microchip development environment MPLAB8x and the new Microchip development environment MPLABX, both with the C32/XC32 compiler suite. Both environments require a number of mandatory project settings. These settings are mentioned in the steps described below referring §5 where the applicable settings are described in more detail.

AVIX virtually works out of the box and requires the following steps to be up and running with a new project:

1	Install AVIX See §4 for details. The names and relative order of directories created by the setup utility must remain as specified since AVIX depends on this. The location where AVIX is installed is not important but advised is to choose a project related location.
2	Create a new project Create a new MPLAB project based on the C32/XC32 compiler suite. For MPLAB8x use ***Project – Project Wizard...***, for MPLABX use ***File – New Project…***
3	Create the projects main source file and add to the project Manually add a source file (.c) to the project containing function ‘`void avixMain(void)`‘. An AVIX project does not contain a user provided function ‘`main`’ which is implemented by AVIX. The application must offer ‘`avixMain`’ instead which is the entry point of an AVIX based application.
4	Add AVIX configuration file (AVIXConfig.c) to the project This AVIX file must be built as part of the project like any other project source file.
5	Add AVIX library file (AVIX_PIC32MX_MICROCHIP_MPLAB_E_050000_C32_2.x.a[1], [2]) to the project. This AVIX file must be linked against the other project binaries.
6	Add AVIX interface include path to project settings The location of the AVIX ‘Interface’ directory must be added to the project settings. See §5.1.1 for details on the MPLAB8x environment and §5.2.1 for details on the MPLABX environment.
7	Allow the compiler to pass ‘C’ structs by value AVIX is highly type safe, a feature requiring the compiler to pass ‘C’ structs by value. Change the project settings to allow this by adding compiler flag –fno-pcc-struct-return. See §5.1.2 for details on the MPLAB8x environment and §5.2.2 for details on the MPLABX environment. ***This setting is highly important. An AVIX based application build without this setting will not work.***

Table 1: Project Setup Steps

[1] The capital E in the file name denotes the library of an Extended distribution. Depending on the type of distribution, the filename may be different.

[2] The AVIX library is present in directory _AVIX\Lib. Note the mentioned library is to be used when using a version 2.x of the C32 compiler or the XC32 compiler. When using a version 1.x of the C32 compiler, use library AVIX_PIC32MX_MICROCHIP_MPLAB_E_050000_C32_1.x.a instead.

▶ *The result of the above steps is a project that builds. At this point however, the application does not yet use any AVIX function resulting in the linker ignoring the AVIX library. This in turn results in the linker complaining function `'main'`, which is implemented in this library, cannot be found.*

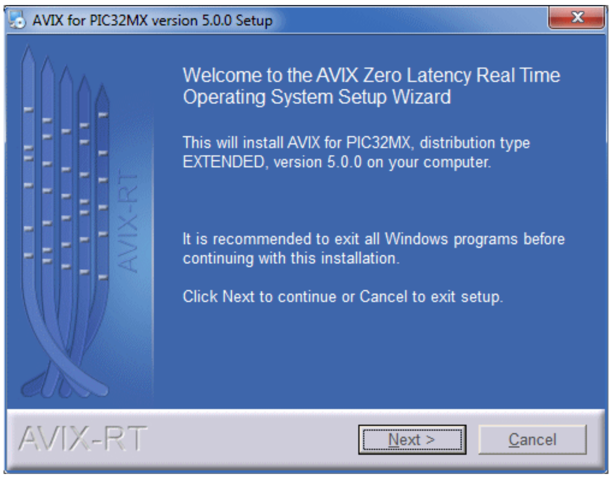
To solve this, AVIX functions must be called from application code, a step required anyway.

From this point on, application code will be added to the project in the form of threads, ISR's and DIH's. Threads are separate 'C' functions that are registered with AVIX so they will run as a thread. Thread functions are typically registered from '`avixMain`' using function `avixThread_Create`. See the AVIX User Guide and Reference Guide for details.

▶ *Function avixMain is executed with all interrupts disabled. For AVIX to work correctly it is essential during execution of avixMain interrupts remain disabled. Make sure not to enable interrupts during execution of avixMain, neither direct nor indirect by (third party) functions being called.*

4 How to Install AVIX

AVIX is distributed in the form a Windows based setup utility, the opening screen of which is shown in Figure 1. Note that the content of this screen may vary based on the type of distribution you have acquired.


AVIX for PIC32MX version 5.0.0 Setup

Welcome to the AVIX Zero Latency Real Time Operating System Setup Wizard

This will install AVIX for PIC32MX, distribution type EXTENDED, version 5.0.0 on your computer.

It is recommended to exit all Windows programs before continuing with this installation.

Click Next to continue or Cancel to exit setup.

Figure 1: AVIX Main Setup Screen

► *Installing AVIX is very straightforward and the only relevant information that is to be provided is the directory where AVIX is to be installed.*

When pressing Next, the install procedure will start. During the install process you will be asked a number of questions allowing you to install those parts of AVIX you need in the directory of your choice.

Note that the AVIX setup utility does not install any executables apart from an uninstall utility named uninstall.exe. This utility is installed in a directory named Uninstall which is created in the main installation directory that is selected as part of the install procedure. Besides this uninstall utility, directory Uninstall contains a number of additional files needed by the uninstall utility.

AVIX can be uninstalled by running this utility through “Add or Remove Programs” present in the Windows Control Panel or from the Windows Start Menu shortcut created as part of the install process. It is not possible to directly run uninstall.exe.

4.1 Directory structure and files

After installing AVIX, the following directory/file structure is created where the root, <Install Directory>, is the directory provided during the install procedure[3]:

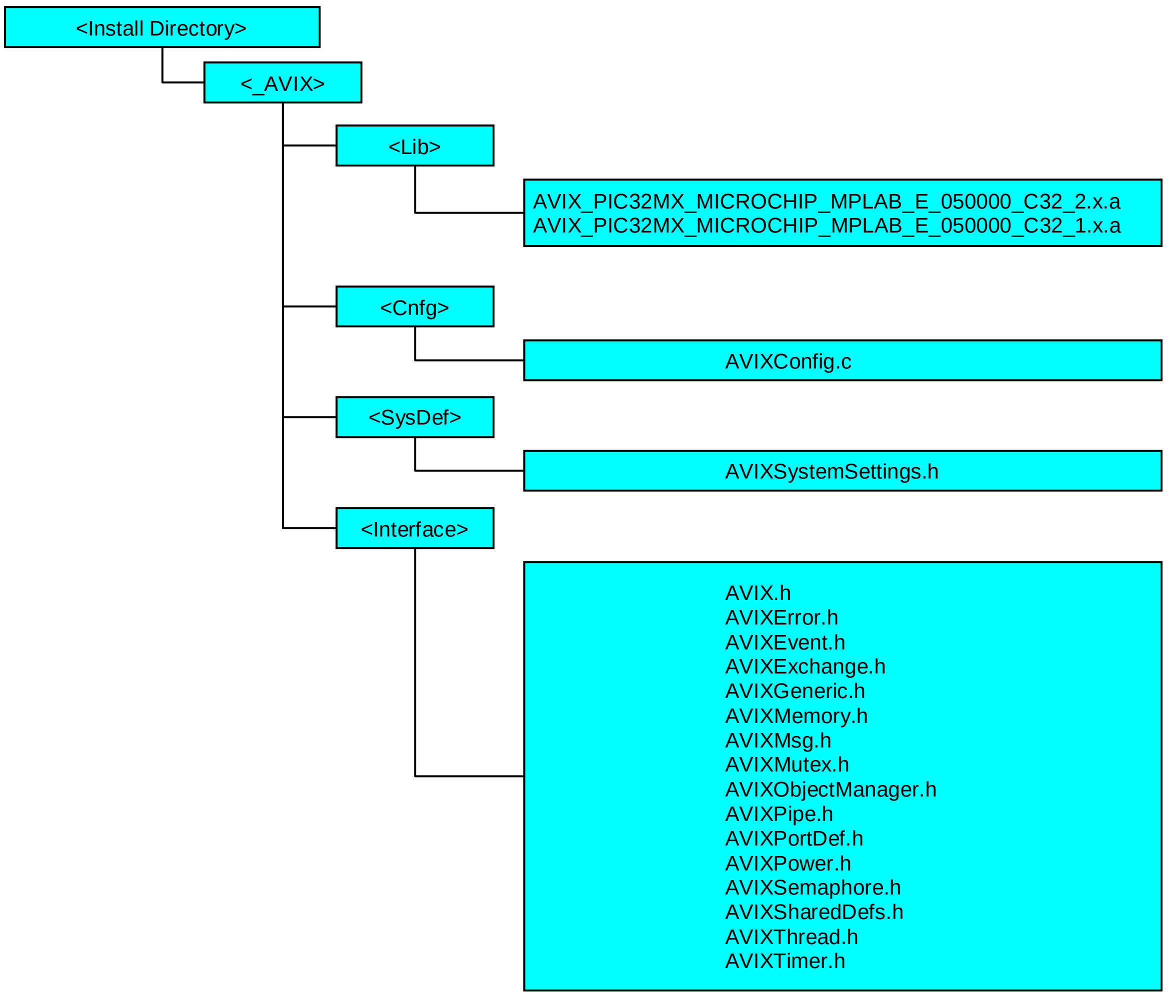


Figure 2: AVIX Directory Structure

The names and relative order of directories ‘Lib’, ‘Cnfg’, ‘SysDef’ and ‘Interface’ must remain as shown in Figure 2 since the build process depends on this.

Below an overview is given of the installed files. Source files requiring AVIX functionality can include file AVIX.h. Optionally the specific header for the desired functionality can be included. For ease of use this is however not advised since AVIX.h in itself is always sufficient.

[3] Depending on options offered during installation, additional directories may be created under the <Project Directory> containing for instance utility software. These additional directories and the files contained in them do not belong to the core of AVIX but are intended to be used with AVIX. When applicable, these files are documented in separate documents.

- **AVIX_PIC32MX_MICROCHIP_MPLAB_E_050000_C32_2.x.a**: Contains the AVIX code. This is a library file that must be linked in the project. The capital ‘E’ in this filename denotes an extended distribution. Depending on the type of distribution used, another character may be present in the filename. This library is to be used when using a 2.x version of the C32 compiler or the XC32 compiler. When using version 1.x of the C32 compiler AVIX library **AVIX_PIC32MX_MICROCHIP_MPLAB8x_E_050000_C32_1.x.a** must be used.

- **AVIXSystemSettings.h**: File containing the configuration settings as described in §6.
This file does never need to be explicitly included.

- **AVIXConfig.c**: Source file needed for configuring AVIX through the settings in AVIXSystemSettings.h.
This file must be compiled and linked as part of the project under development.

- **AVIX.h**: Header file including all other header files.
Including this header file is sufficient to use all AVIX functions without including one of the specific header files.

- **AVIXError.h**: Header file to include when the error facility is used.

- **AVIXEvent.h**: Header file to include when event group functionality is used.

- **AVIXExchange.h:** Header file to include when exchange functionality is used.

- **AVIXGeneric.h**: Header containing generic definitions used from the other header files. ***This file does never need to be explicitly included.***

- **AVIXMemory.h** Header containing definitions for memory pool functions.

- **AVIXMsg.h**: Header file to include when message functionality is used.

- **AVIXMutex.h**: Header file to include when mutex functionality is used.

- **AVIXObjectManager.h**: Header file containing generic definitions related to kernel object handling used for the other header files.
This file does never need to be explicitly included.

- **AVIXPipe.h**: Header file to include when pipe functionality is used.

- **AVIXPortDef.h:** Header containing platform definitions, used by other headers.
This file does never need to be explicitly included.

- **AVIXPower.h:** Header file to when power management functionality is used.

- **AVIXSemaphore.h**: Header file to include when semaphore functionality is used.

- **AVIXSharedDefs.h**: Header file with definitions shared between the configuration file and the AVIX library.
This file does never need to be explicitly included.

- **AVIXThread.h**: Header file to include when thread functionality is used.

- **AVIXTimer.h**: Header file to include when timer functionality is used.

5 Development Environment Settings

AVIX can be used with the legacy Microchip development environment MPLAB8x and the new MPLABX, both with the C32/XC32 compiler suite. A description of these environments falls outside the scope of this document. To use AVIX from within these environments, a number of AVIX specific project settings are however required which are described in this chapter.

5.1 Settings for the MPLAB8x environment

This chapter contains mandatory projects settings when using AVIX with the legacy Microchip development environment MPLAB8x.

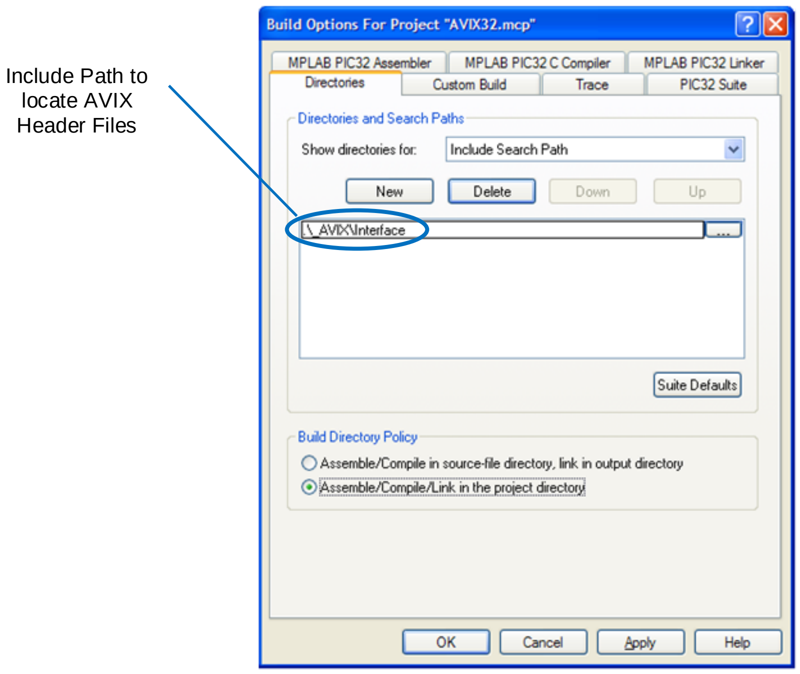
5.1.1 Specify AVIX Include Path

An AVIX based project must have access to the directory containing the AVIX interface files, this location must be added to the MPLAB project settings.

Open the applicable dialog through ***Project - Build Options… - Project – Directories Tab.*** Select 'Include Search Path' from the 'Show Directories for:' dropdown control.

The dialog is shown in Figure 3.

The include path entered depends on the location where AVIX is installed. The value shown is based on AVIX being installed in the project directory.



Show directories for: Include Search Path

._AVIX\Interface

Assemble/Compile in source-file directory, link in output directory [✓ Assemble/Compile/Link in the project directory]

Figure 3: Set AVIX Interface Include Path for MPLAB8x

5.1.2 Structure based function parameters

The functions offered on the AVIX API use ‘C’ structures. The C32/XC32 compiler suite allows for structures to be passed by value or by reference. To implement the high level of type safety the AVIX library is built based on passing structures by value. This implies that code using AVIX functions has to use the same mechanism in order to be compatible with the library.

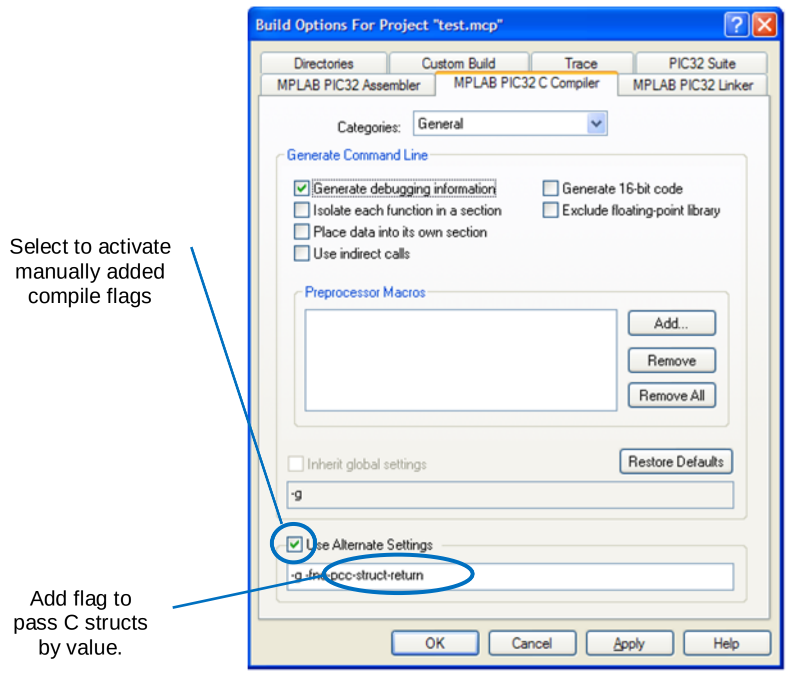
This is accomplished by using compiler flag **-fno-pcc-struct-return**.

Open the applicable dialog through ***Project - Build Options… - Project – MPLAB C32 Tab*** and add the required flag according Figure 4.

It is essential to select checkbox ‘Use Alternate Settings’ in order for the setting to be used.

A consequence of using ‘Alternate Settings’ is that settings made through one of the dialogs (like for instance the compiler optimization level) have to be copied manually to the Alternate Settings field in order to be effective. When for instance changing the compiler optimization level to O3 through the applicable entry, this value will be entered in the default dialog field for settings. After making the setting, manually copy this to the field also used to enter –fno-pcc-struct-return.

► *Setting flag **no-pcc-struct-return** requires use of alternate settings (checkbox “Use Alternate Settings” in the dialog shown in Figure 4. Activating alternate settings implies that changes to the project settings made in the regular way have to be manually copied to the same field where flag –fno-pcc-struct-return is contained in, in order to be persistent when closing this dialog.*



Categories: General

Generate Command Line

- [✓] Generate debugging information
- [ ] Isolate each function in a section
- [ ] Place data into its own section
- [ ] Use indirect calls
- [ ] Generate 16-bit code
- [ ] Exclude floating-point library

[ ] Inherit global settings

-g

[✓] Use Alternate Settings

-g -fno-pcc-struct-return

Figure 4: Struct parameter project setting for MPLAB8x

5.2 Settings for the MPLABX environment

This chapter contains mandatory projects settings when using AVIX with the new Microchip development environment MPLABX.

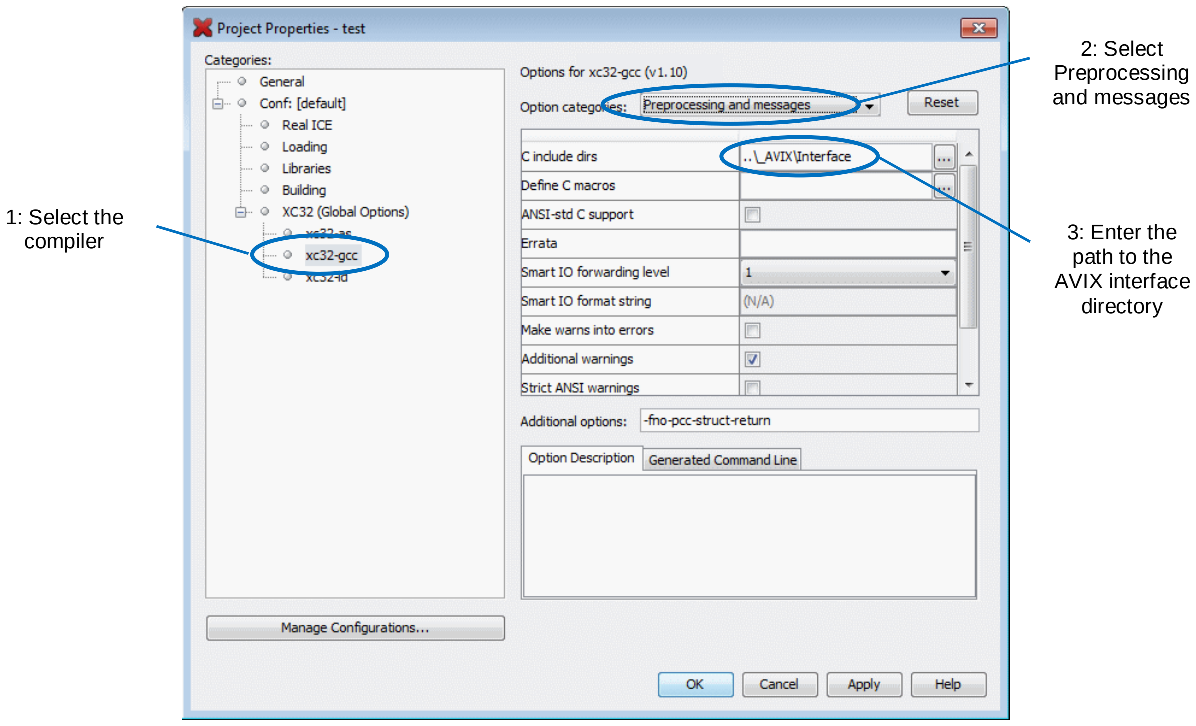
5.2.1 Specify AVIX Include Path

An AVIX based project must have access to the directory containing the AVIX interface files, this location must be added to the MPLABX project settings.

Open the applicable dialog by clicking the left mouse button on the project name and select entry ***Properties*** from the drop down menu.

The dialog is shown in Figure 5 where the selections to make in this dialog are highlighted.

The include path entered depends on the location where AVIX is installed. The value shown is based on AVIX being installed in the same directory where the MPLABX project files directory is located.



Options for xc32-gcc (v1.10)

Option categories: Preprocessing and messages

C include dirs	.._AVIX\Interface
Define C macros	
ANSI-std C support	☐
Errata	
Smart IO forwarding level	1
Smart IO format string	(N/A)
Make warns into errors	☐
Additional warnings	☑
Strict ANSI warnings	☐

Additional options: -fno-pcc-struct-return

Figure 5: Set AVIX Interface include path for MPLABX

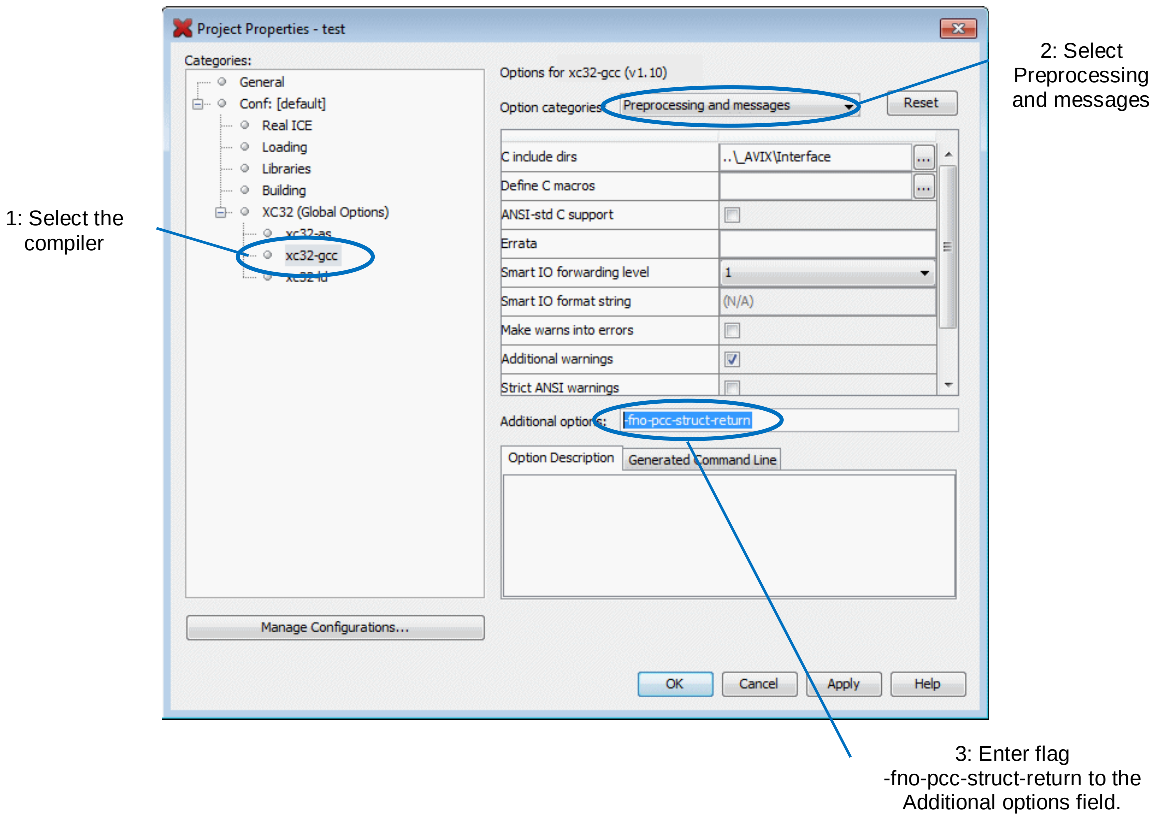
5.2.2 Structure based function parameters

The functions offered on the AVIX API use ‘C’ structures. The C32/XC32 compiler suite allows for structures to be passed by value or by reference. To implement the high level of type safety the AVIX library is built based on passing structures by value. This implies that code using AVIX functions has to use the same mechanism in order to be compatible with the library.

This is accomplished by using compiler flag **-fno-pcc-struct-return**.

Open the applicable dialog by clicking the left mouse button on the project name and select entry ***Properties*** from the drop down menu.

The dialog is shown in Figure 6 where the selections to make in this dialog are highlighted.



Options for xc32-gcc (v1.10)

Option categories: Preprocessing and messages

C include dirs	.._AVIX\Interface
Define C macros	
ANSI-std C support	☐
Errata	
Smart IO forwarding level	1
Smart IO format string	(N/A)
Make warns into errors	☐
Additional warnings	✓
Strict ANSI warnings	☐

Additional options: -fno-pcc-struct-return

Figure 6: Struct parameter project setting for MPLABX

6 Configuring AVIX

AVIX is highly configurable. All configuration settings are made by manipulating the content of file AVIXSystemSettings.h. Details are provided in §6.1.

Besides the configuration settings, AVIXSystemSettings.h contains include statements for the header files required by controllers of the PIC32MX family of microcontrollers.

When including one of the AVIX header files (preferably AVIX.h, see §4.1 for more details), AVIXSystemSettings.h is automatically included and as a result the applicable controller family include file is included. Therefore all definitions of the applicable controller family include file are available to the application and there is no need for the application to explicitly include this file.

6.1 Configuration Parameters

Configuring AVIX is done by manually editing the content of file AVIXSystemSettings.h. The following settings are present[4]:

avix_DEVICE_CLOCKhz

The controller's core speed and device speed are to be set by the application. Based on these settings, the speed at which the AVIX used hardware timer is running must be configured through this parameter. This parameter specifies the frequency in Hertz used for clocking the hardware timer configured through configuration parameter `avix_SYSTMR`.

From this value the applicable timing related settings used by AVIX are derived. An incorrect value for this parameter will lead to imprecise timing of timers and round robin time slice duration.

The value specified here is related to configuration parameter `avix_TIMER_CLK_SECONDARY`:

- When using an internal clock for the hardware timer, the value for `avix_DEVICE_CLOCKhz` must equal the instruction frequency the controller core is operating at.
- When using an external clock for the hardware timer, the value for `avix_DEVICE_CLOCKhz` must equal the frequency of this external clock which most of the time is 32,768Hz.

avix_TIMER_CLK_SECONDARY

Specify whether the hardware timer is clocked from an internal or an external clock.

0: Hardware timer is clocked from internal processor instruction clock.
1: Hardware timer is clocked from external clock.

When using AVIX power management, certain power reduction modes require the system timer to be clocked from an external clock in order for the AVIX timing functions to continue operating. For more details see §9.

[4] Depending on the type of distribution, some configuration parameters are fixed. The applicable parameters are clearly marked in AVIXSystemSettings.h. Details are found in Table 2.

avix_ROUND_ROBIN_CYCLEus

The time in microseconds a thread is allowed to run before another thread at the same priority will be activated. A typical value for this parameter is 10,000 (10 ms).

The value for this parameter must be at least five (5) times the value you choose for `avix_SYS_CLOCK_TICKus`.

Optionally this parameter can be given the value 0 effectively disabling round robin scheduling. When doing so and using multiple threads at the same priority, such a thread must explicitly call `avixThread_Relinquish` to allow the next thread at the same priority to be activated.

avix_SYS_CLOCK_TICKus

Hardware timer period in microseconds. This parameter specifies the period of the AVIX system tick.

This value must be at least five (5) times lower than the value chosen for `avix_ROUND_ROBIN_CYCLEus`.

This parameter specifies the accuracy available for all timing related functionality. A typical value is between 100µs and 500µs where the lower the number, the more accurate application timing can be obtained.

The maximum value that can be given to this parameter depends on the configured value for `avix_DEVICE_CLOCKhz`. Configuring this parameter to 80,000,000, the maximum value for parameter `avix_SYS_CLOCK_TICKus` is 800µs.

When using AVIX power management, to place the controller in SLEEP mode, the timer period configured through this parameter may need to be substantially longer than the times mentioned above. For more details see §9.

Based on the resolution of the clock used for the hardware timer, the actual period may differ from the period configured through this parameter. The actual hardware timer period is available through symbol `AVIX_SYS_CLOCK_ACTUAL_PERIOD`. When the application needs the hardware timer period, make sure to use this symbol i.s.o. `avix_SYS_CLOCK_TICKus`.

avix_MEM_POOL_COUNT

Maximum number of memory pools that can be created by the application.

For every memory pool that can be created, four bytes are reserved. As such the value of this parameter has a marginal effect on RAM usage.

avix_MSG_POOL_NR_MESSAGES

Number of messages in the message pool available to the application.

When this parameter is given a value of zero (0), no message pool is created.

Based on this parameter, during initialization, a memory pool is created from which messages are allocated. Make sure to configure this parameter to the lowest possible value for the lowest memory consumption.

avix_MSG_BODY_NR_BYTES
The message mechanism is based on messages with a fixed size of the message data section. This parameter specifies the size in bytes of a message data section. A value of zero (0) is allowed meaning no data can be placed in messages and only the message type field can be used to differentiate between messages.
avix_SYSTEM_STACK_SIZE
AVIX implements a system stack for use by Interrupt Service Routines. Using this system stack for interrupts means that stacks of individual threads can be smaller. This configuration parameter specifies the size in bytes allocated for the system stack. The configured size is scaled such that the stack size adheres to the stack alignment requirements for the applied microcontroller. Besides ISR's, this system stack is used by some AVIX internal functions. A typical value for this parameter is 800 to 1,000 bytes.
avix_MAX_PRIORITY
The maximum thread priority an application can use. A thread can be given any priority from 1 up to and including `avix_MAX_PRIORITY`. The maximum value for this configuration parameter is 254. Adapt the value of this parameter to the actual priorities used by the application. A lower value for this parameter has a positive effect on both memory usage and performance.
avix_TRACING
Control Thread Activation Tracing. This parameter may have one of the following values: 0: Tracing **disabled**, trace code **executed**. On every context switch trace code is executed but no trace port assigned to any thread will be asserted. Instead of this a dummy memory location is asserted. This allows disabling tracing retaining the same performance as using value 1. 1: Tracing **enabled**, trace code **executed**. On every context switch, trace code is executed. For threads having a trace port assigned this port will be asserted. For threads not having a trace port assigned, a dummy memory location is asserted. This allows enabling tracing retaining the same performance as using value 0. 2: Tracing **disabled**, trace code **not executed**. Using this value, AVIX will not execute the trace code leading to optimal performance. Using this value AVIX executes a different instruction sequence than with either value 0 or 1. Values 0 and 1 exist to allow having a build with or without tracing having the same performance. These modes are intended to be used during development where switching between these mode enable/disable tracing without influencing performance. This forms the basis of the non-intrusiveness of tracing. Regardless if value 0 or 1 is used, the exact same instruction sequence will be executed. Regardless the value of this configuration parameter, trace ports can be assigned to a thread using function `avixThread_SetTracePort`. The value of this configuration parameter only influences how these ports are used by AVIX.

avix_SWI
Specify the internal software interrupt used by AVIX. Allowed values are CS0 or CS1
avix_SYSTMR
Specify the hardware timer used by AVIX. This parameter is a numeric value from 1 to 5 identifying the related hardware timer.

Table 2: AVIX Configuration Parameters

6.2 Distributions

AVIX is available in three different retail distributions. These distributions differ in user configurable parameters and the number of kernel resources (thread, mutex, etc.) that can be used. Every distribution comes with an AVIXSystemsSettings.h file, containing all configuration parameters mentioned in §6. Based on the type of distribution, a configuration parameter is either fixed (as determined by the build process of that distribution) or user changeable. Fixed configuration parameters are clearly denoted as such in AVIXSystemsSettings.h.

Table 3 shows a summary of the fixed configuration parameters and their values together with the maximum number of kernel objects that can be used for each type of distribution.

Distribution	BASIC	STANDARD	EXTENDED
Controller Type	Configurable		
Device Speed	Configurable		
Round Robin Cycle Time	Configurable		
System Clock Period	Configurable		
Internal Interrupt	Configurable		
Hardware Timer	Configurable		
System Stack size	600	800	Configurable
Maximum Priority	8	16	Configurable
Message Body Size	4	4	Configurable
Nr. Messages in Message Pool	10	20	Configurable
Maximum number of Memory Pools	2	4	Configurable
Maximum number of Threads	8	16	Memory dependent
Maximum number of Mutexes	4	8	Memory dependent
Maximum number of Semaphores	4	8	Memory dependent
Maximum number of Event Groups	4	8	Memory dependent
Maximum number of Pipes	4	8	Memory dependent
Maximum number of Timers	4	8	Memory dependent
Maximum number of Exchange Objects	4	8	Memory dependent

Fixed configuration parameter. Value present in AVIXSystemSettings.h for reference purposes. Changing has no effect.

Fixed value in Distribution Library code

Table 3: AVIX Distribution Capabilities

The retail version of AVIX is distributed as a binary library. This library is controller neutral and can be used with any controller of the families targeted by AVIX.

7 Stack usage and Interrupt Service Routines

Controllers belonging to the PIC32MX family do not implement a hardware system stack for use by Interrupt Service Routines (ISR's).

To reduce memory usage, AVIX does implement a software system stack

ISR's can be declared using the standard compiler mechanism. Doing so however, the system stack is not used and interrupts use the stack of the then active thread.

An example of a basic ISR declaration is shown in Code sample 1.

```
// ISR declaration for timer 4 based on the regular C32/XC32 compiler suite mechanism
//
void __ISR(_TIMER_4_VECTOR, ipl7) _ISRforTimer4(void)
{
  …;                              // Application specific ISR code
  …;

  IFS0CLR = _IFS0_T4IF_MASK;    // Reset the interrupt flag.
}
```

Code sample 1: How to declare an ISR using the compiler syntax

Alternatively ISR's can be declared using AVIX provided macros. ISR's using these macros make no use of the stack of the active thread and will mainly use the software system stack.

Using the software system stack for ISR's leads to a significant reduction of RAM usage, at the cost of seven additional instruction cycles consumed by the ISR.

ISR's based on the regular compiler mechanism and ISR's based on the AVIX provided macros may be used together. So for every individual ISR a choice can be made whether reduction of RAM usage or ultimate performance is the most important.

The macros to declare an ISR using the AVIX system stack are shown below.

`avixDeclareISR(vectorName)`
This macro defines an ISR that will use the AVIX system stack. As a result, the ISR uses zero (0) bytes of the stack of the interrupted thread. All stack usage of the ISR is placed on the system stack. The macro must be followed by 'C' style curly brackets {}, in between which the code of the ISR is present. Effectively this defines a 'C' style function for the ISR. Parameter vectorName is one of the vector definitions present in the controller specific header.

`avixDeclareISRShadow(vectorName)`
This macro defines an ISR that will use the AVIX system stack. As a result, the ISR uses zero (0) bytes of the stack of the interrupted thread. All stack usage of the ISR is placed on the system stack. The macro must be followed by 'C' style curly brackets {}, in between which the code of the ISR is present. The ISR declared by this macro uses the controllers shadow register set. Parameter vectorName is one of the vector definitions present in the controller specific header. This macro may only be used with interrupts having priority 7. Using this macro with interrupts at other priorities will lead to an unstable application.

▶ *Make sure when using `avixDeclareISRShadow` for multiple ISR's, each of the interrupts these ISR's belong to have priority 7. Failure to do so will lead to an unstable application.*

Note that for PIC32MX controllers belonging to the PIC32MX3/4 family shadow register usage is tied to interrupt priority 7 by hardware. PIC32MX controllers belonging to the PIC32MX5/6/7 families allow the user to configure the interrupt priority level for which shadow registers are used. Make sure when using `avixDeclareISRShadow`, for controllers belonging to these families, always to configure interrupt priority 7 for shadow register usage. Restricting shadow register usage to interrupt priority 7 has the advantage to offer the best possible performance.

An example of an AVIX macro based ISR declaration is shown in Code sample 1.

```
// ISR declaration for timer 4 based on the AVIX mechanism
//
avixDeclareISR(_TIMER_4_VECTOR)
{
  …;                              // Application specific ISR code
  …;

  IFS0CLR = _IFS0_T4IF_MASK;    // Reset the interrupt flag.
}
```

Code sample 2: How to declare an ISR using the AVIX software system stack syntax

An ISR declared using one of the AVIX macros uses zero (0) bytes of the stack of the interrupted thread. The ISR uses the software system stack for all required storage.

When specifying the threads stack size no bytes need to be reserved for ISR usage. This is of course only applicable when all ISR's are based on the AVIX provided mechanism. When also using ISR's based on the standard compiler mechanism, the thread stack usage cannot be predicted and the above is not applicable.

How much RAM is preserved now when using the software system stack?

An ISR saves 104 bytes on the stack of the interrupted thread. Using all possible interrupt priorities, this implies each thread has to preserve 104 * 7 equaling 728 bytes. For an average application using 15 threads, this equals 10,920 bytes.

Using the AVIX mechanism, zero (0) bytes are placed on the stack of the interrupted thread. The entire ISR stack requirement (728 bytes in case of worst case nesting) is placed on the software system stack.

This implies that using the AVIX software system stack mechanism, RAM usage is lowered by as much as ~10,000 bytes for this particular example.

In practice more RAM is preserved since the above calculation is based on the minimal ISR stack requirement. When using ISR local variables and/or allow ISR's to make function calls, stack requirement will increase and thus the positive effect of using the AVIX software system stack will be even higher.

8 Extended Memory

►

AVIX offers functionality to use extended RAM. PIC32MX controllers do not offer extended RAM. For reasons of portability between the different AVIX ports however, the function and macros used for this functionality are present in AVIX for PIC32MX.

A number of controllers targeted by AVIX offer extended memory which is RAM that can only be accessed using custom addressing mechanisms.

AVIX allows such memory to be used for memory pools which are created using function `avixMemPool_CreateExt`. For access to this type of ram, also two macro's are offered, `AVIX_MEM_BLOCK_PTR_EXT` converts a memory block id to a plain 'C' pointer and macro `AVIX_EDS` is used to tag the 'C' pointers used for this purpose.

This function and these macros are also present in AVIX for PIC32MX. Controllers belonging to the PIC32MX family do however not offer extended RAM.

The reason this function and the related macros are present in AVIX for PIC32MX is for portability between the different AVIX ports. Sources generated for a different port that do use extended RAM can be compiled for AVIX for PIC32MX without any problem.

For AVIX for PIC32MX, the function and the related macros are implemented as follows:

- `avixMemPool_CreateExt` behaves exactly the same as the basic function `avixMemPool_Create`. When only using AVIX for PIC32MX, advised is not to use `avixMemPool_CreateExt` but use `avixMemPool_Create` instead.

- `AVIX_MEM_BLOCK_PTR_EXT` behaves exactly the same as the basic macro `AVIX_MEM_BLOCK_PTR`. When only using AVIX for PIC32MX, advised is not to use `AVIX_MEM_BLOCK_PTR_EXT` but use `AVIX_MEM_BLOCK_PTR` instead.

- AVIX_EDS is an empty macro for AVIX for PIC32MX. When only using AVIX for PIC32MX, advised is not to use this macro.

9 Power Management

This section specifies the AVIX power management implementation aspects specific to the PIC32 controllers.

► *When using AVIX power management, detailed knowledge of the controller specific power reduction features is required. Although this section provides some controller specific power reduction information, this manual is no substitute for the controller's data sheet.*

9.1 Power mode mapping

The AVIX generic power mode constants have the following mapping to the controller specific power modes:

- `AVIX_POWER_REDUCTION_NONE`: No mapping to a controller specific power mode. When using this mode, no controller power reduction mode is used.
- `AVIX_POWER_REDUCTION_LOW`: Maps to the controller IDLE mode. For a description of the IDLE mode consult the controller's data sheet.
- `AVIX_POWER_REDUCTION_HIGH`: Maps to the controller SLEEP mode. For a description of the SLEEP mode consult the controller's data sheet.

In the remainder of this section, the terms IDLE and SLEEP are used.

► *Be aware that with waking up from the SLEEP mode significant delays can be involved influencing the timing of your application. These delays depend, amongst others, on the controller's oscillator configuration. Details can be found in the controller's data sheet.*

9.2 Power modes and interrupt disabling

For PIC32 it is required before activating a specific power reduction mode to either clear (IDLE) or set (SLEEP) the SLEEPEN bit in the OSCCON register. The OSCCON register is protected against accidental writes and in order for the SLEEPEN bit to be changed, PIC32 defines a dedicated unlock sequence. During execution of this unlock sequence, interrupts must be disabled. This is taken care of by AVIX[5]. The time interrupts are disabled is shorter than is the case when executing an interrupt prologue meaning that using power management does not influence interrupt latency. When using `AVIX_POWER_REDUCTION_NONE` no changes to the OSCCON register are required and thus interrupts are not disabled.

9.3 Power modes and the system timer

The configured system timer is initialized by AVIX to continue operation during IDLE mode. This implies that when using IDLE mode, any desired hardware timer can be configured to be used as the system timer.

When using SLEEP mode the controller's core oscillator is stopped meaning that hardware timers clocked from this oscillator also stop. As a result, AVIX timing related functionality will no longer function. This might be acceptable when waking up the application from SLEEP mode does not depend on AVIX timers, and when AVIX timers that are in use are allowed to stop during SLEEP. Note that also round robin scheduling depends on the system timer being operational.

[5] Disabling interrupts to select the desired power mode is forced by the PIC32 hardware architecture and not by the AVIX software architecture.

When AVIX timing functionality must continue to operate, even during SLEEP, the system timer must be clocked from an external oscillator. This is accomplished by setting configuration parameter `avix_TIMER_CLK_SECONDARY` to value 1. In this situation not all available hardware timers can be configured to be used as the system timer. Typically hardware timer 1 is the timer that can be clocked from the external oscillator and thus it is also required to configure timer 1 as the AVIX system timer.

When configuring the system timer to be clocked from an external oscillator, AVIX configures the hardware timer accordingly. Also AVIX takes care of setting the applicable bit in the OSCCON register to activate the external oscillator.

9.4 Power modes and the system timer period

The system timer period specifies the period with which the system timer will generate an interrupt. When using IDLE or SLEEP, these modes are aborted by interrupts and thus also by the system timer interrupt provided the system timer is active (see above).

Waking up from IDLE virtually happens instantaneous. This implies that any allowed system timer period can be configured.

Waking up from SLEEP may take longer depending on the system clock configuration. When using a PLL based clock, it can take up to 2ms for the controller to wake up with a locked oscillator.

It shall be clear that in this case it makes no sense for the system clock to expire faster than this 2ms period without losing accuracy. Therefore when using SLEEP it is advised to select a system timer period substantially longer than this 2ms wake up time. Advised is to set the system timer period 2 to 5 times longer than the SLEEP wake up time. For details about the SLEEP wake up time which is related to your oscillator configuration consult the controller's data sheet.

► *Under all circumstances, AVIX configures the system timer without prescaler for the highest possible accuracy. This implies that when the system timer is clocked from the core oscillator, for a 80MHz part where the device clock equals the core clock, the maximum period is ~800µs. As a result system timer periods exceeding this value are only possible when using an external oscillator which most of the time is running with a frequency of 32,768Hz.*

9.5 Other power reducing capabilities

Besides the power management mechanisms used by AVIX, additional power management capabilities are offered by the controllers in the form of clock switching and DOZE mode. Both mechanisms allow the controller to operate at a lower frequency. Since the controllers power consumption is proportional to its clock frequency, a lower operating frequency results in a reduction of power consumption. The lower clock frequency does however also result in an increase in processing time proportional to the clock frequency. As a result a lower clock frequency leaves less time available for the idle thread to run and thus less time the controller can be set in IDLE or SLEEP. Clock frequency based power regulation is intended to be used by applications that do not work event driven making it difficult to use IDLE or SLEEP mode. When using AVIX as the basis for power management these power management capabilities do not offer any added value and there is no need to make use of them.

10 Resource Usage

This chapter describes the controller's resources used by AVIX.

- **Section 10.1, Overview**, provides a global overview of the resources used by AVIX.
- **Section 10.2, Registers used by AVIX**, provides a detailed description of the controllers registers used by AVIX and guidelines how to manipulate bit fields in registers only partly used by AVIX.

10.1 Overview

Internally AVIX makes use of one of the controller's software interrupts, one of the controller's hardware timers and a small amount of RAM. This section provides information on these resources and interrupt configuration in general, which is important to know when building an application with AVIX.

10.1.1 Interrupts

Internal activation of the AVIX scheduler core is based on a software interrupt. This interrupt is for exclusive use by AVIX and thus cannot be used by the application. The specific interrupt being used is configurable, allowing an interrupt to be selected which is not required by the application. Details on how to configure this interrupt are found in §6.

When the internal scheduler is active, effectively this runs as a hardware priority 1 interrupt.

A second interrupt used by AVIX is related to the configured hardware timer. The hardware timer interrupt handler runs at hardware priority 2.

Because of these two interrupts being used, the AVIX zero latency feature is available for hardware interrupt levels 3 up to and including 7.

AVIX assumes the controller to be initialized to use prioritized nested interrupt handling. AVIX never disables interrupts and expects your application to do so neither. Under no circumstance should you disable all interrupts since this prevents the AVIX scheduler and system timer from working. Because of the rich set of functions AVIX offers for interrupt handler-thread integration it is very unlikely all interrupts ever need to be disabled.

It is however allowed to disable specific interrupts by setting the applicable bit in the control register for that specific interrupt to zero (0) might your application require this.

► *Under no circumstance should the controllers global interrupt level be manipulated. Doing so will stop the AVIX internal timer and/or the scheduler from working for the time interrupts are disabled. In case you do need to block interrupts for a specific period of time, you should do so by disabling that specific interrupt by setting its IE bit to zero.*

10.1.2 Timer

The second hardware resource AVIX uses is a timer. Here too, the specific timer being used is configurable and a timer must be configured not needed by the rest of the application. For the timer also, information on how to configure this is found in §6.

Some of the hardware timers are related to other hardware devices in a dedicated fashion so when selecting a hardware timer for use by AVIX make sure this that it is not needed for some other hardware device which is required by the application[6].

Since AVIX offers a number of software timers limited by the amount of memory only, using one of the hardware timers for AVIX will be no problem since most of the applications timing will be based on these software timers anyway.

10.1.3 Memory

For its internal bookkeeping, AVIX uses a small amount of RAM which is reserved. Furthermore, some RAM is needed for each of the AVIX kernel objects created by the application. The memory layout of an AVIX based application is shown in Figure 7.

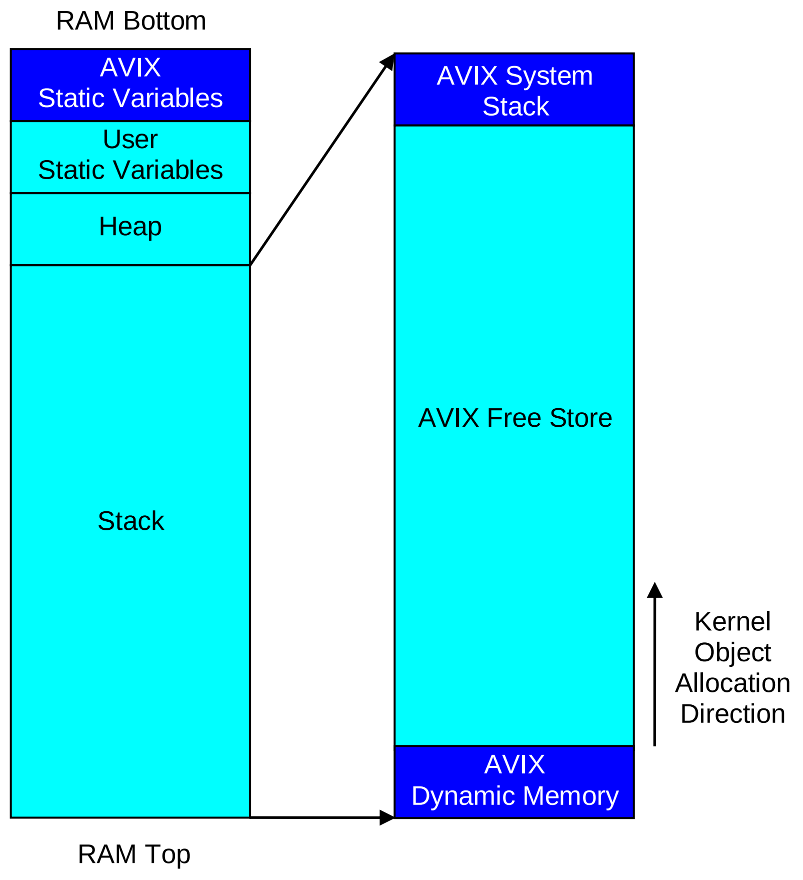


Figure 7: AVIX based application memory layout

When building an AVIX based application use is made of the default linker scripts as installed with the C32/XC32 compiler suite. AVIX requires no modifications to these linker scripts resulting in a standard memory layout which corresponds to the left part of Figure 7.

The lower addresses of RAM contain static variables used by AVIX and the application. Next there is some memory reserved for the heap. A heap only needs to be present when the application makes use of malloc and free functions, as offered by the standard runtime. Since AVIX offers its

[6] The ADC device present in a number of controllers can use hardware timer 3 for starting the conversion. Configuring AVIX to use timer 3 as the base for its internal timing, prevents this timer from being used with the ADC device.

own memory management, advised is to define the heap to have a size of zero bytes. In case the application uses the heap, precautions must be taken since the heap is not 'thread safe'.

When building a C32/XC32 compiler suite based application, al memory left when subtracting static variables and the heap is allocated to the stack. Note this is not the AVIX stack for which reason it is called C32/XC32 compiler suite stack.

This C32/XC32 compiler suite stack memory is entirely claimed by AVIX and used for three purposes. At the lower addresses a section is used for the system stack, the size of which is configurable again. At the upper addresses, some memory is reserved for internal bookkeeping purposes. All RAM left now is called the 'AVIX Free Store'. This section is used for kernel objects that are created by the application. Whenever a …`_Create` service is called, the required amount of RAM for the bookkeeping related to the specific kernel object is claimed from this 'AVIX Free Store'. This is a zero overhead implementation since AVIX kernel objects cannot be deleted implying no management information is required in order to be able to return memory to this 'AVIX Free Store'.

For memory to be allocated and freed for reuse again by the application, AVIX offers a memory pool mechanism allowing fixed size memory blocks to be allocated and freed again.

10.2 Registers used by AVIX

Table 4 contains an overview of the registers (partly) used by AVIX. Partly, because for some registers AVIX only uses a subset of the bits of these registers while the other bits are available to the application.

With 'used by AVIX', what is meant is exclusive write access. Under no circumstance may the application write the bits in the registers mentioned in Table 4. Also, when of a specific register AVIX only uses a subset of the available bits and the application writes the other bits, the application must guarantee such writes are atomic. Under no circumstance should the application manipulate the content of a register partly used by AVIX using separate Read-Modify-Write operations since this will lead to unpredictable behavior and application failure.[7]

Note that using the 'C' language under no circumstance operations are guaranteed to be atomic.

The application is allowed read access to all registers since this will not change the content of the register. Table 4 contains the following columns:

- **Register**; name of the register as defined in the applicable Microchip documentation and header files. Note that not all registers are known on forehand since some depend on configuration parameters. In this case a placeholder character is used in the register name and an explanation is given what the concrete value of this placeholder character depends on.

- **Field**; name of the bit field in the register as defined in the applicable Microchip documentation and header files. Note that not all fields are known on forehand since some depend on configuration parameters. In this case a placeholder character is used in the field name where an explanation is given what the concrete value of this placeholder character depends on.

  When this column contains value <all>, this means AVIX uses all bits of that register and the application may not write a single bit of such a register.

- **Description**; textual explanation of the register and the field.

[7] Manipulating bits in a SFR is not exclusive to the use of AVIX. Also in a non-AVIX based application, atomic manipulation of bitfields in registers shared between main code and interrupt handlers or between multiple interrupt handlers is required. In this situation also it is the applications responsibility to guarantee this.

Register	Field	Description
IEC0	<i>IE	AVIX uses one the controllers interrupt sources. The specific interrupt source used is user configurable. An interrupt source can be enabled through a bit in one of the IEC registers. The interrupt enable bit for the configured interrupt source is exclusively used by AVIX. The bit field is <i>IE where <i> is equal to the value of configuration parameter `avix_SWI` being either CS0 or CS1.
IFS0	<i>IF	AVIX uses one the controllers interrupt sources. The specific interrupt source used is user configurable. An interrupt being active is identified by a bit in one of the IFS registers. The interrupt bit for the configured interrupt source is exclusively used by AVIX. The bit field is <i>IF where <i> is equal to the value of configuration parameter `avix_SWI` being either CS0 or CS1.
IPC0	<i>IP	AVIX uses one the controllers interrupt sources. The specific interrupt source used is user configurable. For each interrupt the priority can be set in a bit field in one of the IP registers. The priority bit field for the configured interrupt source is exclusively used by AVIX. The bit field is <i>IP where <i> is equal to the value of configuration parameter `avix_SWI` being either CS0 or CS1.
IEC0	T<t>IE	AVIX uses one the controller's hardware timers. Each hardware timer can act as an interrupt source. An interrupt source can be enabled through a bit in one of the IEC registers. The interrupt enable bit for the configured hardware timer is exclusively used by AVIX. The bit field is T<t>IE where <t> is equal to the value of configuration parameter `avix_SYSTMR` being a number in the range 1..5.
IFS0	T<t>IF	AVIX uses one the controller's hardware timers. Each hardware timer can act as an interrupt source. An interrupt source being active is identified by a bit in one of the IFS registers. The interrupt bit for the configured hardware timer is exclusively used by AVIX. The bit field is T<t>IF where <t> is equal to the value of configuration parameter `avix_SYSTMR` being a number in the range 1..5.
IPCy	T<t>IP	AVIX uses one the controller's hardware timers. Each hardware timer can act as an interrupt source. For each interrupt the priority can be set in a bit field in one of the IP registers. The priority bit field for the configured hardware timer is exclusively used by AVIX. The specific IPC register depends on the timer configured through parameter `avix_SYSTMR` where <y> in IPCy equals the value of this parameter. The bit field is T<t>IP where <t> is equal to the value of configuration parameter `avix_SYSTMR` being a number in the range 1..5..
TMRy	<all>	Timer count register for the configured hardware timer where y is the timer number specified by configuration parameter `avix_SYSTMR`.
PRy	<all>	Timer period register for the configured hardware timer where y is the timer number specified by configuration parameter `avix_SYSTMR`.
TyCON	<all>	Timer control register for the configured hardware timer where y is the timer number specified by configuration parameter avix_SYSTMR.
OSCCON	LPOSCEN	Used to manage the controllers low power mode

Table 4: Registers (partly) used by AVIX