Ezairo® 8300 Software Development Kit Getting Started Guide

M-20865-013
February 2025

Table of Contents

CHAPTER 1

Introduction

1.1 Purpose

IMPORTANT: onsemi acknowledges that this document might contain the inappropriate terms “white list", "master" and "slave”. We have a plan to work with other companies to identify an industry wide solution that can eradicate non-inclusive terminology but maintains the technical relationship of the original wording. Once new terminologies are agreed upon, future products will contain new terminology.

This group of topics describes how to begin using the Ezairo 8300 Software Development Kit (SDK). It provides the prerequisites and instructions necessary to install the relevant software, connect the hardware, and develop applications for Ezairo 8300 using the Synopsys® ASIP Designer™ toolchain. Sample applications are included for introduction and practice purposes.

NOTE: If you are a developer who is moving from using other onsemi Ezairo products to working with Ezairo 8300, consult *Ezairo 8300 for Users of Other Ezairo Products* for additional important information.

1.2 Intended Audience

This group of topics is for software developers who are designing and implementing Ezairo 8300 applications.

1.3 Conventions

The following conventions are used in this group of topics to signify particular types of information:

`monospace font`	Macros, functions, defines and addresses.
italics	File and path names, or any portion of them.
`<angle brackets>`	Optional parameters and placeholders for specific information. To use an optional parameter or replace a placeholder, specify the information within the brackets; do not include the brackets themselves.

1.4 Further Reading

For more information about Ezairo 8300, refer to the following documents:

- *Ezairo 8300 Hardware Reference*
- *Ezairo 8300 Firmware Reference*
- *Introduction to Ezairo 8300 Programming*
- *Ezairo 8300 Evaluation and Development Board Manual*
- *Ezairo 8300 Datasheet*

CHAPTER 2

Overview

Ezairo 8300 is an open-programmable DSP-based System-on-Chip (SoC) specifically designed for use in ultra-low-power high-performance portable audio devices.

Ezairo 8300 includes five programmable or semi-programmable processing cores, providing a high degree of parallelism and flexibility:

- The CFX processor is an open-programmable dual-Harvard 24-bit digital signal processor (DSP), providing support for any type of audio signal processing.
- The HEAR configurable accelerator core is optimized for pre-programmed functions that are frequently needed in audio signal processing.
- The Filter Engine allows time domain filtering and supports an ultra-low-delay audio path.
- The Arm® Cortex®-M3 core functions as the system master.
- The LPDSP32 is an open-programmable dual-Harvard 32-bit DSP.

2.1 High-Level Overview

Each of the Ezairo 8300‘s programmable cores has a separate toolchain which is integrated in the SDK. Section 4.4 “Preparing the System to Build Source Code with the ASIP Designer toolchain” on page 21 contains more information about the ASIP Designer toolchain for the CFX as a starting point. Information about the other toolchains, and more information about the CFX toolchain, can be found in the relevant sections of related documentation.

For a comprehensive index of documentation included with the SDK, see Section 7 “More Information” on page 44.

2.2 Features

Ezairo 8300 includes the following features:

- Four ADCs with signal detection mode and 2 direct digital output drivers, with high quality and ultra-low power performances
- Peripherals and interfaces needed to make it a complete hardware platform, when combined with non-volatile memory, wireless transceivers or multiple sensors
- A neural network accelerator that allows the Ezairo 8300 to perform neural network computations in a highly efficient and flexible way

The architecture of the Ezairo 8300 is shown in the figure "Ezairo 8300 Architecture Overview" (Figure 1).

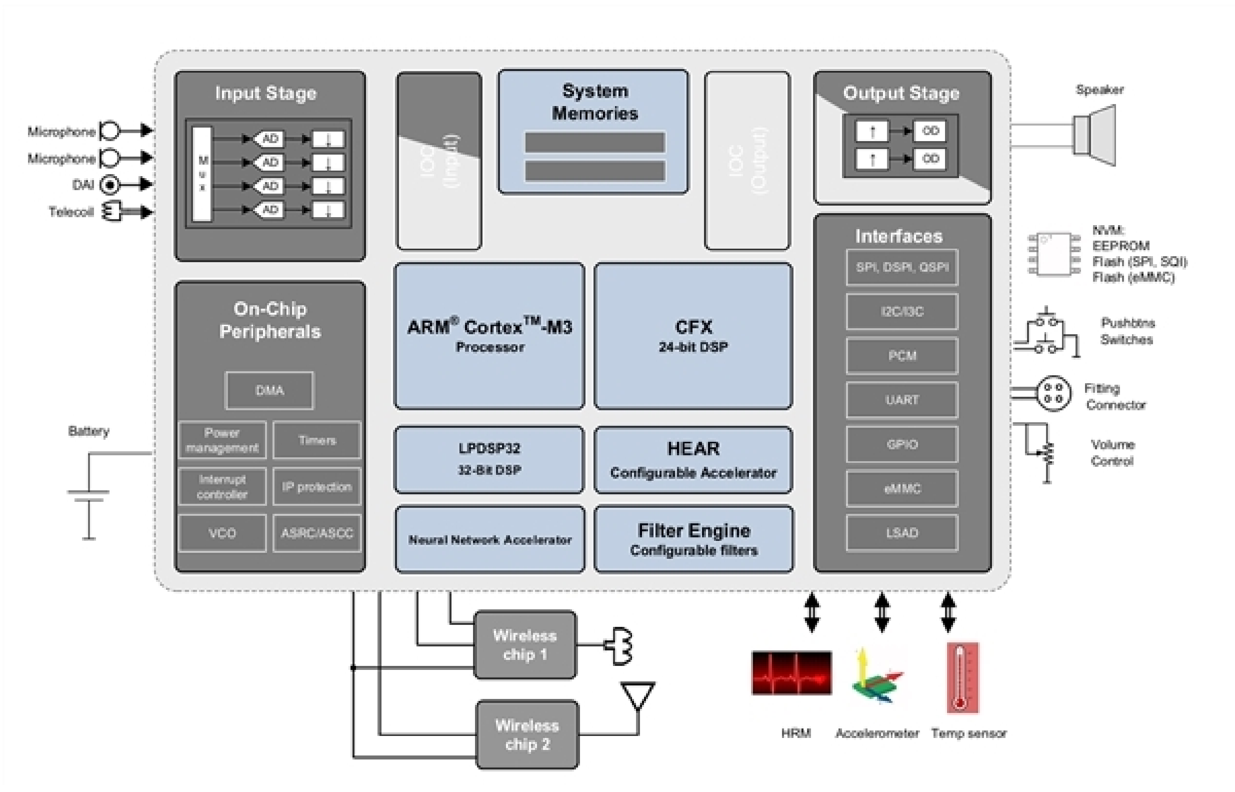


Figure 1. Ezairo 8300 Architecture Overview

CHAPTER 3

Design Information

For all new users of Ezairo 8300, the following sections provide design recommendations to make efficient use of the chip's resources from a programmer's perspective.

NOTE: When compared with Ezairo 7100, the Ezairo 8300 device includes an additional core (the LPDSP32 DSP), allows the Arm Cortex-M3 processor to be the system master, and includes several new blocks and capabilities. For more information on the architecture updates included in Ezairo 8300, refer to Chapter 1 "Architecture Overview" on page 1.

3.1 Programming the Ezairo 8300 System

Optimum power efficiency is achieved by using all cores equally, or load-balancing. Aside from external components connected to Ezairo 8300 (such as a wireless chip), the amount of power consumed is primarily determined by the system clock frequency; so if work is evenly divided among all cores, you can reduce the overall clock frequency, consuming less power. The figure "A Conceptual Illustration of Load-Balancing" (Figure 2) is an illustration of this concept for a simple case with a dual-core system.

Since the programmable cores have their own strengths besides sharing certain common capabilities, you can take apart your algorithm and assign different portions of it to run on different cores (called *algorithm partitioning*). If your algorithm requires a low-delay path, use the Filter Engine to construct this low-delay path first. If you need wireless communication, use the Arm Cortex-M3 core to enable the wireless audio functionality. It is also generally desirable to use the Arm Cortex-M3 core to handle tasks not essential to the audio processing, so that you can leave the CFX and the HEAR to focus on the heavy processing that they are best at. We recommend that the vectorized, number-crunching portions run on the HEAR, and that other signal processing tasks run on the CFX or the LPDSP32. Being a 32-bit processor (rather than a 24-bit processor like the CFX), the LPDSP32 is ideally suited for implementing codecs that might need to share data with the Arm Cortex-M3 core connected to a wireless chip. The LPDSP32 is linked closely in the system with the Neural Network Accelerator (NNA), and could therefore be a logical choice for performing other signal processing tasks related to deep learning applications. Algorithm partitioning is key to optimizing the power efficiency of programs running on Ezairo 8300, as well as to taking advantage of the specialties of each of the programmable cores in the Ezairo 8300 system.

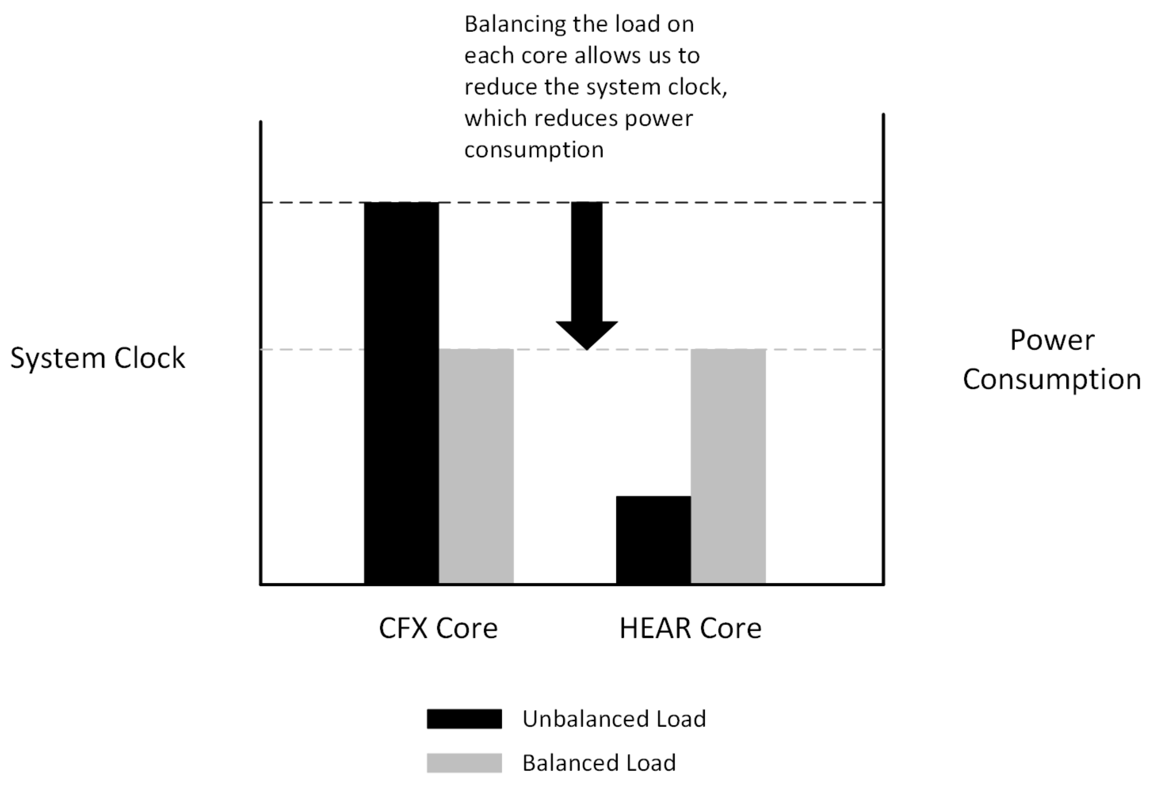

Balancing the load on each core allows us to reduce the system clock, which reduces power consumption
System Clock
Power Consumption
CFX Core
HEAR Core
Unbalanced Load
Balanced Load

Figure 2. A Conceptual Illustration of Load-Balancing

3.1.1 Partitioning Algorithms

Partitioning the algorithms for Ezairo 8300 is somewhat related to task scheduling for multiprocessor systems, a topic that has been studied extensively in the field of multiprocessor computing. This section describes a few key concepts that are useful to developers working on partitioning algorithms for Ezairo 8300.

The guidelines for partitioning algorithms for Ezairo 8300 are:

- Make each core do what it does best, and what no other core can do.
- Schedule tasks among the programmable cores so that the workload is distributed among them as evenly as possible.

Following these two guidelines, when analyzing an algorithm for partitioning, it is important to consider the algorithm from two perspectives: function, and timing.

Functional partitions
: Divide the algorithm based on the type of operations (or functions) that need to be run.

Timing partitions
: Divide the algorithm based on when the parts of the algorithm needs to be run, and how often and how quickly they need to be completed.

When analyzing your algorithm from the functional perspective, keep in mind several important factors, which are described below.

Sometimes certain tasks must be assigned to a particular core, because either this core is the only one to perform them, or no other core would be nearly as efficient even if it could perform them. For example, if your algorithm requires a low-delay path, you must configure the Filter Engine to establish it. If you need to use a CVSD codec, the Arm Cortex-M3 core hardware-based CVSD codec would be the natural choice.

The HEAR is typically more power-efficient than the CFX in running the functions provided by the microcode modules. If the cycle count for a given task is about the same on both cores, the HEAR is still likely to be more power-efficient. Therefore it is usually better to partition your algorithm in such a way that you can utilize the functions available from the HEAR and not duplicate the same functions on the CFX. See the *HEAR Configurable Accelerator Reference Manual* for the complete description of the available microcode modules.

The communication scheme between the CFX, the IOC, and the HEAR is presented in the table "HEAR Inter-Processor Communication" (Table 1). You can start a function chain from a CFX command or a FIFO interrupt, and you can configure each interrupt from the HEAR to the CFX at various points within a function chain. For example, if a function chain contains an FIR filter function followed by a filterbank analysis function, you can configure the HEAR to raise an interrupt on the CFX after the FIR filter function has completed, and raise another interrupt after the filterbank analysis function has completed.

Table 1. HEAR Inter-Processor Communication

From	To	Number of Interrupts
HEAR	CFX/Arm Cortex-M3 core	8
CFX/Arm Cortex-M3 core	HEAR	8
FIFO	HEAR	8

The communication scheme between the CFX, the IOC, the LPDSP32, and the Arm Cortex-M3 core is presented in the table "Inter-Core Communication" (Table 2). There are several ways of using interrupts. For instance, you can start an audio encoding process on the Arm Cortex-M3 core as soon as an input FIFO is full. As another example, the Arm Cortex-M3 core can raise an interrupt on the CFX when a particular function has completed.

Table 2. Inter-Core Communication

From	To	Number of Interrupts
Arm Cortex-M3 Core	CFX	8
CFX	Arm Cortex-M3 Core	8
FIFO	Arm Cortex-M3 Core	8
FIFO	CFX	8
Arm Cortex-M3 core	LPDSP32	6
LPDSP32	Arm Cortex-M3 core	8

Use the CFX and the Arm Cortex-M3 core to handle initialization and control of all peripherals and external interfaces on Ezairo 8300.

There are also timing considerations when analyzing your algorithm:

For algorithms that require a low-delay path (LDP), ensure that a new output sample in the LDP is available at its sampling rate to maintain the continuity of the audio output. For example, if the system clock, and therefore the Filter Engine clock, is at 2.56 MHz, and the sampling rate of the LDP is 128 kHz, then you have a maximum of 20 cycles to complete any processing for each new sample. Typically, the CFX processing is not a part of the LDP. If you must use the CFX in the LDP, use caution and ensure that the timing constraint of the LDP is met.

For algorithms that operate at the decimated sampling rate, you need to ensure that a new block of audio data is available at the block rate to maintain the continuity of the audio output. Thus, the critical path of your task schedule is typically constrained by the block rate. The block rate is calculated as the sampling frequency divided by the input block size. For example, a 16 kHz sampling frequency divided by a block size of eight samples equals 2 kHz, meaning that a new block of audio data must be available every 0.5 ms.

Time-slicing is a technique for partitioning your algorithm over the time dimension, to free up clock cycles for the most critical computations. It involves spreading parts of your algorithms over several blocks instead of having all the computations done within one block. For example, in a dynamic range compression algorithm, instead of calculating new compression gains for all 16 frequency bands within one block, time-slicing divides up the algorithm so that only the compression gains for bands 1-4 are calculated in the first block, then only the gains for bands 5-8 are calculated in the next block, and so forth. This way, the algorithm is partitioned so that it only needs to calculate the compression gains for 4 frequency bands at a time.

Implementing time-slicing might compromise the performance of your algorithm if it is not implemented in a way that accounts for the different sample rates needed.

NOTE: The multi-rate signal processing capability of the Filter Engine means that it can offer convenient time-slicing. For tasks that run at a decimated rate (as compared to the highest sampling rate for the root program sector), you can spread them across multiple program phases. Consult the *Filter Engine Reference Manual* for the concepts of multi-rate processing, program phase, and program sector.

Either a command from the CFX or an event from the FIFO controller can launch a function chain on the HEAR. It is therefore important to consider the flow of the data in your algorithm, to see if the FIFO controller should directly launch a function chain on the HEAR while the CFX is running other computations.

For those tasks that run on the Arm Cortex-M3 core, we recommend taking advantage of any available DMA channels for data transfers. If all four DMA channels have been used, you can reconfigure some or all of them at run-time. This might be power-efficient if the program overhead associated with DMA run-time reconfiguration is less than the computation cycles that manual data transfers would cost.

3.1.2 Other Design Considerations

Designing an audio processing system usually involves trade-offs between power consumption, circuit size, and noise, as illustrated in the figure "Trade-offs Between Power, Size and Noise when Designing an Audio System" (Figure 3). A lower noise level implies a higher RMS current and more filtering, typically on the power supply, which requires more space.

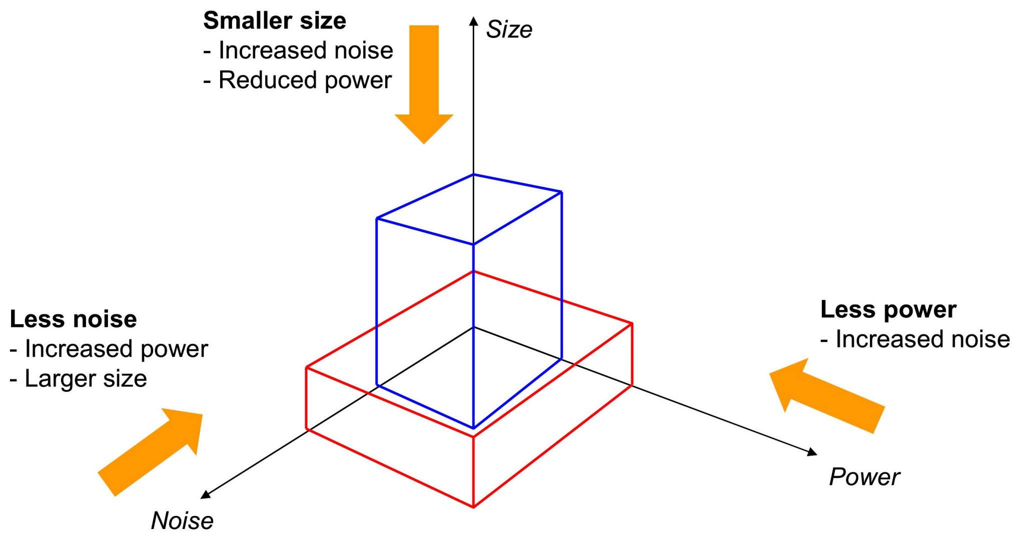


Figure 3. Trade-offs Between Power, Size and Noise when Designing an Audio System

In an audio processing system that emphasizes low power consumption and small circuit size, one way in which noise could be induced is to periodically vary the dynamic current drawn by a processor, and to ensure that the peak-to-peak level of that dynamic current is large enough. A tonal noise can be induced in the output signal because of current coupling with the rest of the system.

For instance, for an unbalanced algorithm on Ezairo 7100, the HEAR might periodically draw a dynamic current because it first runs the filterbank analysis function, then idles when the CFX processes the analysis results. After it finishes processing, the HEAR runs more filterbank functions before idling again, and so on. If the HEAR and CFX processors periodically operate and idle, the peak-to-peak level of the dynamic current drawn by the processors might be very large, which can induce a tonal noise in the output signal. This is known as the *block rate tone* problem.

Ezairo 8300 has much improved power management schemes to mitigate this problem, but its complete elimination still requires careful load-balancing from your algorithm. With Ezairo 7100 you are encouraged to evenly distribute the workload between the CFX and the HEAR (load-balancing). Similarly, the flexibility of the Ezairo 8300 system allows you to implement efficient algorithm partitioning to evenly distribute the workload among all the programmable processors, so that each processor is not periodically operating and idling. This means that the peak-to-peak level of the dynamic current will be very small, and no tonal noise is induced. The figure "A Conceptual Illustration of Producing Noise and Avoiding It " (Figure 4) illustrates the concept of implementing load balancing to reduce the periodic peak-to-peak variations in the dynamic current.

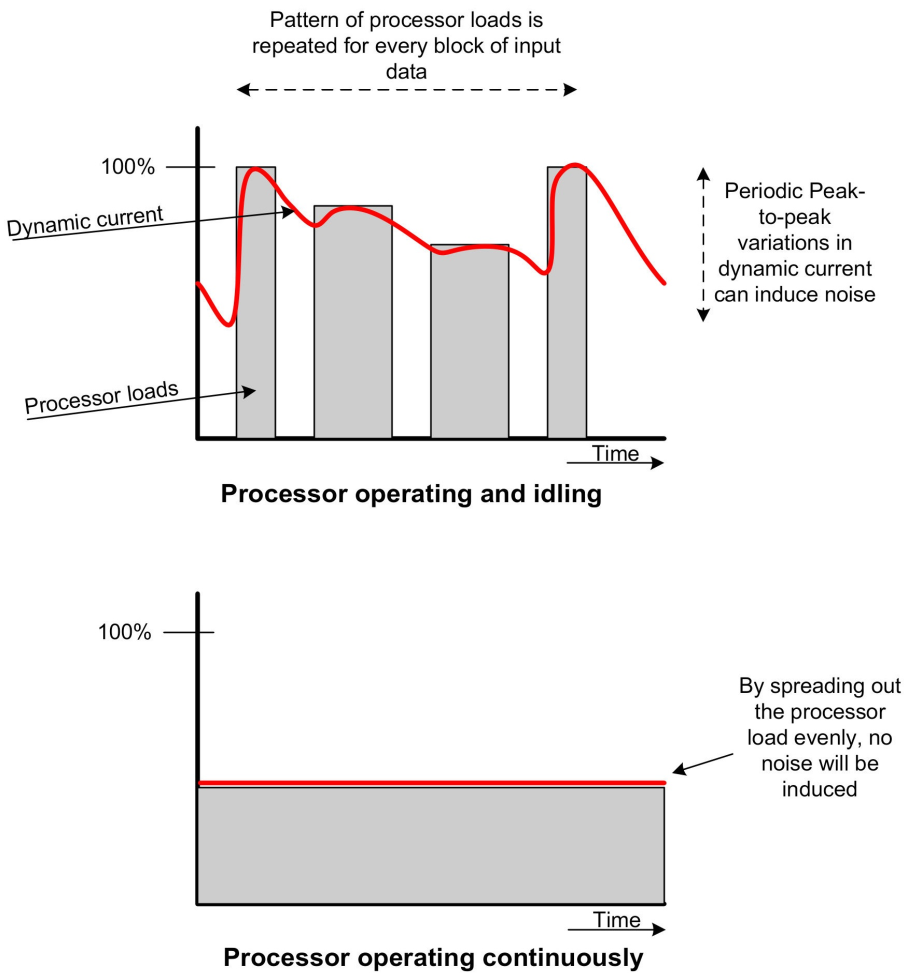

Pattern of processor loads is repeated for every block of input data
100%
Dynamic current
Processor loads
Periodic Peak-to-peak variations in dynamic current can induce noise
Time
Processor operating and idling
100%
By spreading out the processor load evenly, no noise will be induced
Time
Processor operating continuously

Figure 4. A Conceptual Illustration of Producing Noise and Avoiding It

Implementing load-balancing on Ezairo 8300 allows you to optimize the trade-off between power consumption and noise. It not only leads to lower dynamic power consumption, but it also causes lower average power draw, while at the same time avoids inducing tonal noise. By reducing the peak-to-peak variations that require more milli-Watts when a processor is operating in short bursts, you achieve the same results with continuous operation of the processors. In effect, the same amount of work is done when all active processors are running continuously, but with lower overall energy consumption.

NOTE: Load-balancing does not mean you have to use all the programmable cores. If a core is not in use and is disabled, it does not consume power and does not induce noise.

IMPORTANT: If you do not implement proper load-balancing, your program running on Ezairo 8300 can induce tonal noise.

3.2 Data Flow

Ezairo 8300 features flexible data flow through the Input/Output Controller (IOC), the FIFO controller, the Filter Engine for audio multiplexing, and the DMA controller on the Arm Cortex-M3 core, as shown in the figure "Flexible Data Flow via the IOC and FIFO Controller" (Figure 5).

The IOC can be configured to automatically route data between the system memory on one side, and the input/output stage and PCM interface on the other side. The FIFO controller supports up to 32 FIFOs that can be configured to different sizes and locations within the system memory. If you are familiar with Ezairo 7100, you will notice the increased number of FIFOs on Ezairo 8300.

Each of the FIFOs can be connected to one or more data sources or data sinks. A data source is something that writes data into the FIFO, and a data sink reads data from the FIFO. For example, the input stage is a data source for an input FIFO, whereas a PCM port can both receive and transmit data, so it can be both a data source and a data sink. Moreover, a data source or sink is not required to be a physical interface connected to the IOC. For example, an FIR filter running on the HEAR can be a data sink that reads from an input FIFO, while at the same time it can be a data source for another FIFO. FIFOs that are connected to a physical interface through the IOC are called *hardware FIFOs*. FIFOs that are controlled by the software on the CFX and HEAR are called *software FIFOs* (see the figure "Comparing Hardware FIFOs to Software FIFOs" (Figure 6)). The IOC, CFX, HEAR and Arm Cortex-M3 core all have access to the FIFOs via the FIFO controller, which maintains a fixed access priority scheme. See the *Ezairo 8300 Hardware Reference* for details about the IOC and FIFO controller, and also Section 1.1 “FIFO Controller Input and Output Blocks” on page 1.

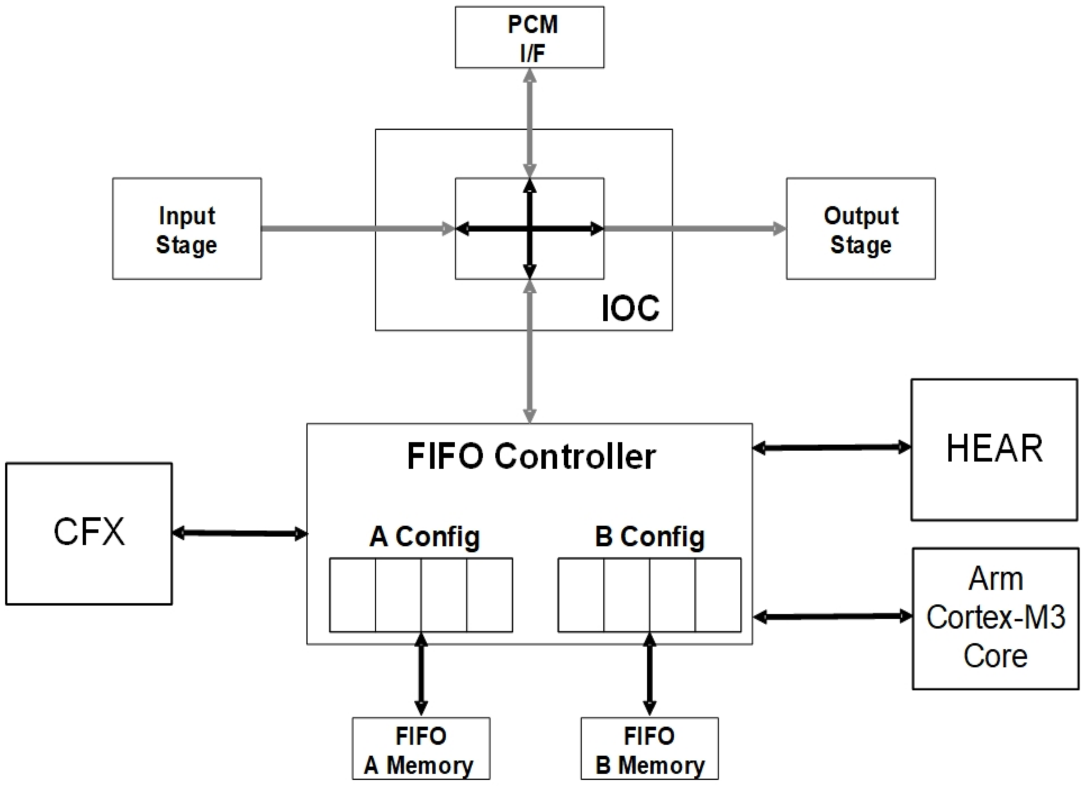


Figure 5. Flexible Data Flow via the IOC and FIFO Controller

To design your algorithms to take advantage of the flexibility provided by the IOC and FIFO controller, consider all the components that can be data sources, data sinks, or both. Configure the IOC and the FIFO controller such that they connect the data sources and sinks to the FIFOs that you have specified. Access to the FIFOs from within your program is straightforward—the FIFO controller handles all the pointer updates to the FIFOs, and automatically maps the direct FIFO access to the appropriate location in the physical memory, so that your program does not need to calculate new addresses due to data updates in the FIFOs. For example, an input FIFO can be configured in such a way that the most recent block of data from the input stage is always located at the base of the FIFO. Thus, an FIR filter that processes data from the input FIFO can simply start from the same memory address every time a new block of data is available in the FIFO.

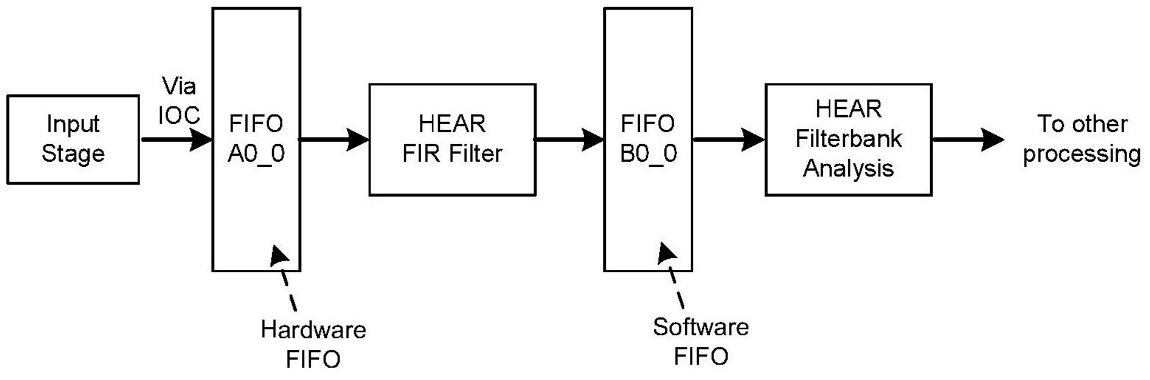


Figure 6. Comparing Hardware FIFOs to Software FIFOs

With audio multiplexing, you can route audio data through the Filter Engine for filtering and basic arithmetic operations. Typical examples of Filter Engine configuration include:

- Low-delay path
- Audio pre-processing on the input
- Audio post-processing on the output

See the *Ezairo 8300 Hardware Reference* and *Filter Engine Reference Manual* for details about audio multiplexing.

The DMA controller on the Arm Cortex-M3 processor has flexible addressing modes to enable you to efficiently transfer data between Arm Cortex-M3 processor data memory (including the mapped HEAR FIFO memory) and peripherals (see the figure "Flexible Data Flow Through the DMA Controller" (Figure 7)). The DMA controller supports up to four independent channels and step sizes between -4 and 4. Each channel can be connected to one data source and one data sink. A DMA data source or sink can be a data buffer in the Arm Cortex-M3 processor memory map, or a peripheral such as the G.722 encoder or the SPI. See the *Ezairo 8300 Hardware Reference* for more information about the DMA controller.

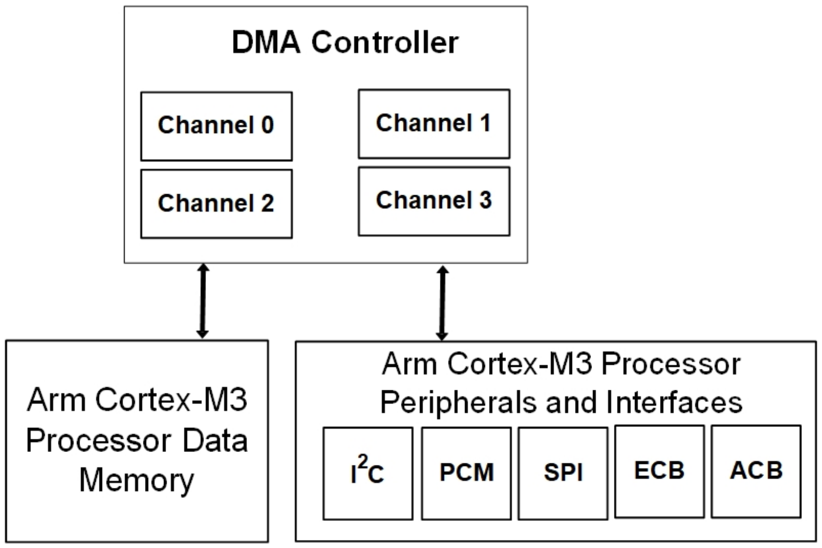

DMA Controller
Channel 0
Channel 1
Channel 2
Channel 3
Arm Cortex-M3 Processor Data Memory
Arm Cortex-M3 Processor Peripherals and Interfaces
I2C
PCM
SPI
ECB
ACB

Figure 7. Flexible Data Flow Through the DMA Controller

CHAPTER 4

Connecting Hardware and Installing Software

This topic explains what you need for using the Ezairo 8300 SDK, how to connect the onsemi Evaluation and Development Board, and how to install the necessary software before you begin working.

4.1 CONNECTING THE HARDWARE

4.1.1 Hardware Prerequisites

The following items are needed before you can make connections:

- Ezairo 8300 Evaluation and Development Board and a USB to Micro-USB cable
- A Binho Pulsar USB Host Adapter and its associated cables
- A computer running Windows
- A SEGGER J-Link™, if you are debugging the LPDSP32 processor
- A SEGGER J-Link device capable of running JFLASH, if you are accessing the NVM via the JLINK header

4.1.2 Connecting the Board

To connect the Evaluation and Development Board to a computer:

1. Check the jumper positions. The figure "Ezairo 8300 Evaluation and Development Board Diagram" (Figure 8), below, depicts the jumper positions necessary to power the Evaluation and Development Board from the USB port and connect the Board to the Windows computer. The black rectangles in the diagram indicate the required jumper positions.

   NOTE: For the onboard flash (LE25S161) to work, the VDDO1−S header must be shorted across only pins 7 and 8. This is because pins 1 and 2 use the Ezairo 8300's VDDIF regulator, and pins 3 and 4 use the EVB's 1.25 V regulator.
2. Ensure that the power switch (SW1) is in the ON position.
3. A connection to the EVB can be performed using the USB J-Link connection for Arm Cortex-M3 debugging, I^2C via the Pulsar DIN connector for CFX and Arm Cortex-M3 core programming, and via the JLINK pins for LPDSP32 debugging.

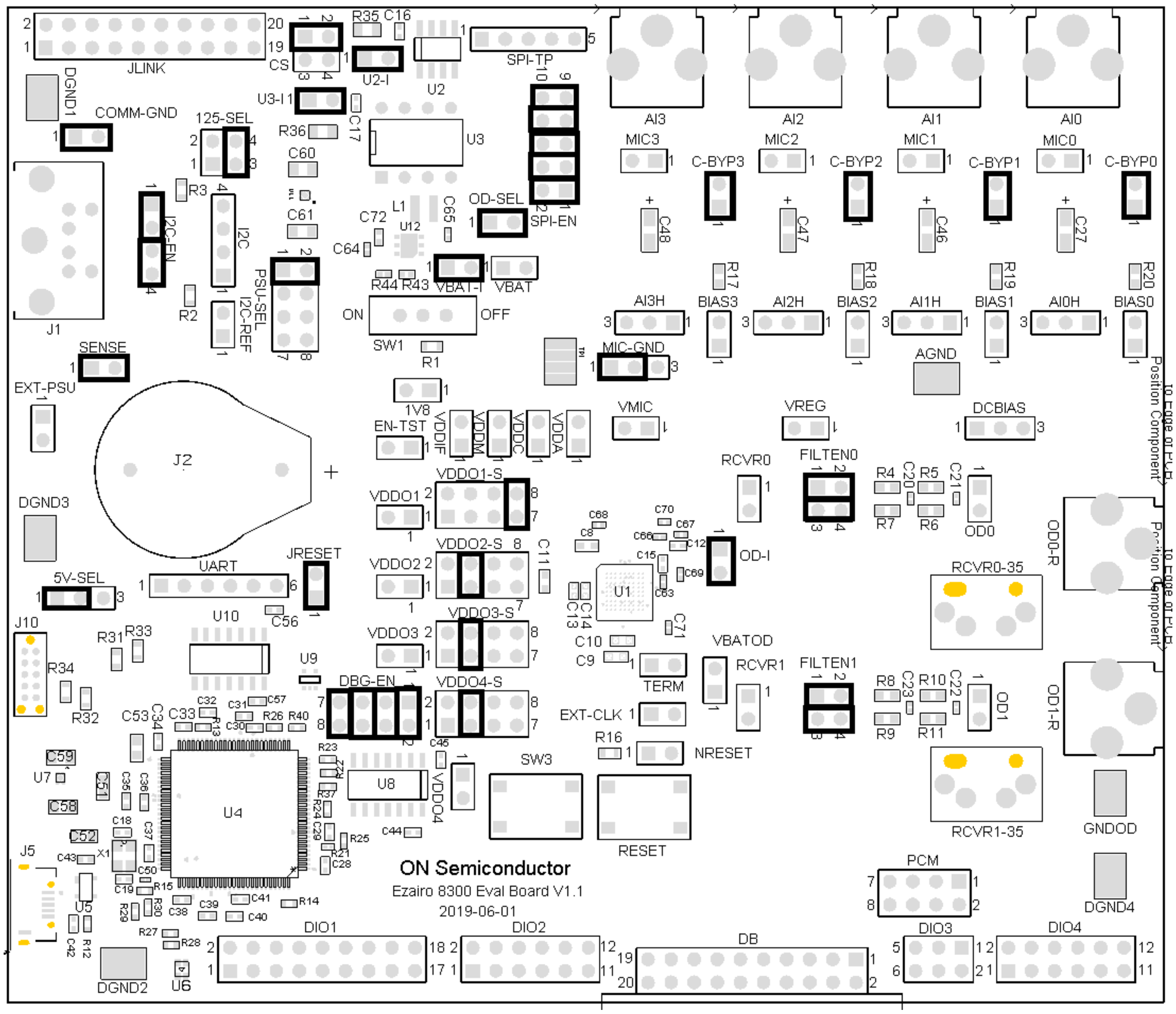


Figure 8. Ezairo 8300 Evaluation and Development Board Diagram

4. Once the jumpers are in the right positions, you can plug the micro USB cable into the socket (J5) on the board. The LED close to the USB connector flashes green when it is initially plugged in. After a second or two, the green LED turns a steady green to show an established connection. If there are any communication issues and the USB port cannot establish a connection with the Ezairo 8300, the LED flashes red instead of green, and continues to flash. If this occurs, double check the jumper configuration and ensure that the ON-OFF switch is set to ON.
5. To enable debugging of a CFX program, connect the cable from the Pulsar to the development board socket J1. If you are only debugging a CFX program and not an Arm Cortex-M3 core program, it is also possible to power the board from the Pulsar by moving the `PSU_SEL` jumper to the 3-4 position, and then the USB cable is not required. More information about this can be found in the *Ezairo 8300 Evaluation and Development Board Manual*.

4.2 Software Prerequisites

Ezairo 8300 SDK (available at www.onsemi.com)

The ASIP Programmer tools require the Visual C++ 2015 64-bit runtime be installed. If you do not have Visual Studio or Visual C++ 2015 installed, you can download the runtime redistributable from the address below:

https://www.microsoft.com/en-us/download/confirmation.aspx?id=52685

NOTE: The ASIP Designer is a license based product which requires access to a license server. To configure the license server location, use the *User_Setup.exe* tool located in the following directory, where **`<version>`** is the installed version number of the tool (e.g.: P-2019.03-SP2):

*C:/Program Files (x86)/ON Semiconductor\Ezairo 8300 SDK\ASIP Programmer\***`<version>`***\win64\bin\WINbin*

The recommended J-Link version to use with the Ezairo 8300 SDK is 7.84.

4.3 Installing the Software

To install the SDK, run the *Ezairo_8300_SDK_***<version>***.exe* application, where **<version>** is the the SDK version number. If you are using Windows 10, the installer prompts you to run as an administrator.

The installer copies the files to *C:/Program Files (x86)/ON Semiconductor\Ezairo 8300 SDK* by default.

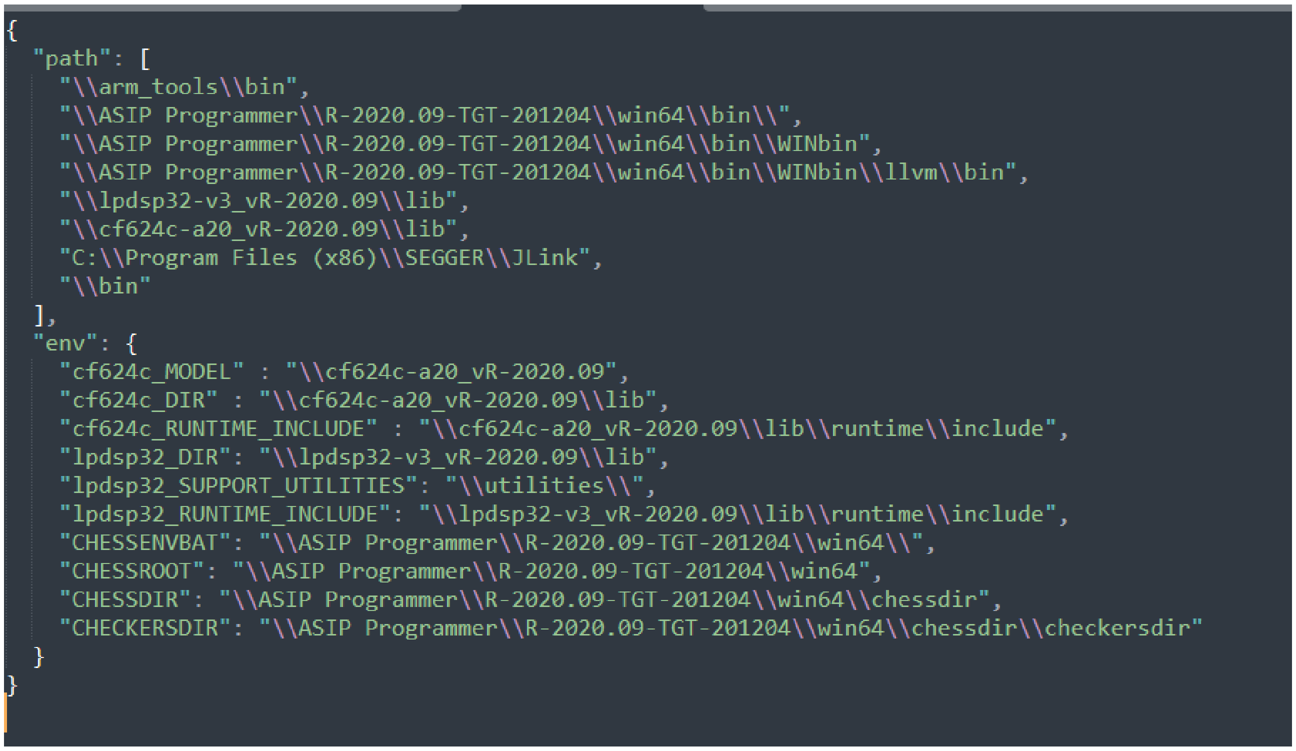
The installer also copies the Ezairo 8300 device support files into *C:\Program Files (x86)\Common Files\SignaKlara\CTK*.

A desktop link to the Ezairo 8300 SDK is installed. Use this for opening the SDK, so that it can properly configure the build environment for the SDK.

4.3.1 Using the ide.json File

ide.json is a JSON-format file that includes all the paths that need to be in your system variable path, and all the necessary environment variables, so that your Ezairo 8300 SDK code projects can work properly.

The *ide.json* file is included with your Ezairo 8300 SDK install. The figure "Default ide.json File" (Figure 9) shows an example of the file’s contents, which can change from release to release.



```
{
  "path": [
    "\\arm_tools\\bin",
    "\\ASIP Programmer\\R-2020.09-TGT-201204\\win64\\bin\\",
    "\\ASIP Programmer\\R-2020.09-TGT-201204\\win64\\bin\\WINbin",
    "\\ASIP Programmer\\R-2020.09-TGT-201204\\win64\\bin\\WINbin\\llvm\\bin",
    "\\lpdsp32-v3_vR-2020.09\\lib",
    "\\cf624c-a20_vR-2020.09\\lib",
    "C:\\Program Files (x86)\\SEGGER\\JLink",
    "\\bin"
  ],
  "env": {
    "cf624c_MODEL" : "\\cf624c-a20_vR-2020.09",
    "cf624c_DIR" : "\\cf624c-a20_vR-2020.09\\lib",
    "cf624c_RUNTIME_INCLUDE" : "\\cf624c-a20_vR-2020.09\\lib\\runtime\\include",
    "lpdsp32_DIR": "\\lpdsp32-v3_vR-2020.09\\lib",
    "lpdsp32_SUPPORT_UTILITIES": "\\utilities\\",
    "lpdsp32_RUNTIME_INCLUDE": "\\lpdsp32-v3_vR-2020.09\\lib\\runtime\\include",
    "CHESSENVBAT": "\\ASIP Programmer\\R-2020.09-TGT-201204\\win64\\",
    "CHESSROOT": "\\ASIP Programmer\\R-2020.09-TGT-201204\\win64",
    "CHESSDIR": "\\ASIP Programmer\\R-2020.09-TGT-201204\\win64\\chessdir",
    "CHECKERSDIR": "\\ASIP Programmer\\R-2020.09-TGT-201204\\win64\\chessdir\\checkersdir"
  }
}
```

Figure 9. Default ide.json File

As the figure shows, the *ide.json* file's `path` array contains all the recommended system variable paths, whereas `env` is a dictionary containing all the necessary key value pairs for the environment variables.

The *ide.json* file contains the default paths when you download it along with the SDK. However, the tools you want to use in your coding projects might be different locations from the default ones shown in the file. If so, you can carefully edit the file, by following these steps:

1. Before you do anything else, back up the file in its original default configuration.

   CAUTION: Do not neglect this step! If ***ide.json*** is deleted, or if its syntax is incorrect, the SDK cannot work properly, so be sure you have a backup of the original.

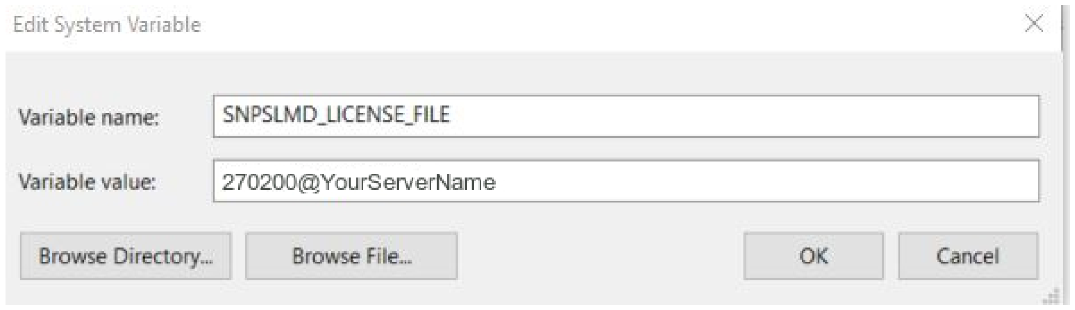
2. Edit the file so that it points to locations appropriate for your system.

NOTE: You need administrator rights to edit *ide.json* if the Ezairo 8300 SDK has been installed in its default location.

3. Save the edited file; and keep the backup of the original version, just in case.

4.4 Preparing the System to Build Source Code with the ASIP Designer toolchain

Prior to starting the IDE you need to create a new environment variable, which the IDE uses to find the Synopsys license server. The environment variable needs to be named `SNPSLMD_LICENSE_FILE`. The value of this variable needs to be the location of the license server being used, in the format `port@ServerName`. The default port is 27020, so the name of the license server needs to resemble `27020@YourServerName`. See the figure "Adding the SNPSLMD Environment Variable" (Figure 10) for an example. For further information, see https://www.synopsys.com/support/licensing-installation-computeplatforms/licensing.html.

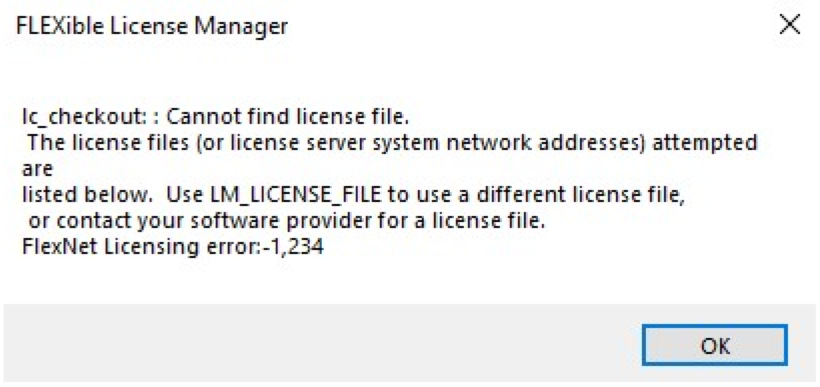

Edit System Variable

Variable name: SNPSLMD_LICENSE_FILE

Variable value: 270200@YourServerName

Figure 10. Adding the SNPSLMD Environment Variable

If this environment variable is entered incorrectly, any attempt to create an ASIP Debug configuration fails, with a message indicating that the plugin cannot load a debug configuration class. (See the figure "ASIP Debug Configuration Error" (Figure 11).)


FLEXible License Manager

```
lc_checkout: : Cannot find license file.
 The license files (or license server system network addresses) attempted
are
listed below.  Use LM_LICENSE_FILE to use a different license file,
 or contact your software provider for a license file.
FlexNet Licensing error:-1,234
```

Figure 11. ASIP Debug Configuration Error

The IDE can also fail in other ways when attempting to use ASIP plugins if this environment variable is incorrect.

Once the SDK has been started, there is one more step to perform before you can build code using the ASIP Designer toolchain: configure the location of the processor model project file *cf624c.prx*. To do this, follow the steps below:

1. Click on the menu item **ASIP Designer** and select **Add/Remove ASIP Processor(s)**.
2. The dialog that pops open must not have any previous processor models populated.
3. Select **Add ASIP**. A new dialog appears, asking for the location of the ASIP processor's *.prx* file. If the installation has been performed to the default location, you can point to the following file: *C:/Program Files (x86)/ON Semiconductor\Ezairo 8300 SDK \cf624c-a20_***`<version>`***\lib\cf624c.prx*
4. Click **Open**, then click **Done** to save the new location of the project file.

5. A window pops up, asking you to select the processors you want to activate. Select them, and choose **Activate**.
6. Follow a similar process to add the processor model for LPDSP32.

CHAPTER 5

Introduction to Sample Applications

To help you learn about the programming aspects of Ezairo 8300, the Software Development Kit (SDK) includes a number of sample code applications as part of its installation. Each sample code application is intended to be simple, easy to use and modify, and practical. You might want to experiment with programming Ezairo 8300 by building an application based on a sample code template. But while the sample code does contain recommended programming practices for Ezairo 8300, it is primarily intended to be a learning tool and does not necessarily represent the most efficient or optimal way to implement programs for Ezairo 8300 unless specified as such.

NOTE: Additional sample code is available in the Communication Toolkit (CTK) Developer Kit. It includes sample applications for communicating with a host system such as a PC.

5.1 Accessing the Sample Applications

The sample code applications are included in the SDK installation. They are located in **`<SDK_HOME>`***\source\samples*.

NOTE: The default installation folder is *C:\Program Files (x86)\ON Semiconductor\Ezairo 8300 SDK*. Your installation folder is represented as **`<SDK_HOME>`** in this document.

Alternatively, the sample code can be viewed through the IDE. To modify, build, and run a particular sample, follow the steps in either Section 5.1.1 “IDE Method #1: Importing sample applications through the Sample Code Explorer”, or Section 5.1.2 “IDE Method #2: Creating a new sample from a template”.

If you would like to import all of the sample applications at once, you can follow the instructions in Section 5.1.3 “IDE Method #3: Importing a sample code application as an existing project”, but select the source folder instead of a particular project's folder.

A text file in Markdown format called *readme.md* accompanies each sample code application, and contains information about how that specific sample works. Most sample applications are intended to be used in the Hierarchical view in Project Explorer, available by selecting the drop-down arrow at the top right, selecting **Projects Presentation**, and choosing **Hierarchical**.

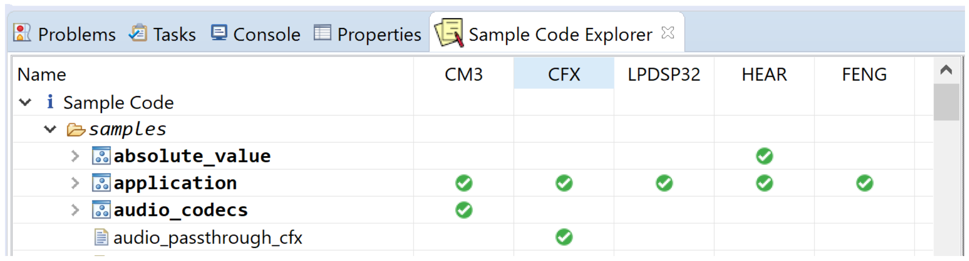
5.1.1 IDE Method #1: Importing sample applications through the Sample Code Explorer

To use this method, perform the following steps:

1. In the C/C++ perspective of the IDE, select from the menu **Window** > **Show View** > **Other...**, and browse to **onsemi** > **Sample Code Explorer...**.
2. A new view opens that allows you to browse the available sample applications included in the SDK. Check marks show which cores are used in each of the sample applications. See the figure "Sample Code Explorer Showing the Cores Used in Each Sample Application" (Figure 12).
3. To import a sample application, right click the sample application name and choose either **Import Single Project** or **Import Multiple Projects** as applicable.
4. Once imported, the sample application can be built as usual.

This method copies one sample code application at a time into your workspace. You can use this copy as a starting point for your own applications.

If you are deleting a project from your workspace that is using hierarchical view, delete all sub-projects first before deleting the top-level application project.



Name	CM3	CFX	LPDSP32	HEAR	FENG
Sample Code					
samples					
absolute_value				✓	
application	✓	✓	✓	✓	✓
audio_codecs	✓				
audio_passthrough_cfx		✓			

Figure 12. Sample Code Explorer Showing the Cores Used in Each Sample Application

For even more detail on using the Sample Code Explorer to access sample applications for the CFX, CFX C, and the Arm Cortex-M3 processor, see Section 6 “Working with Sample Applications” on page 27.

5.1.2 IDE Method #2: Creating a new sample from a template

To use this method, perform the following steps:

1. In the C/C++ perspective of the IDE, select from the menu **File** > **New** > **Project...**, or right-click in the empty space in **Project Explorer** and select **New** > **Project...**.
2. Select **SK5** > **SK5 Project**, and click **Next**.
3. Type in a project name without spaces; from the list, select the sample code that you want to view; and click **Finish**. The IDE creates a project in your workspace.

This method creates a new project for either the HEAR or the Filter Engine, based on a template. You can use this as a starting point for your own applications. NVM images can be created through a similar interface.

5.1.3 IDE Method #3: Importing a sample code application as an existing project

For this method, follow these steps:

1. From the menu, select **File** > **Import**.
2. Select **General** > **Existing Projects into Workspace** and click **Next**.
3. From **Select root directory**, browse to the location of the sample code applications in your file system. (The location is typically **<SDK_HOME>***source**samples*\<*sample_name*>, though some sample applications are grouped into sub-folders by category.) Check **Copy projects into workspace**. Then click **Next**.

The selected sample code from the SDK installation is copied into your workspace. You can modify this to build your own application, just as in Method #1. As some samples are grouped into sub-folder categories, you need to right-click and choose **Import as Project** after the initial import is finished.

NOTE: Remember to change the **Project View** to **Hierarchical**.

5.2 Code Structure and General Information

A possible basic starting point for writing a new program is using the sample application simply called *application*, which provides a shell project for each of the programmable cores in Ezairo 8300. Each core has a separate toolchain, and this application contains a project for each of the toolchains. It also contains some recommendations about project structure. See the *readme* file for more details.

The sample applications are named descriptively. Given the manageable number of sample applications available, we recommend exploring the available samples and seeing what is of most use for you as a starting point or as a learning tool for your development needs. In a subset of cases, the samples are located in a subfolder based on the core they are intended to demonstrate, but typically they are located at the top level and are based on functionality rather than on using a particular core.

Note that the chip ID used in the build settings might need to be modified to match the version you are using. The final version uses chip ID 101, and so the preprocessor build setting would be set to `SK5_CID=101`. Check this for any sample application you wish to use for debugging on hardware. There is also a different build setting available for the CFX to switch debugging on the hardware versus debugging on the instruction set simulator (ISS). The debug configuration would also need to be changed accordingly.

5.2.1 Shared Data Elements

When creating an application that uses several of Ezairo 8300's cores, sharing data between application elements is often required. Some common methods of sharing data includes:

- Using FIFOs or the DMA to pass input/output data
- Using shared memory references at pre-defined shared locations, and using signaling or semaphores to isolate which core controls the data contents at any given time

A common method of defining symbols is to add the shared symbols to the linker scripts. For example, to define a variable `data_store` symbol located at an offset of 0x200 from the base of HEAR shared memory for use by the CFX, a user application could add a define in C as:

```
extern volatile int data_store
```

or in assembly as:

```
.undef global data data_store
```

by also defining this shared resource in the linker scripts as:

```
_symbol data_store (D_HEAR_SHARED_MEM_BASE + 0x200)
_extern data_store
```

CHAPTER 6

Working with Sample Applications

This topic contains basic information about working with the CFX Assembler, CFX C, and Arm Cortex-M3 processor sample application code, taking you through importing, building, and debugging three different kinds of sample applications. For additional in-depth information about working with sample applications, including those for the HEAR, Filter Engine, and LPDSP32 cores, see Section 5 "Introduction to Sample Applications" on page 24, and the *readme* files included with the sample code. Note that the intended usage for sample applications is for familiarizing yourself with possible uses for Ezairo 8300 and becoming comfortable with the development environment. Since Ezairo 8300 relies on external non volatile memory and does not contain on-chip flash memory, the expected usage for sample applications is debugging applications from RAM.

If you wish to build the sample applications into an NVM image for use with an external NVM device, take care to read the appropriate information in the documentation to determine which modifications are needed. The NVMUpdate utility found in the *utilities* folder of the SDK can be used to download NVM images using I^{2}C or JTag. See the *Integrated Development User's Guide for Ezairo 8300* or the *readme* file along with NVMUpdate for more information.

IMPORTANT: Ezairo 8300 code, whether sample applications or user-built, can operate out of RAM (Debug) or NVM. However, there is no specific test to see whether a clock calibration table is present in NVM during RAM Debug, so the CCO defaults to an uncalibrated trim setting of 7.68MHz. This can sometimes cause erratic behavior in applications, and out-of-spec clock speeds for I^{2}C, SPI, I^{2}S, and I^{3}C.

6.1 Starting the Ezairo 8300 SDK

1. If you are upgrading the Ezairo 8300 SDK, create a new workspace at, for example, *c:\workspace* — using either Windows Explorer or the onsemi Launcher in step 2.
2. Open the onsemi IDE by going to the Windows Start menu and selecting **onsemi** > **Ezairo 8300 SDK**. From the Ezairo 8300 Development Tools IDE Launcher screen, browse to your new workspace, select it, and click **Launch**.

6.2 CFX Assembler Sample Application

6.2.1 Importing and Building the CFX Assembler Sample Project

To use the Ezairo 8300 CFX DSP sample project included with this release, first start the Ezairo 8300 SDK as described in Section 6.1 "Starting the Ezairo 8300 SDK" on page 27.

To import sample applications in the Ezairo 8300 SDK, use the Sample Code Explorer view. This view is open by default in the C/C++ perspective when starting the SDK for the first time. To access the Sample Code Explorer view at any other time, choose **Window** > **Show View** > **Other > onsemi > Sample Code Explorer**. The explorer appears in the views panel in the lower section of the IDE.

In the Sample Code Explorer view, follow these steps to import the CFX assembler sample application:

1. In the **Name** panel at the lower part of the view, click the arrow next to **Sample Code**. This reveals the **samples** directory.
2. Click the arrow next to **samples**. This opens a list of the sample applications. Clicking the arrow next to **CFX** reveals the **sample_CFX_assembly** application.
3. Optional step: if you want to read the sample application's *readme* file, click on the application name. The *readme* file appears in a separate panel, providing a description of what the sample does.

4. To import the CFX sample project, right-click **sample_CFX_assembly** and select **Import Single Project** from the dropdown menu.
5. A copy of the sample application is imported directly into your workspace, as can be seen in the Project Explorer panel on the left side of the view. This copying does not modify the original source in the SDK installation folder.
6. Right-click on the project's name in the Project Explorer panel, and choose **Build Project** from the dropdown menu. This builds the project with no errors.

Before starting development with the ASIP Programmer tools the developer must first point to the CFX or LPDSP32 model library. To do this:

1. Click on the **ASIP Designer** menu item in the top menu and select **Add/Remove ASIP Processor(s)**.
2. Click on **Add ASIP** in the right column.
3. When selecting the CFX processor point to a directory similiar to this: *C:/Program Files (x86)/ON Semiconductor\Ezairo 8300 SDK\cf624c<version>\lib*, or *C:/Program Files (x86)/ON Semiconductor\Ezairo 8300 SDK\lpdsp32<version>\lib* for the LPDSP32 processor.
4. Click on **Done** to save and close the dialog.
5. After selecting the processors to include click on **ASIP Designer** > **Select Active ASIP programmer processor**.
6. The dialog shows all selected processors and blank checkboxes beside them. Enable the checkboxes for the processors you want to use with the ASIP programmer tools; multiple processors can be selected. When you are done, click **Finish** to close the dialog box.

(See the *ASIP Designer - ASIP Programmer Eclipse Reference Manual* (*eclipse-manual.pdf*) for more information.)

At this point the ASIP Programmer tools are configured for use with projects and debuggers.

6.2.2 Debugging the CFX Sample Code

6.2.2.1 Changing CTK Configuration for CFX Debugging

To change the Communication Toolkit (CTK) configuration used for CFX debugging, you can set the `E8300_CTK_CONFIG` environment variable to any existing CTK configuration. By default without the environment variable, the `CFX_CFG_8XXX` configuration is used for CFX debugging, which provides the ideal settings for Ezairo 8300 when using the Pulsar. Another approach to changing the configuration used for CFX debugging (and loading of HEAR and Filter Engine applications) is to edit the `CFX_CFG_8XXX` CTK configuration. To change or view CTK configurations such as this one (for example if you are using a CAA rather than the Pulsar), open the CTK Configuration Manager using the Windows Search or Start Menu. It is not possible to change the CTK configuration for a CFX debug session through the IDE, nor the CTK configuration itself.

6.2.2.2 Debugging the CFX with SEGGER J-Link

This section explains Ezairo 8300's newly-added support for debugging the CFX core via a SEGGER® J-Link® (JTAG/SWD) interface. This feature allows the user to debug the CFX core using the onboard J-Link connection, and to perform simultaneous debugging of the CFX and Arm Cortex-M3 cores.

NOTE: At the moment, the CTK only supports communicating with the CFX via JTAG.

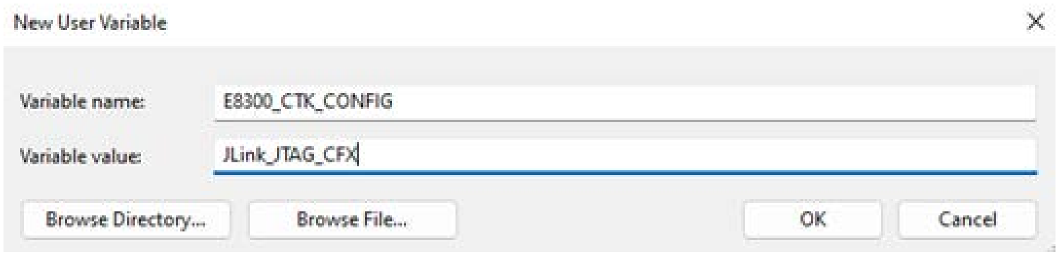
The J-Link communication interface is designed to communicate through the SEGGER J-Link using JTAG/SWD protocols.

- Communication Interface Name: `JLink`
- Communication Module: *JLinkModule.dll*
- Required Drivers: *JLinkARM.DLL*, *JLink_x64.dll* (These are included with the SEGGER J-Link software package, which can be downloaded from SEGGER. The *.dll* files must be found on the path for the module to use.)

This feature has been tested with the SEGGER J-Link driver version 7.68a.

Currently, the CTK provides `J-Link_JTAG_CFX` and `J-Link_SWD_CFX` configurations. If you want to set up the J-Link for CFX deubgging using these provided J-Link JTAG configurations, perform the following steps:

1. Connect the Ezairo 8300 Evaluation and Development Board with a micro USB cable.
2. Create/update the Ezairo 8300 environment variable (`E8300_CTK_CONFIG`) to use the new configuration, as shown in the figure "Creating/Updating the Environment Variable" (Figure 13).



Variable name: E8300_CTK_CONFIG
Variable value: JLink_JTAG_CFX

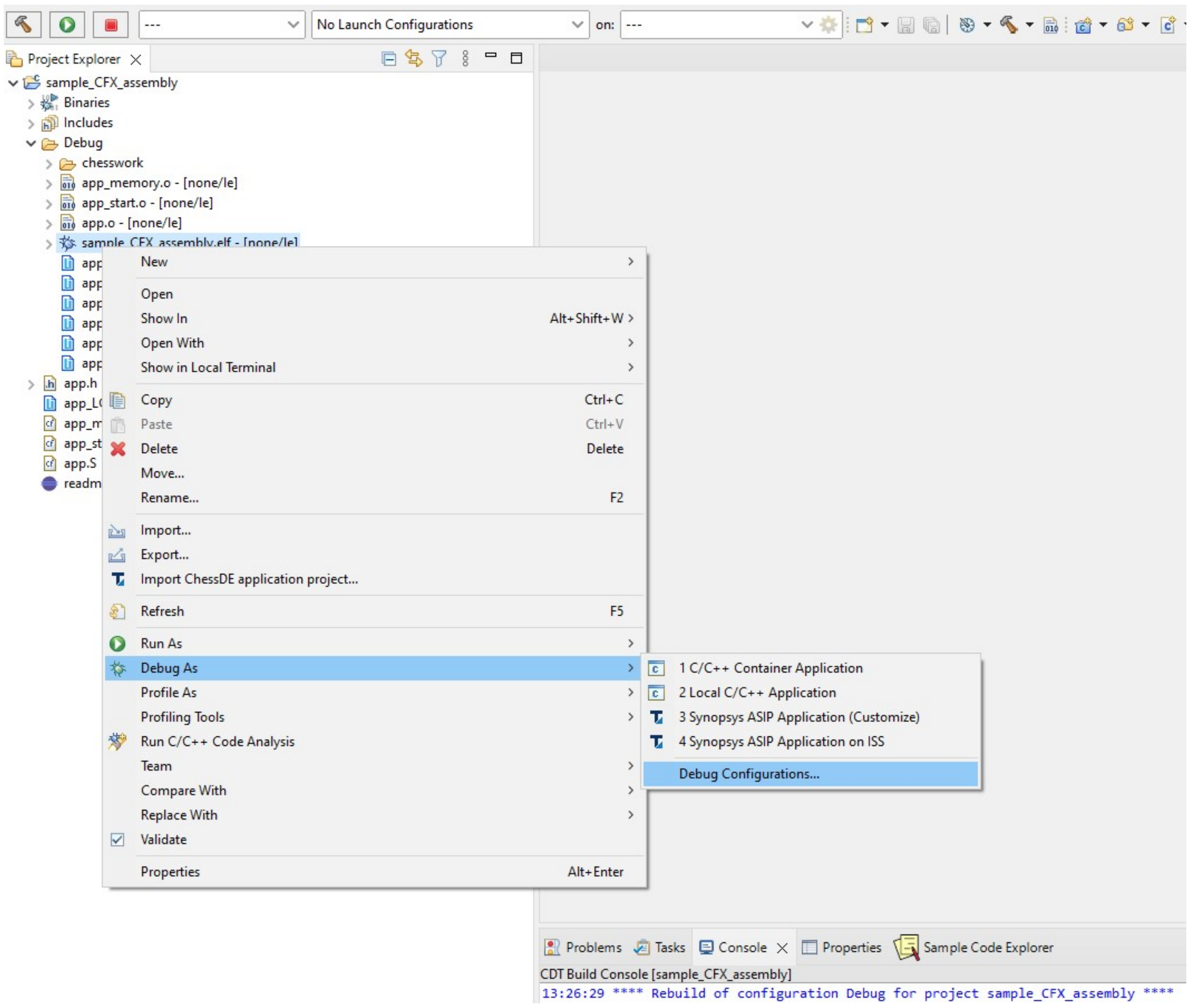
Figure 13. Creating/Updating the Environment Variable

This enables the user to debug the CFX core with the `J-Link_JTAG_CFX` configuration.

IMPORTANT: When debugging the CFX simultaneously with the Arm Cortex-CM33 processor, the configuration for the CFX must have Silent set to `True`. Only start the CFX debug after the Arm Cortex-CM33 processor debug has been started.

If you do not want to use the provided J-Link JTAG debug configuration, create a new debug configuration by following these steps:

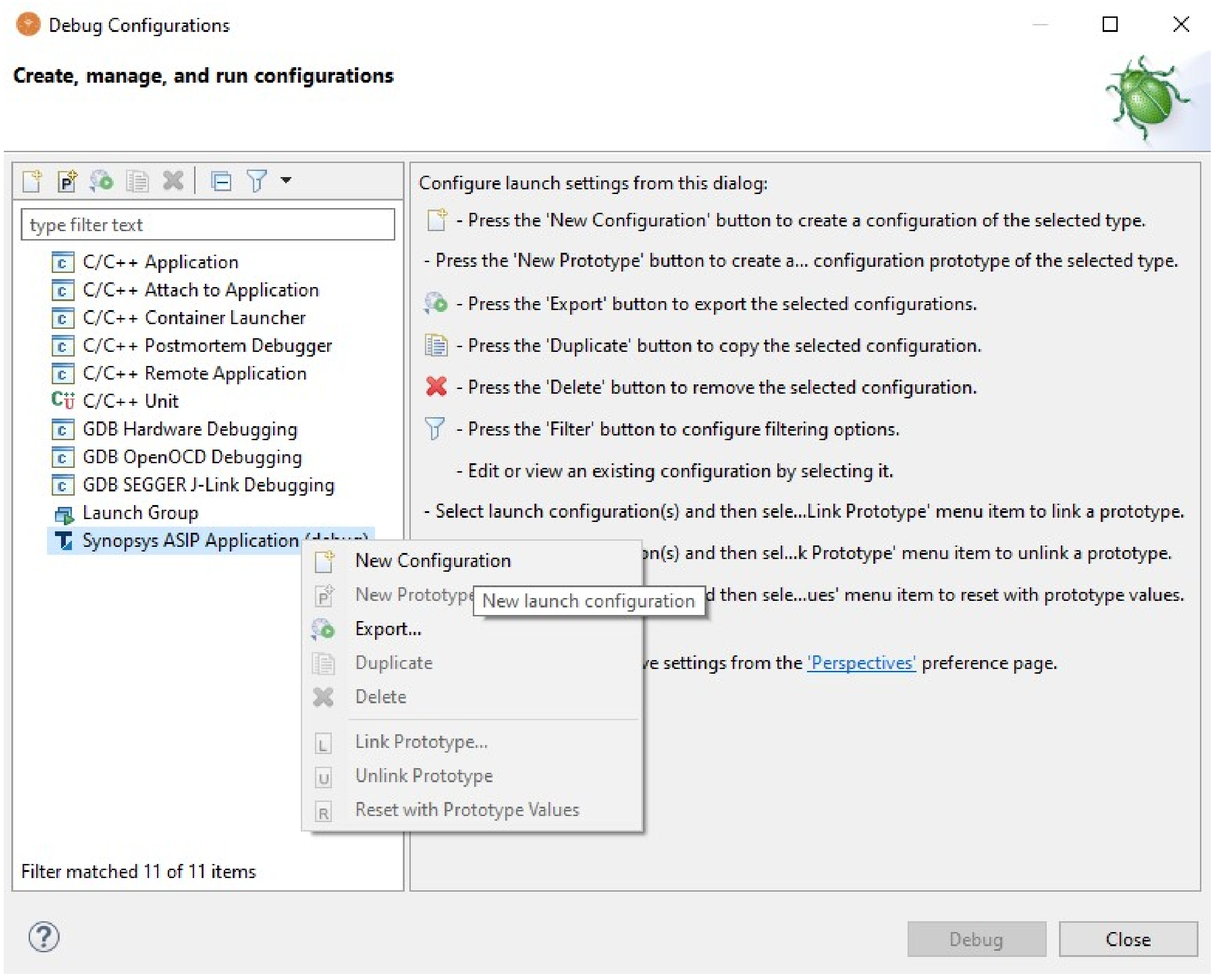
1. Right click on the built *.elf* file (**sample_CFX_assembly** in this case) and select **Debug As** > **Debug Configurations** or select **Synopsys ASIP Application (Customize)** to directly create a new debug configuration with pre-filled values. (See the figure "Creating CFX Assembler Debug Configuration, Step 1" (Figure 14)



```
13:26:29 **** Rebuild of configuration Debug for project sample_CFX_assembly ****
```

Figure 14. Creating CFX Assembler Debug Configuration, Step 1

2. The **Debug Configurations** dialog is displayed. Right click on the **ASIP Designer Application** configuration type and select **New** (as shown in the figure "Creating CFX Assembler Debug Configuration, Step 2" (Figure 15)).


Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Container Launcher
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- C/C++ Unit
- GDB Hardware Debugging
- GDB OpenOCD Debugging
- GDB SEGGER J-Link Debugging
- Launch Group
- Synopsys ASIP Application (debug)

Filter matched 11 of 11 items

Configure launch settings from this dialog:

- Press the 'New Configuration' button to create a configuration of the selected type.
- Press the 'New Prototype' button to create a... configuration prototype of the selected type.
- Press the 'Export' button to export the selected configurations.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.
- Select launch configuration(s) and then sele...Link Prototype' menu item to link a prototype.

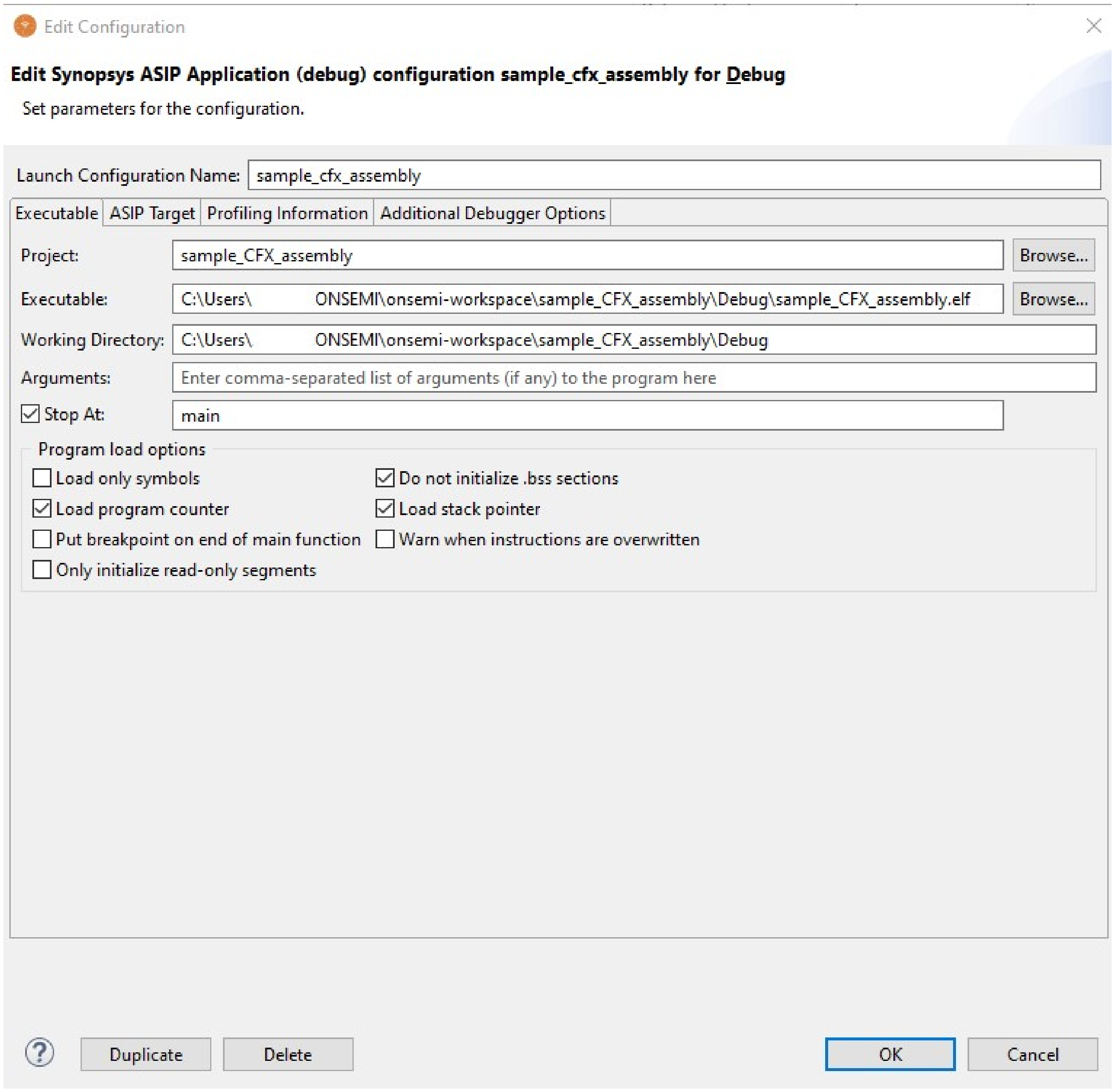
on(s) and then sel...k Prototype' menu item to unlink a prototype.

d then sele...ues' menu item to reset with prototype values.

e settings from the 'Perspectives' preference page.

Figure 15. Creating CFX Assembler Debug Configuration, Step 2

1. A new dialog is displayed, allowing you to create your debug configuration. Within this dialog the following information must be added (shown for the **sample_CFX_assembly** executable):
   a. In the **Executable** tab view, select your desired **Project** (in the figure "Creating CFX Assembler Debug Configuration, Step 1" (Figure 14), **sample_CFX_assembly**) and **Executable** (in the figure "Creating CFX Assembler Debug Configuration, Step 2" (Figure 15), **C:\eclipse-workspace\sample_CFX_assembly\Debug**). See the figure "Selecting Project and Executable for Debug Configuration" (Figure 16).



Edit Configuration

Edit Synopsys ASIP Application (debug) configuration sample_cfx_assembly for Debug

Set parameters for the configuration.

Launch Configuration Name: sample_cfx_assembly

Executable | ASIP Target | Profiling Information | Additional Debugger Options

Project: sample_CFX_assembly

Executable: C:\Users\ ONSEMI\onsemi-workspace\sample_CFX_assembly\Debug\sample_CFX_assembly.elf

Working Directory: C:\Users\ ONSEMI\onsemi-workspace\sample_CFX_assembly\Debug

Arguments: Enter comma-separated list of arguments (if any) to the program here

☑ Stop At: main

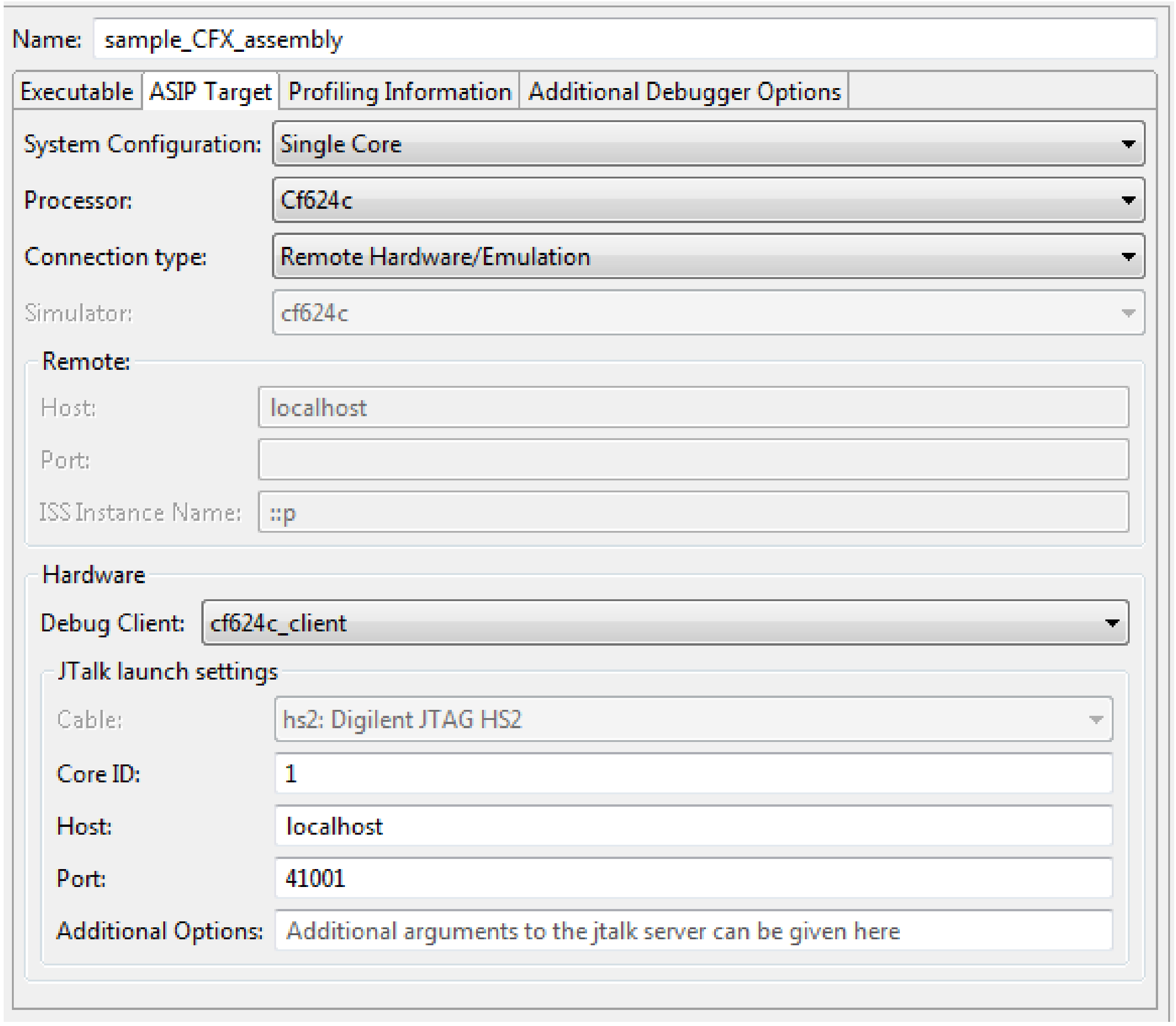
Program load options

- ☐ Load only symbols
- ☑ Load program counter
- ☐ Put breakpoint on end of main function
- ☐ Only initialize read-only segments
- ☑ Do not initialize .bss sections
- ☑ Load stack pointer
- ☐ Warn when instructions are overwritten

Figure 16. Selecting Project and Executable for Debug Configuration

NOTE: If **Put breakpoint on end of main function** is selected and the last instruction in main is not a valid breakpoint location, the debug console reports `ERR_INVALID_HWBREAK_LOCATION` when debugging the application.

b. In the **ASIP Target** tab view, click the **Connection Type** dropdown menu and select **Remote Hardware/Emulation** (see the figure "Selecting Connection Type for Debug Configuration" (Figure 17)).



Name: sample_CFX_assembly

System Configuration: Single Core

Processor: Cf624c

Connection type: Remote Hardware/Emulation

Simulator: cf624c

Remote:

Host: localhost

Port:

ISS Instance Name: ::p

Hardware

Debug Client: cf624c_client

JTalk launch settings

Cable: hs2: Digilent JTAG HS2

Core ID: 1

Host: localhost

Port: 41001

Additional Options: Additional arguments to the jtalk server can be given here

Figure 17. Selecting Connection Type for Debug Configuration

NOTE: The **Debug Client** is an application that is located in the model (selected during setup) under the ISS directory. If **Remote Hardware/Emulation** is selected and a red **X** appears on the dialog, this indicates that *cf624c_client* might be missing. Check the ISS directory; if the file is missing (*cf624c_client.exe*), it can be found in the *Debug_Client* directory off the root of the SDK installation. Copying the client application into ISS resolves the issue.

All other information in the debug configuration can remain set to defaults. The debug client uses the CTK to communicate with the chip, and always uses the `CFX_CFG_8XXX` CTK configuration, which must be left as the default Pulsar configuration. **Host** and **Port** are not used in this case.

IMPORTANT: If you are having trouble establishing a connection to the ASIP target, we recommend that you reduce the speed of the I^2C to its lowest setting.

To begin debugging, follow these steps:

1. Click on **Apply** and **Debug**. The application begins loading to the device. If this is successful, an ASIP Designer Debug session opens.
2. You are now able to step through the code, look at registers and memory views, and more.

If you encounter issues with debugging, refer to *CFX Debug Troubleshooting*.

6.3 CFX C Sample Application

IMPORTANT: The IDE cannot disable interrupts during a debugging session.

A workaround for the CFX core is clearing the master interrupt register manually in the memory view (0x840 in IOMEM).

6.3.1 Importing and Building the CFX C Sample Project

To use the Ezairo 8300 C sample project included with this release, start the Ezairo 8300 SDK from the desktop link.

To import sample applications in the Ezairo 8300 SDK, use the Sample Code Explorer view. This view is open by default in the C/C++ perspective when starting the SDK for the first time. To access the Sample Code Explorer view at any other time, choose **Window** > **Show View** > **Other > onsemi > Sample Code Explorer**. The explorer appears in the views panel in the lower section of the IDE.

In the Sample Code Explorer view, follow these steps to import the CFX C sample application:

1. In the **Name** panel at the lower part of the view, click the arrow next to **Sample Code**. This reveals the **samples** directory.
2. Click the arrow next to **samples**. This opens a list of the sample applications. Clicking the arrow next to **CFX** reveals the **factorial_calculation_demo** application.
3. Optional step: if you want to read the sample application's *readme* file, click on the application name. The *readme* file appears in a separate panel, providing a description of what the sample does.
4. To import the CFX sample project, right-click **factorial_calculation_demo** and select **Import Single Project** from the dropdown menu.
5. A copy of the sample application is imported directly into your workspace, as can be seen in the Project Explorer panel on the left side of the view. This copying does not modify the original source in the SDK installation folder.
6. Right-click on the project's name in the Project Explorer panel, and choose Build Project from the dropdown menu. This builds the project with no errors.

Before starting development with the ASIP Programmer tools the developer must first point to the CFX or LPDSP32 model library. To do this:

1. Click on the **ASIP Designer** menu item in the top menu and select **Add/Remove ASIP Processor(s)**.
2. Click on **Add ASIP** in the right column.
3. When selecting the CFX processor point to a directory similiar to this: *C:/Program Files (x86)/ON Semiconductor\Ezairo 8300 SDK\cf624c<version>\lib*, or *C:/Program Files (x86)/ON Semiconductor\Ezairo 8300 SDK\lpdsp32<version>\lib* for the LPDSP32 processor.
4. Click on **Done** to save and close the dialog.
5. After selecting the processors to include click on **ASIP Designer** > **Select Active ASIP programmer processor**.
6. The dialog shows all selected processors and blank checkboxes beside them. Enable the checkboxes for the processors you want to use with the ASIP programmer tools; multiple processors can be selected. When you are done, click **Finish** to close the dialog box.

(See the *ASIP Designer - ASIP Programmer Eclipse Reference Manual* (*eclipse-manual.pdf*) for more information.)

At this point the ASIP Programmer tools are configured for use with projects and debuggers.

6.3.2 Debugging CFX C Sample Code

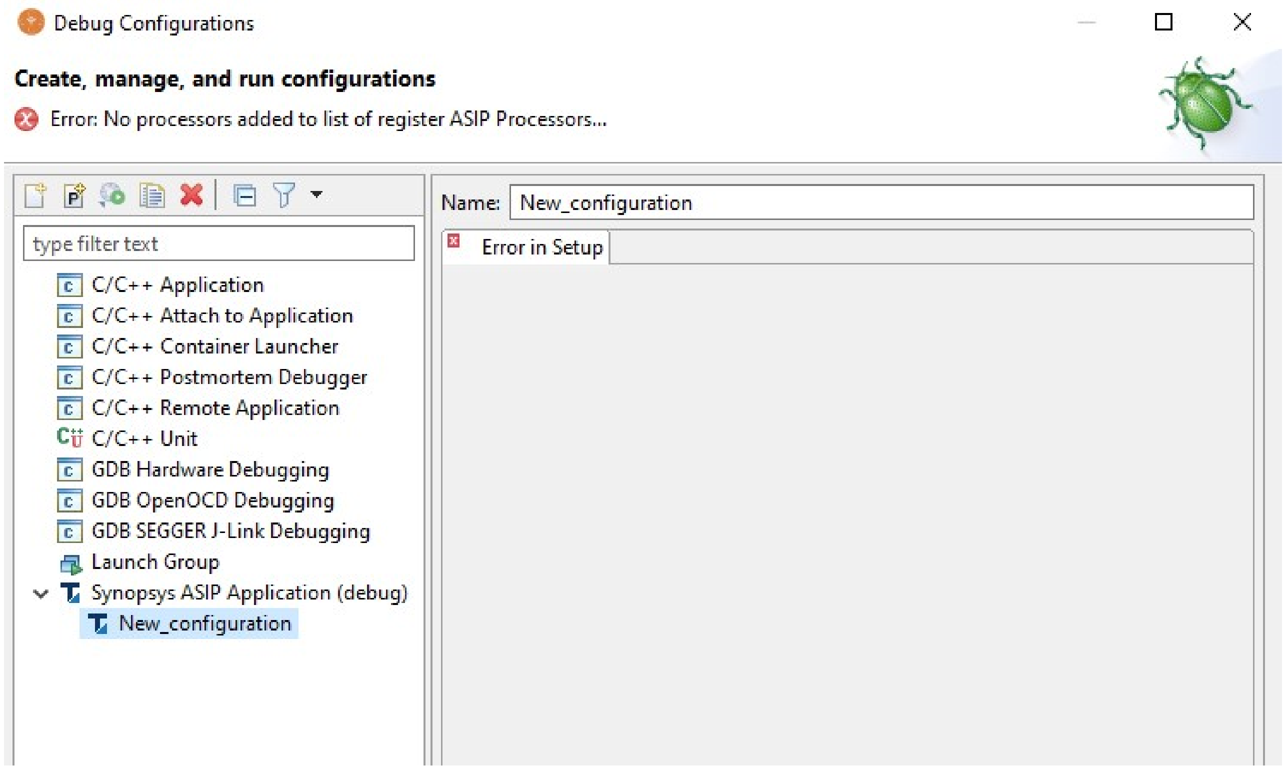
To debug the built *factorial_calculation_demo* sample code, which is implemented in C, follow the same procedure described in Section 6.2.2 “Debugging the CFX Sample Code” on page 28.

6.3.3 CFX Debug Troubleshooting

The following are some possible issues that may arise while debugging CFX sample code, along with recommended solutions to the problems:

Configuration is Blank with red X in tab:

If you see a screen like figure "CFX C Debug Configuration Setup Error" (Figure 18), below, when you create a new ASIP Designer debug configuration, it indicates that you have not configured the ASIP Designer with a model project yet, or that the processor has not been selected as active. To resolve this, follow the instructions in Section 4.4 “Preparing the System to Build Source Code with the ASIP Designer toolchain” on page 21.



Debug Configurations

Create, manage, and run configurations

Error: No processors added to list of register ASIP Processors...

Name: New_configuration

Error in Setup

Figure 18. CFX C Debug Configuration Setup Error

When the debugging operation begins, it stops with a launch error, including the non-obvious example of:

```
Error in services launch sequence. Invalid argument: localhost=localhost
```

This indicates that the loading process has attempted to start, but has encountered an error. Some possible causes and solutions for this are:

- There might be no device or Pulsar connected, and the debugger needs such a connection to be able to create a local debug host (needed to resolve the localhost argument in a standard debug configuration). Make sure your Pulsar and device are properly connected, powered, and are able to communicate.
- You can confirm your CTK configuration by using the CTK Configuration Manager (found in the Windows Start menu under **onsemi**). Edit the `CFX_CFG_8XXX` configuration and choose **Test**. Seeing the result **Configuration is okay** confirms the configuration. If communication with the device is fine, the utility returns a chip family of 10 (SK5).
- PSU-SEL pins 3 and 4 need to be shorted to power the board through the Pulsar. See the *Ezairo 8300 Evaluation and Development Board Manual* for more information.
- A license is required during debugging; if a license cannot be found for the Synopsys debugging tools, the launch fails.
- There might be issues with the Pulsar programmer. Two specific issues can cause problems:
  - When using the SV1 evaluation board, the Pulsar needs to have its pull-ups disabled; otherwise, false NAKs occur.
- The CTK might have become corrupted. Either run the SDK installer and use the repair function to fix the CTK installation, or click uninstall from the Windows Control Panel and then click on **repair**.

6.4 ARM CORTEX-M3 PROCESSOR SAMPLE APPLICATION

6.4.1 Importing and Building an Arm Cortex-M3 Processor Sample Project

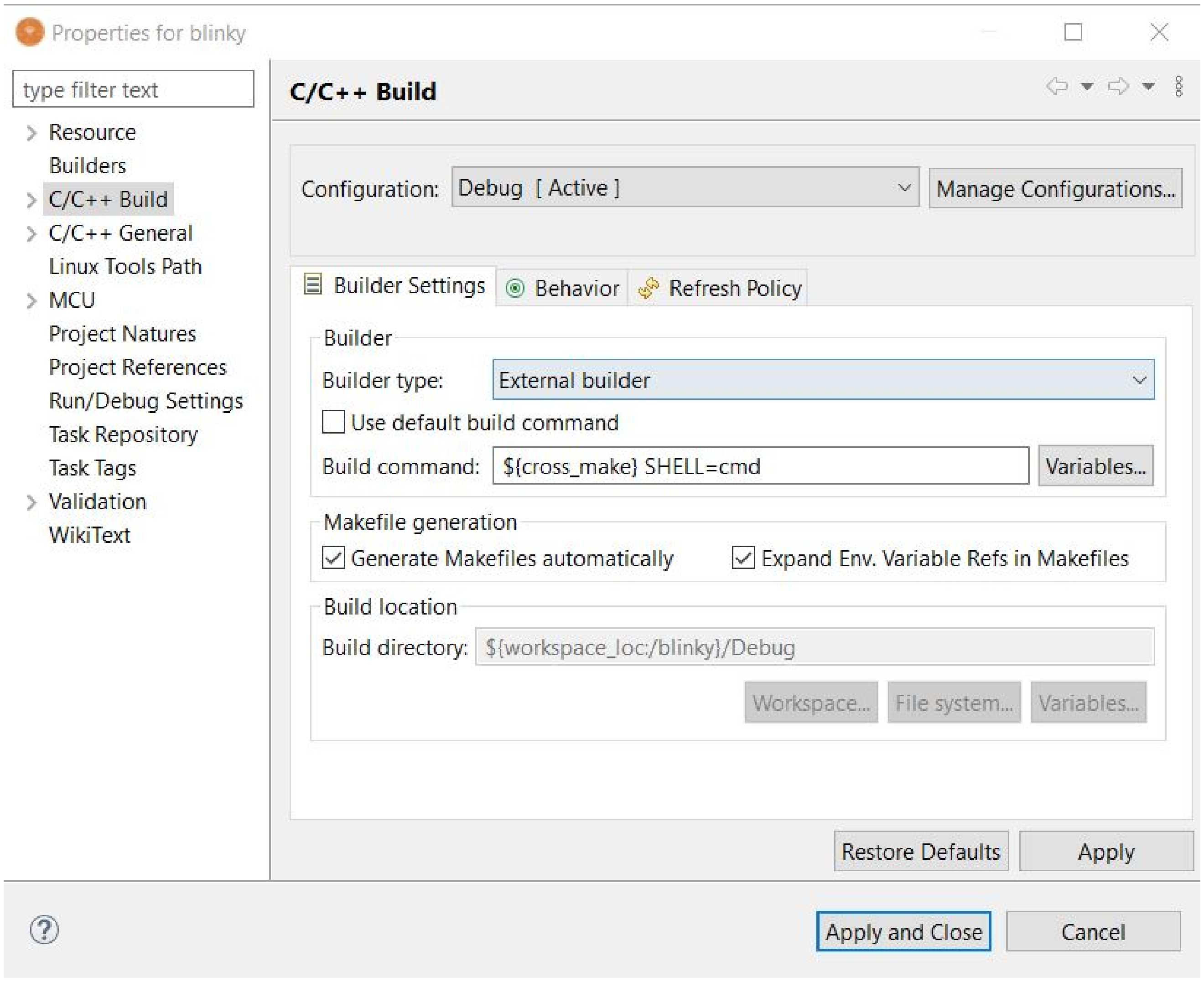
To use the Arm Cortex-M3 processor sample projects, start the Ezairo 8300 SDK from the desktop link.

To import sample applications in the Ezairo 8300 SDK, use the Sample Code Explorer view. This view is open by default in the C/C++ perspective when starting the SDK for the first time. To access the Sample Code Explorer view at any other time, choose **Window** > **Show View** > **Other > onsemi > Sample Code Explorer**. The explorer appears in the views panel in the lower section of the IDE.

In the Sample Code Explorer view, follow these steps to import an Arm Cortex-M3 processor sample application:

1. In the **Name** panel at the lower part of the view, click the arrow next to **Sample Code**. This reveals the **samples** directory.
2. Click the arrow next to **samples**. This opens a list of the sample applications. Clicking the arrow next to **Cortex-M3** reveals the Arm Cortex-M3 core applications.
3. Optional step: if you want to read the sample application's *readme* file, click on the application name. The *readme* file appears in a separate panel, providing a description of what the sample does.
4. To import an Arm Cortex-M3 processor sample application, right-click on the desired application name (**ASRC**, **blinky**, **cm3_bootloader**, **i2c_cmsis**, **swmTraceExample**, or **uart_cmsis**) and select **Import Single Project** from the dropdown menu.
5. A copy of the sample application is imported directly into your workspace, as can be seen in the Project Explorer panel on the left side of the view. This copying does not modify the original source in the SDK installation folder.
6. Right-click on the project's name in the Project Explorer panel, and choose Build Project from the dropdown menu. This builds the project with no errors.

IMPORTANT: The Ezairo 8300 SDK requires certain files to be present in the path when building Arm Cortex-M3 processor-based applications. Depending on the system path configuration and/or command shells present, the build might fail due to these files not being found. In some cases *sh.exe* is found, which also causes a build to fail on Windows. To prevent such failures when generating a new project for the Arm Cortex-M3 core using File->New->C/C++ Project, add "SHELL=cmd" (no quotes, case sensitive) to the build command, and uncheck "Use default build command", as shown in the figure "Build Argument Shell Command" (Figure 19). This is done in the Project Properties under the C/C++ Build->Builder Settings Tab.



Configuration: Debug [Active]

Builder

Builder type: External builder

☐ Use default build command

Build command: ${cross_make} SHELL=cmd

Makefile generation

☑ Generate Makefiles automatically ☑ Expand Env. Variable Refs in Makefiles

Build location

Build directory: ${workspace_loc:/blinky}/Debug

Figure 19. Build Argument Shell Command

6.4.2 Debugging Arm Cortex-M3 Processor Sample Code

For debugging the built code on the Arm Cortex-M3 processor, create a new debug configuration with the following steps:

1. Right click on the built *.elf* file (**blinky.elf** in this case) and select **Debug As** > **Debug Configurations**, as shown in the figure "Arm Cortex-M3 Processor Debug Configuration, Step 1" (Figure 20).

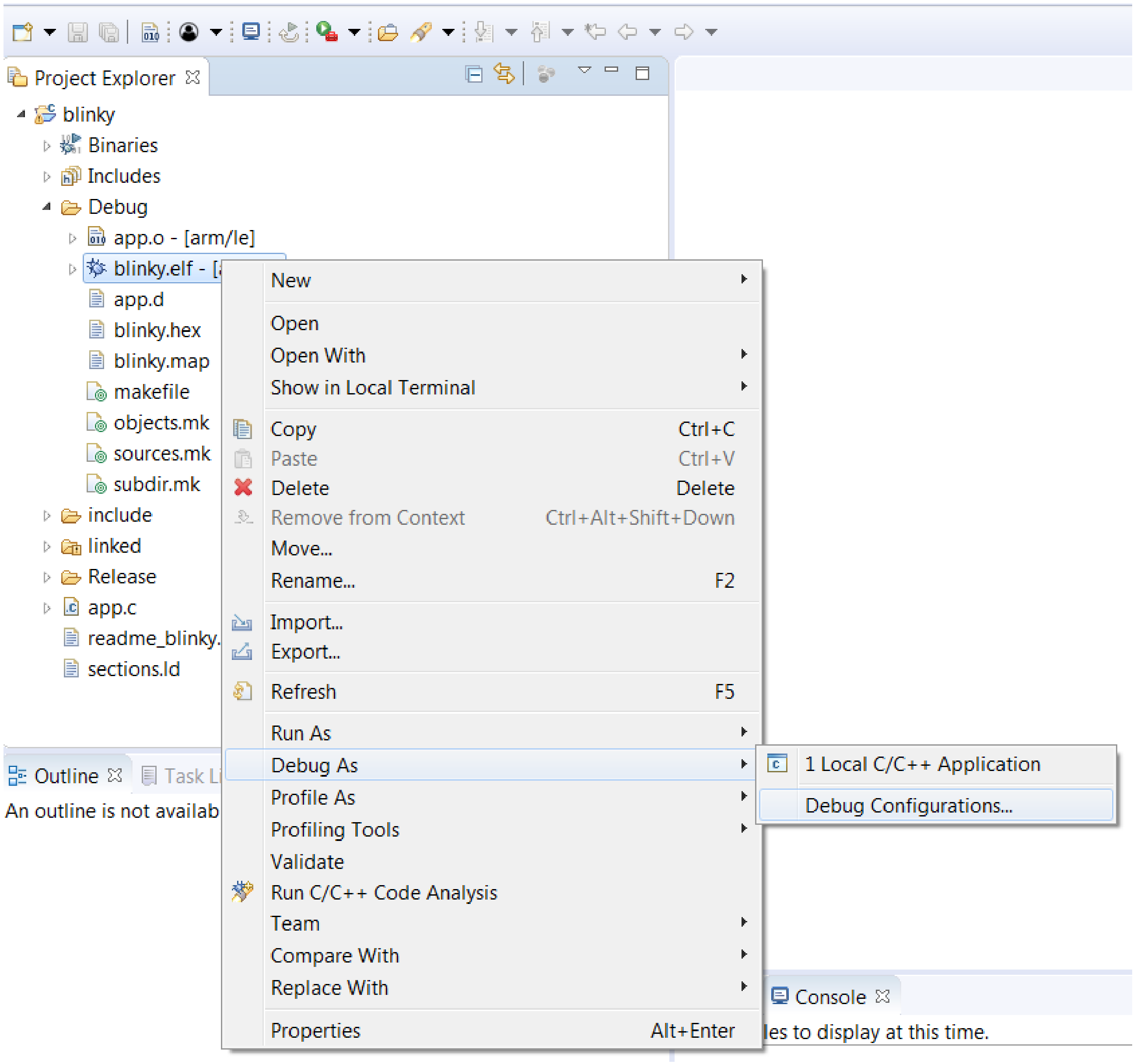
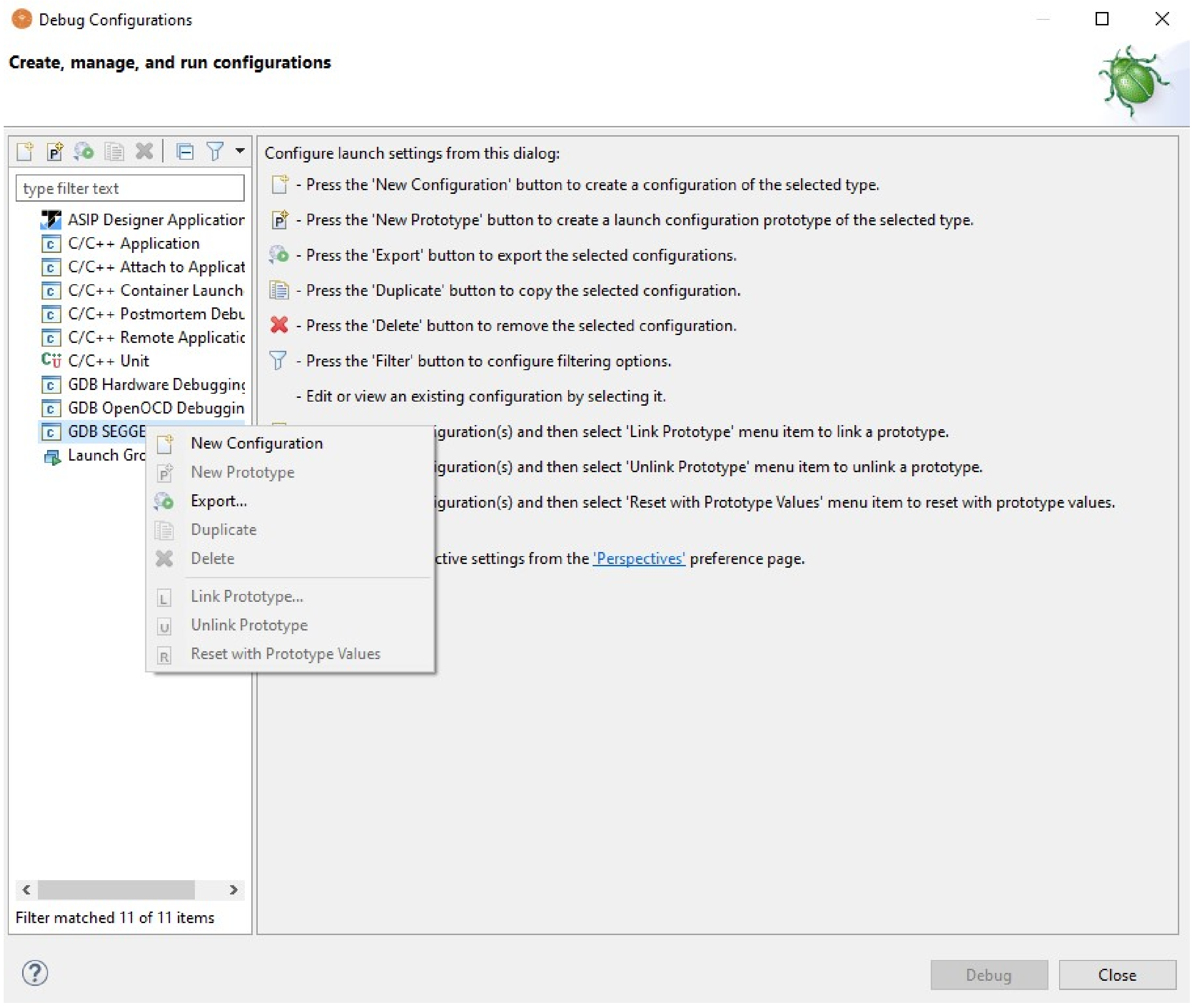


Figure 20. Arm Cortex-M3 Processor Debug Configuration, Step 1

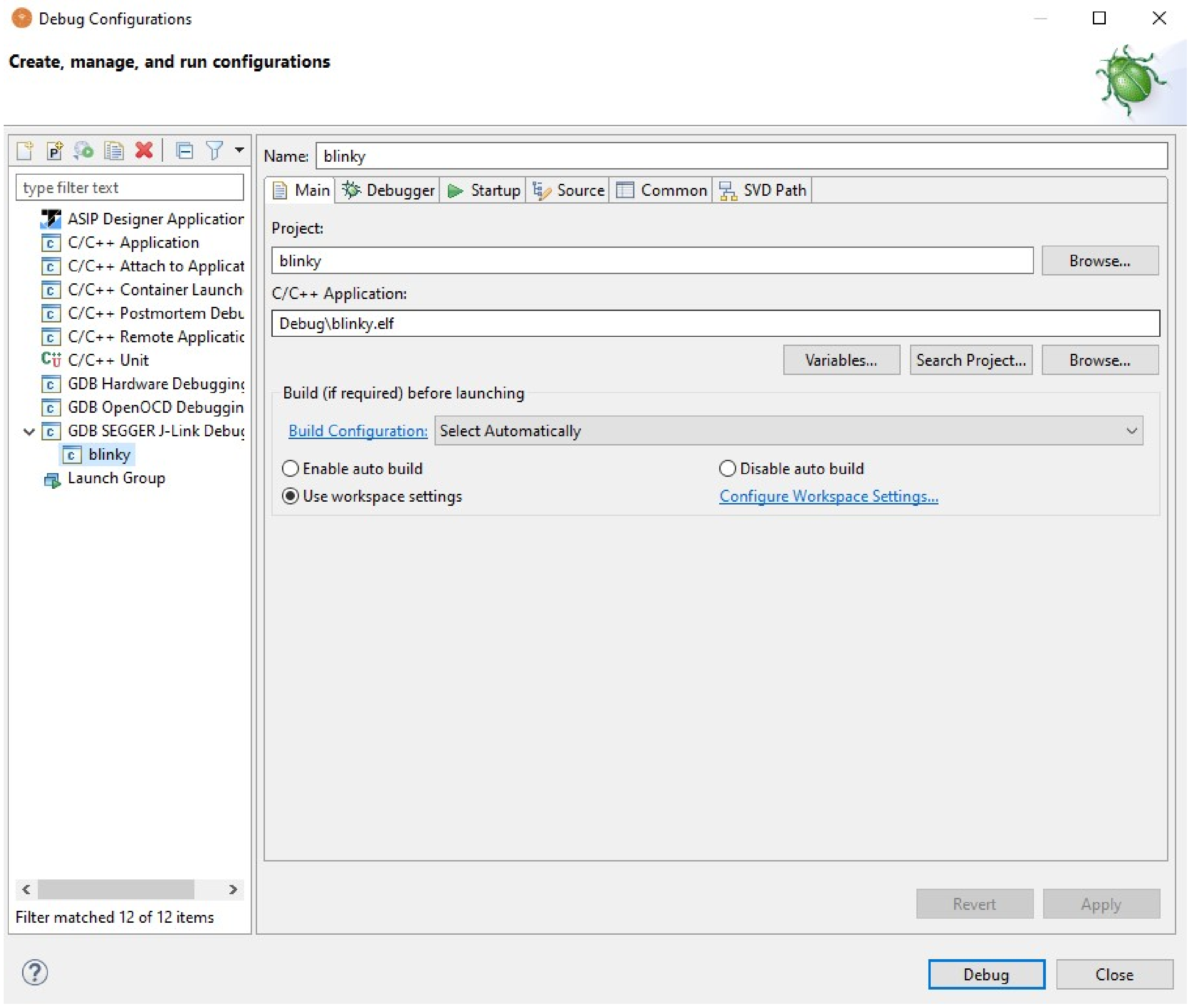
2. The **Debug Configurations** dialog is displayed. Right click on the **GDB SEGGER J-Link Debugging** configuration type and select **New** (see figure "Arm Cortex-M3 Processor Debug Configuration, Step 2" (Figure 21)).



Configure launch settings from this dialog:
- Press the 'New Configuration' button to create a configuration of the selected type.
- Press the 'New Prototype' button to create a launch configuration prototype of the selected type.
- Press the 'Export' button to export the selected configurations.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.
iguration(s) and then select 'Link Prototype' menu item to link a prototype.
iguration(s) and then select 'Unlink Prototype' menu item to unlink a prototype.
iguration(s) and then select 'Reset with Prototype Values' menu item to reset with prototype values.
ctive settings from the 'Perspectives' preference page.

Figure 21. Arm Cortex-M3 Processor Debug Configuration, Step 2

3. A new dialog is displayed, allowing you to create your debug configuration. Within this dialog the following information must be added:
   a. In the **Main** tab view, select the project you are working on (in this case **blinky**), and the C/C++ application (in this case **Debug\blinky.elf**), as shown in figure "Selecting Project and Application Type" (Figure 22).


Debug Configurations

Create, manage, and run configurations

type filter text

ASIP Designer Application
C/C++ Application
C/C++ Attach to Applicat
C/C++ Container Launch
C/C++ Postmortem Debu
C/C++ Remote Applicatic
C/C++ Unit
GDB Hardware Debugging
GDB OpenOCD Debuggin
GDB SEGGER J-Link Debug
blinky
Launch Group

Filter matched 12 of 12 items

Name: blinky

Main | Debugger | Startup | Source | Common | SVD Path

Project: blinky

C/C++ Application: Debug\blinky.elf

Build (if required) before launching

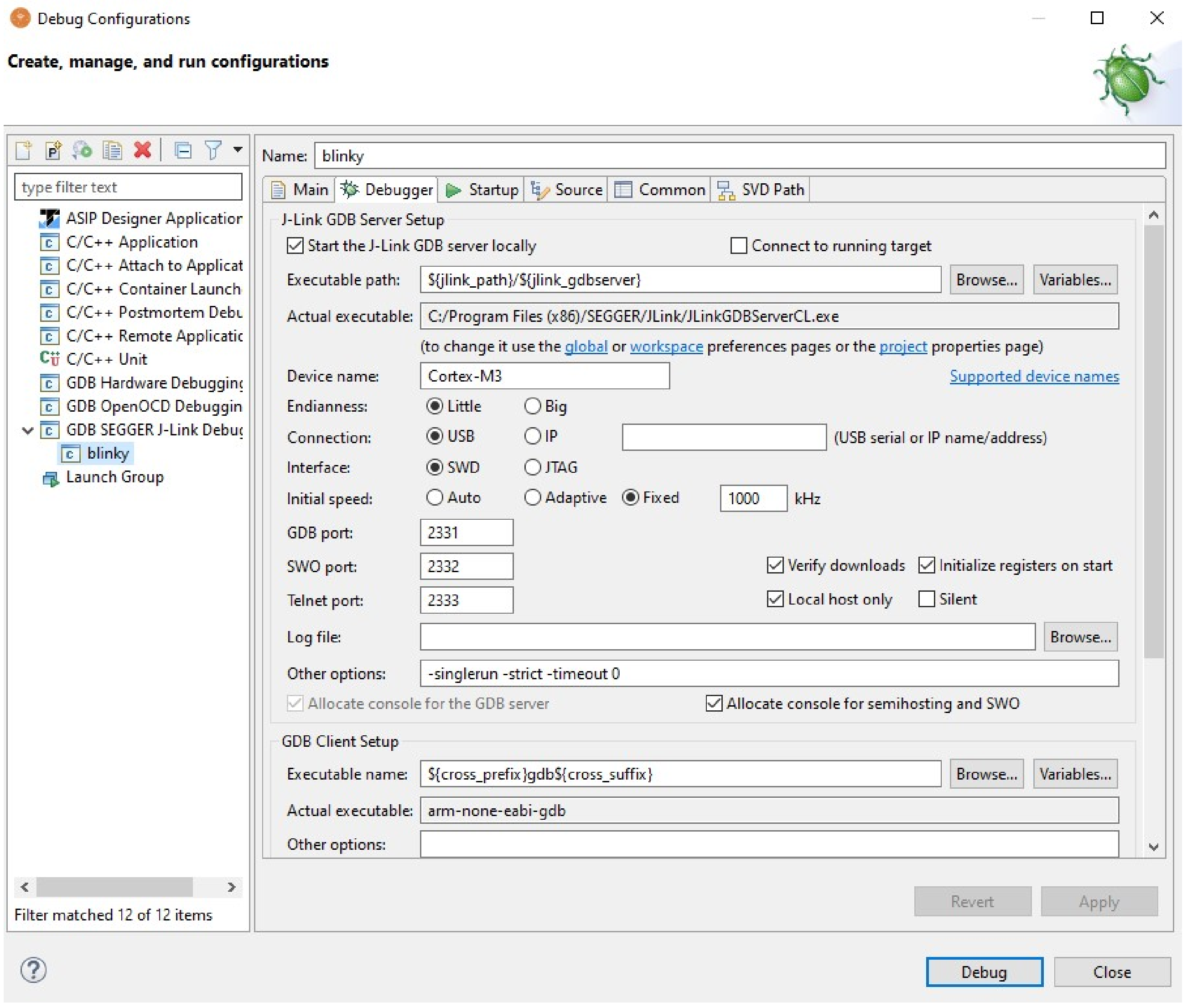
Build Configuration: Select Automatically

Enable auto build | Disable auto build

Use workspace settings (selected) | Configure Workspace Settings...

Figure 22. Selecting Project and Application Type

b. In the **Debugger** tab view, enter **Cortex-M3** in the Device Name window, and make certain that the **Other Options** field does not include the `-nogui` option, as shown in figure "Device Name Window" (Figure 23).



Name: blinky

J-Link GDB Server Setup

[✓] Start the J-Link GDB server locally [ ] Connect to running target

Executable path: ${jlink_path}/${jlink_gdbserver}

Actual executable: C:/Program Files (x86)/SEGGER/JLink/JLinkGDBServerCL.exe

(to change it use the global or workspace preferences pages or the project properties page)

Device name: Cortex-M3

Endianness: [✓ Little] Big

Connection: [✓ USB] IP (USB serial or IP name/address)

Interface: [✓ SWD] JTAG

Initial speed: Auto Adaptive [✓ Fixed] 1000 kHz

GDB port: 2331

SWO port: 2332 [✓] Verify downloads [✓] Initialize registers on start

Telnet port: 2333 [✓] Local host only [ ] Silent

Log file:

Other options: -singlerun -strict -timeout 0

[✓] Allocate console for the GDB server [✓] Allocate console for semihosting and SWO

GDB Client Setup

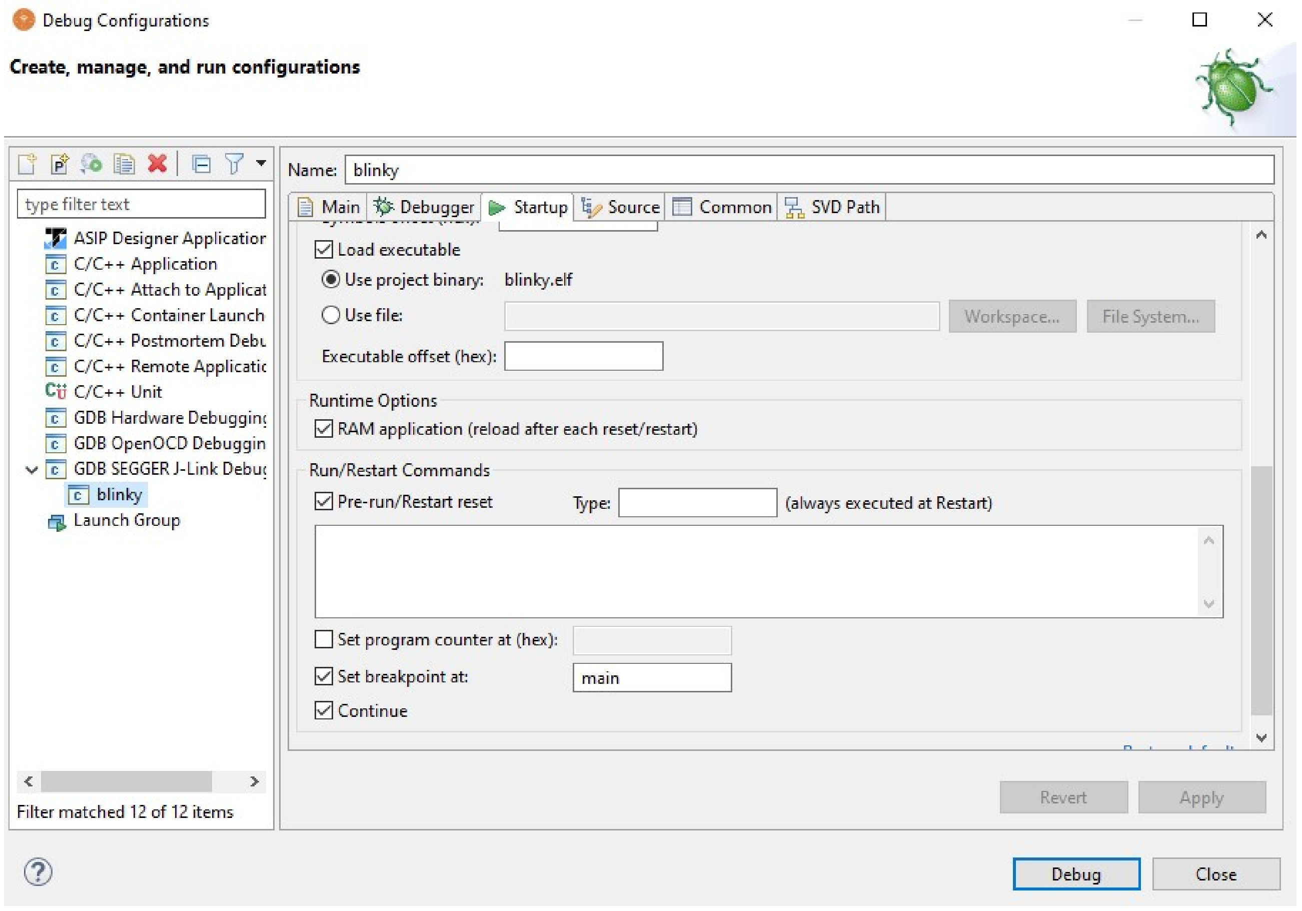
Executable name: ${cross_prefix}gdb${cross_suffix}

Actual executable: arm-none-eabi-gdb

Other options:

Figure 23. Device Name Window

c. In the **Startup** tab view under **Runtime Options**, tick the box labeled **RAM application (reload after each reset/restart)**, as seen in figure "Selecting Runtime Options" (Figure 24).



Name: blinky

[✓] Load executable
(●) Use project binary: blinky.elf
( ) Use file:
Executable offset (hex):

Runtime Options
[✓] RAM application (reload after each reset/restart)

Run/Restart Commands
[✓] Pre-run/Restart reset Type: (always executed at Restart)
[ ] Set program counter at (hex):
[✓] Set breakpoint at: main
[✓] Continue

Figure 24. Selecting Runtime Options

All other information in the debug configuration can remain set to defaults.

1. Click on **Apply** and then on **Debug**. The application begins loading to the device. If this is successful, a **J-Link GDB Debug** session opens.
2. You are now able to step through the code, look at registers and memory views, and more.

CHAPTER 7

More Information

Searchable HTML and PDF versions of the Ezairo 8300 user documentation, and PDF versions of manuals from other sources, are included with your Ezairo 8300 installation. This topic explains what they relate to, and where to find them.

7.1 Where to Find Documentation

7.1.1 Finding HTML Documentation

Fully-searchable HTML versions of most Ezairo 8300 user documentation can be found with this shortcut: in *C:/Program Files (x86)/ON Semiconductor\Ezairo 8300 SDK\documentation\*, double-click on *searchable Ezairo 8300 documents.htm* to start the HTML display in your default browser.

7.1.2 Finding PDF Documentation

The tables in Section 7.2 “Documentation Roadmap” on page 45 include locations of the PDF manuals that are available in the *\documentation* folder of this release. In the tables, the placeholder **`<sdk_home>`** represents the folder where you have installed the SDK.

7.1.3 Finding the Information You Need

A documentation index by subject for both PDF and HTML, including locations, is found in Section 7.2 “Documentation Roadmap” on page 45. To help you find the information you are looking for, the topics are organized as follows:

- "Overview Documentation" on the next page
- "CFX Documentation" on page 46
- "Arm Cortex-M3 Processor Documentation" on page 50
- "LPDSP32 Documentation" on page 52
- "HEAR Documentation" on page 55
- "Filter Engine Documentation" on page 56
- "Neural Network Accelerator Documentation" on page 56
- "Documentation for Other Hardware Elements" on page 57
- "NVM Support Documentation" on page 57
- "Communications Support Documentation" on page 57

7.1.4 Publicly-Available Documentation

For even more information, consult these publicly-available documents (not included with your Ezairo 8300 download):

Ezairo 8300:

More Ezairo 8300 information can be downloaded from the onsemi website, at https://www.onsemi.com/products/audio-video-assp/audiology-dsp-systems/ezairo-8300.

Arm Cortex-M3 processor:

- *Cortex-M3 Technical Reference Manual*, *revision r2p1*
- *ArmCortex Microcontroller Software Interface Standard (CMSIS)*
- Yiu, Joseph *The Definitive Guide to the Arm Cortex-M3* (Newnes, December 2009) or a similar book describing the Arm Cortex-M3 core architecture

NOTE: You can find Arm publications on their website at http://infocenter.arm.com/help/index.jsp.

Digital signal processing:

- Crochiere, Ronald E. and Rabiner, Lawrence R., *Multirate Digital Signal Processing*. (Prentice-Hall Signal Processing Series, 1983)
- Any university-level introductory DSP textbook

Bluetooth®:

- *Specification of the Bluetooth System*, found at https://www.bluetooth.org/en-us/specification/adopted-specifications

G.722 encoder/decoder:

- *ITU-T G.722* (09/2012)

Google® TensorFlow:

- Google's TensorFlow Lite Documentation and Pete Warden's GitHub repository sample code:
    - https://www.tensorflow.org/lite
    - https://github.com/tensorflow/tensorflow/blob/master/tensorflow/lite/micro/examples/hello_world/train/train_hello_world_model.ipynb
    - https://www.tensorflow.org/lite/microcontrollershttps://www.tensorflow.org/lite/microcontrollers
- For understanding the TensorFLow Lite C++ Library:
    - https://www.tensorflow.org/lite/microcontrollers/library
- For understanding how the C++ library can be build into your own IDE:
    - https://www.digikey.com/en/maker/projects/tinyml-getting-started-with-tensorflow-lite-for-microcontrollers/c0cdd850f5004b098d263400aa294023

7.2 Documentation Roadmap

This section points you to different kinds of documentation, making it easier to get the answers you need.

7.2.1 Overview Documentation

Sources of documentation that provide overviews of Ezairo 8300 from various perspectives are found in the table "Where to Find Overview Documentation" (Table 3).

Table 3. Where to Find Overview Documentation

Title / Reference	Description	Installed Location	Format
Ezairo 8300 Evaluation and Development Board User's Guide	This document describes the configuration and use of the Ezairo 8300 Evaluation and Development Board.	**<sdk_home>**\documentation	PDF
Ezairo® 8300 Software Development Kit Getting Started Guide	This group of topics provide installation instructions, defines prerequisite hardware and software, and walks you through the steps of building and running a sample code application so that you can confirm that the SDK and evaluation and development board are functioning as expected.	Downloaded alongside the installer, and also found in **<sdk_home>**\documentation	HTML, PDF
Integrated Development Environment User's Guide	This group of topics describes how to use the tools provided by the IDE: an advanced editor, an integrated source code builder, and an integrated debugger. The group of topics includes information about the ASIP programming tool, and plans for updating this document include adding information about the Arm Cortex-M3 processor toolchain.	**<sdk_home>**\documentation	HTML, PDF
Ezairo 8300 for Users of Other Ezairo Products	This group of topics describes the design and architecture of Ezairo 8300 to assist people with programming the chip, and it provides information about porting code from Ezairo 7100 to Ezairo 8300.	**<sdk_home>**\documentation	HTML, PDF
Datasheet for Ezairo 8300	Technical overview information for the Ezairo 8300.	https://www.onsemi.com/products/audio-video-assp/audiology-dsp-systems/ezairo-8300	HTML

7.2.2 CFX Documentation

Sources of documentation related to the CFX DSP are show in the table "Where to Find CFX Documentation" (Table 4).

Table 4. Where to Find CFX Documentation

Title / Reference	Description	Installed Location	Format
CFX DSP Architecture Manual for Ezairo 8300	This group of topics provides an architecture description and instruction set reference for the CFX for Ezairo 8300 DSP. The information in this group of topics provides the software programming basis to help you develop complex algorithms to process, filter and enhance digital signals in the most efficient and compact way. It contains explanatory information for those who are new to the CFX, and it contains reference material for experienced users. Once you are familiar with the functional features described in the *Ezairo 8300 Hardware Reference*, you can use the assembly language documented in this group of topics to develop algorithms. Further information on DSP software support libraries, such as building block libraries of macros to make algorithm development faster, are described in the *Ezairo 8300 Firmware Reference*.	`<sdk_home>`*\documentation*	HTML, PDF
CFX DSP Instruction Set Quick Reference	A companion document for the *CFX DSP Architecture Manual for Ezairo 8300*, this document serves as a concise programming aid. It is designed to remind users of the assembly language syntax.	`<sdk_home>`*\documentation*	HTML, PDF
Ezairo 8300 Firmware Reference especially the following sections: • Hardware Definitions (chapter 3) • CFX System Library (chapter 5) • Program ROM (chapter 9) • NVMLIB Library (chapter 10) • NVM Memory Layout (chapter 11) • CFX AES Library (chapter 12) • Calibration Library (chapter 13) • CFX DSP Library (chapter 15)	The firmware provides basic system functionality and isolates you from the hardware, making it easier to support and maintain your code. This group of topics describes the libraries and other firmware features that assist with the development of your applications. The referenced section describes the use of the CFX AES library.	`<sdk_home>`*\documentation*	HTML, PDF
Ezairo 8300 Hardware Reference especially the following sections: • CFX Digital Signal Processor (chapter 3) • Debug Ports and Security (chapter 17)	This group of topics describes all the functional features available on an Ezairo 8300 chip, how they are used, and how they are configured. This group of topics is a good place to start when you are designing real-time implementations of your algorithms or planning a product based on the Ezairo 8300 chip.	`<sdk_home>`*\documentation*	PDF, HTML
Integrated Development Environment User's Guide	This group of topics describes how to use the tools provided by the IDE: an advanced editor, an integrated source code builder, and an integrated debugger. The group of topics includes information about the ASIP programming tool, and plans for updating this document include adding information about the Arm Cortex-M3 processor toolchain.	`<sdk_home>`*\documentation*	HTML, PDF

Table 4. Where to Find CFX Documentation (Continued)

Title / Reference	Description	Installed Location	Format
Communication Protocols for Ezairo 8300 especially the following sections: • Basic Protocol Information (in chapter 4) • CFX Debug Port Protocol Commands and Responses (in chapter 4) • CFX I2C Debug Port Bridge (in chapter 5)	This group of topics describes how to use the built-in protocol on the chips based on both the CFX for Ezairo 8300 DSP core and the Arm Cortex-M3 core so that a system other than the SDK can communicate with them. This group of topics is primarily intended for use in situations where the user wants to interface with another device and cannot use the Communication Toolkit support software..	**<sdk_ home>***\documentation*	PDF, HTML
Ezairo® 8300 Software Development Kit Getting Started Guide especially the following sections: • CFX Assembler Sample Application (jn chapter 4) • CFX C Sample Application (in chapter 4)	This group of topics provide installation instructions, defines prerequisite hardware and software, and walks you through the steps of building and running a sample code application so that you can confirm that the SDK and evaluation and development board are functioning as expected.	Downloaded alongside the installer	PDF, HTML
Ezairo 8300 for Users of Other Ezairo Products	This group of topics describes the design and architecture of Ezairo 8300 to assist people with programming the chip, and it provides information about porting code from Ezairo 7100 to Ezairo 8300.	**<sdk_ home>***\documentation*	PDF, HTML
CoolFlux DSP 24C a20: Assembly Programmer's Manual	This document provides a reference for the assembly language of the CoolFlux DSP 24C a20 DSP core, and examines the core's architecture. It is intended for programmers who need to analyze the assembly code generated from C source code.	<sdk_home>*\cf624c-a20_* **<version>** *\doc\manuals*	PDF
CoolFlux DSP 24C a20: C Programmer's Manual	This document explains how to create audio applications in C for the CoolFlux DSP 24C a20 DSP core, and how to use the core's accompanying tool suite. It is intended for C programmers who have good general knowledge of DSP algorithms and processors.	<sdk_home>*\cf624c-a20_* **<version>** *\doc\manuals*	PDF
ASIP Programmer Overview of the Manuals	This overview summarizes all the documentation included in the ASIP programmer distribution.	**<sdk_home>***\ASIP Programmer* **<version>** *\win64\doc\manuals*	PDF
Bridge Linker User Manual	Explains the use of the Bridge linker, which is part of the ASIP Programmer. The linker creates a statically linked Elf executable starting from one or more relocatable Elf object files and archives. This manual discusses the command line arguments of the linker and archiver, the linker optimizations, and the linker configuration file.	**<sdk_home>***\ASIP Programmer* **<version>** *\win64\doc\manuals*	PDF

Table 4. Where to Find CFX Documentation (Continued)

Title / Reference	Description	Installed Location	Format
Checkers API Reference Documentation	The API reference documentation describes the functions and data structures available for interacting with ISSs.	**`<sdk_home>`***\ASIP Programmer\* **`<version>`** *\win64\doc\manuals*	PDF
Checkers ISS Interface Manual	This manual describes the different interface possibilities of an ISS. This includes memory interfaces, simulation modes, SystemC interface, and different levels of APIs.	**`<sdk_home>`***\ASIP Programmer\* **`<version>`** *\win64\doc\manuals*	PDF
Checkers Simulator Manual	This manual describes the use of a simulator or debug client. It covers loading of programs, setting breakpoints and watchpoints, profiling, and related topics. This manual also describes the configuration options to build a simulator or debug client. Both the use and building process are fully integrated in CHESSDE.	**`<sdk_home>`***\ASIP Programmer\* **`<version>`** *\win64\doc\manuals*	PDF
Chess Compiler User Manual	Chapter 1 describes the command-line interface to the CHESS compiler. Chapter 3 covers the CHESS C language extensions (compiler directives). Chapter 4 discusses the various compilation settings.	**`<sdk_home>`***\ASIP Programmer\* **`<version>`** *\win64\doc\manuals*	PDF
ChessDE User Manual	This manual describes the CHESSDE development environment, a graphical user interface to the compiler and simulator/debugger tools.	**`<sdk_home>`***\ASIP Programmer\* **`<version>`** *\win64\doc\manuals*	PDF
ChessMP User Manual	This manual explains the use of CHESSMP, a multi-processor simulation and debug environment.	**`<sdk_home>`***\ASIP Programmer\* **`<version>`** *\win64\doc\manuals*	PDF
Darts Assembler User Manual	This manual covers the command-line interface, the syntax of text and data sections, and the use of expressions in assembly source code.	**`<sdk_home>`***\ASIP Programmer\* **`<version>`** *\win64\doc\manuals*	PDF
Eclipse User Manual	This manual describes how to use the Eclipse IDE to build and debug applications in the context of ASIP Programmer.	**`<sdk_home>`***\ASIP Programmer\* **`<version>`** *\win64\doc\manuals*	PDF
Free and Open-Source Licensing Notices	This document lists the copyright notices and license conditions of the FOSS packages in the distribution.	**`<sdk_home>`***\ASIP Programmer\* **`<version>`** *\win64\doc\manuals*	PDF
GDB User Manual	This manual describes how to set up a GDB debugging session in the context of ASIP Programmer.	**`<sdk_home>`***\ASIP Programmer\* **`<version>`** *\win64\doc\manuals*	PDF

Table 4. Where to Find CFX Documentation (Continued)

Title / Reference	Description	Installed Location	Format
GDB User Manual	This manual describes how to set up a GDB debugging session in the context of ASIP Programmer.	**`<sdk_home>`***\ASIP Programmer\* **`<version>`** *\win64\doc\manuals*	PDF
Installation Manual	This manual describes the downloading and the installation of ASIP Programmer, including licensing and user setup (Chapter 3).	**`<sdk_home>`***\ASIP Programmer\* **`<version>`** *\win64\doc\manuals*	PDF
JTalk User Manual	This document describes using JTalk for chip debugging. JTalk interfaces one or more debug clients to the target hardware and is responsible for driving the cable that connects the PC to the hardware target. Several cables are supported.	**`<sdk_home>`***\ASIP Programmer\* **`<version>`** *\win64\doc\manuals*	PDF
ASIP Designer - ASIP Programmer Nmlview Manual	This manual describes using the interactive tool NMLView for analyzing the description of a processor in nML language, viewing and printing the resulting instruction tables, and producing related documentation, which is generated in HTML format.	**`<sdk_home>`***\ASIP Programmer\* **`<version>`** *\win64\doc\manuals*	PDF

7.2.3 Arm Cortex-M3 Processor Documentation

The table "Where to Find Arm Cortex-M3 Processor Documentation" (Table 5) shows where you can find information about the Arm Cortex-M3 processor that is part of the Ezairo 8300 system.

Table 5. Where to Find Arm Cortex-M3 Processor Documentation

Title / Reference	Description	Installed Location	Format
Arm® and Thumb®-2 Instruction Set Quick Reference Card	This quick reference card from Arm provides a short-hand list of instructions for the Arm Cortex-M3 processor.	**<sdk_ home>***documentation*	PDF
Ezairo 8300 Firmware Reference especially the following sections: • Hardware Definitions • CMSIS Library (chapter 6) • CMSIS Drivers (chapter 7) • External Drivers (chapter 8) • NVMLIB Library (chapter 10) • NVM Memory Layout (chapter 11) • Calibration Library (chapter 13) • Arm Cortex-M3 Bootloader (chapter 14) • swmTrace Library (chapter 16)	The firmware provides basic system functionality and isolates you from the hardware, making it easier to support and maintain your code. This group of topics describes the libraries and other firmware features that assist with the development of your applications.	**<sdk_ home>***documentation*	HTML, PDF
Ezairo 8300 Hardware Reference especially the following sections: • Arm Cortex-M3 Processor (chapter 4) • Debug Ports and Security (chapter 17)	This group of topics describes all the functional features available on an Ezairo 8300 chip, how they are used, and how they are configured. This group of topics is a good place to start when you are designing real-time implementations of your algorithms or planning a product based on the Ezairo 8300 chip.	**<sdk_ home>***documentation*	HTML, PDF
Communication Protocols for Ezairo 8300 especially the following sections: • Basic Protocol Information (in chapter 4) • Arm Cortex-M3 Core Debug Port Protocol Commands and Responses (in chapter 4) • Arm Cortex-M3 Debug Port Protocol Quick Reference (in appendix C)	This group of topics describes how to use the built-in protocol on the chips based on both the CFX for Ezairo 8300 DSP core and the Arm Cortex-M3 core so that a system other than the SDK can communicate with them. This group of topics is primarily intended for use in situations where the user wants to interface with another device and cannot use the Communication Toolkit support software.	**<sdk_ home>***documentation*	HTML, PDF

Table 5. Where to Find Arm Cortex-M3 Processor Documentation (Continued)

Title / Reference	Description	Installed Location	Format
Ezairo® 8300 Software Development Kit Getting Started Guide especially the following sections: • Arm Cortex-M3 Processor Sample Application (in chapter 4)	This group of topics provide installation instructions, defines prerequisite hardware and software, and walks you through the steps of building and running a sample code application so that you can confirm that the SDK and evaluation and development board are functioning as expected.	Downloaded alongside the installer	HTML, PDF
Integrated Development Environment User's Guide especially the following sections: • Debugging in the Integrated Development Environment (chapter 3) • Using the Arm Cortex-M3 Toolchain (chapter 5) • Setting Up the Arm Cortex-M3 Core Side (in chapter 6)	This group of topics describes how to use the tools provided by the IDE: an advanced editor, an integrated source code builder, and an integrated debugger. The group of topics includes information about the ASIP programming tool, and plans for updating this document include adding information about the Arm Cortex-M3 processor toolchain.	**<sdk_ home>**\documentation	HTML, PDF
Ezairo 8300 for Users of Other Ezairo Products	This group of topics describes the design and architecture of Ezairo 8300 to assist people with programming the chip, and it provides information about porting code from Ezairo 7100 to Ezairo 8300.	**<sdk_ home>**\documentation	HTML, PDF

7.2.4 LPDSP32 Documentation

You can find sources of documentation on the LPDSP32 DSP in the table "Where to Find LPDSP32 Documentation" (Table 6).

Table 6. Where to Find LPDSP32 Documentation

Title / Reference	Description	Installed Location	Format
Ezairo 8300 Codec Framework	This document provides an overview of the techniques involved when implementing and integrating code for the audio codecs on the LPDSP32 processor.	**<sdk_ home>**\documentation	HTML, PDF
Ezairo 8300 Firmware Reference especially the following sections: • Hardware Definitions (chapter 3)	The firmware provides basic system functionality and isolates you from the hardware, making it easier to support and maintain your code. This group of topics describes the libraries and other firmware features that assist with the development of your applications.	**<sdk_ home>**\documentation	HTML, PDF
Ezairo 8300 Hardware Reference especially the following sections: • LPDSP32 Processor (chapter 8)	This group of topics describes all the functional features available on an Ezairo 8300 chip, how they are used, and how they are configured. This group of topics is a good place to start when you are designing real-time implementations of your algorithms or planning a product based on the Ezairo 8300 chip.	**<sdk_ home>**\documentation	HTML, PDF

Table 6. Where to Find LPDSP32 Documentation (Continued)

Title / Reference	Description	Installed Location	Format
Integrated Development Environment User’s Guide	This group of topics describes how to use the tools provided by the IDE: an advanced editor, an integrated source code builder, and an integrated debugger. The group of topics includes information about the ASIP programming tool, and plans for updating this document include adding information about the Arm Cortex-M3 processor toolchain.	**<sdk_ home>**\documentation	HTML, PDF
Ezairo® 8300 Software Development Kit Getting Started Guide	This group of topics provide installation instructions, defines prerequisite hardware and software, and walks you through the steps of building and running a sample code application so that you can confirm that the SDK and evaluation and development board are functioning as expected.	Downloaded alongside the installer	HTML, PDF
Ezairo 8300 for Users of Other Ezairo Products	This group of topics describes the design and architecture of Ezairo 8300 to assist people with programming the chip, and it provides information about porting code from Ezairo 7100 to Ezairo 8300.	**<sdk_ home>**\documentation	HTML, PDF
LPDSP32-V3 Block Diagram	A diagram for the LPDSP32 core that provides information on its inputs, outputs, components and process blocks.	**<sdk_ home>** \documentation\lpdsp32_ 3rd_party_docs	PDF
LPDSP32-V3 Hardware Reference Manual	Describes the hardware aspects of the LPDSP32-V3 core and its operations to provide an understanding of the core architecture and various kinds of supported operations.	**<sdk_ home>** \documentation\lpdsp32_ 3rd_party_docs	PDF
LPDSP32-V3 Interrupt Support Manual	Describes how the LPDSP32 core's interrupts are supported.	**<sdk_ home>** \documentation\lpdsp32_ 3rd_party_docs	PDF
User IP Programmers Guide for LPDSP32-V3	Describes the C application layer used for the LPDSP32 core, the flow generally followed when any application is ported to LPDSP32, various tips for optimization to make the best use of the processor and compiler resources, and certain things the programmers should be aware of when porting applications. It also provides a few examples to show the usage of LPDSP32 intrinsic functions and to give an idea of how certain DSP functions can be ported to and optimized for LPDSP32.	**<sdk_ home>** \documentation\lpdsp32_ 3rd_party_docs	PDF
LPDSP32-V3 Assembly Programmer’s Manual	A description of the LPDSP32 core's assembly syntax.	**<sdk_home>**\lpdsp32-v3_ **<version>**\doc\manuals	PDF

Table 6. Where to Find LPDSP32 Documentation (Continued)

Title / Reference	Description	Installed Location	Format
User Guide IP Programmer for LPDSP32-V3	A user guide for the IP Programming, using the suite of LPDSP32 tools.	**<sdk_home>***\lpdsp32-v3_***<version>***\doc\manuals*	PDF
Checkers ISS Interface Manual	This manual describes the different interface possibilities of an ISS. This includes memory interfaces, simulation modes, SystemC interface, and different levels of APIs.	**<sdk_home>***\ASIP Programmer***<version>***\win64\doc\manuals*	PDF
Checkers Simulator Manual	This manual describes the use of a simulator or debug client. It covers loading of programs, setting breakpoints and watchpoints, profiling, and related topics. This manual also describes the configuration options to build a simulator or debug client. Both the use and building process are fully integrated in CHESSDE.	**<sdk_home>***\ASIP Programmer***<version>***\win64\doc\manuals*	PDF
Chess Compiler User Manual	Chapter 1 describes the command-line interface to the CHESS compiler. Chapter 3 covers the CHESS C language extensions (compiler directives). Chapter 4 discusses the various compilation settings.	**<sdk_home>***\ASIP Programmer***<version>***\win64\doc\manuals*	PDF
ChessDE User Manual	This manual describes the CHESSDE development environment, a graphical user interface to the compiler and simulator/debugger tools.	**<sdk_home>***\ASIP Programmer***<version>***\win64\doc\manuals*	PDF
ChessMP User Manual	This manual explains the use of CHESSMP, a multi-processor simulation and debug environment.	**<sdk_home>***\ASIP Programmer***<version>***\win64\doc\manuals*	PDF
Darts Assembler User Manual	This manual covers the command-line interface, the syntax of text and data sections, and the use of expressions in assembly source code.	**<sdk_home>***\ASIP Programmer***<version>***\win64\doc\manuals*	PDF
Eclipse User Manual	This manual describes how to use the Eclipse IDE to build and debug applications in the context of ASIP Programmer.	**<sdk_home>***\ASIP Programmer***<version>***\win64\doc\manuals*	PDF
Free and Open-Source Licensing Notices	This document lists the copyright notices and license conditions of the FOSS packages in the distribution.	**<sdk_home>***\ASIP Programmer***<version>***\win64\doc\manuals*	PDF
GDB User Manual	This manual describes how to set up a GDB debugging session in the context of ASIP Programmer.	**<sdk_home>***\ASIP Programmer***<version>***\win64\doc\manuals*	PDF

Table 6. Where to Find LPDSP32 Documentation (Continued)

Title / Reference	Description	Installed Location	Format
GDB User Manual	This manual describes how to set up a GDB debugging session in the context of ASIP Programmer.	**`<sdk_home>`***\ASIP Programmer\***`<version>`***\win64\doc\manuals*	PDF
Checkers ISS Interface Manual	This manual describes the different interface possibilities of an ISS. This includes memory interfaces, simulation modes, SystemC interface, and different levels of APIs.	**`<sdk_home>`***\ASIP Programmer\***`<version>`***\win64\doc\manuals*	PDF
Installation Manual	This manual describes the downloading and the installation of ASIP Programmer, including licensing and user setup (Chapter 3).	**`<sdk_home>`***\ASIP Programmer\***`<version>`***\win64\doc\manuals*	PDF
JTalk User Manual	This document describes using JTalk for chip debugging. JTalk interfaces one or more debug clients to the target hardware and is responsible for driving the cable that connects the PC to the hardware target. Several cables are supported.	**`<sdk_home>`***\ASIP Programmer\***`<version>`***\win64\doc\manuals*	PDF
ASIP Designer - ASIP Programmer Nmlview Manual	This manual describes using the interactive tool NMLView for analyzing the description of a processor in nML language, viewing and printing the resulting instruction tables, and producing related documentation, which is generated in HTML format.	**`<sdk_home>`***\ASIP Programmer\***`<version>`***\win64\doc\manuals*	PDF

7.2.5 HEAR Documentation

Sources of HEAR Configurable Accelerator documentation are found in the table "Where to Find HEAR Documentation" (Table 7).

Table 7. Where to Find HEAR Documentation

Title / Reference	Description	Installed Location	Format
HEAR Configurable Accelerator Reference	The HEAR Configurable Accelerator is a microcode-configurable signal processing core. The microcode is composed of multiple configurable modules which the application developer can configure and arrange in a manner suitable for their applications. This group of topics provides a reference for using the HEAR Configurable Accelerator and in particular for configuring the microcode which executes on the HEAR as part of a developer's application.	**`<sdk_home>`***\documentation*	HTML, PDF
Ezairo 8300 Hardware Reference especially the following sections: • HEAR Configurable Accelerator (chapter 5)	This group of topics describes all the functional features available on an Ezairo 8300 chip, how they are used, and how they are configured. This group of topics is a good place to start when you are designing real-time implementations of your algorithms or planning a product based on the Ezairo 8300 chip.	**`<sdk_home>`***\documentation*	HTML, PDF

7.2.6 Filter Engine Documentation

The table "Where to Find Filter Engine Documentation" (Table 8) tells how to find information on the Filter Engine.

Table 8. Where to Find Filter Engine Documentation

Title / Reference	Description	Installed Location	Format
Ezairo 8300 Filter Engine Reference	This group of topics provides an architecture description and instruction set reference for the Filter Engine processor. It is intended for readers who want to develop programs written for the Filter Engine. It contains explanatory information for those who are new to the Filter Engine, and reference material for experienced users. To learn how to configure the Filter Engine from the CFX processor, see the *Ezairo 8300 Hardware Reference*. Further information on DSP software support libraries, such as building block libraries of macros to make algorithm development faster, is provided in the *Ezairo 8300 Firmware Reference*.	<sdk_ home>\documentation	HTML, PDF
Filter Engine for Ezairo 8300 Instruction Set Quick Reference	This page provides the syntax of the Filter Engine instructions, a brief definition of the operands, and examples.	<sdk_ home>\documentation	HTML, PDF
Ezairo 8300 Hardware Reference Section 1 “Filter Engine” on page 1	This group of topics describes all the functional features available on an Ezairo 8300 chip, how they are used, and how they are configured. This group of topics is a good place to start when you are designing real-time implementations of your algorithms or planning a product based on the Ezairo 8300 chip.	<sdk_ home>\documentation	HTML, PDF

7.2.7 Neural Network Accelerator Documentation

Sources of documentation on the Neural Network Accelerator are show in the table "Where to Find Neural Network Accelerator Documentation" (Table 9).

Table 9. Where to Find Neural Network Accelerator Documentation

Title / Reference	Description	Installed Location	Format
Deep Learning for Ezairo 8300 Programming Guide	This group of topics provides an introduction to deep learning concepts, and both high-level and low-level instructions for developing deep learning applications with the Ezairo 8300 System-on-Chip. Information is provided regarding processor selection and multiple workflow methodologies, which can be tailored to the user’s individual resources and the project’s desired outcomes.	<sdk_ home>\documentation	HTML, PDF
Ezairo 8300 Hardware Reference especially the following sections: • Neural Network Accelerator (chapter 7)	This group of topics describes all the functional features available on an Ezairo 8300 chip, how they are used, and how they are configured. This group of topics is a good place to start when you are designing real-time implementations of your algorithms or planning a product based on the Ezairo 8300 chip.	<sdk_ home>\documentation	HTML, PDF

7.2.8 Documentation for Other Hardware Elements

Consult the table "Where to Find Documentation for Other Hardware Elements" (Table 10) for sources of information on other hardware elements of Ezairo 8300.

Table 10. Where to Find Documentation for Other Hardware Elements

Title / Reference	Description	Installed Location	Format
Ezairo 8300 Firmware Reference especially the following sections: • Hardware Abstraction Layer (chapter 3)	The firmware provides basic system functionality and isolates you from the hardware, making it easier to support and maintain your code. This group of topics describes the libraries and other firmware features that assist with the development of your applications.	**<sdk_ home>***documentation*	HTML, PDF
Ezairo 8300 Hardware Reference	This group of topics describes all the functional features available on an Ezairo 8300 chip, how they are used, and how they are configured. This group of topics is a good place to start when you are designing real-time implementations of your algorithms or planning a product based on the Ezairo 8300 chip.	**<sdk_ home>***documentation*	HTML, PDF

7.2.9 NVM Support Documentation

Consult the table "Where to Find NVM Support Documentation" (Table 11) to find documentation on Non-Volatile Memory support.

Table 11. Where to Find NVM Support Documentation

Title / Reference	Description	Installed Location	Format
Ezairo 8300 Firmware Reference especially the following sections: • NVMLIB Library (chapter 10) • NVM Memory Layout (chapter 11)	The firmware provides basic system functionality and isolates you from the hardware, making it easier to support and maintain your code. This group of topics describes the libraries and other firmware features that assist with the development of your applications.	**<sdk_ home>***documentation*	HTML, PDF

7.2.10 Communications Support Documentation

Documentation sources for communications support can be found in the table "Where to Find Communications Support Documentation" (Table 12).

Table 12. Where to Find Communications Support Documentation

Title / Reference	Description	Installed Location	Format
Communication Protocols for Ezairo 8300	This group of topics describes how to use the built-in protocol on the chips based on both the CFX for Ezairo 8300 DSP core and the Arm Cortex-M3 core so that a system other than the SDK can communicate with them. This group of topics is primarily intended for use in situations where the user wants to interface with another device and cannot use the Communication Toolkit support software.	**`<sdk_home>`***\documentation*	HTML, PDF
Communication Toolkit API Reference	This manual provides a complete description of the software interface that you use to access the Communication Toolkit (CTK) from your software.	*C:\Program Files (x86)\Common Files\SignaKlara\CTK\documentation\ctk*	HTML, PDF
Communication Toolkit Programmer's Guide	This manual provides the information that you need to develop software that uses the Communication Toolkit to communicate with onsemi chips in the SignaKlara™ technology family.	*C:\Program Files (x86)\Common Files\SignaKlara\CTK\documentation\ctk*	HTML, PDF
Implementing a CTK Communication Module	This specialized programming paper describes how to add new communication modules to the CTK to support new communication interfaces.	*C:\Program Files (x86)\Common Files\SignaKlara\CTK\documentation\ctk*	HTML, PDF
Using the CTK on Windows CE	Information on how to develop WIndows CE applications that use the CTK API to communicate with onsemi DSP systems.	*C:\Program Files (x86)\Common Files\SignaKlara\CTK\documentation\ctk*	PDF
NOAHlink Analog Audio Streaming with Ezairo Products	This specialized programming paper explains how to use the NOAHlink Hearing Aid interface device to stream analog audio over onsemi Ezairo products.	*C:\Program Files (x86)\Common Files\SignaKlara\CTK\documentation\ctk*	PDF
NOAHlink Analog Audio Streaming with Orela 4500	This specialized programming paper explains how to use the NOAHlink Hearing Aid interface device to stream analog audio over onsemi Orela products.	*C:\Program Files (x86)\Common Files\SignaKlara\CTK\documentation\ctk*	PDF
NOAHlink™ Login Enhancements	This specialized programming paper explains how to modify the NOAHlink drivers to support the loading of new protocol firmware without resetting the NOAHlink device.	*C:\Program Files (x86)\Common Files\SignaKlara\CTK\documentation\ctk*	HTML, PDF

PUBLICATION ORDERING INFORMATION

LITERATURE FULFILLMENT:
Literature Distribution Center for onsemi
19521 E. 32nd Pkwy, Aurora, Colorado 80011 USA
Phone: 303-675-2175 or 800-344-3860 Toll Free USA/Canada
Fax: 303-675-2176 or 800-344-3867 Toll Free USA/Canada
Email: orderlit@onsemi.com

N. American Technical Support:
800-282-9855 Toll Free USA/Canada

Europe, Middle East and Africa Technical Support:
Phone: 421 33 790 2910

onsemi Website: www.onsemi.com

Order Literature: http://www.onsemi.com/orderlit

For additional information, please contact your local Sales Representative

M-20865-013