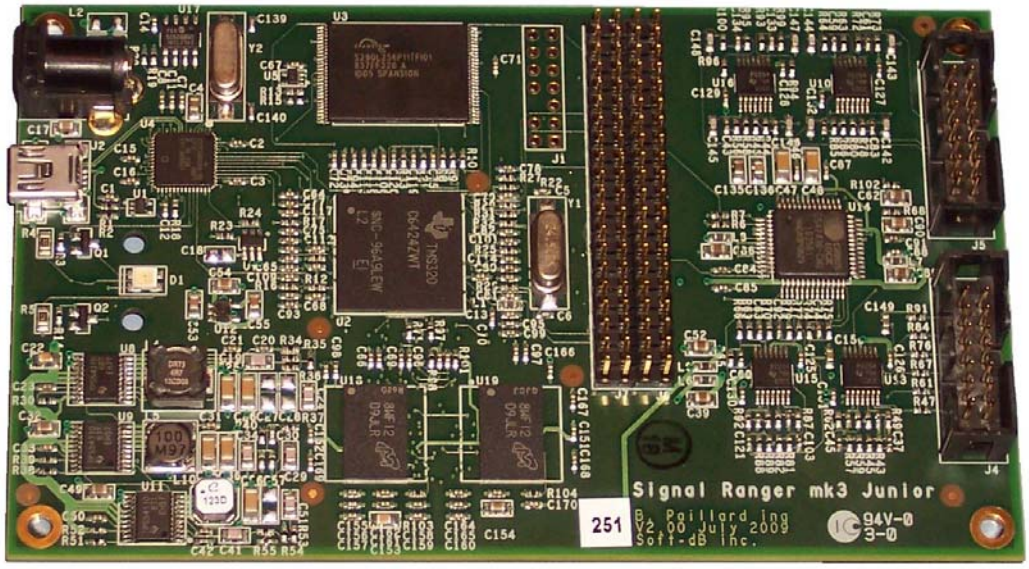
SignalRanger Mk3 Datasheet



SignalRanger Mk3

Hardware Features:

- 6424 DSP from Texas Instruments at 590 MHz (4800 MMACS: Millions Multiply-ACcumulate operations per Second)
- 208 Kbytes of internal memory, 32 Mbytes of Flash memory and 128 Mbytes of external memory DDR2 at 333 MHz
- 6 x 24-bits analog I/Os with an adjustable sampling rate between 4 kHz and 96 kHz (two of the six inputs can accept Electret microphones)
- Input SNR of 101 dB (audio bandwidth) and Output SNR of 104 dB (audio bandwidth)
- USB 2.0 interface (35 Mbit/s)
- Expansion connector: PWMs, GPIOs, McBSP, UARTs and Timers
- Optional expansion board available with Ethernet (RJ45) connector (provided with software that allows remote access to the board through the web)
- Power supply connector (5VDC)
- Size: 12 cm x 8 cm

Software Features:

- USB driver (signed) for Windows XP and Vista 32-bits/64-bits.
- LabView PC communication driver and standard API (DLL)
- DSP driver for the analog I/Os and flash memory
- DSP code written in C for the analog I/Os ready to receive the user signal processing code (IO_Shell)
- PC and DSP example codes for flash memory and analog I/Os management
- Self-test PC application to test the entire hardware on the DSP board
- DDCI Interface (Development to Deployment Code Instrumentation) for an easy and fast application development (see SignalRanger *DDCI* interface description for more information). This software can be used with the USB interface or the optional Ethernet interface.

SignalRanger *DDCI* Interface

The centerpiece of the *SignalRanger* architecture is its *DDCI* interface. *DDCI (Development to Deployment Code Instrumentation)* allows a controlling application running on a PC to communicate with, and control, an embedded device based on a *SignalRanger* board.

The interface in-effect provides real-time visibility and control into the code running in the embedded device. Its usefulness is at two stages in the application life-cycle:

- During development it is used to provide real-time debugging at an application level that is usually not achievable using standard debugging and emulation techniques. In particular reading, writing and code-control functions do not require CPU halt. The interface allows the code to be instrumented in real-time and in the real operating conditions.
- After application deployment, where the same interface is used to support user-control and communications with the embedded device via an application-specific PC application developed for that purpose.

Three essential features of the *DDCI* interface are:

- The same physical USB interface and host libraries and functions are used to support the interface at both stages of the application life-cycle.
- The operation of the *DDCI* interface requires no DSP code addition or adaptation, and only requires minimal CPU time or memory overhead.
- The physical USB interface can be connected and disconnected in-operation, without any disruption of the DSP code running on the *SignalRanger* platform

The main result of using the interface is to condense the two stages of the application life-cycle into a single shorter step. In-effect the application is generally deployed “as is” right after the debugging phase.

Another strong advantage of using the interface is that it provides, with very little effort, much greater real-time visibility into the operation of the embedded code. This greater visibility during development directly translates into more reliable embedded code.

In many applications where it is necessary to run simulations of the signal processing algorithm to validate its operation, the real-time instrumentation provided by *DDCI* make it possible to analyze the high-level behaviour of the signal-processing code with ease, and with real-life conditions and data. Often the simulation step can be bypassed completely, with better results based on real data.

When using the LabVIEW interface, as opposed to the C/C++ interface, a third advantage of the interface is that the extensive LabVIEW libraries are available to add powerful real-time signal-processing, analysis and display capabilities, both at the debugging and at the deployed-application phases.

The functions of the interface can be exercised while the embedded code is running. All these functions support *symbolic access*, where the name of DSP variables and functions can be used to access them, instead of their absolute addresses. The advantage of this feature is that the embedded DSP code can be modified and relinked without having to update the associated user-access application running on a PC. As long as the variable and function names remain the same the access will stay operational across DSP code updates.

The following real-time functions are supported directly by the interface:

- RAM read and write
- Flash read and write
- Peripheral read and write
- Force code execution
- CPU reset
- In-service firmware upgrade management
- Automatic target device recognition and management.

The interface is composed of several parts:

- **Signed Driver for Windows XP and Windows Vista (x86 and x64 platforms):** This driver allows the connection of any number of boards to the PC. The driver is installed as part of the *SignalRanger* installation procedure.
- **Communication Kernel:** This kernel resides in DSP memory, along with the user's application-specific code. It enhances PC to DSP communication.
- **LabVIEW Libraries:** These libraries support wide-ranging communication, programming and control functions with the resident kernel. They can be used to build an application-specific LabVIEW executable to control the embedded *SignalRanger* device.
- **C/C++ libraries:** For developers who prefer to work in a C/C++ environment, we provide libraries in the form of DLLs. These libraries have functionality similar to the LabVIEW libraries.
- **Mini-Debugger:** The Mini-Debugger is a general-purpose interface application that supports programming and real-time symbolic debugging of generic DSP embedded code. It includes features such as real-time graphical data plotting, symbolic read/write access to variables, dynamic execution, Flash programming… etc. At its core, the mini-debugger uses the same interface libraries that a developer uses to design a stand-alone DSP application. This insures a seamless transition from the development/debugging environment to the deployed application.