Debugging 25

25.1 Introduction

Embedded Pentium® processor-based system designers intending to use integration tools to debug their prototype systems can interface to the processor using two methods:

- Insert an emulator probe into the processor socket.
- Include simple logic on their board that implements a debug port connection.

Inserting an emulator probe into the processor socket allows access to all bus signals, but capacitive loading issues may affect high speed operation. In contrast, the debug port connection allows debugger access to the processor's registers and signals without affecting high speed operation. This allows the system to operate at full speed with the debugger attached. Therefore, Intel recommends that all embedded Pentium processor-based system designs include a debug port.

25.2 Two Levels of Support

Two levels of support are defined for the debug port, the second level being a superset of first. The system designer should choose the level of support that is appropriate for the particular system design and implement that level. Samples of each level of implementation are given in "Implementation Examples" on page 25-424.

25.2.1 Level 1 Debug Port (L1)

The Level 1 debug port supports systems with a single processor. L1 uses a 20-pin connector to allow a debugger access to the processor's registers and signals.

25.2.2 Level 2 Debug Port (L2)

L2 extends the 20-pin debug port connector to 30 pins. The extra ten pins include a second set of boundary scan signals as well as additional R/S# and PRDY signals. The additional R/S# and PRDY signals are added to support the a dual-processor configuration. This enables a debugger to provide separate control over the two processors during debug.

Signals on pins 1 through 20 of the L2 debug port are identical to the signals on the L1 debug port.

25.3 Debug Port Connector Descriptions

A debugger can have a 30-pin connector on its probe that supports both levels of the debug port (as described previously, L1 or L2). Two cables can be provided, each cable having a 30-pin connector at one end (to mate with the debugger's probe connector) and the appropriate size connector at the

other end to mate with the debug port in the system under debug. (For example, the L1 debug port Cable can be a 20-conductor cable with a 20-pin connector at one end and a 30-pin connector at the other end, leaving pins 21 to 30 unconnected.)

Intel-recommended connectors for mating with debug port cables are available in either a vertical or right-angle configuration. Use the one that fits best in your design. The connectors are manufactured by AMP Incorporated and are in their AMPMODU System 50 line. Table 25-1 shows the AMP part numbers for the various connectors:

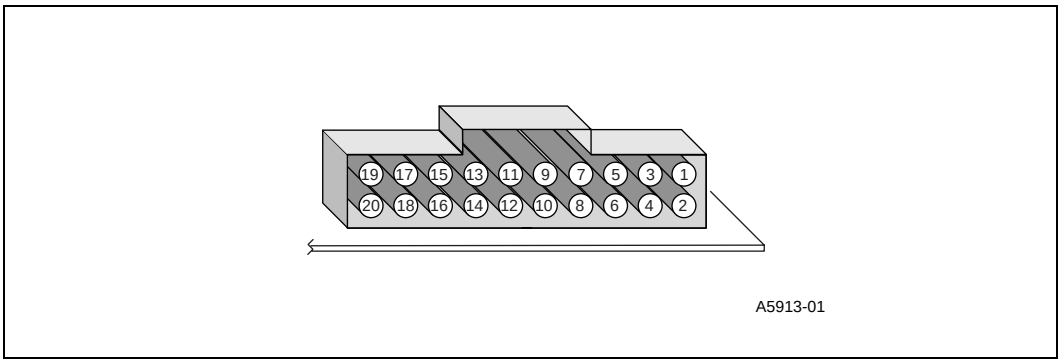
Table 25-1. Recommended Connectors

Connector	Vertical	Right-Angle
20-pin shrouded header	104068-1	104069-1
30-pin shrouded header	104068-3	104069-5

Note: These are high density through hole connectors with pins on 0.050” by 0.100” centers. Do not confuse these with the more common 0.100” by 0.100” center headers.

Figure 25-1 is an example of the pinout of the connector footprint as viewed from the connector side of the circuit board. This is just an example. Contact your third-party tools vendor to determine the correct implementation for the tool you will use. Note that the 30-pin connector is a logical extension of the 20-pin connector with the key aligned with pin 15.

Figure 25-1. Debug Port Connector



25.4 Signal Descriptions

Table 25-2 shows the debug port signals. Direction is given as follows: O = output from the Pentium processor-based board to a debugger; I = input to the Pentium processor-based board from a debugger. These are either 2.5 V or 3.3 V signals, depending on the Pentium processor used in the system. For the L1 debug port, ignore signals on pins 21 through 30.

Note: Target systems should be sure to provide a way for debugging tools like emulators, in-target probes and logic analyzers to reset the entire system, including upgrade processor, chip sets, etc. For example, if you follow the debug port implementation described below, the DBRESET signal provides this functionality. If you are not implementing the debug port, make sure that your system has a test point connected into the system reset logic to which a debug tool can connect.

Table 25-2. Debug Port Signals (Sheet 1 of 2)

Signal Name	Dir	Pin	Description
INIT	O	1	(Pentium® processor signal). A debugger may use INIT to support emulating through the processor INIT sequence while maintaining breakpoints or breaking on INIT.
DBRESET	I	2	Debugger Reset output. A debugger may assert DBRESET (high) while performing the “RESET ALL” and “RESET TARGET” debugger commands. DBRESET should be connected to the system reset circuitry such that the system and processor(s) are reset when DBRESET is asserted. This is useful for recovering from conditions like a “ready hang.” This signal is asynchronous.
RESET	O	3	(Pentium processor signal). A debugger may use RESET to support emulating through the reset while maintaining breaking on RESET.
GND		4	Signal ground.
SMIACT#	0	5	(Pentium processor signal) System Management mode interrupt active.
V_{CC}		6	V_{CC} from the system. A debugger uses this signal to sense that system power is on and to determine signal I/O voltage levels. Connect this signal to V_{CC3} through a 1 KOhm (or smaller) resistor.
R/S#	I	7	Connect to the R/S# pin of the.
GND		8	Signal ground.
NC		9	No connect. Leave this pin unconnected.
GND		10	Signal ground.
PRDY	O	11	From the PRDY pin of the.
TDI	I	12	Boundary scan data input (signal). This signal connects to TDI of the. For dual processor operation, TDI of the Dual would connect to TDO of the.
TDO	O	13	Boundary scan data output (signal). This signal connects to TDO from the for a single processor design, or to TDO from the Dual Pentium for dual processor operation.
TMS	I	14	Boundary scan mode select (signal).
GND		15	Signal ground.
TCK	I	16	Boundary scan clock (signal).
GND		17	Signal ground.
TRST#	I	18	Boundary scan reset (signal).
DBINST#	I	19	DBINST# is asserted (connected to GND) while the debugger is connected to the debug port. DBINST# can be used to control the isolation of signals while the debugger is installed.
BSEN#	I	20	Boundary scan enable. This signal can be used by the system to control multiplexing of the boundary scan input pins (TDI, TMS, TCK, and TRST# signals) between the debugger and other boundary scan circuitry in the system. The debugger asserts (low) BSEN# when it is driving the boundary scan input pins. Otherwise, the debugger drivers are high impedance. If the boundary scan pins are actively driven by the system, then BSEN# should control the system drivers/multiplexers on the boundary scan input pins. See “Example 2: Single Processor, Boundary Scan Used by System” on page 25-426.
PRDY2	O	21	From the PRDY pin of the Dual (for dual processor operation).

Table 25-2. Debug Port Signals (Sheet 2 of 2)

Signal Name	Dir	Pin	Description
GND		22	Signal ground.
R/S#2	I	23	Connect to the R/S# pin of the Dual (for dual processor operation).
NC		24	
NC		25	
NC		26	
NC		27	
NC		28	
GND		29	Signal ground.
NC		30	

25.5 Signal Quality Notes

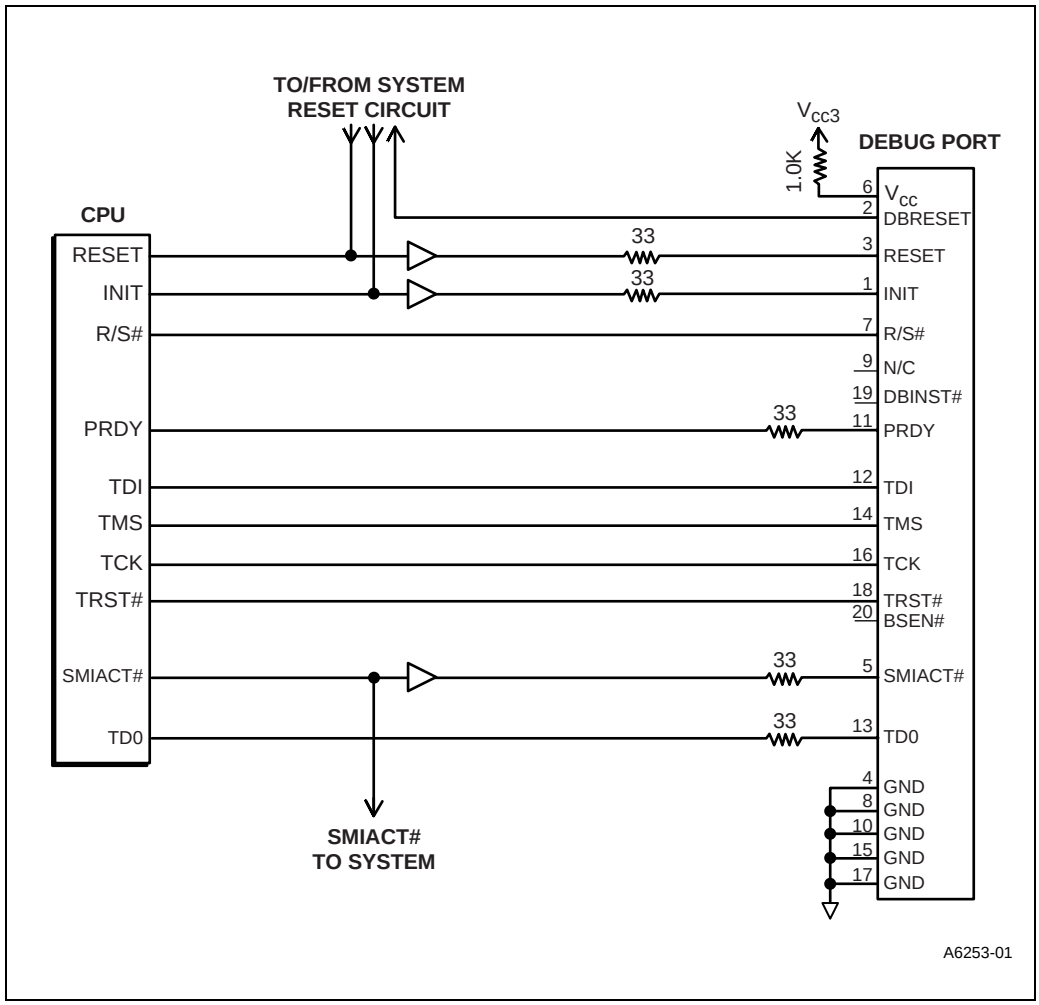
Since debuggers can connect to the system via cables of significant length (e.g., 18 inches), care must be taken in Pentium processor-based system design with regard to the signals going to the debug port. If system outputs to the debug port (i.e., TDO, PRDY, INIT and RESET) are used elsewhere in the system they should have dedicated drivers to the debug port. This will isolate them from the reflections from the end of the debugger cable. Series termination is recommended at the driver output. If the boundary scan signals are used elsewhere in the system, then the TDI, TMS, TCK, and TRST# signals from the debug port should be isolated from the system signals with multiplexers.

25.6 Implementation Examples

25.6.1 Example 1: Single Processor, Boundary Scan Not Used by System

Figure 25-2 shows a schematic of a minimal Level 1 debug port implementation for a Pentium processor, single-processor system in which the boundary scan pins of the are not used in the system.

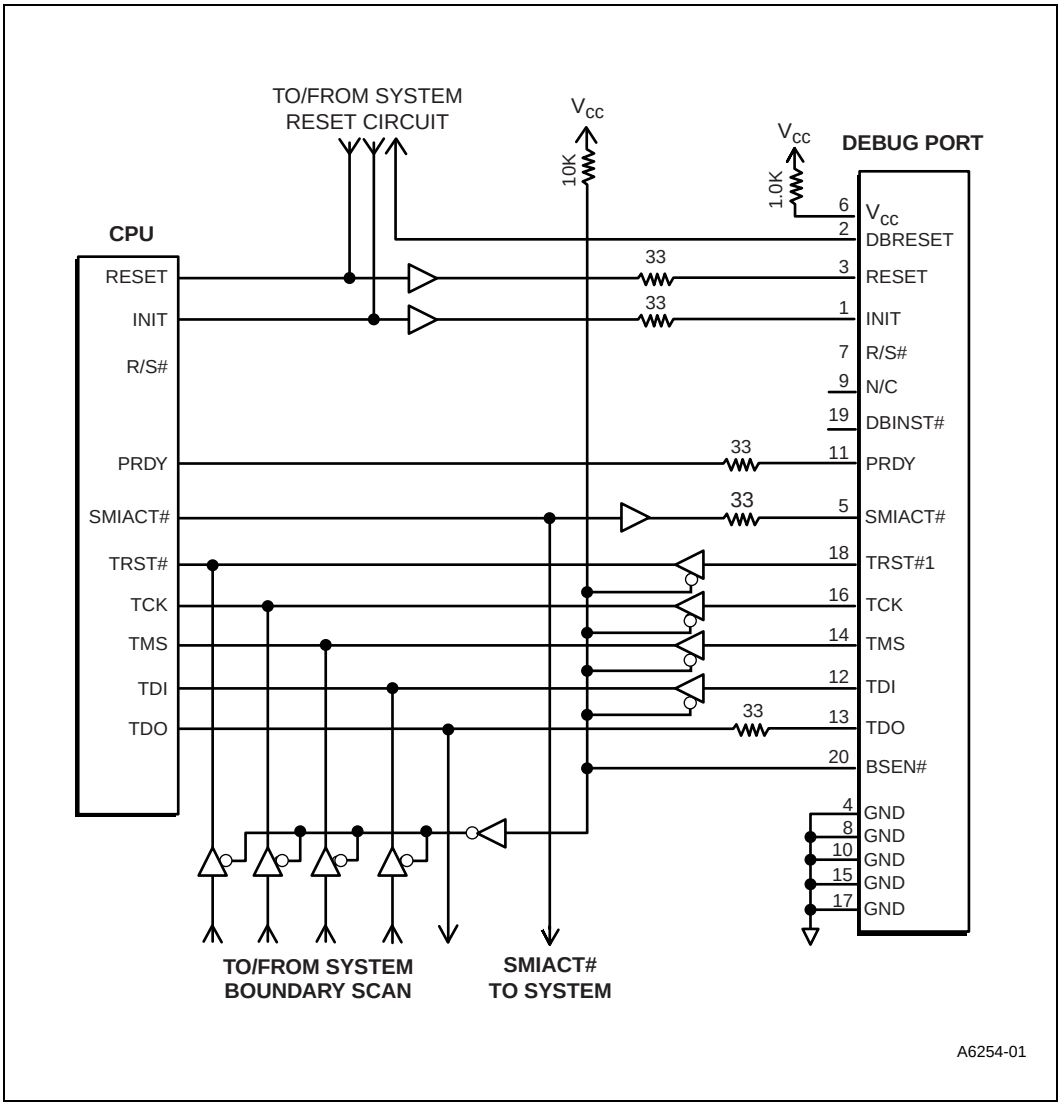
Figure 25-2. Single Processor – Boundary Scan Not Used



25.6.2 Example 2: Single Processor, Boundary Scan Used by System

Figure 25-3 shows a schematic of a Level 1 debug port implementation for a single-processor system in which the boundary scan pins are used. Note that the BSEN# signal controls the multiplexing of the boundary scan signals. With this implementation, the system could use the boundary scan (through the) while the debugger is “emulating,” but could not while the debugger is “halted” (because the chain is broken).

Figure 25-3. Single Processor – Boundary Scan Used

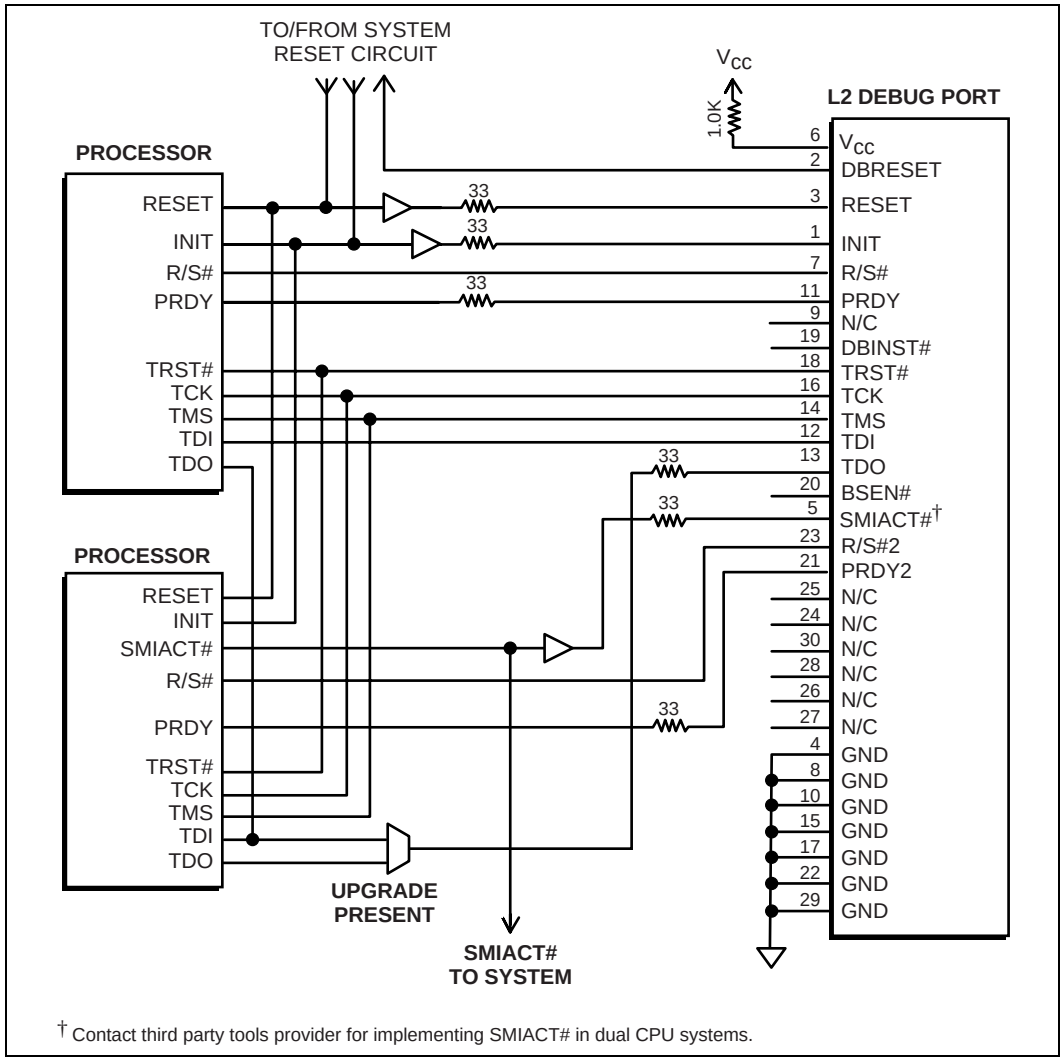


25.6.3 Example 3: Dual Processors, Boundary Scan Not Used by System

Figure 25-4 shows a schematic of a typical Level 2 debug port implementation for a Pentium processor, dual-processor system in which the boundary scan pins are not used. The multiplexer circuit for use with the “upgrade socket” concept is shown, but could be replaced with a jumper.

Note: Contact your third-party tools vendor for information on implementing SMIACT# in a dual processor system.

Figure 25-4. Dual Processor – Boundary Scan Not Used

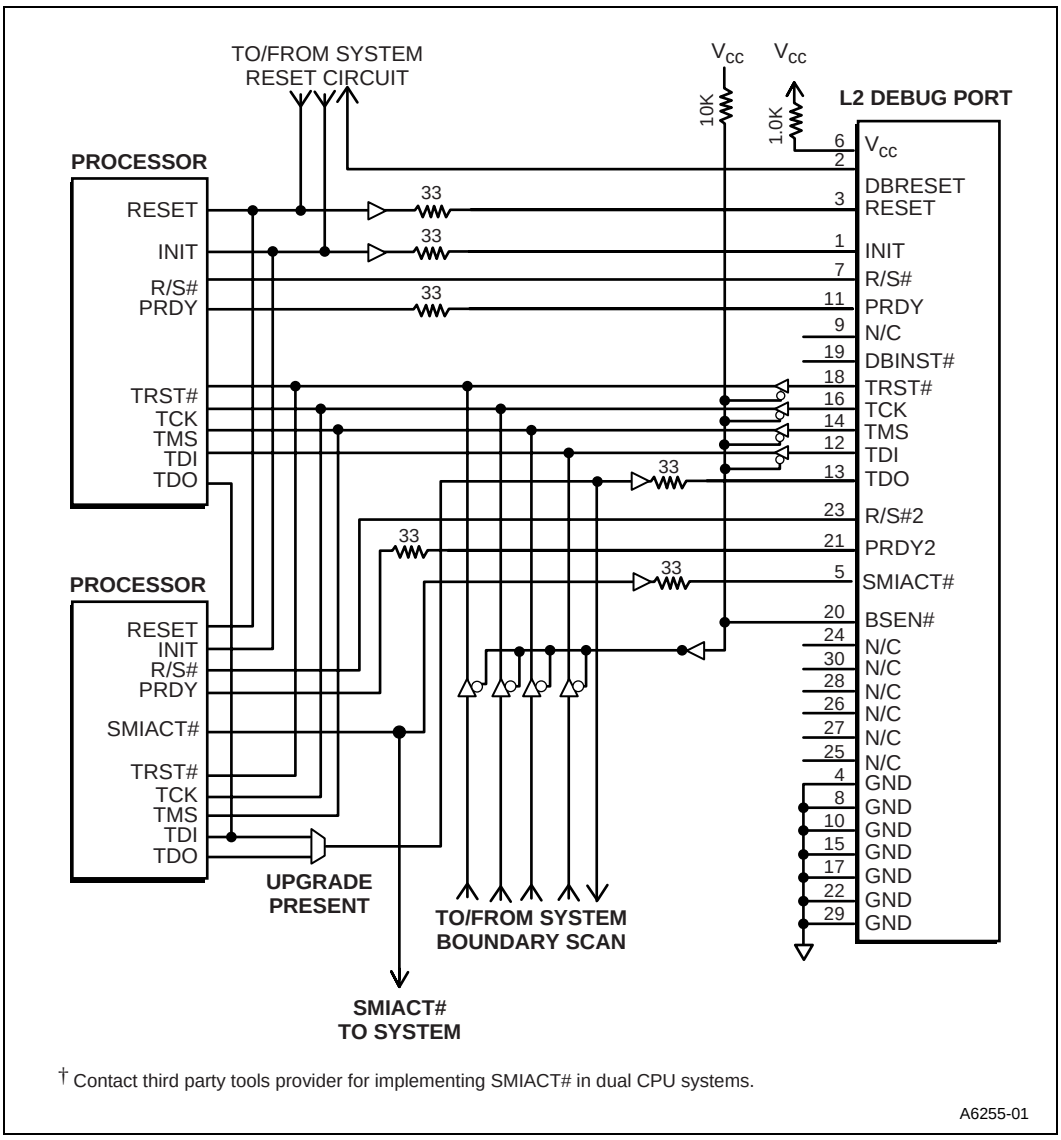


25.6.4 Example 4: Dual Processors, Boundary Scan Used by System

Figure 25-5 shows a schematic of a Level 2 debug port implementation for a dual-processor system that uses boundary scan. Note that the BSEN# signal controls the multiplexing of the boundary scan signals. With this implementation, the system could use the boundary scan (through the) while the debugger is "emulating," but could not while the debugger is "halted" (because the chain is broken).

Note: Contact your third-party tools vendor for information on implementing SMIACT# in a dual processor system.

Figure 25-5. Dual Processor – Boundary Scan Used



25.7 Implementation Details

25.7.1 Signal Routing Note

The debugger software communicates with the processor through the debug port using the boundary scan signals listed above. Typically, the debugger expects the processor to be the first and only component in the scan chain (from the perspective of the debug port). That is, it expects TDI to go directly from the debug port to the TDI pin of the processor, and the TDO pin to go directly from the processor to the debug port (see Figure 25-6). If you have designed your system so that this is not the case (for instance, see Figure 25-7), you will need to provide the debugger software with the following information: (1) position of the processor in the scan chain, (2) the length of the scan chain, (3) instruction register length of each device in the scan chain. Without this information the debugger will not be able to establish communication with the processor.

Figure 25-6. Example of Processor Only in Scan Chain

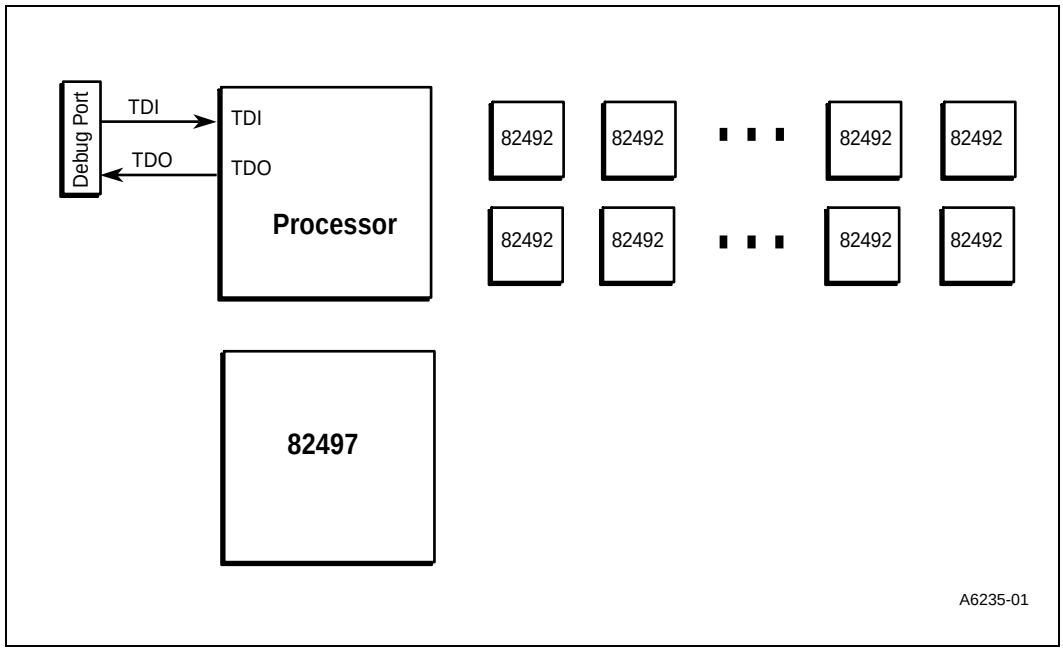
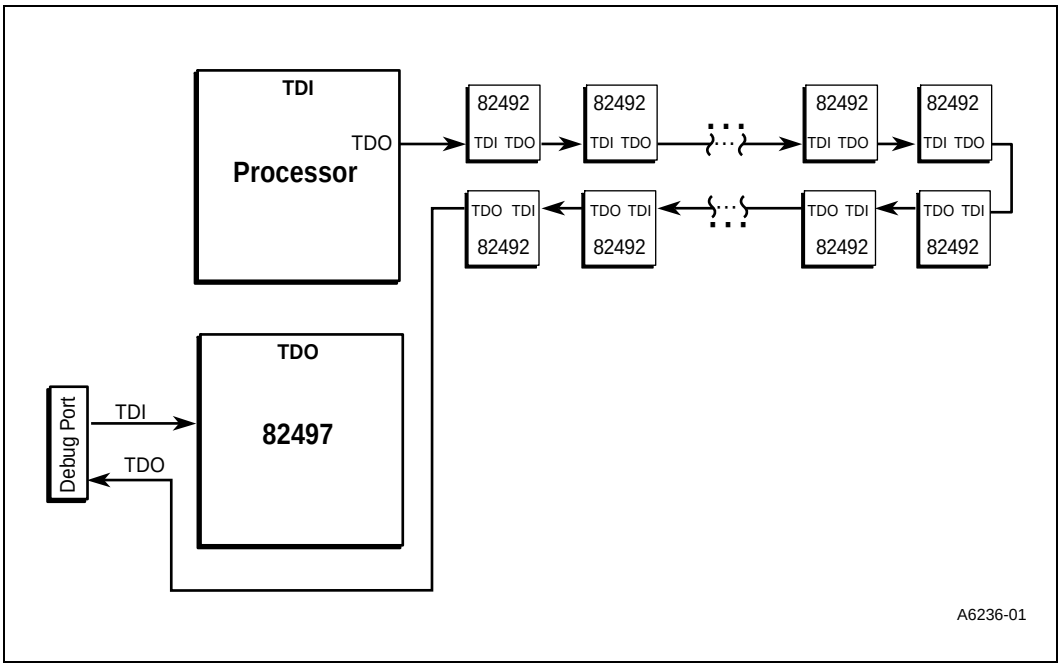


Figure 25-7. Example of Multiple Components in Scan Chain



25.7.2 Special Adapter Descriptions

For those designs where board real estate is a concern or where the design is finished and it is too late to implement the debug port, it may be possible to use a special “debug port adapter” to replace the on-board debug port described in the previous sections. The purpose of the adapter is to provide easy access to the boundary scan signals of the processor(s). For simplicity, the adapter should make the boundary scan signals accessible to the debug tool while at the same time preventing the target system from accessing them. Two debug port adapters are described: (1) for uniprocessor debug, (2) for dual-processor debug.

Standard PPGA adapters are available from many third-party tools vendors.

25.7.2.1 Uniprocessor Debug

A debug port adapter for use in uniprocessor systems, or dual-processor systems where only one processor will be debugged at a time, can be built by reworking two Pentium processor SPGA sockets (see Figure 25-8).

Note: This adapter can be used only when the processor is *not* included in the target system boundary scan string. In addition, when used in dual-processor systems you will only be able to debug the processor to which the adapter is connected.

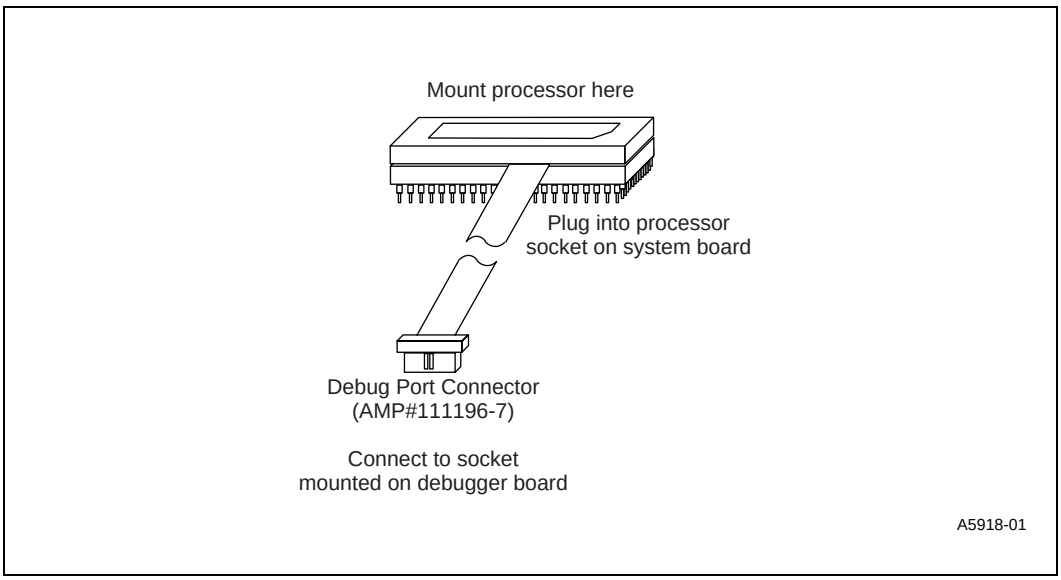
Table 25-3 show which pins to connect lines of appropriate 20- or 30- wire cable to on the top socket.

Table 25-3. SPGA Socket

Cable Wire Number	SPGA Pin Number	Signal
1	AA33	INIT
2	NC	DRESET
3	AK20	RESET
4	AD36 (V_{SS})	GND
5	AG03	SMIACT#
6	U37 (V_{CC3})	V_{CC}
7	AC35	R/S#
8	AB36 (V_{SS})	GND
9	NC	NC
10	Z36 (V_{SS})	GND
11	AC05	PRDY
12	N35	TDI
13	N33	TDO
14	P34	TMS
15	X36 (V_{SS})	GND
16	M34	TCK
17	R36 (V_{SS})	GND
18	Q33	TRST#
19	NC	DBINST#
20	NC	BSEN#

Note: You may connect the GND pins to any pin marked V_{SS} on the SPGA pinout diagram. The NC pins are no connects. You may simply cut those wires.

Figure 25-8. Uni-Processor Debug



Connect a double-row receptacle (AMP# 111196-7) to the debug port connector end of the cable. This is a 30-pin connector, so that it fits into the socket on the debugger buffer board.

Remove the following pins from the bottom socket:

R/S#	AC35
PRDY	AC05
TDI	N35
TDO	N33
TMS	P34
TCK	M34
TRST#	Q33

Connect the two sockets together. Make sure not to crush the wires between the pins.

25.7.2.2 Dual-Processor Debug

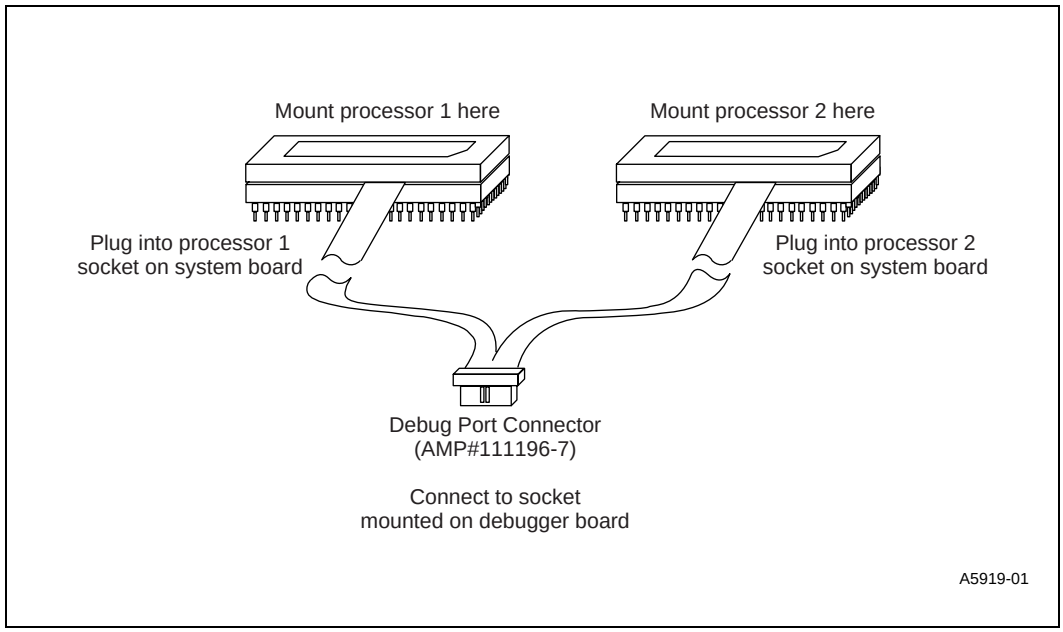
A debug port adapter for use in dual processor debugging can be built by reworking four Pentium processor-based SPGA sockets. (See Figure 25-9).

Note: This adapter can be used only when the processors are *not* included in the target system boundary scan string.

You will need to use two SPGA sockets per processor location. For this discussion, assume that the startup processor is called processor 1 and that the upgrade processor is called processor 2. Thus, you will use two SPGA sockets to connect to processor 1 and two SPGA sockets to connect to

processor 2. Certain debug port signals must be shared by Processor 1 and Processor 2. These signals must be connected from the debug port connector end of the cable (on which you will place a double-row receptacle: AMP# 111196-7) to both double SPGA sockets.

Figure 25-9. Dual-Processor Debug Port Adapter



Connect lines of 30-wire cable to the pins on the top SPGA sockets for both processor 1 and 2. Following are the signals which should be connected to each processor socket. Make sure to connect the shared lines to both top sockets.

Figure 25-10. Shared Pins for Dual-Processor Adapter

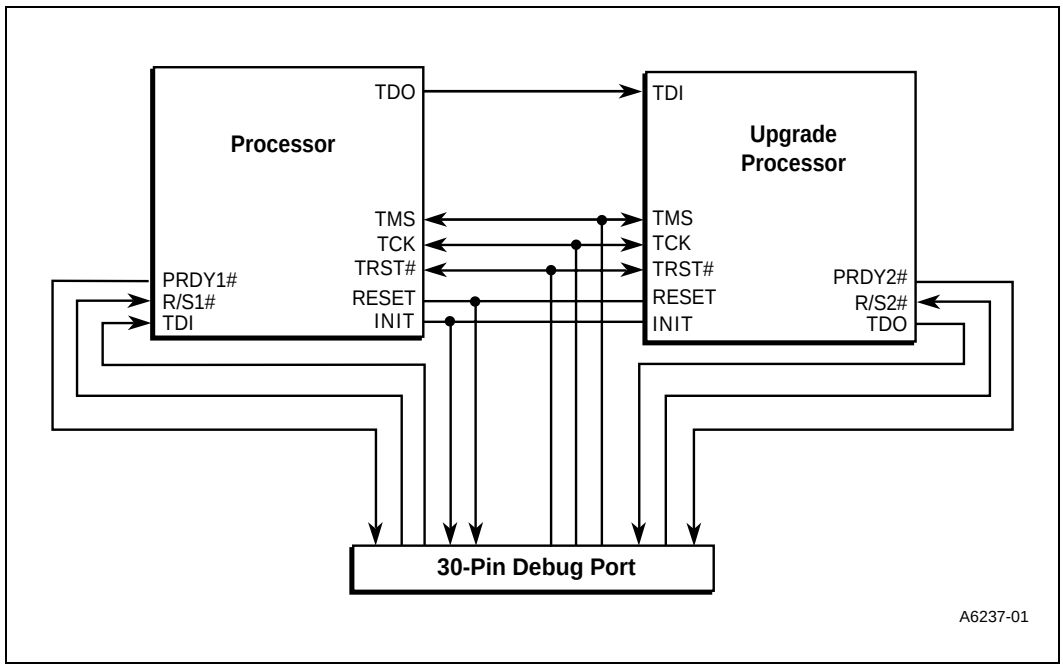


Table 25-4. Debug Port Connector Pinout

Cable Wire Number	SPGA Pin Number	Processor Socket	Signal
1	AA33	1,2	INIT
2	NC		DBRESET
3	AK20	1,2	RESET
4	V_{SS}	1	GND
5	AG03		SMIACT#†
6	V_{CC}	1	V_{CC}
7	AC35	1	R/S1#
8	V_{SS}	1	GND
9	NC		NC
10	V_{SS}	1	GND
11	AC05	1	PRDY1
12	N35	1	TDI
13	N33	2	TDO
14	P34	1,2	TMS
15	V_{SS}	1	GND
16	M34	1,2	TCK
17	V_{SS}	1	GND
18	Q33	1,2	TRST#
19	NC		DBINST#
20	NC		BSEN#
21	AC05	2	PRDY2
22	V_{SS}	2	GND
23	AC35	2	R/S2#
24	NC		NC
25	NC		NC
26	NC		NC
27	NC		NC
28	NC		NC
29	V_{SS}	2	GND
30	NC		NC

† Contact your third-party tools vendor for information on implementing SMIACT# in a dual processor system.

Note: You can connect the V_{CC} and GND pins to any convenient power or ground pin.

Connect a double-row receptacle (AMP# 111196-7) to the debug port end of the cable. This is a 30-pin connector, so that it fits into the socket on the debugger buffer board.

Remove the following pins from the bottom of both double sockets:

R/S#	AC35
PRDY	AC05
TDI	N35
TDO	N33
TMS	P34
TCK	M34
TRST#	Q33

Connect each set of two sockets together. Make sure not to crush the wires between the pins.