EBSA-110 and EBSA-285 Flash ROM Access Via JTAG

Application Note

October 1998

Order Number: 278205-001

Contents

Figures

Tables

1.0 Introduction

This document describes the software and hardware required to access Flash memory, via JTAG, on the EBSA-110 and EBSA-285 StrongARM** evaluation boards. It includes information about a software tool allowing program images to be downloaded from a PC, to the Flash memory on these boards.

This application note is based on the development work of a JTAG-based programming utility, for the Flash memory on the EBSA-110 and EBSA-285 platforms. It complements the existing Flash Management Utility (FMU).

1.1 New System Development

All microprocessor systems require a boot mechanism from the reset address. This program is typically kept in ROM. However, it is becoming increasingly attractive to replace ROM with Flash technology for the following reasons:

- It is non-volatile; contents are preserved when powered down (same as ROM).
- Can be reprogrammed in-situ to cater for upgrades, different uses, added features etc.
- Can be selectively reprogrammed by Flash block (typically 64 KB blocks in byte-wide devices).

Systems often had socketed ROMs to allow programming at the manufacturing stage prior to assembly, or in the case of EPROMs, removal for erasure and reprogramming. In System Programming (ISP) is an increasingly popular alternative. This technology allows boards to be populated without sockets, which saves cost, saves space (through enabled use of smaller package types), and improves reliability. Even these systems may require the boot-block to be preprogrammed into the device before assembly, and require expensive rework should the boot image get corrupted or require upgrading.

JTAG is a potential solution to the problem of programming a blank Flash part, in system, prior to any normal processor code execution. It also fits well with the demands of increasing system complexity, miniaturization and automation during the development, verification, manufacture and deployment of new products. This issue (not the JTAG ISP) is well documented in the IEEE 1149.1 JTAG Standard.

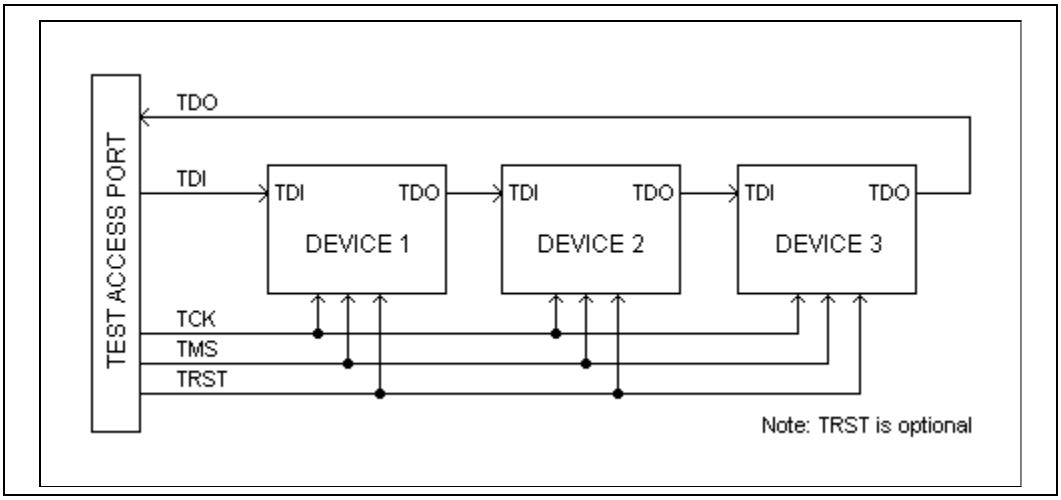
1.2 JTAG - Joint Test Action Group

Rather than use conventional test equipment, the JTAG standard describes the design of special on-component test circuitry. This can access the external pins of an IC and its' internal circuits, when placed in a designated test mode. Some devices support only the access of pins through JTAG, but not the internal circuitry. The JTAG register associated with the functional pins of a device is called the Boundary Scan Register. A collection of these from a serial link of parts is known as the Boundary Scan Chain. During normal device operation, the JTAG logic is transparent to the system.

Tests are achieved by providing stimuli and sampling the response. General operational tests may be applied to verify that parts behave correctly. The JTAG standard also suggests that entire hierarchical systems could be tested, using "daisy-chained" devices as illustrated in Figure 1.

JTAG fundamentals are described in more detail in Section 2.0. Please consult the full IEEE 1149.1 JTAG specification for more comprehensive information.

Figure 1. Typical JTAG Daisy-Chain



1.2.1 IEEE 1149.1 Utilization

JTAG compliant integrated circuits are gaining popularity, especially those with a high pin count. Various semiconductor manufacturers now provide a range of IEEE 1149.1 compliant logic parts.

1.2.2 Extended Use of JTAG Test Circuitry

Whilst primarily intended for manufacturing test, the JTAG boundary scan technology inherently provides an ISP interface for Flash technology. The main disadvantage is slow speed of operation. However, using JTAG to program a small primary boot loader with minimal driver capability, in order to get a system running and load a more substantial image, is a very feasible proposition. Providing a simple JTAG programming utility from the parallel port on a PC allows this to be a useful tool for developers, manufacturing, and potentially end-users.

The utility described in this document is slow because of PC parallel port limitations. Download time is also affected by the complexity of accessing memory in a systems' architecture. However, JTAG programming is usable, especially for image sizes of a few kilobytes.

1.3 StrongARM** Evaluation Boards

The SA-110 and SA-1100 StrongARM processors, along with the 21285 core logic device for the SA-110, are JTAG compliant devices. Two SA-110 based platforms were used to develop this application note and associated JTAG software utility.

1.3.1 EBSA-110

The EBSA-110 is the original SA-110 verification vehicle. It consists of an SA-110, fast-static and EDO / burst-EDO DRAM system, plus a programmed I/O subsystem. The boot path is through a link selectable ROM or Flash device, byte packed from 8 to 32 bits. All I/O access, byte packing and memory control is done via a pair of Complex Programmable Logic Devices (for example, the Altera MAX 7000* family CPLDs). ROM execution includes programming the Flash boot block.

The JTAG interface can be used on the SA-110 to generate appropriate bus cycles to program the Flash parts.

This platform is no longer produced. Please check your StrongARM sales channel for product availability.

1.3.2 EBSA-285

The EBSA-285 superseded the EBSA-110 as a general-purpose evaluation and porting vehicle. The board provides SDRAM support, a ROM / Flash boot path, a PCI interface and some ancillary logic, in a device known as the 21285 core logic. The 21285 can operate as a device in a PC system, or as the central function, configuring the PCI. A passive PCI backplane (EBSA-BPL) is available to support the latter case.

While the SA-110 JTAG interface could be used to generate bus cycles to the 21285, and hence the Flash port for programming, it is much simpler to bypass the SA-110 and apply the Flash programming vectors directly to the pins of the 21285 device.

1.4 How to Use This Document

This document provides a detailed overview of the software and hardware required to program Flash memory over JTAG on both the EBSA-110 and EBSA-285 evaluation board designs. The following information is provided.

- Section 2.0 outlines JTAG and its operation.
- Section 3.0 outlines the hardware platforms and memory cycles required to access Flash on the EBSA-110 and EBSA-285 boards. A summary of the JTAG to PC cable requirements is included.
- Section 4.0 describes the software design.
- Section 5.0 summarizes documentation errata relevant to the JTAG interface.
- Section 5.0 provides JTAG circuit information for the SA-110 and 21285 devices.
- Section 6.0 provides a user guide for the programming utility including a summary of the error messages.

2.0 About JTAG - An Introduction

The purpose of this section is to provide an introduction for engineers unfamiliar with the JTAG testing system. It is not a complete definitive reference.

2.1 What Is IEEE Standard 1149.1

The Joint Test Action Group (JTAG) is a committee board of delegates from a number of the world's leading electronic equipment companies and semiconductor manufacturers. It was formed by the IEEE to help establish and govern a methodical approach to implementing on-component testing circuitry. The IEEE 1149.1 Standard for JTAG, titled Standard Test Access Port and Boundary-Scan Architecture, mostly contains details for integrated circuit designers wishing to incorporate the test circuitry.

2.2 Why Use JTAG

When testing some modern design circuit assemblies, problems can arise with conventional test equipment, whether it is automated or not. The difficulties originate from the continuing trend to design products physically smaller, calling for assembly techniques using surface mount components. Prime examples of these are BGAs (Ball Grid Arrays), flat pack devices, and small outline IC packages. Device pin counts are increasing and lead spacing is decreasing. Accessibility of pins and the mechanical durability of solder joints cause these problems to arise in product verification.

Electrical continuity between device leads and the PCB can become questionable, especially after cumbersome and heavy probes are attached. Mechanical stresses can easily break a soldered joint and lift a pin away from the board when a probe is removed. A second concern is whether or not device pins can be accidentally made short circuit by the probes, damaging the device at test power up. This is attributed to the fine spacing of the leads.

A more appropriate way of testing devices and systems is called for. This means including test circuitry on the semiconductor devices, known as a Boundary Scan Test Architecture. This can access the physical pins for monitoring purposes. By strategically setting logical states on pins, and sampling the response, the electrical continuity of signal paths can be checked on the circuit board. General device testing may also be done to check behavioral responses to stimuli data.

A standardized test circuit was adopted by the JTAG committee. It is designed to allow normal operation of the IC, unless a test mode is activated. This forms a circuit given the acronym of BILBO - Built In Logic Block Observer.

2.3 Test Circuit Implementation

2.3.1 Fundamentals

The JTAG system is designed into a device, as a group of registers of various size and use, controlled by a logic block and instruction decoder. A dedicated connection port provides external access to the test circuitry.

The internal controller block governs the operation of the JTAG circuit, which provides the necessary internal enable signals for the multiplexers and registers. It conforms to a state machine, shown in Figure 2. The states are best described in the IEEE 1149.1 JTAG Standard.

The design is semi-static allowing the current state to be retained unless a reset, power loss, or the next JTAG clock transaction occurs.

Figure 2. JTAG State Machine

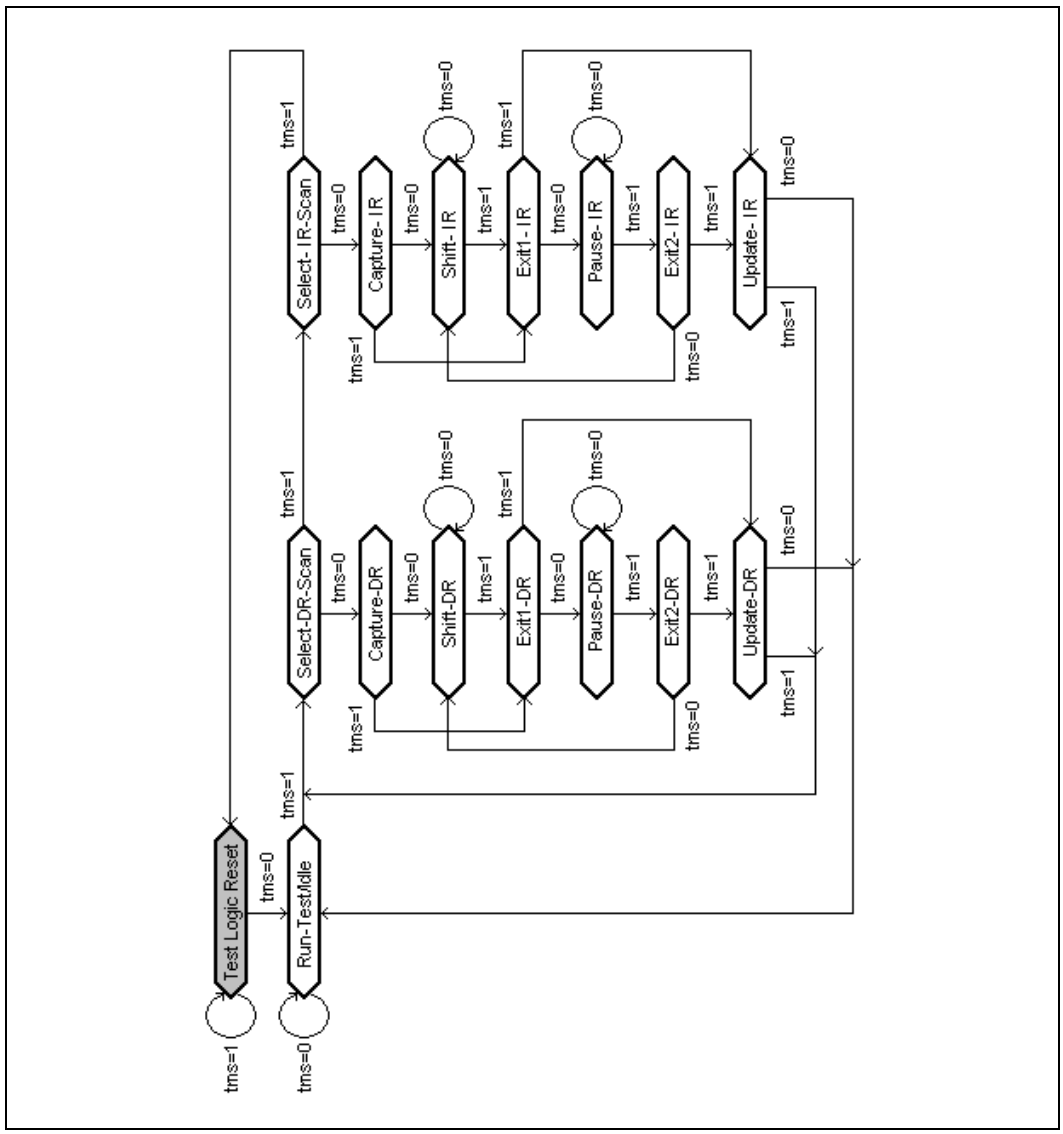
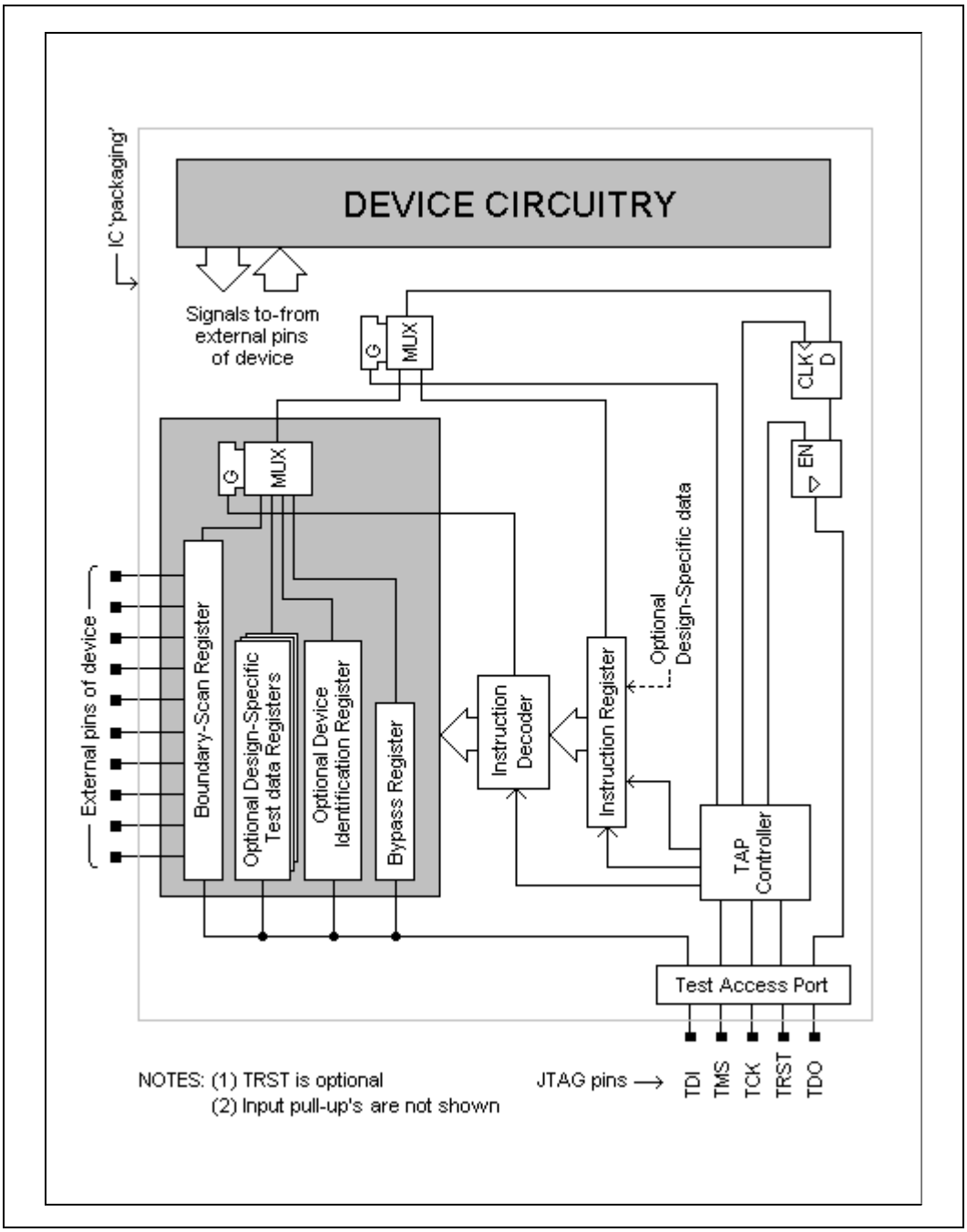


Figure 3 shows a diagram of a typical digital integrated circuit device, with JTAG circuitry included. Notice that the test circuit is separate from the functional core of the IC, and only the external pins of the device are shared by both parts, with the exception of the TAP pins.

Figure 3. IEEE 1149.1 JTAG Standard Circuit Implementation



2.3.2 JTAG Interface

The external interface to the test circuitry is known as the Test Access Port (TAP), and consists of four mandatory functional pins and an optional reset input. Table 1 details these connections.

Table 1. TAP Pin Names

Pin Name	Type	Purpose
Test Data In, TDI	Input, with pull-up	To input data into the device upon rising edges of TCK transitions.
Test Data Out, TDO	Tristatable Output	To output data from the device upon falling edges of TCK transitions.
Test Mode Select, TMS	Input, with pull-up	To signal the intention to proceed onto the next stage of the JTAG state machine, upon rising edges of TCK transitions.
Test Clock, TCK	Input	To provide a strobe signal to sample data into, and clock data out of the device.
Test Reset, TRST	Input, with pull-up	To reset the JTAG circuitry to a known initial condition. It has the same effect as the Test Logic Reset state. This input is asynchronous to TCK and is active low.

Pull-ups on the TMS and TCK inputs give a degree of immunity from electrical noise, and default the JTAG logic to a predictable state.

2.3.3 Registers

Figure 3 shows the arrangement of registers in a typical JTAG implementation.

Register access is dependent upon the JTAG instruction executed. External data between TDI and TDO is serially transferred, least significant bit first. The difference in register sizes cause data packet lengths to be variable. All registers (with exception to the Bypass Register) are parallel accessed for data load functions, but this is unavailable to the JTAG user.

During Test Logic Reset state, the TAP controller selects either the Bypass or Identification Register to be connected between TDI and TDO. This is a safety feature, as these registers have minimal system impact.

Information about the JTAG registers is presented in Table 2.

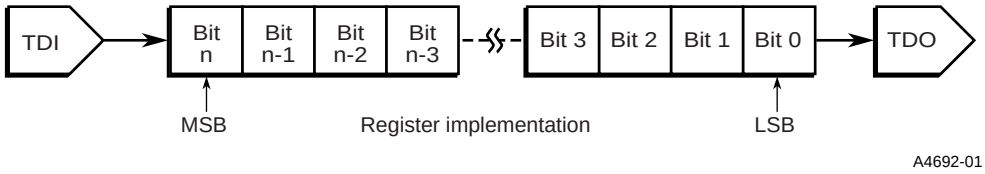
Table 2. JTAG Registers Description

Register Name	Purpose
Instruction Register, IR	To hold the current instruction opcode value, for the intended task.
Boundary Scan Register	To hold the data read from, or to be written to device pins, and is likely to be the largest register length in the JTAG device. It is divided into shift register stages called cells, and can contain various cell types; output, input, internal, linkage, and control.
Bypass Register	To shorten the length of a JTAG daisy-chain, for all subsequent Shift Data-Register loads. When a BYPASS instruction has been loaded into a specific device, this register is connected between the TDI-TDO data path, effectively reducing register size to one cell for that component.
Test data Register (Optional)	To hold specific data for manufacturers testing purposes. In some devices, these registers may provide additional functionality to the part.
Identification Register (Optional)	To hold a 32-bit composite word of predefined data, containing a manufacturer's number, a part number, and version information. The least significant bit is always set to logic 1, intended to be used by automated JTAG processes to identify the data as being from this register.

2.3.4 Endian Issues

All IEEE 1149.1 JTAG compliant designs require registers to be little endian, where the least significant bit or cell is “nearest the TDO pin”, as depicted in Figure 4.

Figure 4. JTAG Register View



2.3.5 Instruction Register

The Instruction Register is typically 8-bits in length or smaller, and contains valid instruction data in the Update-IR state. During a Shift-IR loading sequence, data can be clocked through this register out of TDO. Instructions can be passed to any subsequent devices in the JTAG daisy-chain, in this way.

Note that in the Capture-IR state, the previous instruction will be overwritten each time with a predetermined binary value of '01' for the two least significant bits. Remaining bits may be utilized as status flags or are set to fixed logic states.

2.3.6 Boundary Scan Register

The largest of the registers is the Boundary Scan Register, which reads and writes logical states of JTAG device pins. It is implemented as a shift-register of macrocells containing several multiplexers and flip-flops. In the Update-DR state, it contains valid stimuli data. In the Capture-DR state, response data is sampled. This means that data clocked into a device in the Shift-DR state can accordingly set pin outputs in the following Update-DR state. At the same time, the clocking action will shift out sampled pin data from the previous Capture-DR state.

Note that output cells can be overwritten to predetermined “safe” logic states. This is implementation specific.

2.3.7 Bypass Register

The Bypass Register is only one shift register stage. It is used in daisy-chained systems to reduce the number of register stages that need to be clocked through. This allows specific test data to reach the intended device more quickly. The Bypass Register is NOT effective for instruction loading sequences, therefore Instruction Register lengths cannot be shortened in the JTAG daisy-chain.

2.3.8 Identification Register

The ID register holds a formatted 32-bit data word containing device information. It can be used to distinguish JTAG parts in a daisy-chained system. The register structure is explained in Table 3.

Table 3. Identification Register Format

Bit field	Usage	Description
0	Pre-set to logic 1	An identity packing bit intended for distinguishing that this is valid data.
1..11	Manufacturer's Identity code	An 11-bit compressed version of a standard JEDEC number uniquely identifying the manufacturer.
12..27	Part number	A 16-bit word of unique value for a particular device.
28..31	Version number	A 4-bit word holding the silicon revision number of the particular IC.

The Manufacturer's Identity code field is formed by taking the most significant byte of the JEDEC code unique to the manufacturer, and discarding its parity bit. This determines the seven least significant bits. The remaining four bits hold a binary count value of the number of 0x7F continuation bytes in the original code. JEDEC codes are governed by the EIA / JEDEC office. Information can be found in their publication, JEP106.

2.4 The JTAG Instruction

JTAG instructions can be either public or private. The public type are available to all users. The private instructions are intended for reserved use by manufacturers and may invoke special tests. Table 4 describes the JTAG commands.

Table 4. JTAG Commands

Command Name	Nature	Register Selected	Purpose
EXTEST	Mandatory, public, opcode value of 0.	Boundary Scan	To permit external tests where data is asserted to drive output pins, and response is sampled from input pins.
SAMPLE / PRELOAD	Mandatory, public.	Boundary Scan	To take a sample of the logical state of the device pins, without asserting test data, that is, transparent to the normal operation of the device. Not meant for real-time work!
BYPASS	Mandatory, public, opcode binary value of all 1's.	Bypass	To shortcut the data path, reducing the number of data register stages in a JTAG daisy-chain. It does not affect normal IC operation.
INTEST	Optional, public.	Boundary Scan	To permit internal tests where stimuli data is asserted at inputs, and response is sampled at output pins.
HIGHZ	Optional, public.	Bypass	To force all output pins into high impedance state.
IDCODE	Optional, public.	Identification	To present the 32-bit device specific identity code.
USERCODE	Optional, public or private.	User defined	To execute customized functions.
CLAMP	Optional, public.	Bypass	To force all output pins into a predetermined state governed by the contents of the Boundary Scan Register.
RUNBIST	Optional, public.	User defined	To execute a self-test on the internal core circuitry of the device. External signals are controlled to suppress any interfering activity.

Not all of the JTAG instructions may be implemented into a given device. The IEEE 1149.1 standard requires inclusion of EXTEST, SAMPLE / PRELOAD and BYPASS commands in all compliant designs.

Most commands are given device specific opcode values, with the exception of EXTEST and BYPASS which are predefined.

2.5 Using IEEE 1149.1 JTAG

2.5.1 System Analysis

Before JTAG can be used, the precedence order of devices in the application daisy-chain should be examined. Each IEEE 1149.1 compliant device will have permitted architectural variations, such as in register lengths and register omissions. The Boundary Scan Register will be architecturally unique for a given device type, and information about this can be found in BSDL “descriptive language” files. Instruction opcodes will be different for each device type, except for EXTEST and BYPASS, which are predefined. All of this information needs to analyzed carefully.

2.5.2 State Machine

The state machine diagram shown in Figure 2 aids the applications engineer in using the JTAG test circuitry. There are only three basic actions performed by this state machine, in general:

- Do nothing; Reset-idle.
- Load a new instruction.
- Load new data into a selected data register.

Most of the states perform internal action, such as initializing and updating registers. Other states are regarded as temporary, which exit or wait between the operative ones. From the JTAG user’s point of view, the Shift-DR and Shift-IR states are of most concern. These allow new data to be clocked into the selected register from TDI, and captured data to be clocked out of TDO.

The typical sequence of events is to load an instruction, and then load the data into the register selected by the command.

2.5.3 Navigating States

By toggling the state of the TMS pin and strobing TCK, the state machine can be traversed. After power up or assertion of TRST, the circuitry waits in Test Logic Reset state for as long as TMS is logic high, and TCK is strobed. Before the JTAG circuitry is activated, Run-Test / Idle state must be entered.

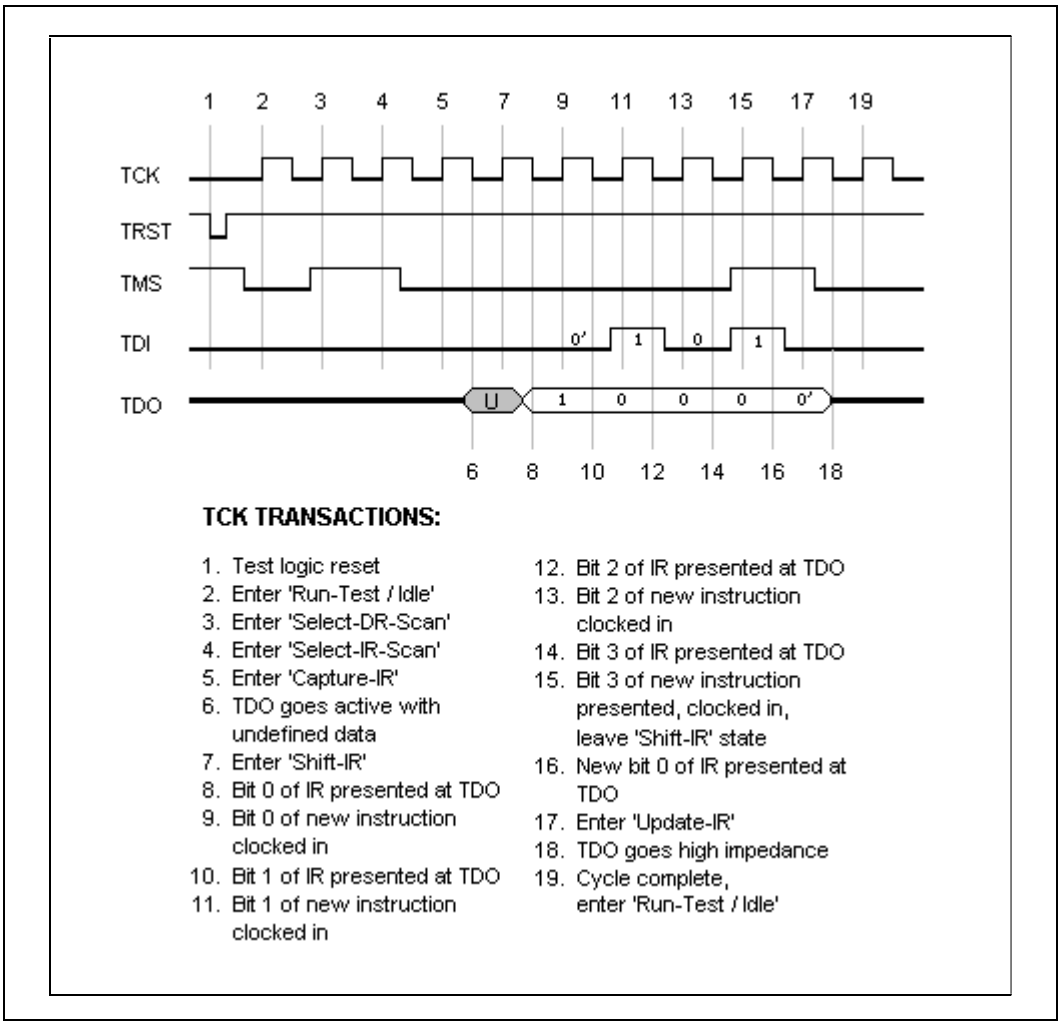
2.5.4 Loading Registers

To begin loading a register, the appropriate Select-*xx*-Scan state must be entered. The Instruction Register is exclusively loaded by following the Select-IR-Scan sequence of the state machine diagram. All other registers are loaded by the Select-DR-Scan path.

The data word or instruction word is loaded in the Shift-*xx* state, one bit at a time per strobe of TCK, keeping TMS logic low. An important point to note is that the last bit of the word is clocked in when leaving the Shift-*xx* state. Care should be taken to meet this requirement.

Figure 5 shows a typical sequence of signal events to load a four bit command word (10 decimal) into the Instruction Register. The Pause-*xx* and Exit2-*xx* temporary states are not used in this example. Note that the old Instruction Register contents are clocked out of TDO - see Section 2.3.5 for a description of this data. On the final bit load, TCK transaction 16 presents bit 0 of the new instruction word on TDO. A similar procedure is followed to load a data register, except transaction 4 is omitted.

Figure 5. TAP Signals Example



2.6 General Considerations

2.6.1 Requirements

Requirements for the clock signal, TCK, are quite loosely defined by the IEEE 1149.1 JTAG Standard. Specifications include the maximum operating frequency, full timing details of the implementation, standard logic threshold levels, and TDO drive / fan-out characteristics. To the JTAG user, only the maximum TCK frequency and minimum assertion time of TRST will be of concern, assuming that all set-up and hold timings are met for the TAP signals.

To permit normal function of the integrated circuit, the onboard JTAG boundary scan circuitry must be forced into an inactive condition by execution of the BYPASS or IDCODE commands, or by forcing a reset state by assertion of TRST.

2.6.2 Access of Tristate Outputs or Bidirectional Designs

An important concern is presented in the case of devices with tristate outputs or bi-directional technologies. Care must be exercised when accessing such pins in test mode, to avoid possible contention between external signals and output drivers. This means examining the design of each JTAG device for the enabling signals of each output pin driver, and controlling these as appropriate.

2.6.3 Irrelevant Device Pins and Use of JTAG

In most JTAG applications, only a subset of available pins in the Boundary Scan Register will need to be used. The irrelevant pins will still need consideration. Logic levels on inputs may be regarded as “don't care”, because all inputs are high impedance and need not be controlled. Outputs should be treated so that they retain their quiescent states, as exercised in normal operation. This means active low outputs remain logic high, and active high outputs remain logic low.

3.0 Evaluation Boards, Devices, and Using JTAG

Section 1.0 and Section 2.0 outlined the application problem and the IEEE 1149.1 JTAG Standard. This section describes the necessary specifics for implementing software to access Flash memory, via JTAG, on the EBSA-110 and EBSA-285. It should be read in conjunction with the following documents from Intel:

- 21285 Core Logic for the SA-110 Microprocessor Data Sheet (Order number 278115-001)
- EBSA-285 Evaluation Board Reference Manual (Order number 278136-001)
- SA-110 Microprocessor Technical Reference Manual (Order number 278058-001)
- EBSA-110 Schematic Directory
- EBSA-285 Schematic Directory

Note: Please consult Section 5.0 for documentation errata.

3.1 Nomenclature

- 32-bit data is referred to as a 'Dword'.
- The JTAG programming software is known as "the software".
- The Boundary Scan Register is abbreviated to 'BSR'.
- 4-byte hexadecimal values are shown in the format: 'xxxx.xxxx'.

3.2 Device Background Information

3.2.1 SA-110 Microprocessor

The StrongARM** microprocessor is based on Version 4 of the ARM** architecture. It is optimized for use in low power, high performance applications.

The CPU has 32-bit wide data and address buses, which may be disabled externally to allow other system peripherals to access specific devices. Control bus signals are used for Flash memory programming and need to be appropriately asserted or de-asserted by the software.

The Boundary Scan Register of this device is 130 stages long, and the Instruction Register holds five bits. The SA-110 supports six JTAG instructions. Section 5.0 has information about this part.

3.2.2 21285 System Core Logic IC

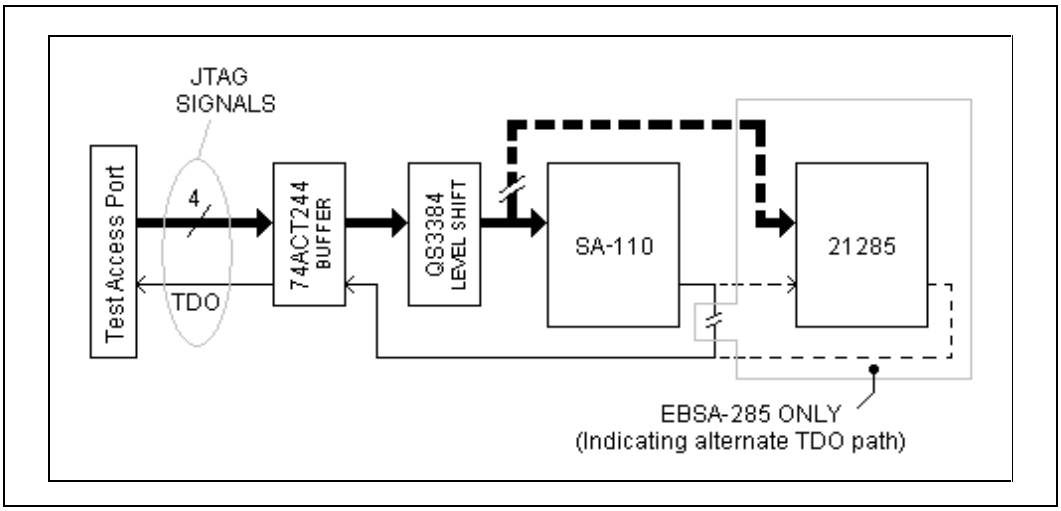
The 21285 is a device providing core logic to simplify systems design with the SA-110. The main features of the device include an interface to a PCI bus system, and a memory controller for different types of SDRAM.

The Boundary Scan Register of this device is 360 stages long, and the Instruction Register holds four bits. The 21285 supports six JTAG instructions. Section 5.0 has information about this part.

3.3 Using the System Architecture

The JTAG device chain on the EBSA-110 and the EBSA-285 is illustrated in Figure 6.

Figure 6. EBSA Daisy-Chain



3.3.1 EBSA-110

The EBSA-110 has only a single JTAG device in the daisy-chain; the SA-110. Flash memory communications must be performed via this part.

The 'CTA' and 'CTB' programmable logic system control blocks must be used to access the Flash by emulating CPU memory read and write cycles. The drawback of working around the system architecture is a greater overhead of processing data. Only a small amount of this is data to or from the Flash memory.

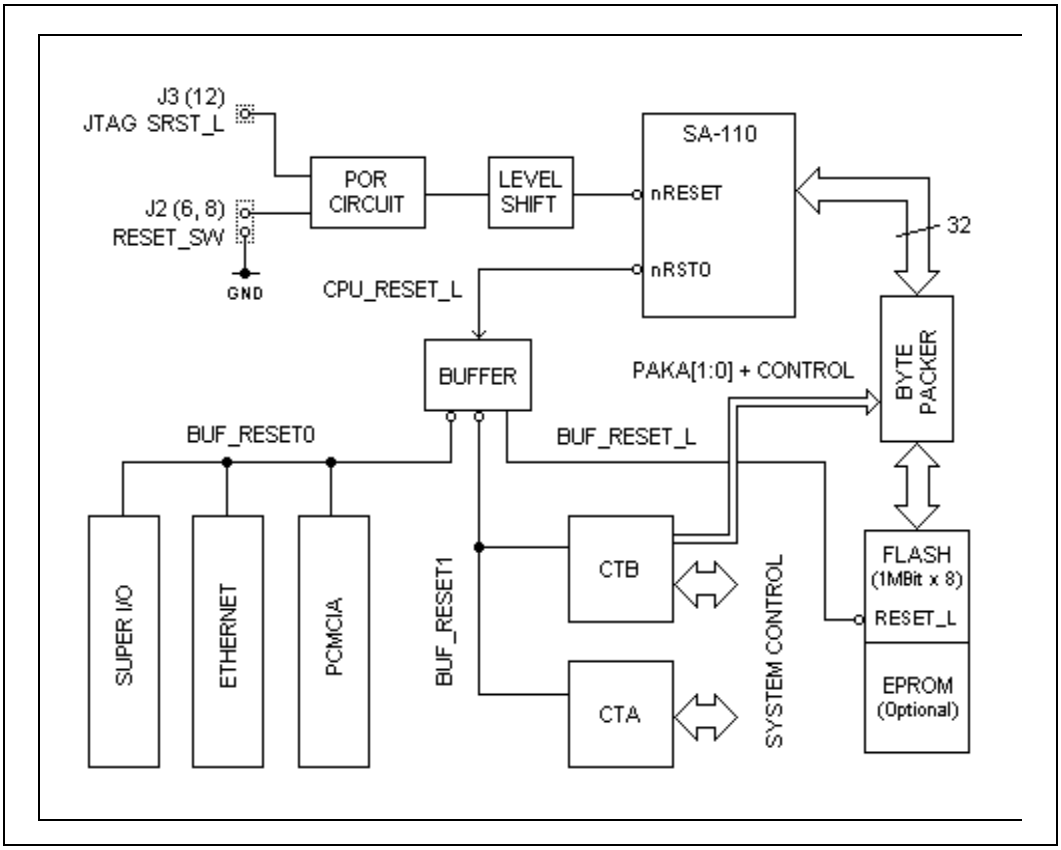
The CTB device operates several data bus buffers, which are arranged to act as byte packing stages. This allows only a byte-wide ROM memory to be required in the design.

Dwords may be read from Flash, being automatically assembled by the byte packing stages, from four contiguous bytes. Data is constructed in two steps, where the first set of packing buffers make two 16-bit words from four 8-bit bytes. The second set of buffers assemble a Dword from two 16-bit words. Two read modes are supported; single-beat (non-sequential) and multi-beat (burst).

Only single bytes may be written to the Flash, presenting the data on the low order byte lane and controlling the byte select lines, PAKA[1:0]. Dwords can be formed by separately writing the four bytes in contiguous memory address space.

SRST on the JTAG TAP must be asserted low to halt the SA-110. The nRSTO output of this device should be set logic low initially, to reset other components in the EBSA-110 system. To allow the Flash and system logic parts to function, this reset output must be subsequently taken high. Control of the nRSTO signal is achieved via the BSR. Figure 7 shows the reset system in detail.

Figure 7. Reset System on the EBSA-110



3.3.2 EBSA-285

The SA-110 precedes the 21285 in the EBSA-285 JTAG daisy-chain. Flash memory should be accessed via the latter device, whilst the CPU is placed into BYPASS mode. This provides the simplest programming method.

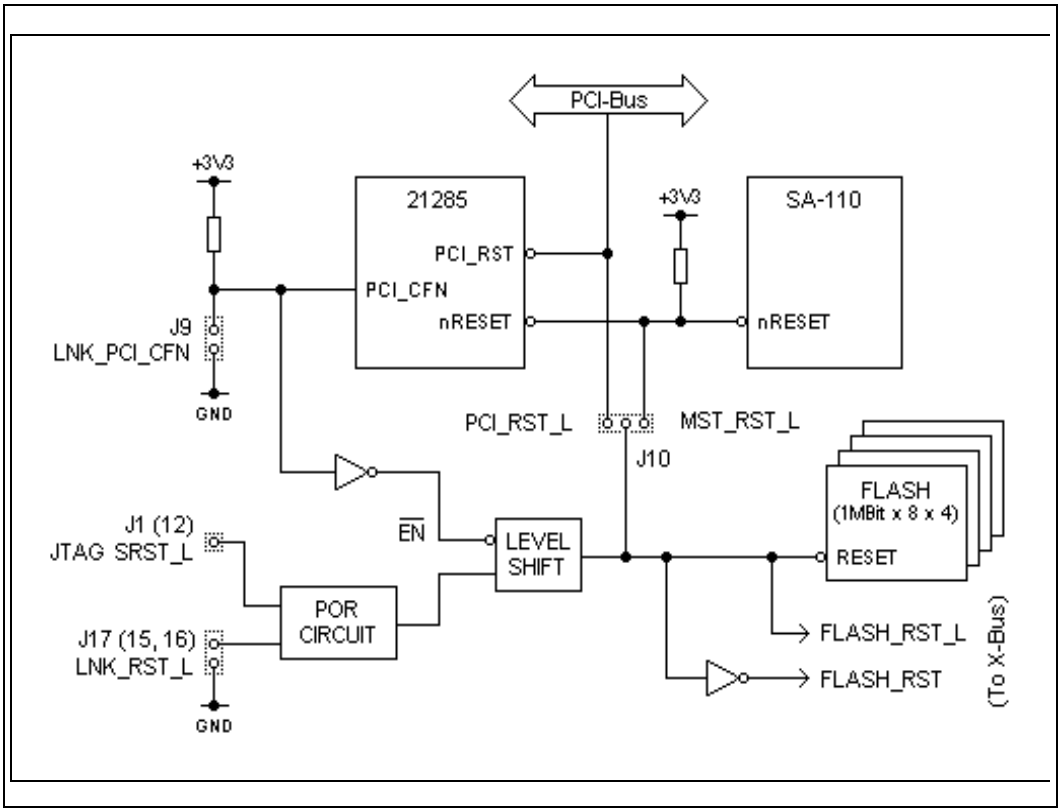
The board has four Flash parts, which are byte wide and arranged in parallel. They are simply interfaced to the system, allowing easy memory communication.

The SA-110 should be held in reset state by asserting the nRESET pin low via the respective 21285 output pin. This grants address and data bus access. SRST should be set logically high and jumpers J9 and J10 should be left open. Jumper J17 has a connection for an external reset push-button which should also be left open. These settings permit the Flash memory to operate, otherwise the Flash stays in reset state where it does not drive its' outputs.

In between the 21285 and Flash memory are LVT16244 buffers on the data and address buses. The 'bus-hold' feature of these buffers will operate when the Flash outputs are tristated, that is, when held in reset. Since the bus-hold technology is like a transparent latch, the jumper problem can be verified by reading the held data. Please refer to the manufacturers data for more details.

The reset system for the EBSA-285 is illustrated in Figure 8.

Figure 8. Reset System on the EBSA-285



3.3.3 BSR Signals Analysis and Use

Please consult Section 2.3.6 for a description of the BSR functionality.

The JTAG device used for communications is loaded with the EXTEST command. This needs to be performed only once for the whole series of Flash operations that follow. These operations are accomplished by loading several sequences of data in the format of the BSR.

The appropriate BSDL file should be carefully analyzed for signals relevant to the Flash memory device. The evaluation board schematics will provide information about which lines to control. These will include data and address buses, chip enable, read/write enable, output enable, and some internal controls. Note that some higher order address bits perform a few of these signal functions.

From BSDL files for the SA-110 and 21285 devices, the following tables have been drawn up:

- Data bus cells for both SA-110 and 21285
- Address bus cells for both SA-110 and 21285
- Control cells for the SA-110
- Control cells for the 21285

3.3.4 Special Address Lines

On the EBSA-110, memory reads automatically present Dword data on D[31:0]. For memory writes, data must be presented on D[7:0] and the PAKA[1:0] byte select lines must be used, programmed via A[23:22]. A binary code on the address lines will generate the same effect on the byte select lines.

An example addressing sequence of writing two consecutive Dwords on the EBSA-110 is shown below:

- 0x8000 0000, <Dword 1, byte 0>
- 0x8040 0000, <Dword 1, byte 1>
- 0x8080 0000, <Dword 1, byte 2>
- 0x80C0 0000, <Dword 1, byte 3>
- 0x8000 0004, <Dword 2, byte 0>
- 0x8040 0004, <Dword 2, byte 1>
- 0x8080 0004, <Dword 2, byte 2>
- 0x80C0 0004, <Dword 2, byte 3>

On the EBSA-285, address lines A[1:0] are set to '00' in the Dword mode. The 21285 device routes CPU address lines A[3:2] to the Flash memory address pins A[1:0], respectively. This gives an incremental address of four, owing to that many bytes per Dword.

ROM_WE_L and ROM_OE_L are generated by respective high-order address lines, A[31:30]. To facilitate different ROM widths in a system, A[29:28] are used by the 21285 to deal with their addressing, but are not required in this case. The remaining lines, A[23:4], address the Flash memory.

3.3.5 Memory Mapping

For simplicity of accessing Flash memory via JTAG on the evaluation boards, the following address information should be applied. These values ignore all high-order address line control signal bits as described in Section 3.3.4. (This does not mean that the software should ignore controlling them!)

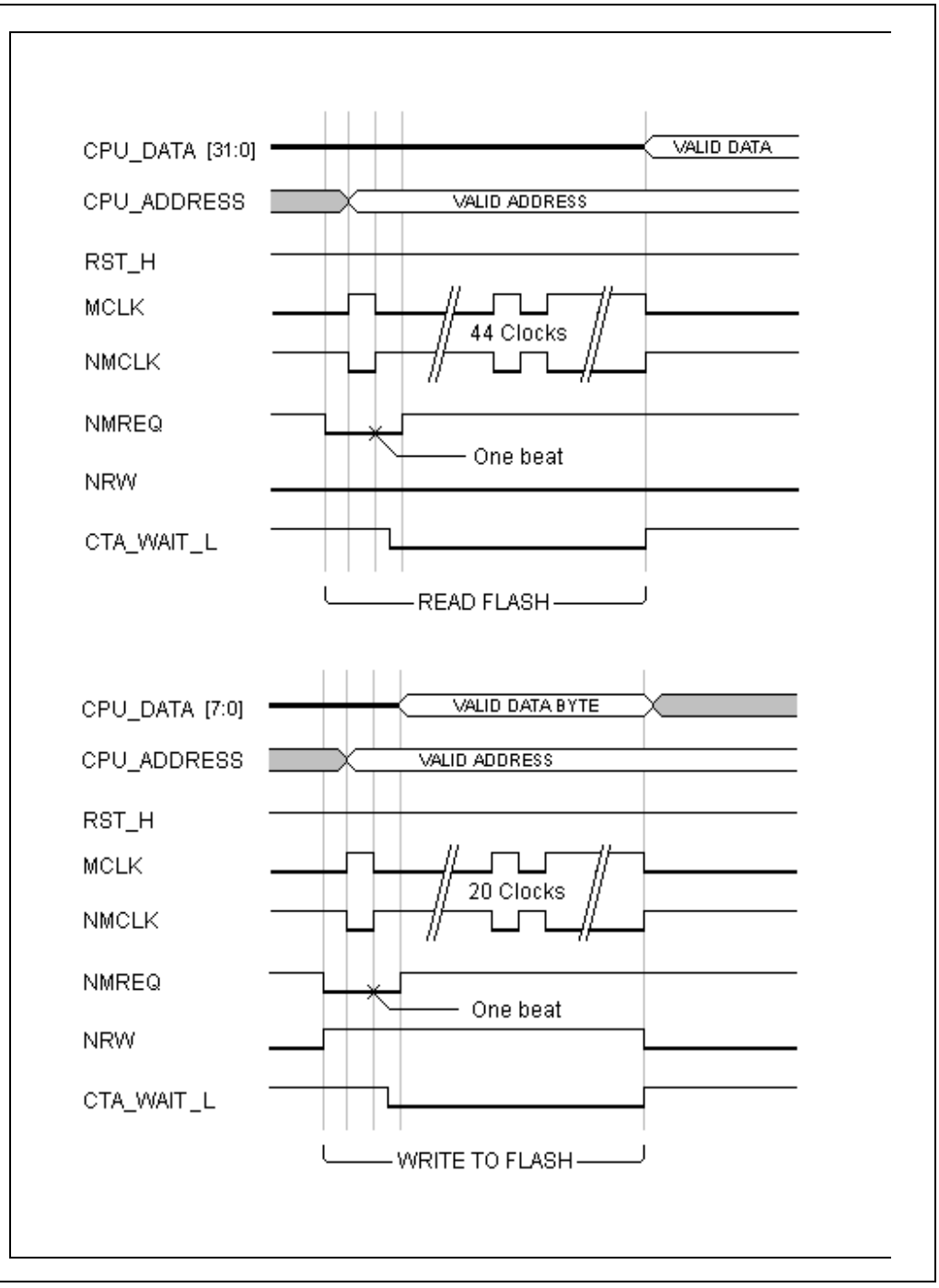
- EBSA-110 Flash memory can be addressed from 0x8000 0000 up to 0x800F FFFF (20-bits).
- EBSA-285 Flash memory can be addressed from 0x0000 0000 up to 0x004F FFFF (22-bits).

The origin of these address values can be explained. In normal operation, the address map quadrants 1 and 3 will swap upon the first detected CPU memory write. For the EBSA-110 this is also true when programming via JTAG, but it is automatic. However, for the EBSA-285, this is not the case since programming is achieved via the 21285 device and does not involve the CPU.

3.3.6 CPU Memory Cycle on the EBSA-110

Appropriate timing diagrams of Flash memory access are given in the EBSA-110 reference manual. However, these diagrams contain more information than needed and are somewhat difficult to read. A simplified timing diagram is given in Figure 9, which is suitable for accessing Flash via JTAG.

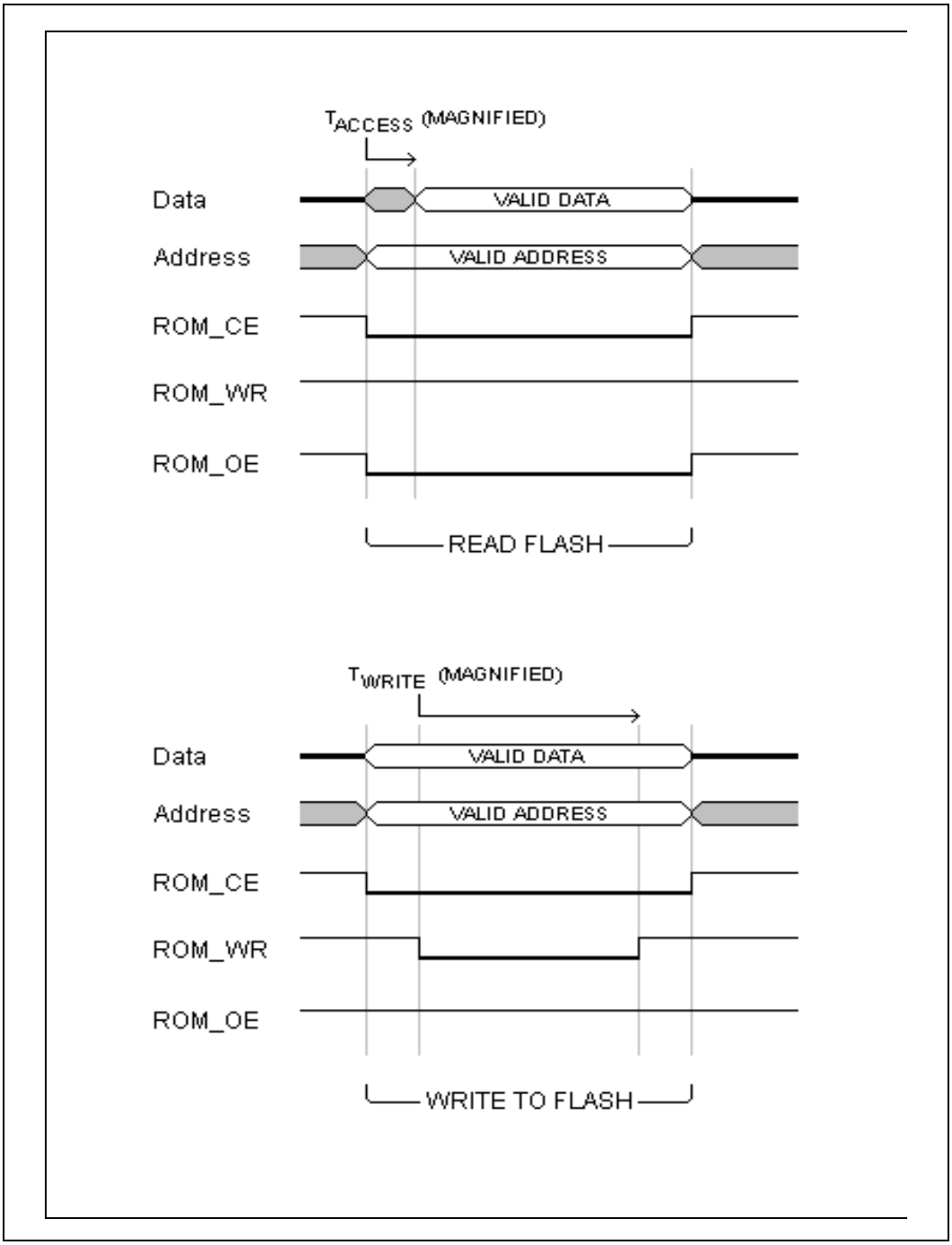
Figure 9. Simplified EBSA-110 Memory Read-Write



3.3.7 CPU Memory Cycle on the EBSA-285

The timing diagrams for the EBSA-285 are straightforward, but they may be further reduced as shown in Figure 10. This allows faster throughputs of data to be realized in the code implementation, when using the software.

Figure 10. Simplified EBSA-285 Memory Read-Write



3.3.8 TAP Connector

The TAP is buffered from the host PC parallel port, on the evaluation board. The physical connection is made by a 14-way IDC connector. The pinout is shown in the two rightmost columns of Table 5.

The connector originates from an existing microprocessor board produced by Advanced RISC Machines, Ltd. To respect conventions, this pinout has been adopted for the EBSA-110 and EBSA-285. A newer 20-way connector (Multi-ICE Standard) may be used on future boards.

3.3.9 Suggested Cable Connections

During the development of the software, the following connections were used in the prototype communications cable. Details are given in Table 5. Note that this part is not commercially available.

Approximately two meters of standard 14-way IDC was used, terminated with a matching female connector. The PC parallel port plug should be a male 25-way D-type connector, housed in an IDC shell hood.

A 2K pull-down resistor should be fitted between GND and the POWER_OK input pin 10, in the D-type connector assembly, on the cable.

Table 5. Transceiver Cable Connections

From D-Type Pin	Description	To IDC Pin(s)	Description
2	Data, PD[0]. (TCK)	9	TCK
3	Data, PD[1]. (TMS)	7	TMS
4	Data, PD[2]. (TRST_L)	3	TRST_L
5	Data, PD[3]. (SRST_L)	12	SRST_L
8	Data, PD[6]. (TDO)	5	TDI
9	Data, PD[7]. Wired back to pin 12, (SENSE_OUT)	-	-
10	ACK signal, Status bit 6. (POWER_OK)	1, 13	+5V output via a 33R limit resistor on the evaluation board.
11	BUSY signal, Status bit 7. (TDI)	11	TDO
12	PE signal, Status bit 5. From pin 9, (SENSE_IN)	-	-
17-25	GND (Linked pins)	2, 6, 8, 10, 14	GND

Note: Pins not listed in this table are regarded as “No connection”.

4.0 Program Implementation

4.1 Structures and Libraries

The rom_pgm.exe software is comprised of four hierarchical libraries and a main module. With exception to jtag_bscan.c, each one has a header file containing a small global structure of status and data information. Some libraries contain procedures for diagnostics and code development, but are not invoked unless you call them in modified source code. Table 6 briefly summarizes each of the libraries and the main module.

Most functions are 8-bit unsigned char types, returning an exit code. This is either a zero for no-error, or non-zero (1) for error. The program includes simple error trapping which will flag up messages, and exit all called routines until the main module is reached. The code is not designed to handle recovery from errors.

Note: In the following section, 'port' refers to the PC parallel port.

Please refer to the source code listings for specific information.

Table 6. ROM_PGM.EXE Libraries, Structures, and Main Module

Library or Module Source-Name	Global Structure	Description
rom_pgm.c	ProgSet	Main module containing the user interface along with the program functionality algorithms for memory block read, write, delete, and so forth.
jtag_pp.c	Sig	Contains the parallel port toggling functions and cable connections/power test function.
jtag_ops.c	JTAG	Handles most of the initialization of JTAG, and navigates the state machine as appropriate.
jtag_bscan.c	N/A	Loads/unloads data and sets control signals in the BSR. Shifts the assembled BSR bit vectors in and out of JTAG devices. It is an extension to the library code in jtag_ops.c.
flashrom.c	FLASH	Contains routines for Flash memory communications, and dealing with the system architecture for either evaluation board.

4.2 Initializing JTAG and System Checks

To ensure that the JTAG system is correctly set up, the following steps should be performed by the software in the given chronological order, before Flash memory is accessed.

In rom_pgm.exe, JTAG initialization is achieved by these procedures:

- Main module: rom_pgm.c, Statement: switch(Stage), case: 5.

This places JTAG devices into EXTEST or BYPASS and also asserts an appropriate reset, depending upon the board connected.

- Library: jtag_pp.c, Routine: CommsTest().

This checks for cable presence and power-okay, returning a fault flag.

- Library: jtag_ops.c, Routine: InitJTAG().

This initializes the JTAG system and validates the connected board. If criteria are met, the program is ready for Flash memory accessing.

- Library: jtag_ops.c, Routine: GetDeviceIDs().

This reads the JTAG identification register, and sets the operation specifics of the software to work with the board.

4.2.1 Cable Connections and Power

The following basic checks should be carried out:

- Is the transceiver cable connected between the specified/default port and the evaluation board?
- Is the target board +5V power healthy?

The port output should be reset to a value of 0x80, waiting for a few seconds. This allows the evaluation board power supply decoupling capacitors to discharge, as a precaution.

Detecting the cable on the port is achieved via a signal loop-back path, made in the connections for this purpose. As SENSE_OUT will be set logic high, testing SENSE_IN allows cable presence at the PC to be determined.

Verifying the +5V power on the target board is done by testing the POWER_OK input bit.

4.2.2 Asserting Resets

Please refer to Section 3.3 for more information about handling system resets.

The objective is to halt normal system activity but allow the parts required to function. JTAG reset requires two signals to be appropriately asserted, which are SRST_L and TRST. The previous value of 0x80 on the port output satisfies this.

The software should then release TRST, respecting the minimum assertion time, giving a port output value of 0x86. TMS can be left low since TCK is not strobed at this point.

4.2.3 Device Counting

Device counting allows a simple check to determine which JTAG instructions should be loaded into the devices, in the daisy-chain. This must be done in order to get the ID codes from the parts.

Having reset the JTAG circuitry, the devices present will be in Test Logic Reset state. The software must set TMS low and strobe TCK once, to exit this state.

Counting is achieved by placing all possible devices into BYPASS mode. To do this, an oversized stream of logic 1's is clocked in during the Shift-IR state. The number of these to be clocked in must be more than the combined IR length of devices expected in the daisy-chain. (rom_pgm.exe uses 32). All bypass registers should then be flushed out with logic 0's. Device counting begins by setting TDI logic high and strobing TCK, until a logic 1 emerges at TDO. The number of clocks required is the number of devices in the daisy-chain.

4.2.4 Identification Check

By loading the IDCODE instruction for each device in the daisy-chain, the parts on a board may be identified against known values. After clocking out the 32-bit data words, the upper four bits of each ID code should be masked off for the comparison. This is the silicon revision number, which will vary. However, the full code may be presented on the screen as useful information.

4.2.5 Recognizing the Connected Board

The combination of correct ID values for the parts and device count gives enough evidence to decide what board is connected. Unrecognized boards should cause the software to abort further action. From here on, the operation of the software must be customized to handle the differences between the two boards.

4.2.6 Loading BYPASS/EXTEST

At this point, the corresponding BYPASS and EXTEST modes should be entered by the software, for corresponding devices shown below:

- EBSA-110: SA-110 enters EXTEST.
- EBSA-285: SA-110 enters BYPASS, and the 21285 enters EXTEST.

Apart from a few system specific nuances, the JTAG system has now been initialized and is ready to work with the Flash memory.

4.3 Accessing the Flash Memory

4.3.1 BSR Modeling in Software

A bit vector of length and format of the BSR should be modeled in software by a one dimensional unsigned char array. Each location should hold either a 1 or 0. A suggestion is to have two arrays for incoming and outgoing vectors.

The software procedure for outgoing transactions should “fit” the address and data values into their respective cells. All control signals should be set appropriately for the desired operation, that is, tristate control buffers set for a write to Flash and memory control signals asserted.

Once the BSR bit vector has been assembled, it is then dispatched in the Shift-DR JTAG state. This is done one bit at a time per TCK strobe, using a program loop. Simultaneously, the previous data from Flash is shifted in and this will require processing to extract the data from the bit vector cells.

4.3.2 Flash ID Check

Flash memory device ID should be checked to determine whether the parts are the expected type. It is also useful during software development to check if the code is correctly accessing the Flash, as the ID word is a known value given in the manufacturer's data sheet.

For the EBSA-285, the procedure DetermineFlashType() in flashrom.c also checks for the jumpers discussed in Section 3.3.2. If the jumper error occurs, the latched data will be 0x9090.9090. This is the 'INTELLIGENT_ID' Flash command last written to the device.

4.3.3 Block Size

The EBSA-110 uses only one Flash part, giving a physical memory size of 1MB by 8-bits. (This is accessed through the 32-bit to 8-bit byte packing system). Since the memory device contains 16 sub-blocks, this gives a block size of 64KB. On the EBSA-285, there are four Flash devices giving sixteen 256KB sized blocks, accessed using a 32-bit data path.

4.3.4 Flash Type

The EBSA boards use Intel 28F008SA-L parts. Information important to the JTAG user will be the Flash instruction set, 'Write State Machine' operation, and some timing diagrams. Please consult the data sheet for details.

It is worth noting that some Flash instructions automatically output the status register contents upon subsequent memory reads.

4.3.5 Software Functions

The functionality of rom_pgm.exe is made up of several algorithms of basic Flash operation macros. These reside in the main module. This layer of software should be independent of system architecture and of memory device type.

The libraries, flashrom.c and jtag_bscan.c, contain driver code for accessing and working the Flash memory devices. Two fundamental procedures are written to emulate CPU memory read and write cycles. Using these to access Flash, the operation macros are implemented for:

- Read Dword data
- Write Dword data
- Delete block
- Identify Flash type

4.3.6 Program Image Types

The rom_pgm.exe software does not care what the image format is, providing that binary files are used. Images may be copied to any block, but may not run if they have been built for a specific block. Since the target address given in the headers of some file types is not used, the software cannot handle images larger than a single block size, that is, chained images are not supported. Raw data images and AIF files are treated the same, where the message “Invalid image file” will be displayed. The image shall still be copied, but the full block size default will be used. An image with an FMU header is treated differently. The filename is shown on the screen with image length, and only that number of bytes will be copied.

A standard AIF image file includes a 128 byte header. The existing FMU software adds a further 64 byte header to the beginning of an AIF file. The rom_pgm.exe JTAG programming software does not do this, but will identify valid FMU headers and use parts of them when present in a file through inheritance. The task of writing an extra code utility to rebuild files with a header is not too difficult.

The FMU image header format is given in Table 7. The size of this is 64 bytes, and uses little-endian format. More information can be found in the appropriate evaluation board reference manuals.

Table 7. Image Header Format (FMU)

Inclusive Byte Position	Description
0..3	Image type word. Always a value of 0xEB00002E.
4	Image number.
5..7	Three byte signature. Always a value of 0x00AA55.
8..11	Mapping information word.
12..15	Checksum word.
16..19	Image length word.
20..35	Filename of 16 bytes; ASCII characters.
36..39	Boot flags word.
40..63	Reserved bytes (24).

4.3.7 Speed Optimizations

The slowest factor of the software appears to be with the PC parallel port hardware, giving a maximum data throughput limit of 200KB per second. The ‘C’ language macros for I/O, outp() and inp(), were thought to be the cause but assembler versions of these functions were tried, giving no better results.

A number of software optimizations have been implemented to give the best possible results within reason. The fundamental optimizations are shown in Example 1.

Code optimizations have been made to routines in jtag_bscan.c library, where loading and unloading of the BSR bit vector array is done. The code has been reduced to the minimum needed by resetting all cells to logic 0, then only dealing with the important signals. This reduces the number of conditional statements and therefore overhead time.

In the flashrom.c library, FlashWriteCyc() and FlashReadCyc() routines were originally made to emulate “at speed” CPU memory cycles. Optimizations simplified these to the minimum number of signal transactions, since the working speed of the JTAG software is very much slower than normal conditions. The diagrams in Section 3.3.6 and Section 3.3.6 show the reduced waveforms.

Writing to the screen using printf statements within program loops is reduced to a minimum, as this is a relatively slow 'C' function.

In general, the best results are achieved in reviewing the fundamental level of the software, that is, the signal toggling port functions. This is because they are called so frequently, and quickly compound the delay.

Example 1. Fundamental Speed Optimizations

```
unsigned char WritePin(unsigned char Pin, unsigned char Value)
{
    /* This function will change the logic state of the pin specified,
        to a specified state */

    unsigned char State;

    /* Get the current state of the data register, if not known initially.
        This means subsequent pin writes only take one I/O port instruction
        instead of two. A global flag, Sig.NewWrite, is used to determine if
        the function has not been initially called. Speed optimization. */

    if (Sig.NewWrite == TRUE)
    {
        Sig.Last_out = inp(Sig.Current_Port + OFFS_OUT);
        /* Set non-zero - initialization not needed again */
        Sig.NewWrite = FALSE;
    }
```

```
    /* With last port out data, clear the bit, then update */
    State = ((Sig.Last_out & ~(1 << Pin)) | ((Value & 1) << Pin));
    outp((Sig.Current_Port + OFFS_OUT), State);
    /* Update the signals status info byte */
    Sig.Last_out = State;
    /* Return the status byte */

    return(Sig.Last_out);
}

void TCK_Strobe()
{
    /* This clocks the JTAG TCK line hi-lo. NOTE: Assume TCK is low
        initially from reset condition. Therefore, only two I/O routines
        are called, not three in the case of "ultra-safe code" */

    WritePin(B_TCK, H);
    Sig.Last_out = WritePin(B_TCK, L);
}
```

4.3.8 Power-Down Caution

When the target board is powered down, and a number of PC parallel port outputs are set logic high, the evaluation board can sink current. This is not desirable as damage to equipment could result! The software should clear the port outputs as necessary, using a value of 0x80, or 0 in the case of program exit. This has been catered for in rom_pgm.exe.

4.4 Future Development

In a further attempt to speed up the JTAG communications, research into alternative parallel port modes could be done. Most of today's PC machines support two additional port modes. These are Enhanced Parallel Port (EPP) and Extended Capabilities Port (ECP) modes. The latter is more suitable.

4.4.1 ECP Ideas

Although ECP supports a handful of submodes, no speed increase is likely with the present system setup. Submodes 0 and 2 are suggested, which are 'Compatible' and unidirectional (outgoing) 'FIFO' modes respectively.

Submode 3, 'ECP parallel port mode', will require additional hardware and protocol handling as it is incompatible with the present system setup. ECP features automatic DMA accesses to fill or empty address and data FIFOs. Bidirectional data flow is supported. Feasible data throughputs of up to 500KB per second can be achieved.

Information about ECP may be found in semiconductor manufacturer's data books on Super I/O integrated circuits.

5.0 Integrated Circuit JTAG Information

Table 8. SA-110/21285 JTAG Instructions

Instruction	Opcode Value; SA-110	Opcode Value; 21285	Description
EXTEST	00000	0000	To permit external tests where data is asserted to drive output pins, and response is sampled from input pins.
SAMPLE / PRELOAD	00001	0001	To take a sample of the logical state of the device pins, without asserting test data, that is, transparent to the normal operation of the device. Not meant for real-time work!
BYPASS	11111	1111	To shortcut the data path, reducing the number of data register stages in a JTAG daisy-chain. It does not affect normal IC operation.
IDCODE	00110	0100	To present the 32-bit device-specific identity code.
CLAMP	00100	0011	To force all output pins into a predetermined state governed by the contents of the Boundary Scan Register.
HIGHZ	00101	0010	To force all output pins into high impedance state.

Note: Opcodes are stated as binary values. Only six JTAG instructions have been implemented into these two devices.

Table 9. SA-110/21285 JTAG Identity Codes

Device	Manufacturers ID Number; Bits 1 to 11	Part Number; Bits 12 to 27
SA-110	0000 0110 101 (0x35)	0001 0000 0010 1100 (0x102C)
21285	0000 0110 101 (0x35)	0100 0001 1001 0100 (0x4194)

Table 10. BSR: Data Bus Cells

Signal Name	Input Cell (SA-110)	Input Cell (21285)	Output Cell (SA-110)	Output Cell (21285)
D[0]	65	357	64	358
D[1]	63	355	62	356
D[2]	61	353	60	354
D[3]	59	351	58	352
D[4]	57	349	56	350
D[5]	55	347	54	348
D[6]	53	345	52	346
D[7]	51	343	50	344
D[8]	49	336	48	337
D[9]	47	334	46	335
D[10]	45	332	44	333
D[11]	43	330	42	331
D[12]	41	328	40	329
D[13]	39	326	38	327
D[14]	37	324	36	325
D[15]	35	322	34	323
D[16]	33	315	32	316
D[17]	31	313	30	314
D[18]	29	312	28	312
D[19]	27	309	26	310
D[20]	25	307	24	308
D[21]	23	305	22	306
D[22]	21	303	20	304
D[23]	17	301	16	302
D[24]	15	295	14	296
D[25]	13	291	12	292
D[26]	11	289	10	290
D[27]	9	287	8	288
D[28]	7	285	6	286
D[29]	5	283	4	284
D[30]	3	281	2	282
D[31]	1	279	0	280

Table 11. BSR: Address Bus Cells

Signal Name	Output Cell (SA-110)	Output Cell (21285)	Input Cell (21285 Only)
A[0]	112	259	258
A[1]	111	257	256
A[2]	110	255	254
A[3]	109	253	252
A[4]	108	251	250
A[5]	107	249	248
A[6]	106	247	246
A[7]	105	245	244
A[8]	104	243	242
A[9]	103	241	240
A[10]	102	239	238
A[11]	101	237	236
A[12]	100	235	234
A[13]	99	233	232
A[14]	98	231	230
A[15]	97	229	228
A[16]	96	227	226
A[17]	95	225	224
A[18]	94	223	222
A[19]	93	221	220
A[20]	92	219	218
A[21]	91	217	216
A[22]	90	215	214
A[23]	89	213	212
A[24]	88	211	210
A[25]	87	209	208
A[26]	86	207	206
A[27]	85	205	204
A[28]	84	203	202
A[29]	83	201	200
A[30]	82	199	198
A[31]	81	197	196

Table 12. SA-110 BSR: Miscellaneous Cells

Signal Name	Cell	Purpose
'Control'	19	Output direction control for D[31:0]. Set this high for a write output, low for a read.
'Control'	68	NMCLK output drive enable. Set high always.
NWAIT	75	Input from CTA_WAIT signal.
NMCLK	76	Negated MCLK. (IEEE 1149.1 compliant). Toggle this line with the compliment MCLK, to work the system state machines in CTA and CTB logic blocks, for the EBSA-110 design.
MCLK	78	Memory clock output. Note: This cell is not IEEE 1149.1 compliant as it will drive the output whenever SNA input pin is high. See description for NMCLK.
MAS[1]	113	Byte mask line. Set high always.
MAS[0]	114	Byte mask line. Set low always.
'Control'	115	Output drive enable for A[31:0]. Set high always.
RSTOUT	116	CPU Reset output. Set low to initialize, then set high for further operations.
NRW	123	Read-Write output. Toggle this line; write when high, read when low.
SEQ	126	Sequential address. Compliment of NMREQ. Set high always.
NMREQ	127	Memory request output. Toggle this line low for a accessing Flash, and conversely.
'Control'	129	NMREQ and SEQ output drive enable. Set high always.

Note: The signals, MAS[1:0], are set this way to safely select byte 0 on D[7:0].

Table 13. 21285 BSR: Miscellaneous Cells

Signal Name(s)	Cell(s)	Purpose
CS_L[3:0]	36, 38, 40, 42	Chip selects [3:0], active low. Set high always.
ROM_CE_L	54	ROM enable line. Toggle this low to activate Flash and conversely.
XIOW_L	57	X-Bus write strobe, active low. Set high always.
XIOR_L	59	X-Bus read strobe, active low. Set high always.
XCS_L[2:0]	63, 65, 67	X-Bus chip selects [2:0], active low. Set high always.
TX	73	Transmit data from UART. Set high always.
PCI_GNT_L	75	PCI grant, active low. Set high always.
REQ_L	84	Bus master request, active low. Set high always.
'Control'	121, 151	Output direction controls, AD[31:16] and AD[15:0] respectively. Set high always.
'Control'	126	Output direction control, FRAME_L. Set high always.
'Control'	129	Output direction control, IRDY_L. Set high always.
'Control'	132	Output direction control, TRDY_L, DEVSEL_L, STOP_L. Set high always.
'Control'	140	Output direction control, PERR_L. Set high always.
'Control'	143	Output direction control, SERR_L. Set high always.
'Control'	146	Output direction control, PAR. Set high always.
PCI_IRQ_L	167	PCI interrupt. Set high always.
'Control'	171	Output direction control, CBE_L[3:0]. Set high always.
PAD_NIRQ	190	Interrupt request. Set high always.
PAD_NFIQ	192	Fast interrupt request. Set high always.
'Control'	264, 265	Output direction controls, PAD[31:15] and PAD[14:0] plus PAD_MAS[1:0], respectively. Normally these are set logic low, but if the address lines are to be read, toggle them high.
PAD_ABE	267	External CPU address bus tristate enable. This allows the 21285 to request use of the bus. Always set logic low.
PAD_DBE	294	External CPU data bus tristate enable. This allows the 21285 to request use of the bus. Always set logic low.
'Control'	317, 338, 359	Output direction controls for D[31:16], D[15:8] and D[7:0] respectively. Set these low for a write output, high for a read.

6.0 User Guide for ROM_PGM.EXE

6.1 Requirements

The software is very easy to use, but only works from a PC compatible running DOS. Note, it cannot run from the MS-Windows 3.1/95/NT DOS window utility. This is due to features of the operating system which inhibits rom_pgm.exe from accessing required PC hardware, properly and quickly.

A transceiver cable is required, with connections given in Table 5.

The software is designed to work with both the EBSA-110 and the EBSA-285, automatically deducing which board is connected. Other system configurations will be rejected.

6.2 Syntax

The program uses command line parameters to specify filenames and paths, port, and desired operation. The syntax for the command line is:

```
c:\rom_pgm <command> </b:block_number> <image_filename> </p:port_name>
```

A basic help screen may be invoked by typing:

```
c:\rom_pgm /helporc:\rom_pgm /?
```

The operations performed by this software via <command> are:

'CHK' Checks the availability of Flash blocks sequentially, starting from the specified block. Pressing the <esc> key will abort the check on the current block. Note that the software will try to look for a valid image header, and if it succeeds, then the image name will be displayed with its size. If it fails then a default sequential search through the first 256 bytes only will be performed. Upon finding data not equal to 0xFF, the message “Invalid image found” will be shown.

'DEL' Erases a specified Flash block. The software will ask for verification. It will warn if block 0 is specified as this is the boot block, and erasing this will render the board un-bootable.

'TST' Writes a test pattern if the block desired is verified empty. If true, a test pattern of “<pattern count> AAAA...ZZZZ 0000...9999 aaaa...zzzz”, is written as many times as will fit in the specified block. The action is aborted if the block is occupied or <esc> is pressed (where only a portion of test data will be written).

'VRD' Verbose Read. Displays the contents of a block in a typical hex memory viewer fashion, where the LSB of each Dword is at offsets 0, 4, 8 and 0x0C. Pressing <return> will print up the next screen of memory data. Hitting <space> will jump one screen full of data and print up the next, which is faster. Hitting <esc> will abort the operation. A column of ASCII print tables is given which is useful to identify image names. A percentage-read indicator is given for guidance.

'RD' Reads out images from the given block to a binary image file specified, or to the default filename. The software tests for a valid image header, and if detected will subsequently get the image length, copying that number of bytes to the file. If no image header is found, then an “Invalid image: Copying *xxx* Kbytes block to file...” message is shown and the full block size is copied. A progress indicator is given. Pressing <esc> will abort, but the file will be incomplete!

'WR' Writes an image to a given block from the specified filename order fault filename. Before an image write can be done, the software reads in a portion of the source file, testing it for a valid image header. If one is found, then the image length is determined and that number of bytes will be written to the Flash, otherwise an "Invalid image: Copying *xxx* Kbytes to memory" message is displayed, and the full block size will be copied. The target block is verified empty, and if not true, the process is aborted. A progress indicator is given, and pressing <esc> will abort. This will mean that the written image is not complete!

Valid values of </b:block_number> are 1 to 15. Illegal values will display an error message which terminates the program.

Image filenames given must have the extensions of *.aif, *.axf, or *.img. Path names may be specified, but the total string length of this must not exceed 80 characters. File operations are automatically disabled for the TST, CHK, VRD, and DEL commands, since they are not needed.

Valid constants for </p:port_name> are, 'LPT1' or 'LPT2'. PC parallel port addresses of 0x378 and 0x278 are used, respectively. Standard port mode is used.

6.3 Defaults

The <image_filename> and </p:port_name> fields are optional. The default filename is flashrom.img. The default port is LPT1.

6.4 Usage

- With the power off on both PC and EBSA board, connect the transceiver cable between PC parallel port LPT1 and the 14-way IDC JTAG TAP connector.
- Power up, then check communications by using the VRD command on any block apart from boot block 0 (unless there is nothing in it anyway).

```
c:\rom_pgm vrd /b:4
```

- The software will display a banner and perform some checks that take a few seconds. If all is okay, then the desired operation will commence, otherwise error messages are displayed.
- Use the WR command to program a valid image into the Flash from a specified file.
- Use the RD command to read back a block to a given file.

6.5 Error Conditions

The software performs some fundamental error checking. When errors occur, messages are displayed and the program terminates. Most of these messages should be self explanatory. Below is a list with a brief explanation, where appropriate.

6.5.1 Fatal Errors

1. Messages from module: rom_pgm.exe, routine: main()

- Could not open the working file [xx...xx]

A file I/O error had occurred, possibly from a file or path not existing or a path was not correct.

- Failed to initialize JTAG successfully
- Failed to read JTAG device identification(s)
- Failed 'BYPASS'/'EXTEST' instructions and reset

A fundamental JTAG error occurred with the devices as they have not responded to the IEEE 1149.1 state machine operation. The software could not assert the appropriate system reset.

- Failed to identify FLASH devices successfully

The Flash parts do not respond as expected. The devices may not be the Intel 28F008SA-L type, or something stopped them from working, such as a reset jumper.

- Failed to complete desired operation

The operation you requested from the software could not be completed due to a previous and more significant error condition.

- An unknown error has occurred!

An unusual error has occurred in the software, which is not identified.

2. Messages from module: rom_pgm.exe, routine: Commands()

- Error! No command specified or wrong syntax
- Error! Block number specified is too large
- Error! No block number specified or wrong syntax

The user caused an error by incorrectly specifying the command line arguments when invoking rom_pgm.exe.

3. Message from module: rom_pgm.exe, routine: ReadBlock()

- Error! File write operation failure!

A file I/O error occurred, possibly from a media error or the disk becoming full.

4. Messages from module: rom_pgm.exe, routine: WriteBlock()

- Error! File read operation failure!

A file I/O error occurred, probably resulting from a media error.

5. Messages from library: jtag_pp.c, routine: CommsTest()

- Error! Comms cable not connected at PC!
- Error! Comms cable not connected at board, or +5V is dead!
- Error! Unknown comms test result!

When the procedure tested for the transceiver cable presence and power healthy, a failure occurred.

6. Messages from library: jtag_ops.c, routine: GetDeviceIDs()

- Error! Cannot cope with more than two JTAG devices!
- Error! The board connected is of an unknown type!

The user has connected a board that is not an EBSA-110 or EBSA-285. The software is not designed to work with any other boards.

7. Message from library: jtag_ops.c, routine: LoadJTAGInstr()

- Error! Couldn't load JTAG instruction!

A fundamental JTAG or communications error occurred previously that prevented the software from further actions.

8. Messages from library: jtag_ops.c, routine: InitJTAG()

- Error! The JTAG daisy-chain is short circuit!
- Error! The JTAG daisy-chain is open circuit!

During preliminary checks, a fault was detected in the JTAG daisy-chain with the TDI-TDO data line.

9. Message from library: jtag_bscan.c, routine: TalkBoundaryScan()

- Error! Couldn't load the boundary scan register!

A fundamental JTAG or communications error occurred previously that prevented the software from performing further actions.

10. Message from library: flashrom.c, routine: WriteToFlash()

- Error! Failed to write the data successfully! SR:[xx...xx]

11. Message from library: flashrom.c, routine: DeleteFlashBlock()

- Error! Failed to erase block x successfully! SR: [xx...xx]

12. Messages from library: flashrom.c, routine: DetermineFlashType()

- Error! Flash is not responding! Ensure removal of J9, J10 and J17 reset(EBSA-285 only)
- Error! Flash is not responding!(EBSA-110 only)

13. Message from library: flashrom.c, routine: FlashWriteCyc()

- Error! Boundary scan comms in cpu-flash write failure!

14. Message from library: flashrom.c, routine: FlashReadCyc()

- Error! Boundary scan comms in cpu-flash read failure!

These are all Flash related problems occurring because of system errors, or otherwise arising from a previous JTAG or communications failure event. Where the message states SR:[xx...xx], the status register contents of the Flash device is given to aid diagnosis.

6.5.2 Non-Fatal Errors

Message from module: rom_pgm.exe, routines: WriteBlock() and TestPattern().

- Error! This block is not empty

This is the only error condition that will terminate the software in ordinary fashion, that is, the operations WR and TST are aborted after this event.

6.6 Completion Times

The software has been tested on a 133 MHz Pentium® processor running MS-DOS V6.22.

For the EBSA-285 (using 256KB images), downloads take approximately 10 minutes for a block read, and 25 minutes for block write.

For the EBSA-110 (using 64KB images), downloads take approximately 30 minutes for a block read and 100 minutes for a block write.

Not all applications will be large enough to use a full block size, so these download times may be significantly reduced.

Support, Products, and Documentation

If you need technical support, a *Product Catalog*, or help deciding which documentation best meets your needs, visit the Intel World Wide Web Internet site:

http://www.intel.com

Copies of documents that have an ordering number and are referenced in this document, or other Intel literature may be obtained by calling **1-800-332-2717** or by visiting Intel's website for developers at:

http://developer.intel.com

You can also contact the Intel Massachusetts Information Line or the Intel Massachusetts Customer Technology Center. Please use the following information lines for support:

For documentation and general information:	
Intel Massachusetts Information Line	
United States:	1–800–332–2717
Outside United States:	1–303-675-2148
Electronic mail address:	techdoc@intel.com

For technical support:	
Intel Massachusetts Customer Technology Center	
Phone (U.S. and international):	1–978–568–7474
Fax:	1–978–568–6698
Electronic mail address:	techsup@intel.com