US005825878A

United States Patent [19]

Takahashi et al.

[11] Patent Number: 5,825,878

[45] Date of Patent: Oct. 20, 1998

[54] **SECURE MEMORY MANAGEMENT UNIT FOR MICROPROCESSOR**

[75] Inventors: **Richard Takahashi**, Phoenix, Ariz.; **Daniel N. Heer**, Newton, N.H.

[73] Assignee: **VLSI Technology, Inc.**, San Jose, Calif.

[21] Appl. No.: **717,106**

[22] Filed: **Sep. 20, 1996**

[51] **Int. Cl.**[6] **H04L 9/00**

[52] **U.S. Cl.** **380/4**; 380/25

[58] **Field of Search** 380/3, 4, 23, 25, 380/52

[56] **References Cited**

U.S. PATENT DOCUMENTS

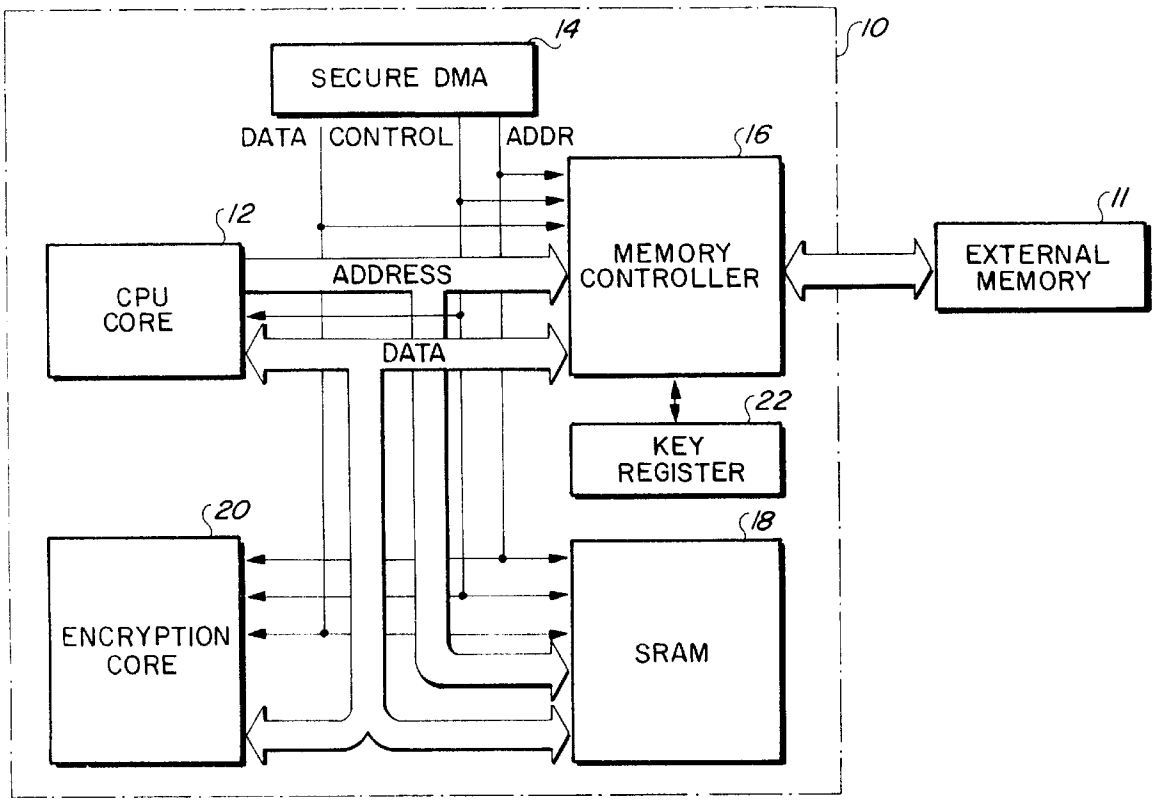
5,377,264 12/1994 Lee .
5,386,469 1/1995 Yearsley .
5,452,355 9/1995 Coli .
5,459,851 10/1995 Nakajima .

Primary Examiner—David Cain
Attorney, Agent, or Firm—LaValle D. Ptak

[57] **ABSTRACT**

A secure embedded memory management unit for a microprocessor is used for encrypted instruction and data transfer from an external memory. Physical security is obtained by embedding the direct memory access controller on the same chip with a microprocessor core, an internal memory, and an encryption/decryption logic. Data transfer to and from an external memory takes place between the external memory and the memory controller of the memory management unit. All firmware to and from the external memory is handled on a page-by-page basis. Since all of the processing takes place on buses internal to the chip, detection of clear unencrypted instructions and data is prevented.

7 Claims, 2 Drawing Sheets



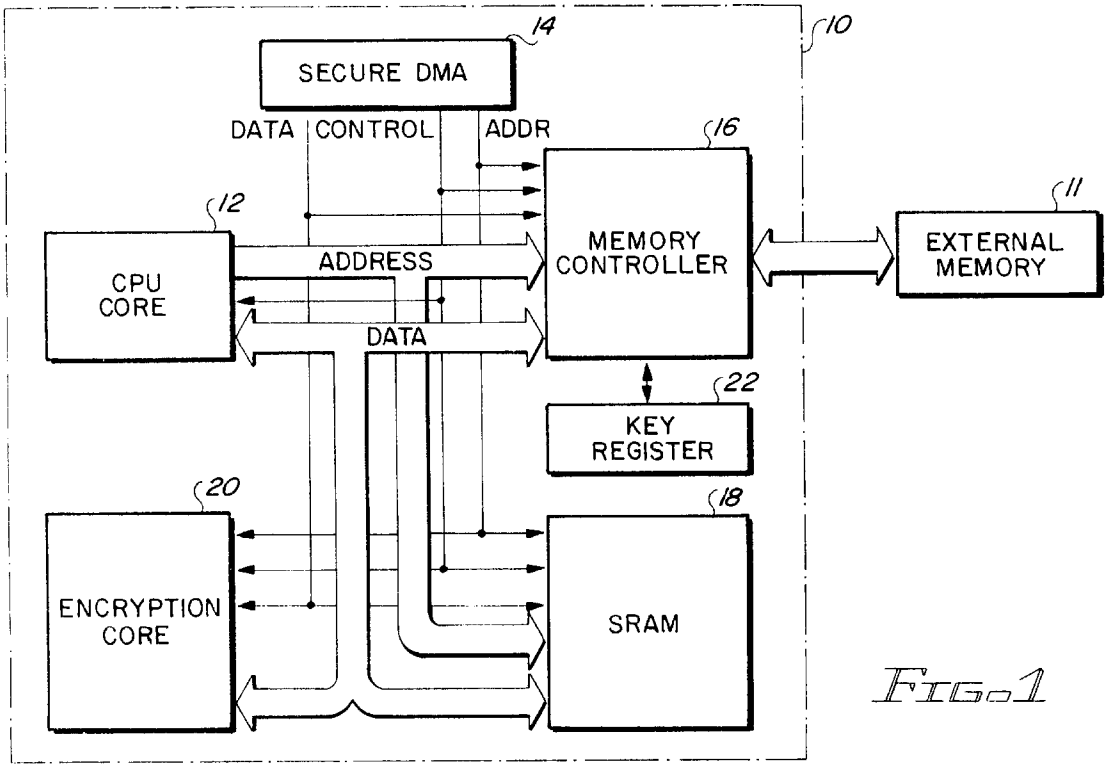


Fig.1

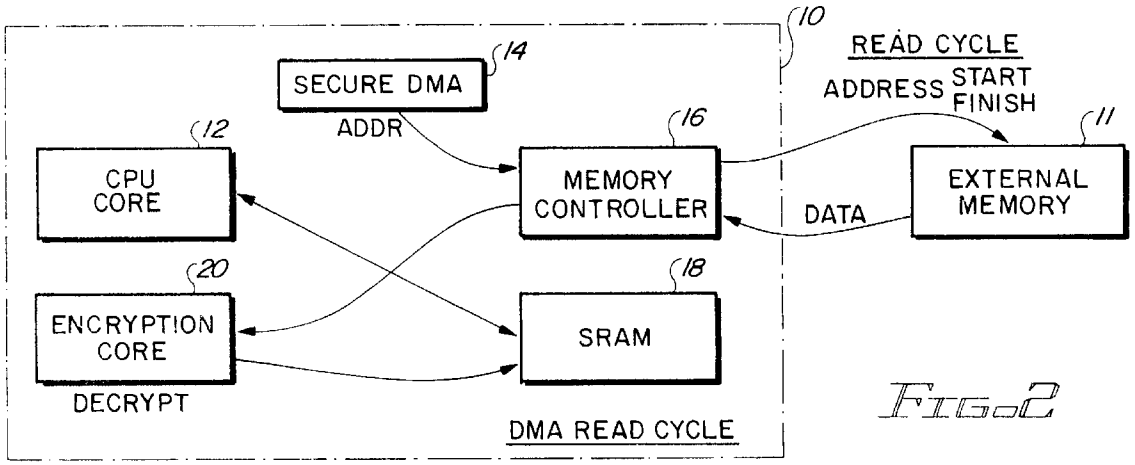


Fig.2

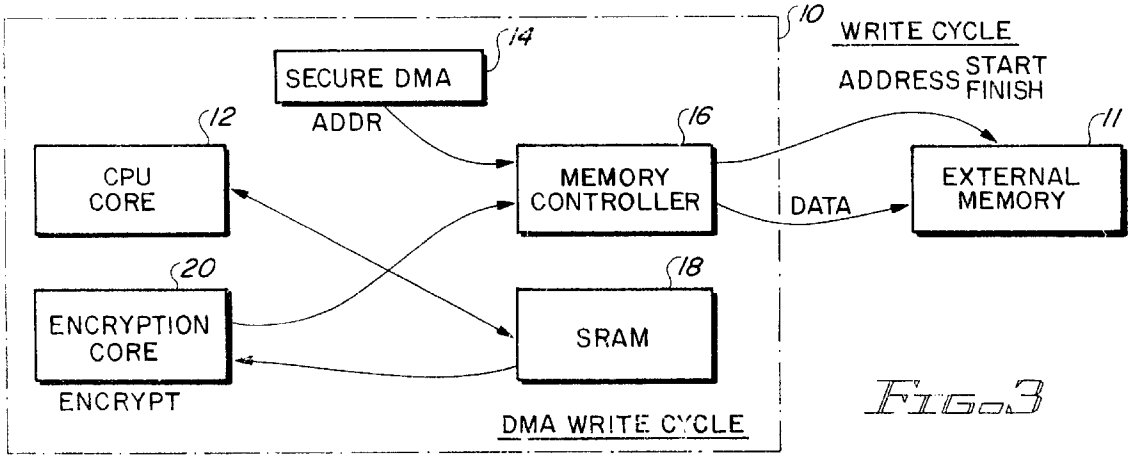


Fig.3

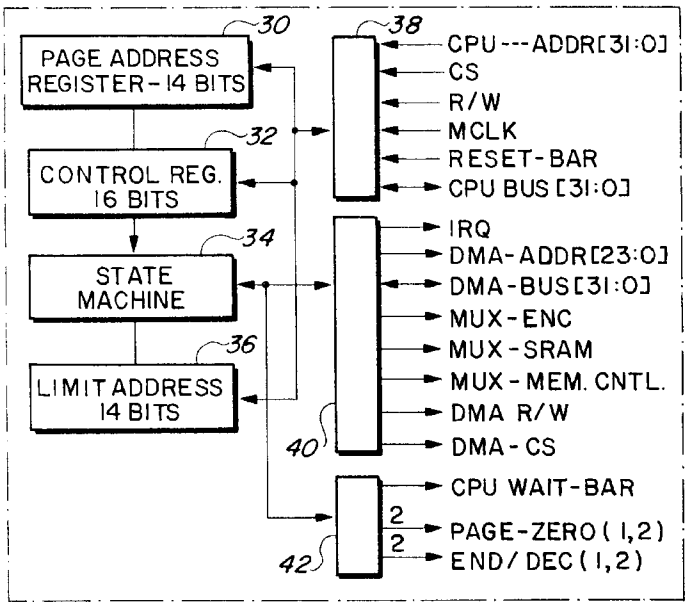


Fig-4

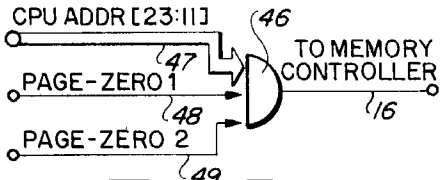


Fig-5

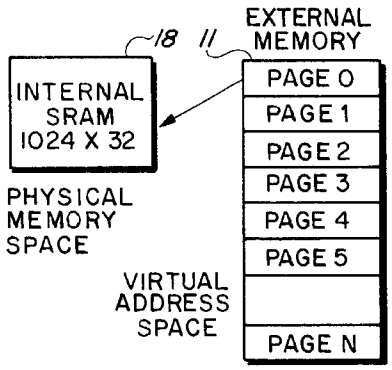


Fig-7

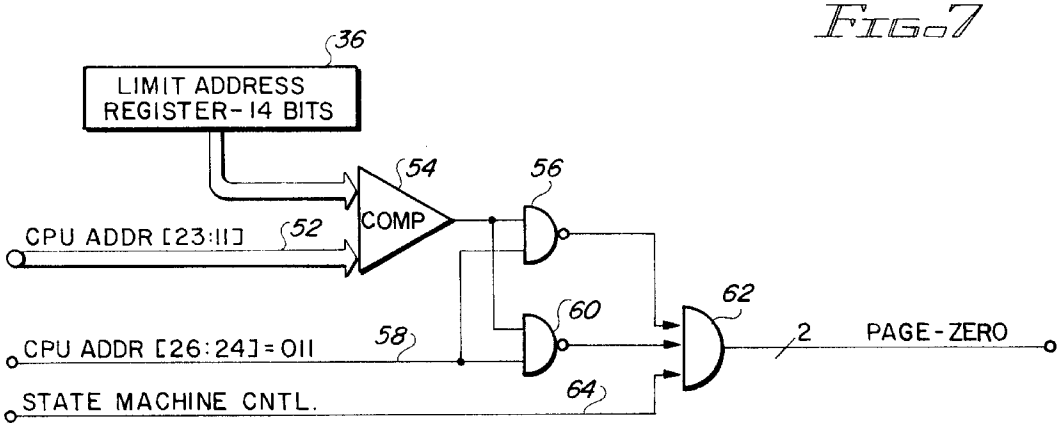


Fig-6

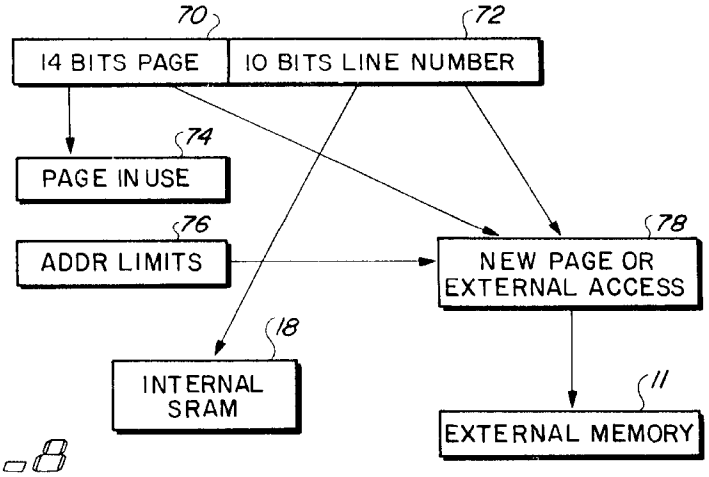


Fig-8

SECURE MEMORY MANAGEMENT UNIT FOR MICROPROCESSOR

BACKGROUND

Various techniques have been employed for decrypting and encrypting firmware stored in an external memory associated with a microprocessor system. Encryption of such firmware is used to prevent unauthorized parties from determining instructions or data stored in the memory by reading out the information and then utilizing the information. When this information is encrypted, unauthorized third parties are not able to use it unless they can obtain access to the unencrypted firmware which is handled by the microprocessor.

To prevent unauthorized access to the clear or unencrypted instructions and data, physical security measures have been developed by forming protective layers over a memory device to limit visual access to the memory, even if the encapsulation material over the chip is removed. Other techniques include employing polysilicon layers to carry the signals; so that the signal transmission is invisible. In addition, using multi-layer chips with criss-crossing signal paths makes it difficult to probe signal paths located in lower layers. As encryption/decryption circuits become more complex, however, it frequently is necessary to modify the microprocessor core in some manner in order to operate with the security systems.

The Yearsley U.S. Pat. No. 5,386,469 is directed to a firmware encryption/decryption system operating in real time to decrypt incoming code from an external memory. This is accomplished by a program counter operating in response to "enable bits" and "seed value" bits to determine when to "mask" the code using an encryption mask generator. The encryption mask of Yearsley is not a true encrypter using a DES (Data Encryption Standard) algorithm. Each clock cycle in the system of Yearsley unmasks the firmware in accordance with the seed and the program counter value in real time. In addition, some modification of the core microprocessor is necessary in order to use it with the Yearsley system.

It is desirable to provide a secure memory management unit which overcomes the disadvantages of the prior art, and which does not require any modification to the core microprocessor with which the memory management unit is used.

SUMMARY OF THE INVENTION

A secure embedded memory management unit for encrypted data and instruction transfer from an external memory includes a microprocessor core, an internal memory, a direct memory access controller and encryption core all formed in the same IC chip. The direct access memory controller is interconnected by a bus to an external memory, where the encrypted instructions and data are stored. Encrypted information supplied to the memory controller from the external memory then is supplied, internally in the chip, from the memory controller to the encryption core, where it is decrypted. The decrypted information then is supplied to the internal memory coupled to the microprocessor core. The information stored in the internal memory is utilized in a conventional manner in its "clear" form by the microprocessor core. The reverse of this operation occurs when information is to be written to the external memory. Information passing from and to the external memory is loaded on a page-by-page basis; and once a page of firmware has been loaded from the external memory onto the chip, the chip disables access to the bus to protect from any external probing.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of a preferred embodiment of the invention;

FIG. **2** is a diagrammatic flow chart illustrating the operation of embodiment of FIG. **1** for a read cycle;

FIG. **3** is a diagrammatic flow chart illustrating the operation of the embodiment of FIG. **1** for a write cycle;

FIG. **4** is a detailed diagrammatic representation of a portion of the circuit shown in FIG. **1**;

FIG. **5** is a circuit detail useful in explaining the operation of the embodiment of FIGS. **1** and **4**;

FIG. **6** is circuit detail of a portion of the embodiment of FIGS. **1** and **4**;

FIG. **7** is a diagrammatic representation of the relationship between memory space in the external memory and the internal memory of the embodiment shown in FIG. **1**; and

FIG. **8** is a diagrammatic representation of the memory organization of the internal memory of the embodiment shown in FIG. **1**.

DETAILED DESCRIPTION

Reference now should be made to the drawings, in which the same reference numbers are used throughout the different figures to designate the same components. FIG. **1** is a block diagram of a preferred embodiment of a secure memory management unit for a microprocessor system. The system shown in FIG. **1** is fabricated on a single integrated circuit chip **10** for communication with an external memory **11**, which may be of any suitable type for storing information used in the operation of the system on the chip **10**. The main function of the secure memory management unit (MMU) on the chip **10** is to read encrypted external program code instructions and data stored in the external memory **11**, to decrypt and store the information in a secure random access memory (RAM) with an internal microprocessor CPU core **12** then utilizing the information, which is stored in the secure internal RAM.

The circuit shown on the chip **10** of FIG. **1** is designed to carry out these functions. It should be noted that all of the parts shown enclosed within the dash-dot line or box **10** of FIG. **1** are fabricated on the same integrated circuit chip utilizing fabrication techniques designed to physically embed the components in the chip and to prevent access to the internal buses and connectors shown interconnecting the various components located within the box **10**.

The secure MMU of FIG. **1** comprises a memory controller **16** and a secure direct memory access controller **14**, along with an SRAM memory **18** for program storage, and a secure internal encryption core logic **20**, along with a microprocessor or central processing unit (CPU) core **12**.

Although an SRAM memory is shown, other types of read/write memories, such as EEPROM or FLASH ROM may be used as well. The memory **18** is divided into multiple cache sections of various sizes. The various buses for interconnecting these components for data, address and control signals are illustrated in FIG. **1**.

The direct memory access (DMA) controller **14** and the memory controller **16** together operate to transfer instructions between the external memory **11** and the internal SRAM memory **18**. Through appropriate instructions, the secure DMA controller **14** moves instruction from the external memory **11** through the memory controller **16** to the decryption core **20**, and finally, to the SRAM memory **18** for a read cycle. For a write cycle, the instruction is moved from

the SRAM memory **18** to the encryption core **20**, then to the memory controller **16**, and finally, from the controller **16** through a connecting bus to the external memory **11**. All of these transfers of information are controlled by the secure DMA/MMU controller **14**/**16**.

Typically, on a first external instruction access, the secure DMA controller **14** puts the CPU core **12** in a wait state mode, or the CPU core **12** executes from an internal ROM (not shown) and reads the page of external encrypted program code or data containing the requested external page address. The system operates to transfer information from and to the external memory **11** on a page-by-page basis. The page address can read or write up to 1,024×32 bit words. After the page of instructions has been written to the secure SRAM **18**, the DMA controller **14** causes these instructions to be decrypted by sequentially transferring the contents of the secure internal SRAM **18** one 32-bit word at a time to the encryption and decryption core block **20**. The cleared word is then written back to the SRAM **18**.

When the full page of instructions has been decrypted by the encryption core **20**, the DMA controller **14** takes the CPU core **12** out of the wait state mode, and the CPU core **12** reads the instruction located in the secure internal SRAM **18**. As noted, this instruction now is clear or decrypted information. If the next external instruction requested by the CPU core **12** is within the page of the secure internal SRAM **18**, the instruction is read in a single cycle from the secure internal SRAM **18**. If the next external instruction requested is not in the page of the secure internal SRAM **18**, the DMA controller **14** operates as described above, and the process is repeated. The process described may be altered, depending upon the configuration of the MMU control register consisting of the DMA controller **14** and the memory controller **16**.

The secure DMA controller **14** and memory controller **16** is the interface which provides input/output (I/O) transfer of data directly to and from the external memory **11** by way of the memory controller unit **16**, the encryption core **20**, and the internal SRAM memory peripheral **18**. The DMA controller **14** is the preferred form of data transfer for use with high speed peripheral devices to speed the encrypted instruction transfer. The CPU core **12** utilizes the DMA controller **14** by sending the selected page address to be transferred, the control configuration, and the limit addresses. This will be explained in greater detail in conjunction with FIG. **4**.

The actual transfer of data is done directly between the external memory **11** and the memory controller **16**, through the DMA controller **14**, which frees the CPU core **12** for other tasks. The major difference between an I/O program controlled transfer and the DMA controller **14** is that data transfer does not employ the registers of the CPU core **12**. The transfer is done in the DMA controller **14** interface by first checking if the memory unit **18** is not used by the CPU core **12**; and then the DMA controller **14** controls the memory cycle to access a word in the external memory **11**.

It should be noted that the system shown in FIG. **1** also employs the usual circuits of an interface, such as an address decoder, a control decoder, and state machine control logic (not shown, since these are standard components). In addition, the system uses a separate page address register, a limit address buffer register, and a page size count register (described in greater detail in conjunction with FIGS. **4**, **5** and **6**). The address and buffer registers are used for direct communication with the memory controller **16**. The page size register specifies the number of words to be transferred within a page. The CPU core **12**, with the DMA controller **14**, includes a special state machine control section for the memory controller **16** to communicate with both the CPU core **12**, the encryption core **20**, the SRAM **18**, and the DMA controller **14** on a priority basis.

Both the CPU core **12** and the DMA controller **14** can communicate with the memory controller **16**; but the DMA controller **14** has priority over the CPU core **12**. A request bit in the control register in the DMA controller **14** is set when the corresponding CPU core **12** requests a memory cycle. The memory control **16** services both the CPU core **12** and the DMA controller **14**, and resolves conflicts between the two requests. Whenever a DMA controller **14** memory cycle request is terminated, the memory controller **16** clears a corresponding request flip-flop (not shown) and the DMA controller **14** waits until a new page load memory cycle is requested.

The design of the DMA controller **14** enables, but is not restricted to, allowing the code stored in the SRAM cache **18** to be accessed by the CPU core **12** while the DMA controller **14** is loading other sections of the cache. This increases the speed of operation of the system, allowing commonly used sections of code to be semi-permanently placed into the SRAM cache **18**. This is accomplished by dividing the SRAM cache **18** into various sections of different sizes. Implementation of this feature may be accomplished by means of a hard division of the cache or a flexible division where the CPU core **12** and the DMA controller **14** access the memory **18** on alternate cycles of the system clock. Collision detection circuitry or software (not shown) also may be employed to prevent access to pages in the SRAM **18** while these pages are being loaded or allow accesses to the pages being loaded when a piece of memory already has been loaded into the cache **18**. Such detection circuitry recognizes a page as "in process", "loaded", or "unloaded". The unloaded and partially loaded pages operate in the same manner where the offset to the page is compared to a pointer loading data. The wait signal (shown in FIG. **4**) is released once the actual data is loaded. For a new cycle, this occurs when the first instruction is loaded and continues a wait/load cycle until the code jumps out of this particular page. For a page in process, the code actually may be able to run a full routine and branch out prior to ever having a collision or a wait cycle.

The initialization process for the system essentially is a program consisting of I/O instructions that include the page address and command codes for the DMA **14** interface. The CPU core **12** checks the status of the peripheral (external memory **11**) and the DMA/MMU controller **14**/**16**; and if all is in order, the CPU core **12** sends the following information through the I/O memory controller lines:

1. The starting address of the page memory block where the instructions or data are available (for output) or where data are to be stored (for input) in the SRAM **18**.
2. The page size, which is the number of words in the page memory block.
3. A control specifying an input or output transfer.
4. A command to start the DMA/MMU controller **14**/**16**.

The starting page address, page size, and the control specifying the direction of transfer are stored in designated control registers in the DMA/MMU controller **14**/**16**. The CPU core **12** then stops communicating with the DMA/MMU controller **14**/**16**. The DMA controller **14**, which controls the memory controller **16**, handles all of the housekeeping operations such as packing characters into words (for output) or unpacking words into characters (for input) and checks the status of the peripheral external memory **11**.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.