International Journal of Electronics Communication and Computer Engineering
Volume 5, Issue (4) July, Technovision-2014, ISSN 2249–071X

Technovision-2014: 1st International Conference at SITS, Narhe, Pune on April 5-6, 2014

Design and Development of a Segmentation Unit on PLD

Dipannita. S. Neogi
E&TC Department,
Sinhgad College of Engineering,
Vadgaon (BK), Pune-41, India
Email: neogi_dipannita@yahoo.co.in

Prof. U. R. More
E & TC Department,
Sinhgad College of Engineering,
Vadgaon (BK), Pune-41, India
Email: umashankar.more@gmail.com

Prof. J. B. Jagtap
E & TC Department,
K.B.P Polytechnic College of Engineering,
Satara, Pune-001,India
Email: jagtapsir@gmail.com

***Abstract* – Virtual Memory has hardware support in modern processors now days. In this work address translation mechanism is explored as a hardware implementation. A segmentation unit is designed and developed which maps virtual to linear address. In this scheme, the values present in the relocation register is added to each and every address generated by a user process at the time when it is sent to memory. The user program generally deals with logical addresses rather than real physical addresses. The actual logical address to physical address translation is done within the MMU. The segmentation unit is implemented designing the descriptor tables. In segment translation scheme the virtual address space is divided into different logical segments where each is a part of physical memory.**

***Keywords* – Design and Development, Segmentation Unit, PLD, Virtual Memory, MMU, Hardware.**

I. Introduction

Virtual memory refers to the technology in which some space in hard disk is used as an extension of main memory so that a user program need not worry if its size extends the size of the main memory. If that does happen, at any time only a part of the program will reside in main memory, and other parts will otherwise remain on hard disk and may be switched into memory later if needed. In this work a virtual address also called as logical address is translated into linear address using segment translation unit.

A. Problem Statement

To design and develop segmentation unit which can translate virtual address requested externally to linear address so that large program can be handled in the memory itself. The segmentation unit needs to be implemented in software using VHDL and ISE /ModelSim Simulator Software's and later on fitted into PLD.

B. Block Diagram

Fig.1 shows the overall block diagram of a segmentation unit where the virtual address also termed as logical addresses are externally given and logical addresses get translated to linear addresses. The virtual address fields contain segment and offset where the segment id indicates the base address of the segment table. The segment table consists of limit and base fields. The limit field marks the size of the segment and the base field indicates the segment base output address. This segment base out gets added to the offset and generates the linear address.

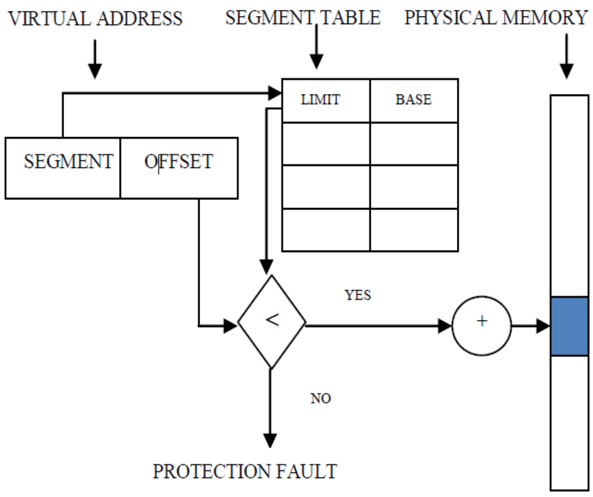

Fig.1. Block Diagram of Segmentation Unit

II. Literature Review

In today's CMOS VLSI technology makes it possible to combine a very powerful processor onto a single chip. Although component-level analysis was well understood, system level models were difficult to develop. The behavior of the system under real workloads was extremely difficult to predict and model. However, it is not easy to predict the impact of a memory-component failure on the computer system. This depends on behavioral aspects such as the contents of the memory, the usage of the memory, and the operating system recovery techniques. The memory-access behavior of programs to determine the likelihood of an error causing a failure was studied [1]. It was found that the access behavior could account for an important number of unobserved faults. It suggested that traditional memory reliability analysis can be fairly pessimistic in regard to field experience.

In 1995, scientists worked upon micro architecture of HAL's memory management unit [2].The HAL MMU is responsible for the functions such as address space translations and hardware handling, protection violation checking, data movement controls and bus interfaces among, exception handling memory coherency among caches and memories, diagnostic and the functions for caches and memories.

In recent research in high-speed network interfaces for commodity networks has focused on removing the operating system from the critical path for sending and receiving messages. An effective solution is to provide user-level messaging so user applications can directly access the hardware of the network while keeping the remaining protected from one another. This allows messages to be sent from and received into user space without kernel intervention. Implementation difficulties arise in the mapping between the virtual addresses of message buffers specified by applications and the physical addresses required for actual transmission and reception. The network interface must be able to translate the virtual buffer addresses to physical addresses, and the translations must be coordinated with the operating system's virtual memory subsystem.

Intellectual Property (IP) core-based hardware-software systems has emerged as a result of a new design paradigm of System-on-Chip (SoC). SoC is usually defined as an integration of complex functional modules or cores, where each core is complex enough to be a complete IC in itself. In order for these IP cores to be as highly reusable as possible, the cores must be soft cores in a synthesizable Hardware Description. Language (HDL) form and can be targeted for different semiconductor process technology. The traditional approach to system design involves combining a microprocessor and other devices on a single chip as in circuit board. Currently advanced submicron technologies enable a complete design on a single chip. Increasing density and speed of Field Programmable Gate Arrays (FPGAs) leads to adoption of IP core-based System-on-Chip (SoC) designs. Since processors are common in system design, integrating them with other functions into single device is the result of these advances. RISC instruction sets gradually have moved to ClSC sets during the 1980s. After twenty years and invention of SoC processors, RISC machines are gaining more importance due to the fact that only 25% of the instructions of a complex instruction set are frequently used about 95% of the time [3]. A RISC instruction set generally contains less than 100 instructions with fixed-length format such as 32 bits. Only three to five addressing modes are used. Most instructions are register-based. Memory access is done by load store instructions SoC approach encourages design engineers to adopt existing IP cores. This promotes reuse. Modern IP core library typically includes features for specific applications such as communication ports, image processing units, and Floating Point Units (FPUs). Nowadays, object-oriented software applications are getting dynamic memory intensive[4]. This creates the need of high-performance memory allocator and deallocator as a core extension. For example, Active Memory Management Unit (AMMU) provides high performance in memory management [5]. AMMU uses hardware accelerated dynamic memory management algorithm based on modified buddy system [6].

III. Experimentation

The segmentation unit is provided with an input of virtual address which is responsible for generating the linear address. A descriptor table is implemented of 512K out of which 256 K is global descriptor table which stores all the drivers information and rest 256 K is used as a local descriptor table which is used to store application codes like notepad etc.

A. Segment Translation

Figure 2. shows in detail the way system converts a logical address into a linear address. The virtual address also called as logical address is divided into selector and offset. The selector used is a 16 bit selector and offset used is a 32 bit offset address. In the work out of 8192 descriptors used by Intel[7] only 512 descriptors have been used due to hardware restrictions. Segment descriptors are stored in either of two kinds of descriptor table one is global descriptor table (GDT) responsible for storing the system code and drivers and other is a local descriptor table (LDT) responsible for storing the application code like notepad. Out of 512 descriptors 0 to 255 is used as Global Descriptor Table and 256 to 511 is used as Local Descriptor Table. These descriptors are created by applications programmers.

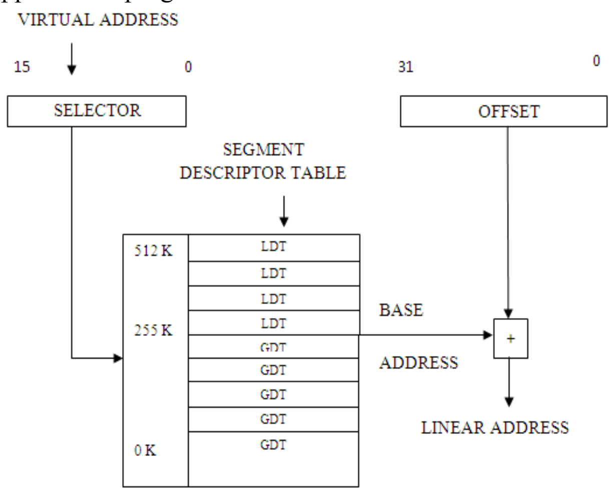

Fig.2. Virtual to Physical Address Translation

B. Algorithm

- Start
- Create descriptor table for storing segment descriptors
- Set table selection bit to 0 for accessing GDT and 1 for accessing LDT
- For rising clock edge and write enable initialized to value 1 update selected descriptor.
- For falling clock edge and write enable initialized to value 0 get requested descriptor.
- Calculate linear address as sum of base address and offset
- End

C. Flowchart

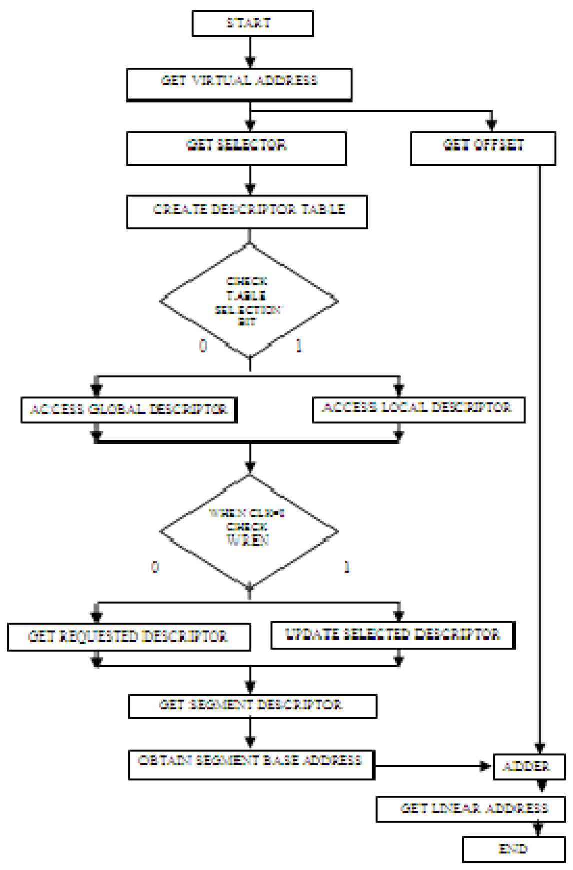

Fig.3. Segmentation Flowchart

IV. Results and Discussions

The results of the segmentation unit is discussed in this chapter and various levels of RTL are shown. The segmentation unit is a unit where the logical or virtual address is translated to linear address. The input to the unit is a 32 bit logical address where the selector is of 16 bit and offset is 32 bit which is later on added to the segment output base address. The MSB 12 bits of selector is selector index which indicates the appropriate segment from the segment descriptor table. The third bit of selector indicates the table selection bit and if this bit is one then it refers to local descriptor table and if zero to global descriptor table.

A. Initialization of the system

The contents of the selector starts varying from x“0000”, x“0010”, x“0020” so on and the contents of offset are x“00000000” to x“00000100”. 8K is the maximum number of descriptor entries that can be accessed through selector but only 512 descriptors are used due to hardware restrictions. The memory depth of descriptor table is 512 words and width is 64 bits. The input to the descriptor table is a 16 bit selector and a 64 bit descIn which is two 32 bit segment base inputs consolidated to form a 64 bit input given to the descriptor table. Table selection bit is set to be the most significant bit in address therefore base of GDT is 0x000 and base of LDT is 0x100 and also due to maximum number of entries it is not possible to access out of range descriptors so there is no need to define limits. When selector is x“0010”, offset is x“00000000” and segbasein is x“10000000” then segbaseout is x“18000000” then the linAddr obtained is a summation of segment base out and offset i.e x“18000000”. Similarly when selector is x“0008” ,offset is x“00000100” and segbasein is x“180000000” then segbaseout is x“18000000” then the linAddr obtained is again a summation of segment base out and offset i.e x“18000100”after two clock cycles.

B. Top Level RTL Schematic

In the Figure 4. all the inputs given to segment translation is translating virtual address to linear address which is obtained here on output linAddr as 32 bit address.

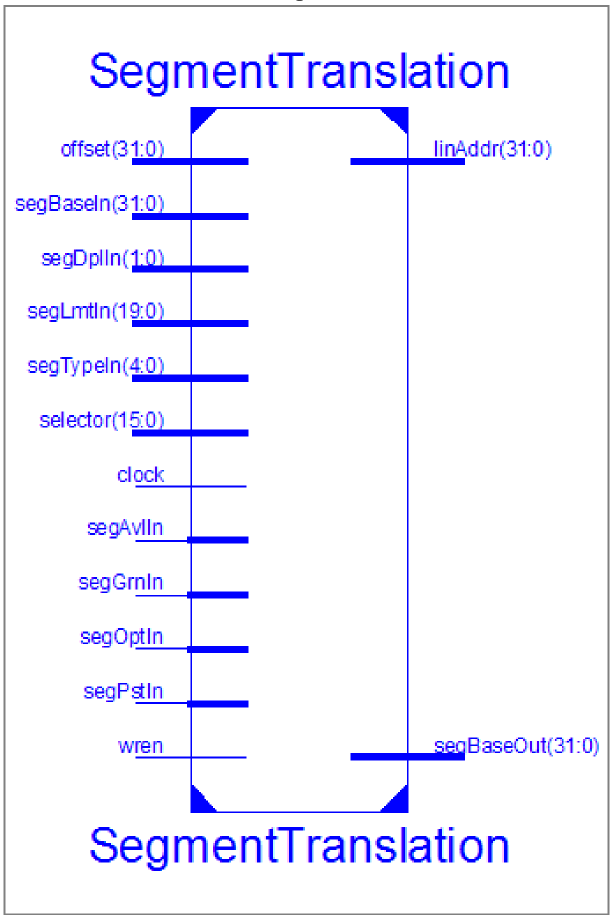

Fig.4. Top Level RTL Schematic

C. Second Hierarchy of RTL

Figure 5 shows the second hierarchy of segmentation unit that converts logical or virtual address to linear address. The descriptor table is implemented which is responsible for storing all the segment descriptors.

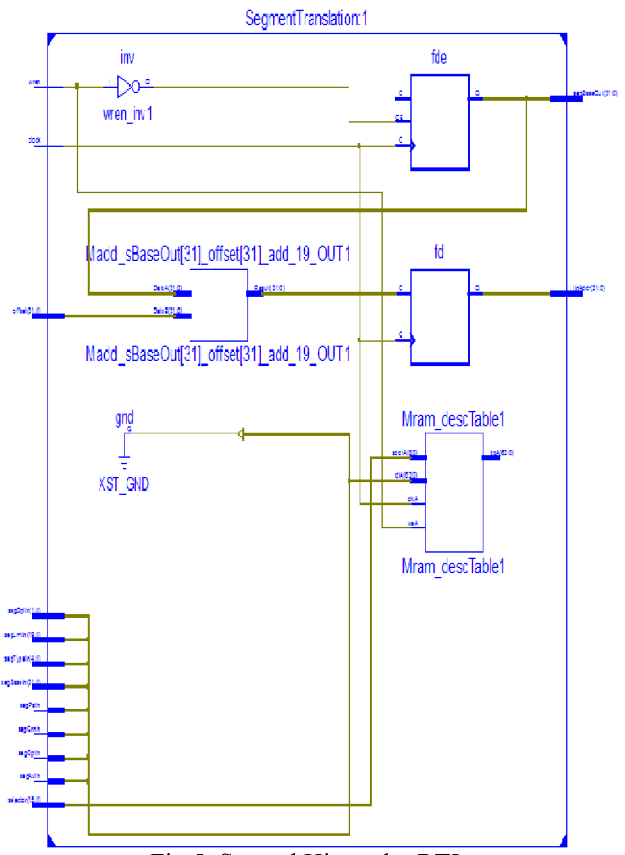

Fig.5. Second Hierarchy RTL

D. Simulation Results

Initialization of segment descriptor tables with present field set to zero is done so that access to any segment by program fails to access invalid segment descriptor. This is done by getting into write mode with wren signal initialized as “1” and setting segpstin bit to “0”. This clears all the entries of Global Descriptor Table (GDT) and Local Descriptor Table (LDT). Then getting back to read mode with wren bit initialized to “0” some entries are checked in order to identify whether all the entries have been cleared. Getting back to write mode some entries are created which checked later and so for creating GDT descriptor 1 the third bit of selector signal is set to “0”, segGrnIn bit set to „0‟ to indicate granularity set to byte, segPstIn bit to '1' indicating segment is present, segBaseIn is initialized to x"18000000" with segTypeIn set to "01000" which is set to indicate segment is of type 8 (system program) ,segDplIn set to "11" set to indicate segment privilege level to 3,segLmtIn set segment to limit 1K bytes. Similarly creating LDT Descriptor 0" again selector(3) bit is set to '1',segGrnIn bit is set to'1' to set granularity to 4K byte, segPstIn set to '1' to indicate segment is present,segBaseIn is set to value of x"10000000" with segTypeIn set to "00100" indicating type 4(application program), segDplIn set to "01";indicating privilege level to 1,segLmtIn set to limit 1M(4K x 256) bytes.Getting back to read mode both the entries are checked. The output is given by linAddr which is linear address where the segBaseout is added to offset. Thus segmentation of virtual to linear address is obtained and in the waveform different values for linAddr output for different segBaseIn values are seen.

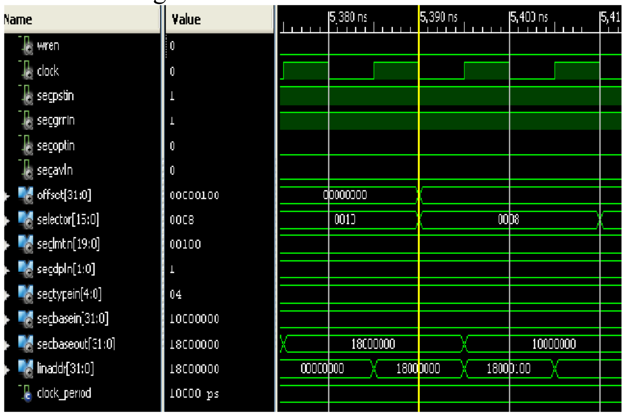

Fig.6. Simulation Result

E. Synthesis Result

As seen in the Table 1. the number of slice registers used is 32 out of 93120 so nearly it’s a 0% utilization. Similarly number of slice LUT’s used is 32 out of 46560 and so 0% utilization is done. But 100 % utilization is there for number of fully used LUTFF pairs all 32 FF’s are used. The number of bonded IOBs is used around 70% and block RAM/FIFO utilization is just 1 % and number of BUFG/BUFGCTRLs is 3%.

Table 1: Device Utilization for Segmentation Unit for VIRTEX 6

Device Utilization Summary (Estimated Values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	32	93120	0%
Number of Slice LUTs	32	46560	0%
Number of fully used LUT-FF pairs	32	32	100%
Number of bonded IOBs	170	240	70%
Number of Block RAM/FIFO	2	156	1%
Number of BUFG/BUFGCTRLs	1	32	3%

REFERENCES

[1] N.S. Bowen, D.K. Pradhan, “Program fault tolerance based on memory access behavior”, 21 Sf Symp. Fault-Tolerant Computing, 1991 June, pp.

[2] David Chih-Wei Chang, David Lyon, Charles Chen, Leon Peng, Mehran Massoumi, Matthew Hakimi, Satish Iyengar, Ellen Li, Roque Remedios “Microarchitecture of HaL‟a memory management unit,1995

[3] K. Hwang, “Advanced Computer Architecture: Parallelism, Scalability, and Programmability”, McCraw-Hill, I993

[4] J. M. Chang, W. H. Lee, "A study on memory allocations in C++", Proceedings of 4th International Conference on Advance Science and Technology, Naperville, Illinois, April 4-5, 1998. pp. 53-62.

[5] W. Srisa-an, C. D. Loo, and J. M. Chang "A Performance Analysis of the Active Memory Module (AMM)", to appear in Proceedings of IEEE International Conference on Computer Design, Austin, Texas, Sep. 23-26, 2001,

[6] J. M. Chang, E. F. Gehringer, "A High-performance memory allocator for object-oriented systems", IEEE Transaction

[7] Reference manual by Intel for 80386 .