© 2014 IJIRT | Volume 1 Issue 6 | ISSN : 2349-6002

A Review Paper on Design of DMA Controller Using VHDL

Vibhu Chinmay , Shubham Sachdeva

Student (B.Tech 5th sem) Department of Electronics and Computers Engineering

Dronacharya College of Engineering, Gurgaon-123506, India

Abstract: **In this paper, an attempt has been made to review the design of Direct Memory Access (DMA) Controller using VHDL. Direct Memory Access is a method of transferring data between peripherals and memory without using the CPU. The 8237A Multimode DMA Controller is a peripheral interface circuit for microprocessor systems. It is designed to improve system performance by allowing external devices to directly transfer information from the system memory. Memory-to memory transfer capability is also provided. The 8237A offers a wide variety of programmable control features to enhance data throughput and system optimization and to allow dynamic reconfiguration under program control.**

Keywords: DMA, VHDL, IP Core.

I.INTRODUCTION

Many system-on-chip (SoC) integrated circuits contain embedded cores with different scan frequencies. To better meet the test requirements for such heterogeneous SoCs, leading tester companies have recently introduced port-scalable testers, and Many IP core design software like Xilinx, Leonardo Spectrum, and Modelsim etc. which can used to design IP core like DMA ,Interrupt Controller etc. These IP core can be Power aware an Implement on SoC by choosing different design technique and various modeling techniques .These all modeling technique and tolls like Xilinx ISE also provide RTL view which will help to make IP cores to use in any Processor design .Here in these project we will see an Intel 8237 DMA IP core design which is using a very different kind of design technique not used up till now. So by this project we will prove that if we are trying & use various modeling techniques for designing IP cores than it may be used in various power level requirement circuits & processors & it may also power aware. These IP cores are ASIC application specific IC so we can control its power, speed, size etc. to implement before on an embedded circuit. So an IP core design is a part of a Main embedded circuit and control the working of that circuit or processor.

Today's SoCs are composed of a wide variety of modules, such as microprocessor cores, memories, peripherals, and customized blocks directly related to the targeted application. To effectively perform simulation-based design verification of peripheral cores, it is necessary to stimulate the description in a broad range of behavior possibilities, checking the produced results. Different strategies for generating suitable stimuli have been proposed by the research community to functionally verify these modules and their interconnection when embedded in a SoC: however, their verification often remains a largely manual and unstructured operation. In this paper we describe a general approach to develop concise and effective sets of inputs by modeling the configuration modes of a peripheral with a graph, and creating paths able to cover all of its nodes: proper stimuli for the device are then directly derived from the paths. The resulting inputs sequences are aimed at design verification of system peripherals such as DMA controllers, and can be applied via simulation by means of dedicated test-benches or by setting up an environment including a processor, which executes a proper test program. In the latter case, the developed programs can be exploited in later stages for testing, by adding suitable observability features. Experimental results demonstrating the method effectiveness are reported.

IP Core (Intellectual Property)

An IP (intellectual property) core is a block of logic or data that is used in making a field programmable gate array (FPGA) or application-specific integrated circuit (ASIC) for a product. As essential elements of design reuse, IP cores are part of the growing electronic design automation (EDA) industry trend towards repeated use of previously designed components. Ideally, an IP core should be entirely portable - that is, able to easily be inserted into any vendor technology or design methodology. Universal Asynchronous Receiver/ Transmitter (UART's), central processing units (CPU's), Ethernet controllers, and PCI interfaces are all examples of IP cores. IP cores fall into one of three categories: *hard cores*, *firm cores*, or *soft cores*. Hard cores are physical manifestations of the IP design. These are best for plug-and-play applications, and are less portable and flexible than the other two types of cores. Like the hard cores, firm (sometimes called *semi-hard*) cores also carry placement data but are configurable to various applications. The most flexible of the three, soft cores exist either as a net-list (a list of the logic gate s and associated interconnections making up an integrated circuit) or hardware description language (HDL) code. In electronic design a semiconductor intellectual property core, IP block, IP core or logic core is a reusable unit of logic, cell, or chip layout design that is the intellectual property of one party. IP cores may be licensed to another party or can be owned and used by a single party alone. The term is derived from the licensing of the patent and source code copyright intellectual property rights that subsist in the design. IP cores can be used as building blocks within ASIC chip designs or FPGA logic designs.

DMA Controller

Direct memory access (DMA**)** is a process in which an external device takes over the control of system bus from the CPU.

DMA is for high-speed data transfer from/to mass storage peripherals, e.g. hard disk drive, magnetic tape, CD-ROM, and sometimes video controllers. For example, a hard disk may boast a transfer rate of 5 M bytes per second, i.e.1 byte transmission every 200 ns. To make such data transfer via the CPU is both undesirable and unnecessary.

The basic idea of DMA is to transfer *blocks of data* directly between memory and peripherals. The data don't go through the microprocessor but the data bus is occupied. "Normal" transfer of one data byte takes up to 29 clock cycles. The DMA transfer requires only 5 clock cycles. Nowadays, DMA can transfer data as fast as 60 M byte per second. The transfer rate is limited by the speed of memory and peripheral devices. A DMA controller interfaces with several peripherals that may request DMA. The controller decides the priority of simultaneous DMA requests communicates with the peripheral and the CPU, and provides memory addresses for data transfer. DMA controller commonly used with 8088 is the 8237 programmable device. The 8237 is in fact a special purpose microprocessor.

Normally it appears as part of the system controller chip-sets. The 8237 is a 4-channel device. Each channel is dedicated to a specific peripheral device and capable of addressing 64 K bytes section of memory.

DMA 8237 IP Core

The 8237 programmable DMA controller core is a peripheral interface circuit for microprocessor systems. The core is designed to be used in conjunction with an external 8-bit address latch. It contains four independent channels and may be expanded to any number of channels by cascading additional controller chips. Each channel has a full 64-K address and word count capability.

Some Important Signal Pins

DREQi (DMA request): Used to request a DMA transfer for a particular DMA channel.

DACKi (DMA channel acknowledge): Acknowledges a channel DMA request from a device.

HRQ (Hold request): Requests a DMA transfer.

HLDA (Hold acknowledge): signals the 8237 that the Micro-processor has relinquished control of the address, data and control buses.

AEN (Address enable): Enables the DMA address latch connected to the 8237 and disable any buffers in the system connected to the microprocessor. (Use to take the control of the address bus from the microprocessor).

ADSTB (Address strobe): Functions as ALE to latch address during the DMA transfer.

EOP (End of process): Signals the end of the DMA process.

IOR (I/O read): Used as an input strobe to read data from the 8237 during programming and used as an output strobe to read data from the port during a DMA write cycle.

IOW (I/O write): Used as an input strobe to write data to the 8237 during programming and used as an output strobe to write data to the port during a DMA read cycle.

MEMW (Memory write): Used as an output to cause memory to write data during a DMA write cycle.

MEMR (Memory read): Used as an output to cause memory to read data during a DMA read cycle.

Internal Register

The current address register (CAR) is used to hold the 16-bit memory address used for the DMA transfer. The current word count register (CWCR) programs a channel for the number of bytes (up to 64K) transferred during a DMA action. The base address (BA) and base word count (BWC) registers are used when auto-initialization is selected for a channel. In this mode, their contents will be reloaded to the CAR and CWCR after the DMA action is completed. Each channel has its own CAR, CWCR, BA and BWC. The command register (CR) programs the operation.

8237 DMA Controller

The mode register (MR) programs the mode of operation for a CHANNAL the request register (RR) is used to request a DMA transfer via software, which is very useful in memory-to-memory transfers. The mask register set/reset (MRSR) sets or clears the channel mask to disable or enable particular DMA channels. The mask register (MSR) clears or sets all of the masks with one command instead of individual channels as with the MRSR. The status register (SR) shows the status of each DMA channel.

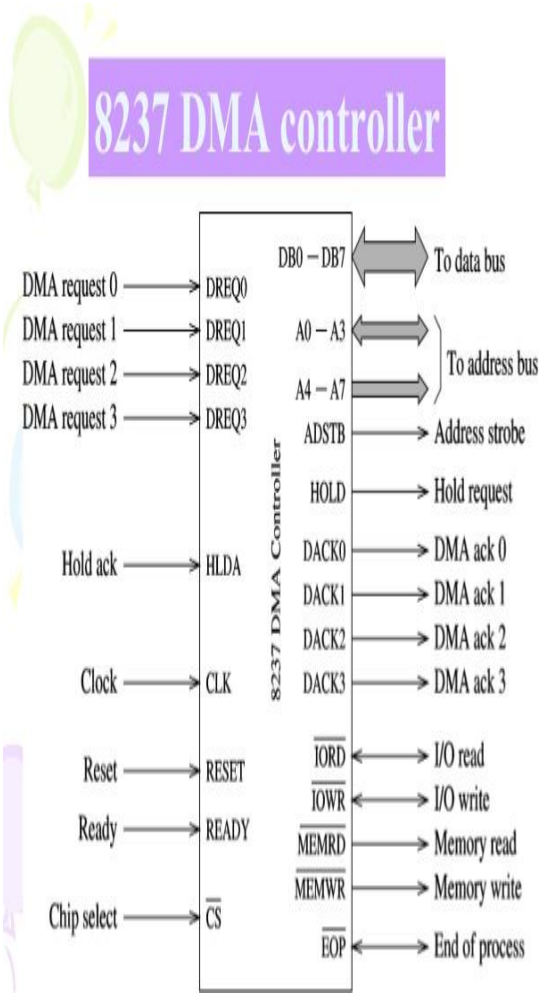

Figure 1: 8237 DMA Controller

Basic DMA Operation

The direct memory access (DMA) I/O technique provides direct access to the memory while the microprocessor is temporarily disabled. A DMA controller temporarily borrows the address bus, data bus, and control bus from the microprocessor and transfers the data bytes directly between an I/O port and a series of memory locations. The DMA transfer is also used to do high-speed memory-to memory transfers. Two control signals are used to request and acknowledge a DMA transfer in the microprocessor-based system.

The HOLD signal is a bus request signal which asks the microprocessor to release control of the buses after the current bus cycle. The HLDA signal is a bus

grant signal which indicates that the microprocessor has indeed released control of its buses by placing the buses at their high-impedance states. The HOLD input has a higher priority than the INTR or NMI Interrupt input.

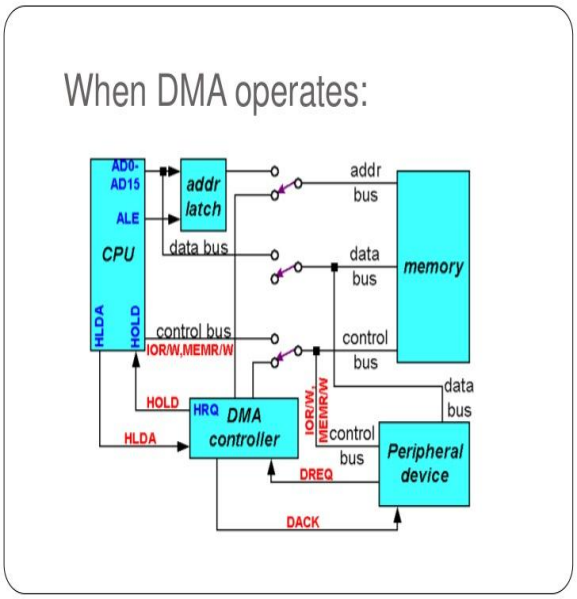

Figure 2: When DMA Operates

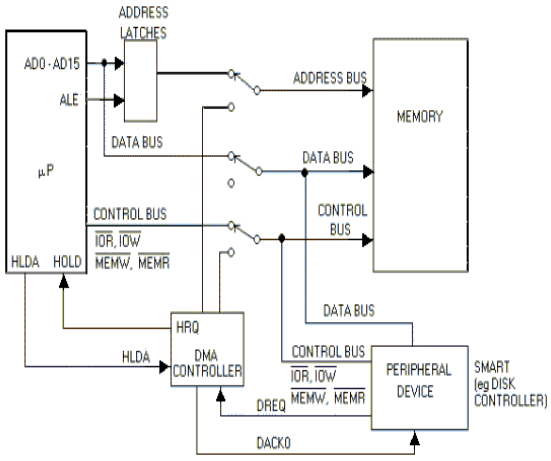

Figure 3: When DMA Does Not Operate

The following sequence of steps takes place for a DMA transfer in cycle-stealing mode:

- Each time the peripheral is able to transfer a byte it asserts its DMA request line to the DMA controller.
- The DMA controller asserts the CPU's hold request pin.
- When the CPU control circuitry is able to suspend execution (at the end of an instruction or by inserting wait states in T3) it asserts the hold acknowledge (HOLDA) signal to the DMA controller and floats the address, data and control bus signals.
- The DMA controller then puts the memory address on the address bus, asserts either MEMR* plus IOW* or MEMW* plus IOR* on the control bus and asserts the appropriate DMA acknowledge line to the peripheral.
- The peripheral responds to the DMA acknowledge signal by reading or writing it's data to the data bus
- At the same time the memory responds to the MEMR*/MEMW* control signal which causes the data to be read/written directly from/to memory.
- At the end of the bus cycle the DMA controller then negates hold request line and the CPU can continue to execute until the next DMA request.

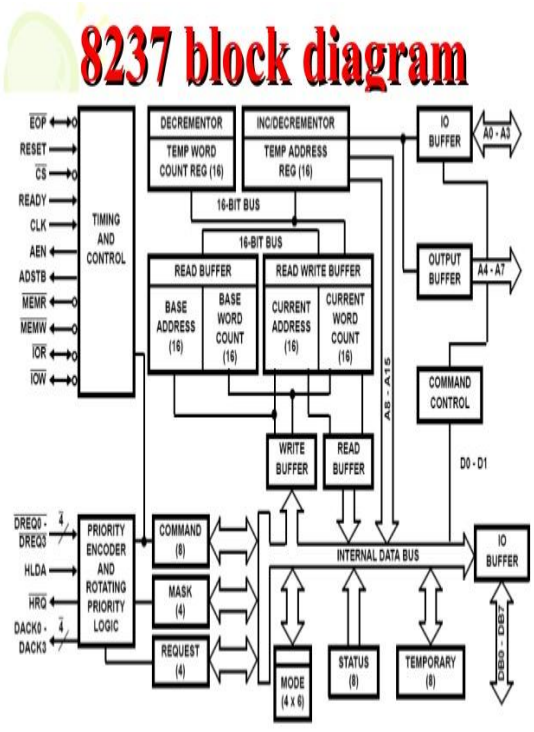

Figure 4: Block Diagram

VHDL

VHDL (*VHSIC hardware description language)* VHSIC: very high-speed *integrated circuit* is a hardware description language used in electronic design automation to describe digital and mixed-signal systems such as field-programmable gate arrays and integrate circuits.

VHDL is a fairly general-purpose language, and it doesn't require a simulator on which to run the code. There are many VHDL compilers, which build executable binaries. It can read and write files on the host computer, so a VHDL program can be written that generates another VHDL program to be incorporated in the design being developed. Because of this general-purpose nature, it is possible to use VHDL to write a *testbench* that verifies the functionality of the design using files on the host computer to define stimuli, interacts with the user, and compares results with those expected.

It is relatively easy for an inexperienced developer to produce code that simulates successfully but that cannot be synthesized into a real device, or is too large to be practical. One particular pitfall is the accidental production of transparent latches rather than D-type flip-flops as storage elements.

VHDL is not a case sensitive language. One can design hardware in a VHDL IDE (such as Xilinx or Quartus) to produce the RTL schematic of the desired circuit. After that, the generated schematic can be verified using simulation software (such as ModelSim) which shows the waveforms of inputs and outputs of the circuit after generating the appropriate testbench. To generate an appropriate testbench for a particular circuit or VHDL code, the inputs have to be defined correctly. For example, for clock input, a loop process or an iterative statement is required.

The key advantage of VHDL when used for systems design is that it allows the behavior of the required system to be described (modeled) and verified (simulated) before synthesis tools translate the design into real hardware (gates and wires). Another benefit is that VHDL allows the description of a concurrent system (many parts, each with its own sub-behavior, working together at the same time). VHDL is a Dataflow language, unlike procedural computing languages such as BASIC, C, and assembly code, which all run sequentially, one instruction at a time.

A final point is that when a VHDL model is translated into the "gates and wires" that are mapped onto a programmable logic device such as a CPLD or FPGA, and then it is the actual hardware being configured, rather than the VHDL code being "executed" as if on some form of a processor chip.
There are various types of Modeling in VHDL Language.

Conclusions

This Project will also provide the knowledge that how to start and design an Processor or IP This project will provide knowledge about the DMA 8237 and also give knowledge about the IP core the cost of IP core is very high in today's Market so the project is beneficial for me as far as it will use in many industries. This design is also used such kind of modeling style and tool that the digits is also power aware in generation type DMA's core.

References

1. Intel 8237 data sheet
2. Design of Two-Dimension DMA Controller in Media Multi-Processor SoC files
3. Direct Memory Access and DMA-controlled I/O
4. Modern development of DMA
5. Device Driver and DMA Controller Synthesis
6. Asynchronous System Bus Enhancement by Interrupt and DMA Technique
7. The research of a parallel DMA control mechanism in DSP
8. DMA control and transmission signal control
9. System-on-Chip (SoC) for Hand-held Compute-intensive Embedded Systems
10. DMA-Aware Memory Energy Management.
11. Stepwise Refinement of Behavioral VHDL Specifications by Separation of Synchronization and Functionality
12. www.arm.com, www.amba.com, www.intel.com