USB DMA INTERFACE FOR DEBUGGING VIDEO APPLICATION

Dhanesh M S
Asst.Professor,Dept of ECE
Rajagiri School of Engg & Tech
Cochin, India
dhaneshms@rajagiritech.ac.in

Tollin Jose
M.Tech student,Dept of ECE
Rajagiri School of Engg & Tech
Cochin, India
tollinjose@yahoo.co.in

Abstract— USB DMA is a design for testability included along with human machine interface to improve its final testing. The design for testability adds certain testability features to a microelectronic hardware product design. The premise of the added features is that they make it easier to develop and apply manufacturing tests for the designed hardware. USB DMA interface establish a direct communication of human machine interface with the host system. So it becomes possible to send data corresponding to image and video directly into the devise. It imitates the behavior of the video processing core, there testing of display driver become possible without video processing core. The core receives input video data from the host system, it transfer the data serially through the USB port to the USB DMA interface. The first state of USB DMA is the EZ-USB Cypress chip which convert input serial data into parallel form. Then this parallel data is processed and finally convert it to AHB standard and stores it in the external memory.

I. INTRODUCTION

Many new and exciting innovations, such as HDTV, High resolution mobile phones, digital camera revolve around video and image processing and this technology's rapid evolution. Leaps forward in image capture and display resolutions, advanced compression techniques, and video intelligence are the driving forces behind the technological innovation

We see this technological advancement in the electronic products around us. Of course the most advanced video processing system is used in Smart Televisions. But there are other electronic products where more sophisticated video processors are used. The technology advancement necessitated the importance of video processing core which is exclusively dedicated for the processing of video signals. High data application requires larger bandwidth, bus width and processor capable to process high data content in the minimum time. With expanding resolutions and evolving compression, there is a need for to capture video input stream and give an output video formatted in variety of digital video formats. The Video core has memory requirements for buffering the data when moved into the core and also when data is outputted. After processing video data in the respective format is moved into the external memory. Any other module can fetch the data from this external memory whenever required.

USB DMA module is a design for testability included along with the video processing core used in the Human machine interfaces. It acts as link between host system and memory module by which direct data transfer from the host system to memory module is made possible. So USB DMA module improves the speed and testing accuracy of the module. The memory data corresponding to particular video frame can be read from the memory by the display driver. Thus debugging of Video processing module is made possible. The USB DMA is a programmable data pump used to transfer data from the USB endpoints of the external Cypress EZ-USB FX2 chip to the video memory, or to parse input data into single AHB operations, using dedicated data parser. The primary function is to transfer image data. Video data transfer is similar to that of image transfer, the only difference is that we must ensure that minimum number of image frames is transferred to satisfy persistence of vision.

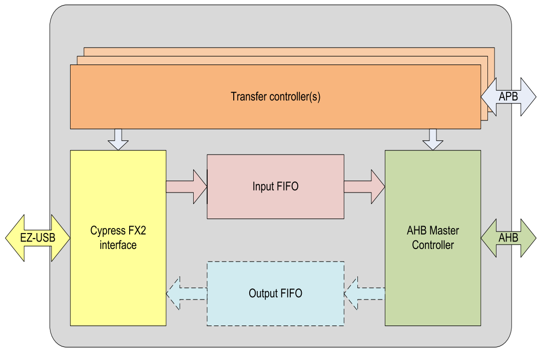

Fig 1. Block diagram of USB DMA

II. GENERAL DISCRIPTION

A. USB Interface

The Universal Serial Bus (USB) has gained wide acceptance as the connection method of choice for PC peripherals. The Host is always the master, this is the fundamental concept of USB. There is exactly one master in a USB system: the host computer. USB devices respond to host requests and USB devices cannot send information among themselves.

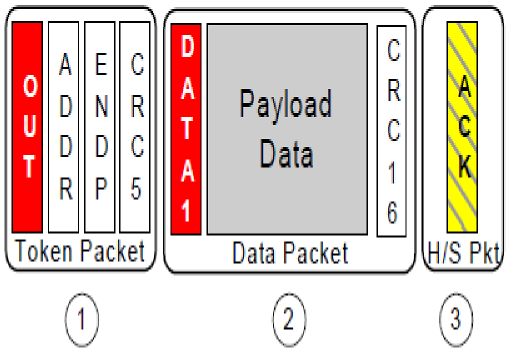

Fig 2. Packet structure of USB

The figure shows USB out transfer which consists of three packets. Packet 1 is the token packet and it contains the Address, Endpoint and CRC code. Packet 2 contains data and packet 3 is a handshaking packet which ensures the error free data transfer. Data transmitted in this format reaches the EZ-USB Interface.

B. *EZ-USB FX2 Interface*

The Cypress Semiconductor EZ-USB family supports the high bandwidth offered by the USB 2.0 high-speed mode. The EZ-USB chips provide a highly-integrated solution for a USB peripheral device.

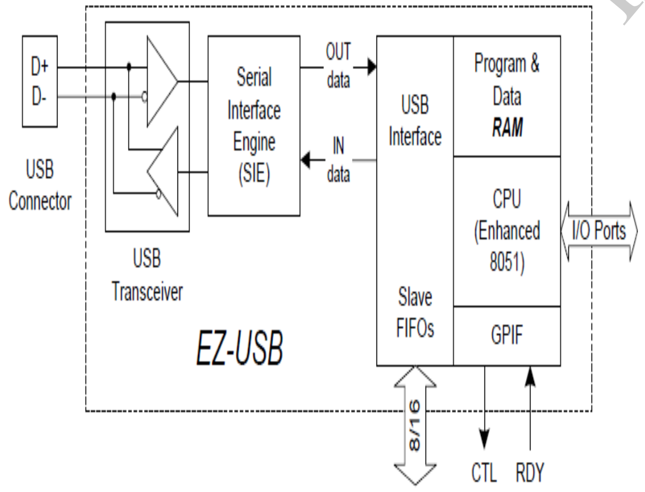

Fig 3. Architecture of EZ-USB module

The EZ-USB packs all the intelligence required by a USB peripheral interface into a compact integrated circuit. As Figure illustrates, an integrated USB transceiver connects to the USB bus pins D+ and D-. A Serial Interface Engine (SIE) decodes and encodes the serial data and performs error correction, bit stuffing, and the other signaling level tasks required by USB. Ultimately, the SIE transfers parallel data to and from the USB interface.

C. *External FIFO Interface*

The large data FIFOs (endpoints 2, 4, 6 and 8) in the EZ-USB are designed to move high-speed (480 Mbps) USB data on and off chip without introducing any bandwidth bottlenecks. They accomplish this goal by implementing the following features:

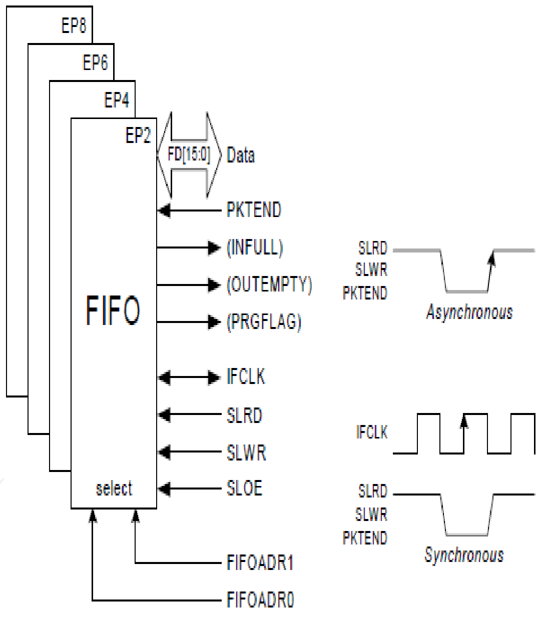

Fig 4. EZ-USB Endpoint FIFO

Fig illustrates the outside-world view of the EZ-USB data FIFOs configured as Slave FIFOs. The outside logic supplies a clock, responds to the FIFO flags, and clocks FIFO data in and out using the strobe signals. Optionally, the outside logic may use the internal EZ-USB Interface Clock (IFCLK) as its reference clock. Output of the EZ-USB is connected to the USB DMA Interface. That is serial data received through the USB interface is converted to parallel format by Cypress FX2 interface and the data is temporarily stored in the input FIFO interface. With its quantum FIFO interface, the Cypress can support high speed USB data streaming to the device over bulk or isochronous USB endpoints.

.

D.CFX2 FIFO IFC

FX2 FIFO IFC is the interfacing module of USB DMA with the EZ-USB chip. This module behaves like a buffer between USB to parallel converter chip and the entire module. Major inputs are system clock, bulk and single access data, packet count. And outputs are bulk and single access data, control signals.

It handles endpoint selection and interface initialization. Token format in the serial communication is converted to parallel format by the EZ-USB so it is easy for this module to fetch the data in the parallel format. Each time when the ready is ready to be transmitted initialization processes is done followed by end point selection. Parallel data contains end point address in decoded format, this module fetches the address bits and selects the end point according to it.

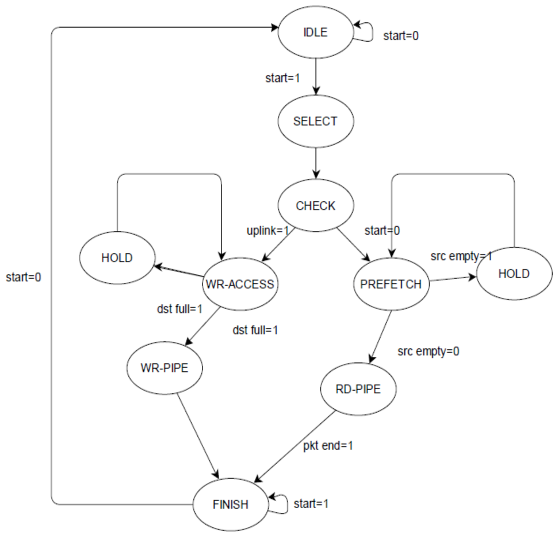

Fig 5. State chart of FX2 FIFO IFC0

Process is divided into three states Idle, start and wait for each endpoint. There are three endpoints in the system one for bulk data, other two for single access data input and output. When operation in each endpoint in completed endpoint address is incremented and next operation in started at the new endpoint selected. When address of the endpoint ran out of range it is reset and will start from the first endpoint.

1)Write branch

Check whether FIFO is full else data transfer is initiated and is continued until any interruption is occurred. So write branch has three states Access, Pipe and Wait states. FSM movers to write Access state only after ensuring that DST FIFO is not full, that is space should be available to write data into it. After reaching Access state programming is done to double check this condition because there is a chance that due some unexpected situation FIFO may get full during the state transfer. To avoid such errors during execution FIFO full flag is cross checked when FSM reaches write Access state.

When FIFO or memory access is permitted system is ready for the data transfer. It waits for the control signal which initiates the same. When enabled continues data transfer started. This is continued until FIFO becomes full and when FIFO is fulls the FSM moves to Wait states. FSM then remains in Wait state until some space is available in the FIFO to continue the data transfer.

2) Read branch

Similar to write branch read branch has three states Access, Hold and pipe. Access give control to the read branch when the FIFO is not empty, it move to Pipe state to start the data transfer and continued until any interruption is occurred. When interrupted it moves to the Hold state. There is a packet count data input to this module, which determine the number of data packets to the read. So while reading the counter is decremented and reading is completed when this count become zero.

After completing both read and write branch FSM moves to the final state Finish. At Finish state signals are produced to indicate its completion depending on the timing on the timing constraints of the FX2 chip.

E.AHB Finite State Machine

Buffered data both bulk and single access format from the FX2 FIFO IFC is then moved to AHB FSM module, where entire data is converted into AHB format and transferred to corresponding modules. It is controlled by the signals generated from the USB DMA control module like bulk empty, AHB write, AHB request. AHB FSM module acknowledge USB DMA control module after completing the requested data transfer.

There will be many components in an SOC and each modules will be communicating with each other. Depending on the data bandwidth and communication speed requirement different communication methods are preferred. In AMBA protocol AHB is preferred for high speed data transfer.

III USB DMA CONTROL

Control signals to the AHB FSM module are generated in this module. USB DMA is defined as FSM with Inactive, Idle, Pump, Jump and End states.

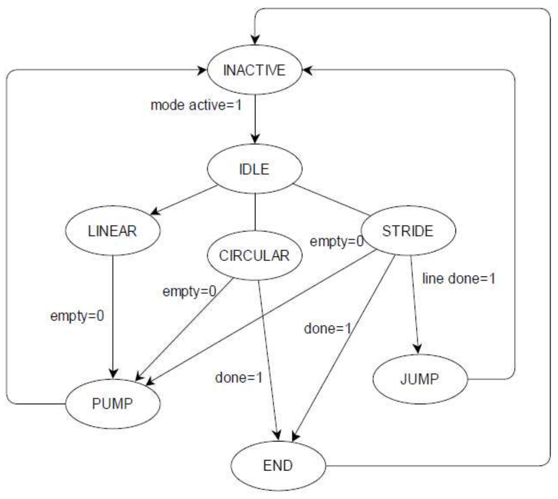

Fig 6 State chart of USB DMA controller

If no mode is selected then the FSM is in the Inactive state. When the mode system is selected then the system move to idle state where it check the mode. It can be in Linear, Circular and Stride based on the requirements. If mode selected is Linear it check for AHB request and ensure that FIFO is not empty, if satisfied FSM move to Pump state (continues data transfer). Similar method is followed for the circular mode with an extra data count register indicating minimum count of data transfer before resetting the address. But for stride mode since it is used for image transfer it check Line done signal if high, then the system moves to Jump state. In all these three modes after data transfer is completed FSM move to End state. For each mode corresponding value is moved to the address register. Data counter, line counter, stride counter registers associate with each mode.

A.Bulk access

Bulk access operation enables the transfer of a data stream from the USB endpoint to the HMI device memory.The USB DMA supports several bulk access operating modes, enabling operation from simple data transfers to image transfers, which require image dimension information for correct alignment. The USB DMA supports several operating modes.

Stride: used for image transfer. The data is written to the AHB bus incrementally from a starting address for a specified number of data elements. After that the new starting address is defined as old starting address incremented by the stride length. After a specific data count has been reached, the new starting address is replaced by the original base address.

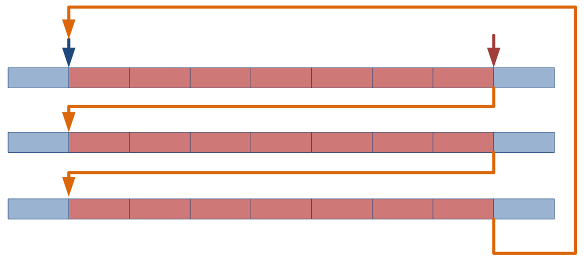
Fig 7. Stride data storage pattern

The USB DMA bulk interface transfers data in 16-word bursts, which is the maximum specified burst size on the AHB bus. Data sizes less than a word (32 bits) are not supported. The destination addresses and counts need to be defined in quantum of the burst size. These limitations reflect the nature of the interface, which supports high speed operation of the interface while keeping the logic simple.

B.FX2 SA IO

This module handles one down end point and one up end point. From idle mode the FSM moves to Load state when data is available in its terminals and remains in the Load state until data transfer is completed. When done state changes to OP. After OP is finished it is ready for transmitting, for that FSM will check transmit FIFO. If space is available then data is Outputted into it else it Wait for FIFO to be free.

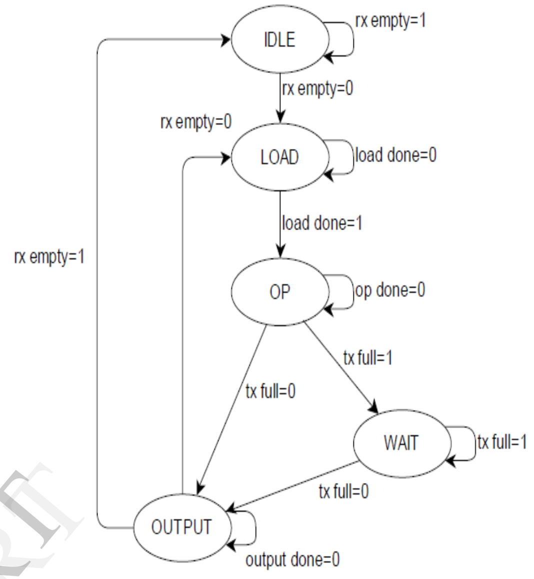

Fig 8. State chart of FX2 SA IO

This module generates all control signals required to control single access protocol. It fetches the data from the input vector, count the data packets to do the same.

C. Single access

The single access mode enables random access to the AMBA address space. It supports single read and write accesses of all sizes. The single access mode utilizes two USB endpoints for a request-response protocol. The requests are sent over one endpoint, while over the second endpoint the responses are received by the USB host.

A request message consists of:

- Tag - the identifier of the operation in the stream, used to identify the corresponding response message in the response stream. The tag of the request will be copied to the response.
- Command/size – the identifier used to define the bus operation and also indicate the data size.
- AMBA address in little endian
- AMBA data (for write operations) in little endian ordering.
- Padding – dummy data to align on 16 byte boundary (to pack messages into a single USB frame and for future use).

A response message consists of:

- Tag - the identifier of the operation in the stream, used to match the response message in the response stream to its corresponding request message The tag of the request is copied to the response.
- AMBA data (for read operations) in little endian ordering Dummy data/padding for write operations.
- Padding – dummy data to align on 8 byte boundary (to pack messages into a single USB frame and for future use).

D.DMA control registers

DMA control registers module is a APB slave in the system. The data value is received from the host system through the APB slave input. Inside the DMA control register module entire data is fetched and de-multiplexed to corresponding registers. Packet count is moved to FX2 FIFO IFC module and rest all registers are moved into USB DMA control module

Finally purpose of the USB DMA interface is to convert serial data from the host system into AHB format. Therefore simulations results show the output in AHB format. In AHB protocol different modules in a system will be connected together by AHB bus. According to the requirement a module can be configured either as a AHB master or as a AHB slave. In our system USB DMA interface is configured as an AHB master.

Since the USB DMA is configured as AHB master the left side of the picture consists of all signals corresponding to AHB master.

Simulation graph shows the initialisation of a AHB master, in our case it is USB DMA. Before data transfer is initiated all signals from the AHB master is set to the default value. Address is not set to it will have a reset or a garbage value. When the system get the control signal, it initiate the data transfer by setting Hwrite signal. Now Hburst value is set to 111 value indicating a Increment 16 format ie address is incremented continuously sixteen times without any break. And when it reaches sixteenth address it wait for a new mode data transfer.

IV SIMULATION RESULTS

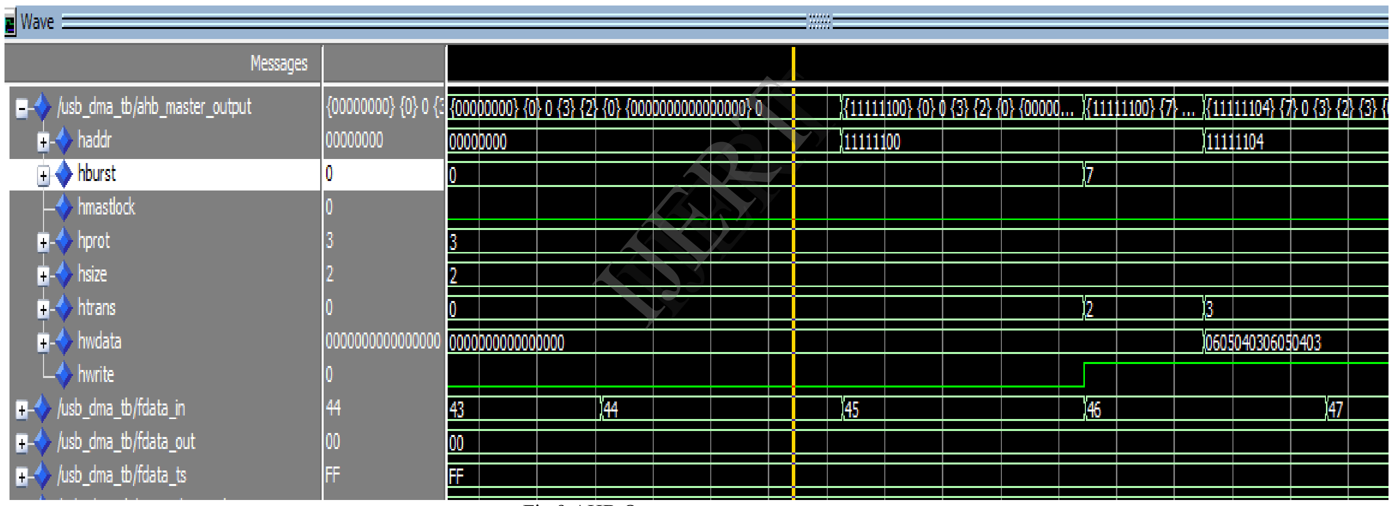

Fig 9.AHB Output

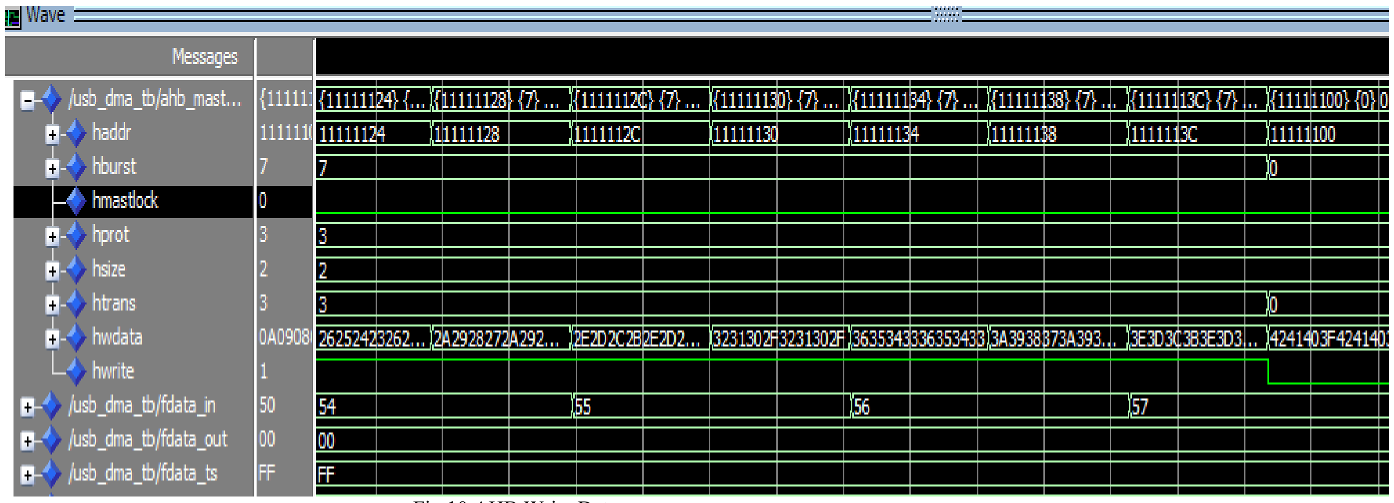

Fig 10.AHB Write Data

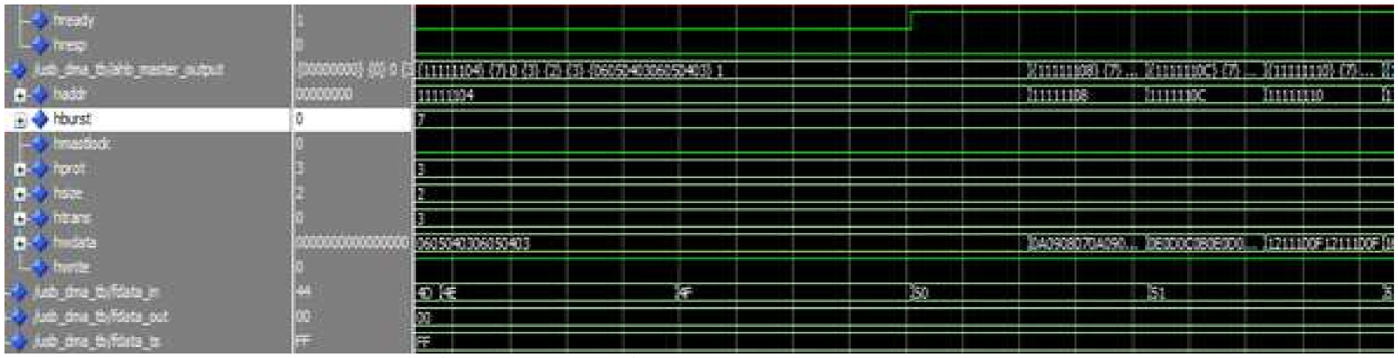

Fig 11.AHB INCR Mode 16

Htrans indicate different states of the system when data transfer is done. It is two bit signal so four states can be represented ie idle, busy, non-synchronous and synchronous. Here before data transfer is initiated the system is in Idle state then it moves to Non synchronous state where an address value is set. Finally to the Synchronous state where address is continuously incremented. When the system reaches Synchronous mode it is ready for data transfer now it wait for the Hready signal from the slave.System waiting for the Hready signal to initiate data transfer is shown in the following simulation graph.

The master wait for the Hready signal from the slaves only then data transfer from the master can be initiated. So till the Hready signal is received all signals from the AHB master will be configured with initial values. When Hready is high as show in the figure data transfer is initiated. Address is incremented based on its configuration, here value of Hsize is set as 010 that indicate 32 bit transfer. For 32 bit data transfer address is incremented my four (8 x 4) for every data transfer.

V CONCLUSION

USB DMA interface was developed using VHDL, simulated in Modelsim and finally synthesised using Latice ISE. The implemented USB DMA interface is very effective and it satisfied all power and timing requirements. The entire circuit was implemented with minimum gate count. It can transfer the image and videos data from the host system to the memory of the system. Serial data in USB format is converted to parallel format in the first phase and finally this parallel data is then converted into AHB format. Display driver module can be tested using USB DMA interface without video processing module.

The module cannot read data from the memory directly from the host machine, only writing into the memory is included in USB DMA. So in future this application can be included along with the USB DMA interface so that it can be developed as a complete debugging system

REFERENCES

[1] Byungyong Kim, Chanho Lee, “High performance on-chip-network architecture with multiple channels and dual routing”, SoC Design Conference, ISOCC 2008, Page(s): III-33 - III-34

[2] Dong Xiang, Shan Gu, Fujiwara H, “Non-scan design for testability for mixed RTL circuits with both data paths and controller via conflict analysis”, Test Symposium, 2003, Page(s): 300 – 303

[3] EZ USB Specification, Cypress

[4] ARM AMBA® 3 AHB-Lite Protocol Specification, ARM Limited

[5] USB DMA Datasheet, Exor International

[6] LatticeXP2 High-Speed I/O Interface, Lattice semiconductor cooperation, technical note TN1138,june 2010