Simulation of a Unidirectional Bus Protocol

Vijaykumar, Akshay, *SID 010276253, SJSU,* and K N, Hemanth, *SID 010259249,* SJSU

***Abstract*—The example design in this document provides a brief overview of a unidirectional bus protocol that can be used to interface multiple masters to multiple slave devices. The concept of bus arbitration, data-flow and control-flow required for a bus to function are explored in some detail.**

I. Introduction

THIS design example specifies a bus architecture as shown in Fig.1. The system has two master and four slave devices combined with a bus arbiter and an address decoder.

The arbitration is based on priority encoding, i.e., based on the priorities assigned to each master device, while ensuring that an ongoing transaction is not corrupted. The master devices request the arbiter for ownership of the bus and transact with the slave devices with the help of an address decoder. The decoder ensures that only one slave is active on the bus at any given point in time. The details of the protocol are explored further in the sections below.

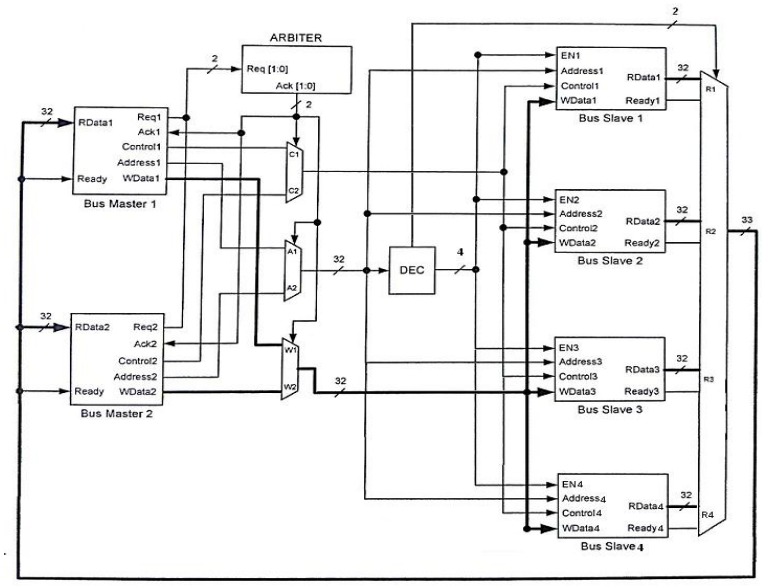

Fig.1 Bus Architecture

II. Overview of the Bus Protocol

A. Control Flow

Each master of the bus generates a set of five control signals viz, Request, Status, Burst Length, Size, Write Enable and receives a Ready signal from the slave. These signals govern the transaction of data between masters and slaves on the unidirectional bus.

For any transaction to take place, the master has to take control of the bus. For this, the master sends a request signal through the 'Req' port to the bus arbiter. Based on the bus arbitration mechanism, the arbiter sends an acknowledgement to the master via the 'Ack' port. The master now owns the bus and can perform transactions with the slaves.

Status is a 2-bit wide control signal that denotes the current state of the master. This signal can take 4 values as shown in Fig.2. When the master initiates a new transaction, it transitions to a 'Start' state. If a transaction involves sending or receiving of more than one data packet, the master transitions to the 'Continue' state (denoted as 'Cont') at the next positive clock edge. On completion of all transactions, the master transitions to the 'Idle' state. At any point in time, if any operations internal to the master are pending, the master transitions to a 'Busy' state at the following positive clock edge. The present Address, Control and Data signals are not modified until the master transitions out of the 'Busy' state. The transitions of Status signal is shown in Fig.3.

Status [1:0]	Bus Master Status
0 0	Start Transfer (START)
0 1	Continue Transfer (CONT)
1 0	Finish Transfer (IDLE)
1 1	Pause Transfer (BUSY)

Fig.2 Status

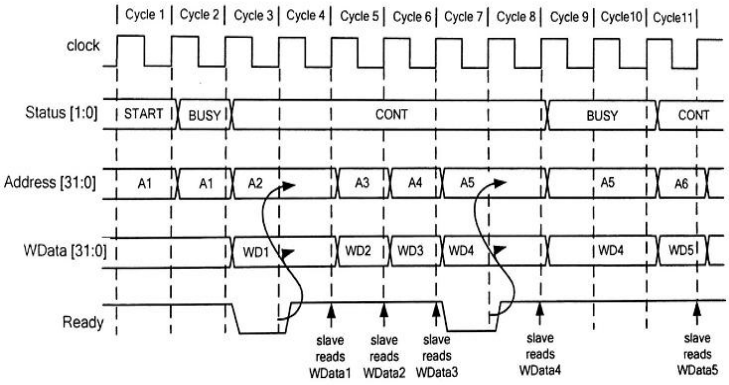

Fig.3 Changing of Status in a transaction

The Write Enable is a 1-bit wide signal that defines the type of transaction. If the Write Enable is high (logic '1'), a write operation is being issued to the slave. If the Write Enable is low (logic '0'), a read operation is being issued to the slave.

Burst length is a 4-bit wide control signal sent from the master to indicate how many requests will be made to the slave in a single transaction. In this design, the slave does not

take into account the 'burst' control signal. As long as the ADL provides the "enable" signal, the slave will process the incoming requests from the master. This signal is more useful for the master to track an ongoing transaction and perform a bus handover, if required, after the transaction is completed.

Size is a 2-bit control signal that indicates the number of bytes transacted between the master and the slave per address provided. For e.g., if the size is a half-word, then the address provided by the master corresponds to 2 bytes of data from the slave memory. If the size is a full-word, then the address provided by the master corresponds to 4-bytes of data from the slave memory. Assumption made in this design is that the master always provides an address which is aligned with the transaction size. Transaction sizes implemented and verified with this design and their corresponding bit encodings are provided in the Fig.4 and Fig.5.

Burst [3:0]	Number of data packets
0 0 0 0	1
0 0 0 1	2
0 0 1 0	4
0 0 1 1	8
0 1 0 0	16
0 1 0 1	32
0 1 1 0	64
0 1 1 1	128
1 0 0 0	256
1 0 0 1	512
1 0 1 0	1024
1 0 1 1	2048
1 1 0 0	4096
1 1 0 1	8192
1 1 1 0	16384
1 1 1 1	32768

Fig.4 Burst Length

Size [1:0]	Number of bits
0 0	8
0 1	16
1 0	32
1 1	64

Fig.5 Bit Encoding for Size

Ready is a 1-bit control signal generated by the slave to indicate to the master that it is ready for the next transaction. The master does not provide the next set of Address/Control/Data signals if the slave is not ready. For e.g., if a transaction requires multiple cycles for the slave to complete, it may pull the busy signal low and thus prevent the master from issuing the next set of Address/Control/Data signals.

The signals 'Status', 'Burst Length', 'Size' and 'Write Enable' are all combined and sent as a single 9-bit wide signal through the 'Control' output port.

B. Data Flow

The data flow and transition of control signals with respect to the data are explained in this section. Every data transaction begins with the master, which has acquired the bus, transitioning to "Start" state and placing the first set of address and control signals on the bus irrespective of whether the slave is ready or not. After this initial state, the master ideally transitions to the "Cont" state in the next cycle but holds the present address and control signals on the bus until the slave becomes ready. The master may also go into a "Busy" state during any clock-cycle, owing to internal functions, during which case as well, the address, control and data signals are held as it is on the bus.

Once the master detects that the slave is ready, it places the next set of control and address signals on the bus. If the transaction is a write transaction, the data to be written to the slave is placed on the bus along with the next control and address signals in the subsequent clock-cycle. This is to say that, for e.g., if the first transaction is a write transaction, the master places address and control signals during clock-cycle one. It provides the data to be written along with the next set of address and control signals in clock-cycle two, provided the slave is ready during clock-cycle one. If the transaction is a read transaction, the slave provides the data in the next clock-cycle corresponding to the control and address signals issued for the read.

In this way, the master proceeds through issuing "Burst" number of transactions to the slave. The master may immediately start a new transaction by transitioning to "Start" status and going through the next set of "Burst" number of transactions or the master may transition to "Idle" state and bequeath the ownership of the bus.

A Read transaction is shown in Fig.6. The first set of address and controls, A1 and C1 are provided in Cycle 1. However, the slave is not ready at positive edge of Cycle 2, the address and control signals are not modified. At the positive edge of Cycle 3, Ready signal goes high. The existing address and control signals are read by the slave at this edge and the next set of address and control signals, A2 and C2 are provided. As per the protocol, the master should read RData at the positive edge of Cycle 4. As Ready is low at this edge, the master reads RData at a positive edge when Ready is high, which is at positive edge of Cycle 6. Similar read transactions occur in the subsequent cycles.

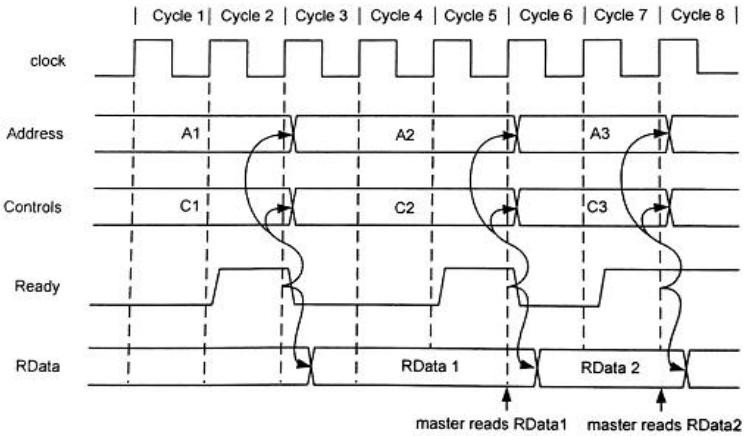

Fig.6 Read Transaction

A Write transaction is shown in Fig.7. The first set of address and controls, A1 and C1 are provided in Cycle 1. However, the slave is not ready at positive edge of Cycle 2, the address and control signals are not modified. At the positive edge of Cycle 3, Ready signal goes high. The existing address and control signals are read by the slave at this edge and the next set of address and control signals, A2 and C2 are provided along with the data WData1 for A1 and C1. The Ready goes low at the positive edge of Cycle 4 and hence, the slave does not sample A2, C2 and WData 1. Master holds these values on the bus until the slave is ready, which is at the positive edge of Cycle 6. At this point, the slave samples A2, C2 and WData1 while the master proceeds to issue the next set of address, control and data signals as per the protocol.

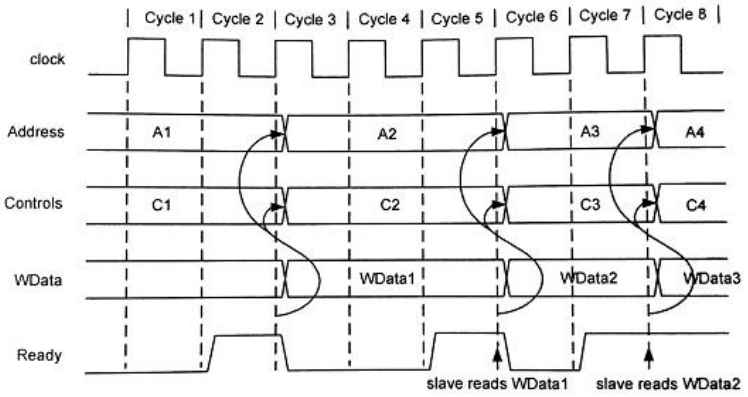

Fig.7 Write Transaction

C. Arbitration Mechanism

The arbiter in the current system employs a one-hot priority encoding scheme. The truth table of the Arbiter is as shown in Fig.8. The 'Req' lines of Master 1 and 2 are connected to Req[1] and Req[0] of the arbiter respectively. Here, Master 1 has a higher priority than Master 2. This is to say that, if both the masters request for the control of the bus at the same time while the bus is not owned, Master 1 would be granted control. However, once a master has taken control of the bus, any requests made by the other masters will not be acknowledged until the current owner bequeaths the bus. This is to ensure that an ongoing transaction on the bus is not corrupted by a different master.

Req1	Req2	Ack1	Ack2
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	0

Fig.8 Truth Table of Arbiter

The mechanism of bus handover is as shown in Fig.9. In Cycle 1, Master 1 requests for control of the bus. In Cycle 2, the arbiter acknowledges this request. From this point on, Master 1 owns the bus and begins transacting. In Cycle 6, Master 2 requests for the bus. The arbiter denies the request as Master 1 holds the control. In Cycle 9, Master 1 is done with its transactions and hence, pulls its request line low. The arbiter then acknowledges Master 2's request and grants him the bus in Cycle 10.

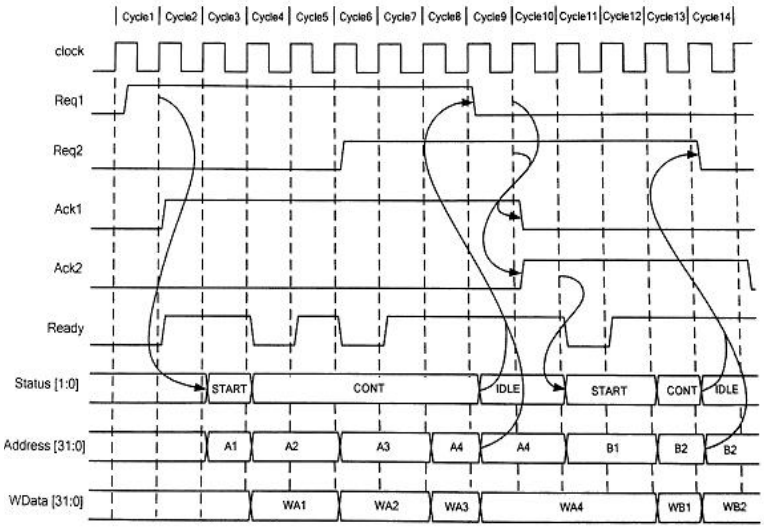

Fig.9 Bus Handoff by the Arbiter

III. Functional Components

A. Overview of Arbiter

The arbiter module has a 2-bit input port 'Req' and a 2-bit output port 'Ack'. The arbiter is implemented as a state machine with 3 states viz Idle, Ack 1 and Ack 2. The state diagram is as shown in Fig.10. The arbitration mechanism is as explained in II.C.

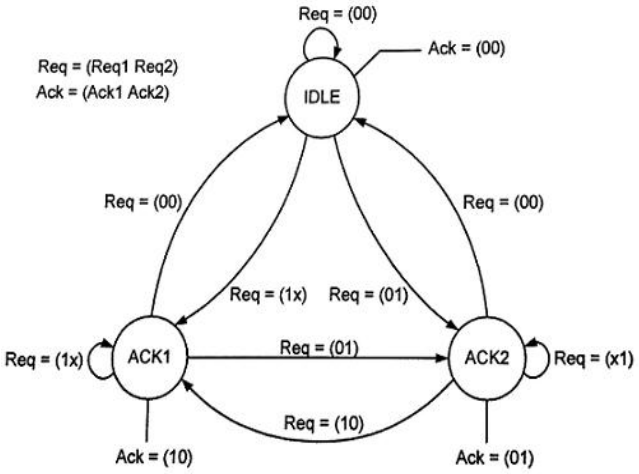

Fig.10 State Diagram of Bus Arbiter

B. Overview of Master

The master modules that transact over the bus as per the unidirectional bus protocol have four output ports 'Address', 'WData', 'Control', 'Req' and three input ports 'Ack', 'RData' and 'Ready'. The 'Address', 'WData' and 'RData' are all 32-bit wide. The 'Control' port is 9-bit wide. Each of the 'Req' and 'Ack' ports are 1-bit wide. The module sends the address and data to the slaves over the 'Address' and 'WData' ports respectively while receiving data from the slaves through the 'RData' port, based on the 'Ready' signal.

The master requests for control of the bus from the arbiter using the 'Req' port and receives an acknowledgement through the 'Ack' port. The block diagram of a Master Module is shown in Fig.11.

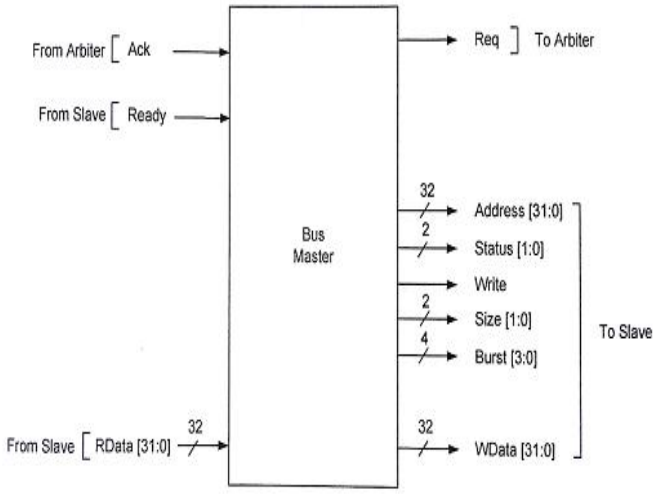

Fig.11 Block Diagram of Master

C. Overview of Address Decoder

Address decoder or 'DEC' as denoted in the block diagram in Fig.1, enables one of the slaves for the master to transact with, over the bus. The decoder considers the two MSBs of the address placed on the address bus and enables the corresponding slave. The same bits of the address are also used by the decoder to 'select' the corresponding slave's output data from the output multiplexer stage.

D. Overview of Slave

The slave is a word-addressable memory device with an 8-bit address line. This means that it stores 256 x 32 bits of data and utilizes a masking logic to provide byte addressing.

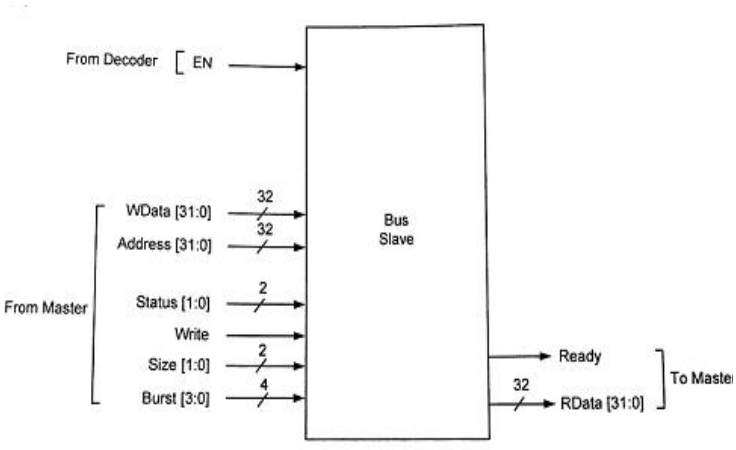

Fig.12 Block Diagram of Slave

The slave module has three input ports; a 32-bit wide "DataIn" port for the input data from the master, 32-bit wide "Addr" port for input address from the master and a 9-bit wide "Control" signal port for the control inputs from the master. It has two output ports; a 32-bit wide "DataOut" port to provide data out of the memory and a one-bit "Ready" port to provide the ready signal to the master. The block diagram in Fig.12 shows how the slave is interfaced with the bus.

All transactions occur according to the bus protocol as explained in II. In this implementation, if a write transaction is requested from the slave for a size less than a full word, the slave pulls down the ready signal for two clock-cycles

IV. Timing Diagram

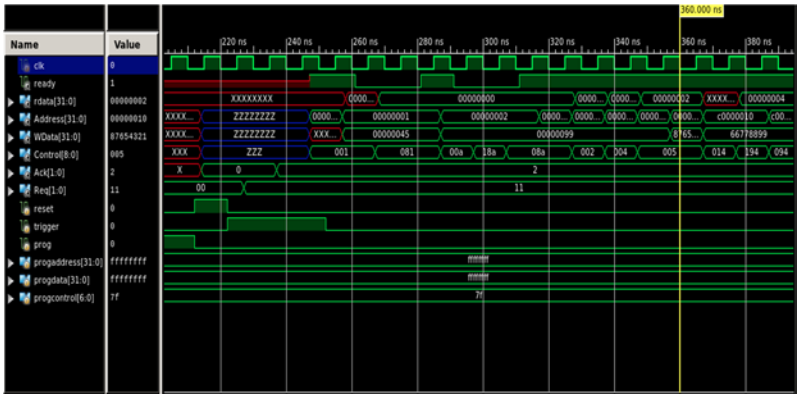

Fig.13 Timing Diagram of a set of read and write transactions

V. Challenges and Lessons Learnt

The first challenge we faced was performing a read transaction across multiple slaves. The master might change the read address from one slave to another in subsequent clock cycles. The new address could cause the output mux stage to select the data from a different slave and the data coming out from the previous slave might go un-sampled by the master. This problem was tackled in two steps. First, a flip-flop was added to the master input to latch older data on the bus. Second, the master was forced to transition into "Busy" state for one clock-cycle before changing address from one slave to another, in case the current command issued was a read. This would latch the incoming data from the slave into the flip-flop while the master is busy and in the next cycle, the new address would cause a different slave to be selected while the master samples the read-data from the flip-flop.

The second challenge was addressing of the slave memory. Implementing a slave as a byte addressable memory would cause a significant delay for both read and write transactions with sizes above a byte whereas implementing the slave as a word addressable memory would cause a fixed and comparatively lesser delay only while writing with a transaction size less than a full word. But the masking logic would become much more complex.

The slave in this design was implemented as a word-addressable memory. A masking logic was implemented where a word from the existing memory location is read-out, only the relevant bytes are modified and the whole word is written back to the memory. As reading by itself from memory has an inherent delay of one clock-cycle the slave had to lower its ready signal to defer the write operation by two clock cycles.

VI. Conclusion

The bus protocol design was implemented and successfully simulated using Verilog HDL and verified. The timing diagram from the simulation is shown in the Fig.13. The results are compliant with the protocol described in section II.

References

[1] Dr. Ahmet Bindal, “System Bus” in *Fundamentals of Computer Architecture and Design, Maple Press Student* ed. San Jose