CS 254 (Digital Logic Design Lab)
Stage -1 Project Report

Project Topic : Memory processor interaction

Team : Scopemaxx
Team members:
Kapil Vaidya (140050006)
Mohit Vyas (140050015)
Govind Lahoti (140050020)
Kushal Babel (140050022)

High Level Architecture

The system consists of a memory and processor where the processor is four times faster than the memory. The processor gets the requests of words and address from the lane interface and then processes it. The processor caters to the requests in round robin fashion and then sends them over to the memory through the memory interface. For effective communication, there are different control signals between the processor and the memory. To make up for the mismatch, we have to use more number of I/O lines and FIFOs instead of a simple communication over a wire. The memory then writes the received word in the bank specified by the address.
Tricky Part :
The efficiency of system is affected by the number of I/O lines and the number of bits the processor processes in each clock cycle. There is a tradeoff between minimization of I/O lines and the number of requests processed in a given time. The parameters of the efficiency of the system are idle time of Lanes, idle time of I/O lines and amount of words processed. The detailed analysis of the tradeoff is shown with the help of a graph in a later section.

Our logic will have two main blocks:

1. Processor
2. Memory

Processor will in turn contain following sub-blocks:

1. Memory Interface
2. Arbiter
3. Main processor

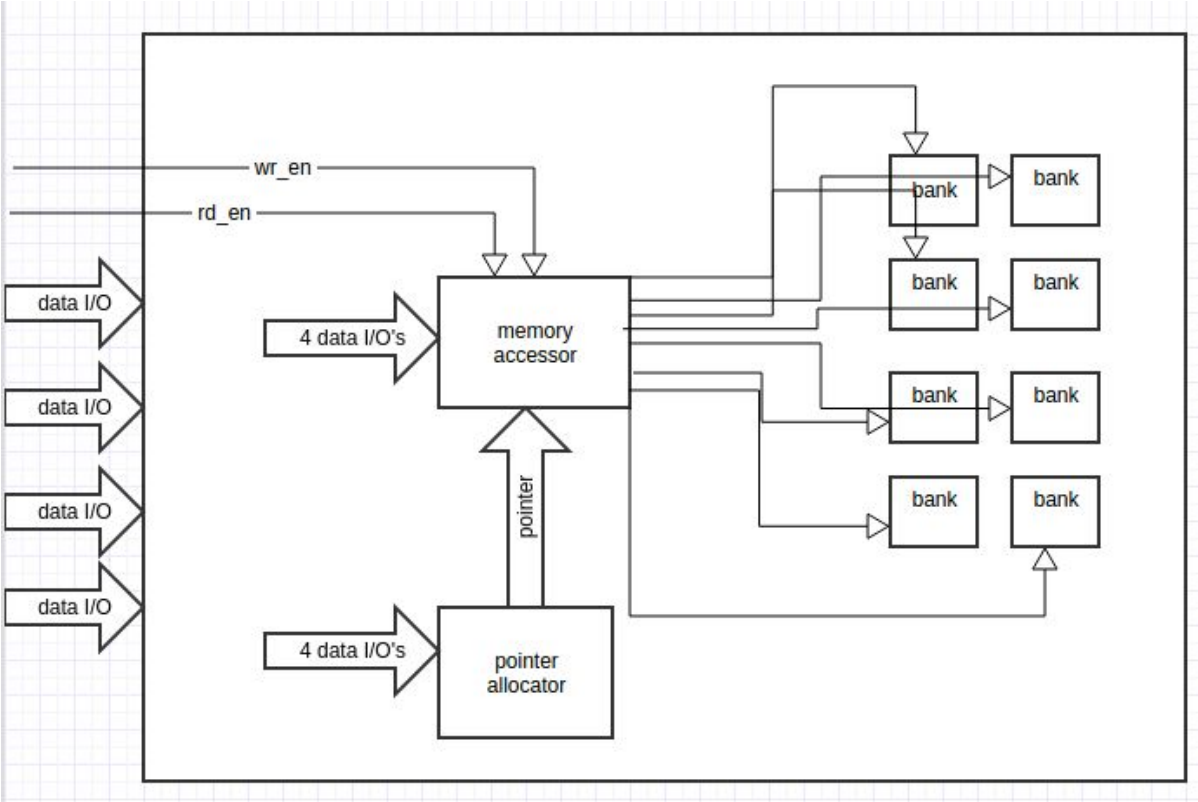
Memory will in turn contain following sub-blocks:

1. Pointer allocator
2. Memory accessor

Functionalities of blocks

NOTE:
The address referred everywhere in the report or in the system has the following format : The first three bits specify the bank number and the rest 6 bits specify the row in the corresponding the bank. This is kind of hierarchical addressing.

MEMORY DATAPATH



Sub-blocks of memory block

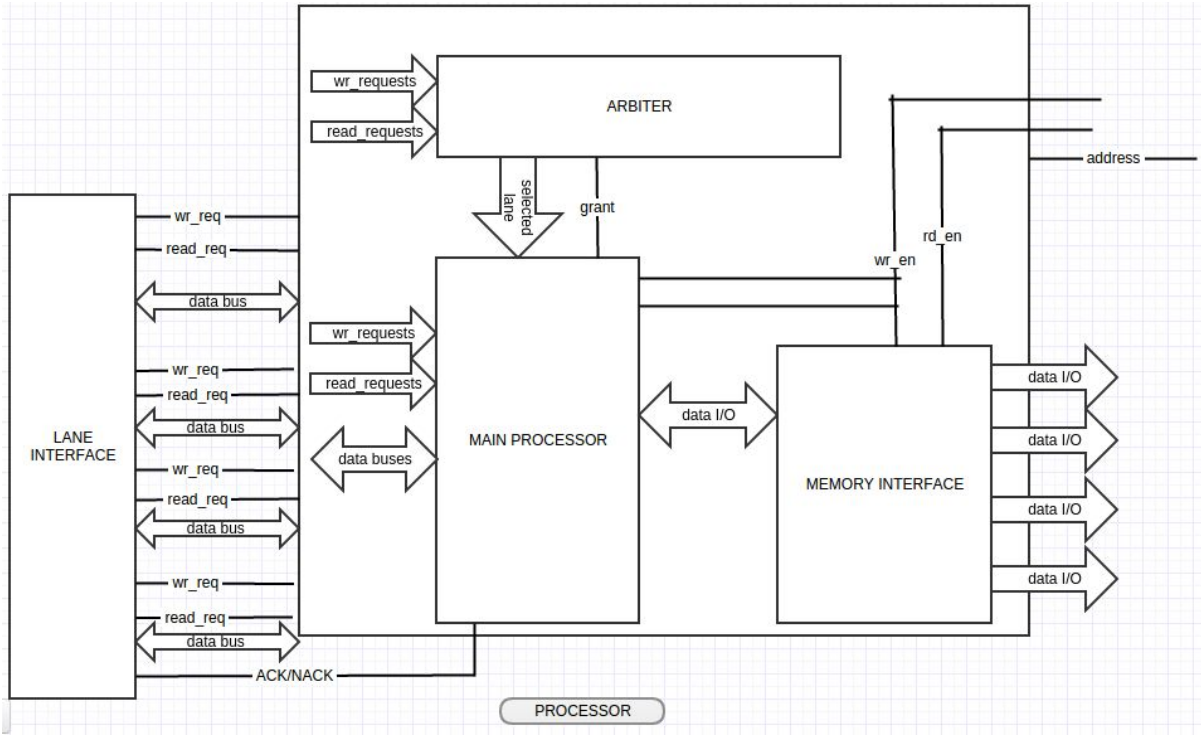
1. Pointer allocator

The pointer allocator receives input from the I/O lines, as well as the address line. When the address line is high, the block takes in the the incoming bits and updates the pointer signal. For this the block has some internal variables for keeping track of the position(in the address) of the 4 bits being received. The functionality can be easily achieved by the shift register mechanism done in one of the labs.
The block does not do anything when the address line is low.

2. Memory accessor

The memory accessor has several inputs, namely pointer, write enable and read enable signals, and also the I/O lines which act as inout.
Suppose the write enable line is high, then the block keeps writing the data received through the input line in the address specified by the pointer input.
If the read enable signal is high, then the block accesses the memory location given by pointer and keeps on sending the next 4 bits via the I/O lines.
It does not do anything when neither of the control line is high.
THe block has output lines to all the memory banks.
The block will have to maintain some internal variables for keeping track of the position(in the word) of the 4 bits being written or read.

PROCESSOR DATAPATH



Sub-blocks of processor block

1. Arbiter

The arbiter receives the read requests and the write request signals from the Lane interface. It then decides which of the four lanes' request should be granted in round robin fashion. The output is two control lines to the processor which convey the lane number whose word should be sent or fetched.

2. Main processor

The main processor receives the lane number from the arbiter whose request should be processed. It also receives the read enable and write enable signals, and the word in the suitable format corresponding to all the four lanes through the lane interface. It then processes the request as follows.
It sends the address bits to the memory interface, and makes the address control signal high for 9 bit duration.
When the write request signal from the lane is high , based on the nature of the request(read or write) the corresponding control line(read or write) is made high till the whole word is sent. The length and the actual word bits are then sent to the memory interface.
When the read signal is high, it receives the word chunks from the memory interface and keeps on building the word. It then returns the word to the lane interface.
As soon as, the entire word is sent or received (determined on the basis of the word length) the corresponding control line is made low.

3. Memory interface

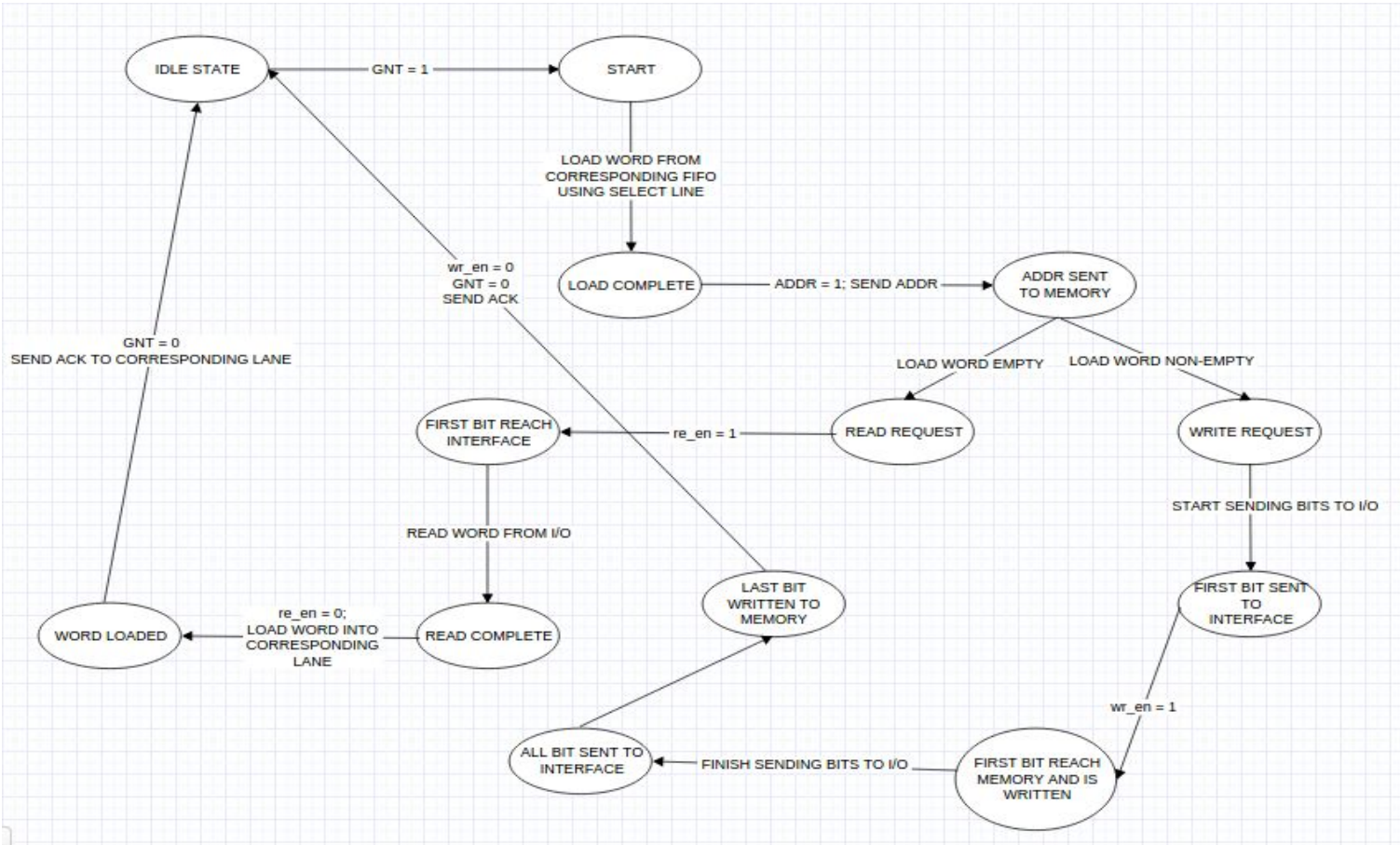
The memory interface consists of one deep FIFO and 4 shallow FIFOs.
The memory interface has the input from the main processor regarding the data to be sent to the memory. It also has the read enable and write enable signal as its input to decide which way the data is flowing. When the write enable signal is high, in each clock cycle it takes the next word from the deep FIFO and gives it to the next shallow FIFO. On the other hand, when the read enable signal is high, it takes the word chunk from the next shallow FIFO and gives it to the deep FIFO.

Note

The processor sends an acknowledgement upon processing the request completely. The acknowledgement is sent to the arbiter by making the grant signal low whereas the acknowledgment is sent in the form of a high ACK/NACK signal to the Lane Interface. It is assumed that the lane interface handles the acknowledgement appropriately, makes the read/write request low for the present request and loads a new request at the interface, if any.

TENTATIVE STATE TRANSITION DIAGRAM

Processor



Assumptions and constraints

1. Block-level assumptions

- Lane Interface provides the packet to the main processor in the desired format only. When writing, first it gives the address(9 bits) followed by length of word(7 bits) followed by the actual word (121 bits). If actual word is of length less than 121 bits, then remaining bits are filled with garbage values. When reading, packet contains just the address(9 bits).
- The re_req/wr_req signal is held high only when the lane interface can provide the desired packet to the main processor in the next clock cycle.
- re_req and wr_req are not held high at the same time. They are held high for at least one clock cycle(1 GHz)
- Address the lane interface provides is always valid.

2. System-level assumptions

- Word are maximum 121 bits long.
- Memory stores in the word in desired format only. For any location(128 bits) in the memory, first 7 bits are reserved for denoting the length, while the next 121 bits contain the actual data. (If word length is less than 121 bits, then garbage value is stored in the remaining bits)

3. Constraints

- Word length can’t be more than 121 bits long
- We have only 8*40 memory locations

Optimisation

Instead of reading the whole 128 bits of the word from the bank, which might consist of redundant bits, we read only those number of bits which are specified by the length(given by the initial seven bits) of the word.

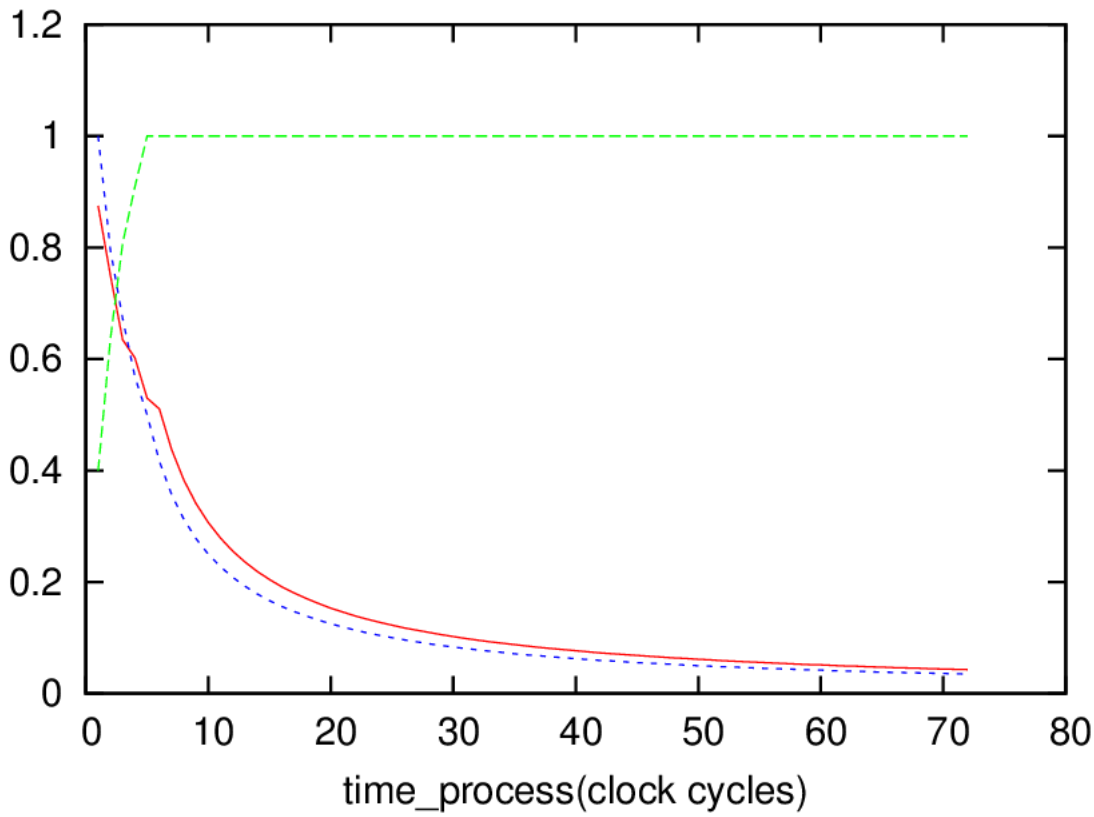
System Efficiency analysis

We need to analyse our system efficiency on various parameters.

1. Efficiency of Lanes :- Percentage of time lanes are busy in given time.
2. Efficiency of I/O lines :- Percentage of time I/O lines are busy in given time.
3. Number of request processed :- The total amount of requests processed by our system in given amount of time.

The analysis was done by writing a code which gives the time for which different lines are free by varying the number of bits processed by the processor in a clock cycle. The wordlength is considered as 72 for the analysis as a representative for words having length from 20 to 121.

Here is a plot against process time for a word.



Efficiency of lanes :- red
Efficiency of I/O lines :- green
Amount of words processed :- blue.

We can see if time_process is less , so no of i/o lines used is large this leads to less efficiency of I/O lines.We can also see if time required for process is high i.e I/o lines are less this leads to less words processed and efficiency of lanes also decreases but efficiency of I/o increases as whole time have to send data.

Plan for testing/verification

2 test benches will be created.

1. Test-bench-1 : This test-bench will cover lots of corner cases. e.g, when none of lane has any request,lanes send read and write requests,checking synchronization of various components. Verification of output for this test-bench will be done manually.
2. Test-bench-2 : Large number of random requests will be generated at each lane using random generators. This test-bench will be used to test the efficiency of our design.
   This is the Test Bench where we will experiment with various word lengths, and number of I/O lines. In these experiments, we will measure the efficiency in terms of the parameters mentioned previously.

Plan for sharing work

All the design level decisions will be thought of and discussed by everyone. State machines too will be prepared by common effort. DIstribution of work will be done only while writing the code.

1. Govind Lahoti : Implementation of memory
2. Mohit Vyas : Implementation of lane arbiter interface and arbiter
3. Kushal Babel : Implementation of processor memory interface
4. Kapil Vaidya : Implementation of main processor

This is tentative work distribution. It may be changed later based on need and difficulties one is facing.

-xxx-