M.EEC041 - Digital Systems Design

2025/2026

Laboratory exercise 4 - V1.1
(new/corrected text in blue)

December 3rd, 2025

In laboratory exercise 3 the sequential divider built in the exercise 2 was integrated in a complete top-level design for a FPGA device (Spartan6LX45, included in the Atlys development board). This project includes an interface based on a UART and a custom module implementing a set of input/output internal ports connected to the divider.

In this exercise you should modify this system to perform the calculation of the arithmetic division on a list of pairs of unsigned integer numbers. The module must execute the same function as the C code below (consider all variables 32-bit unsigned integers):

```
~~for(i=0; i<N*2; i++)~~
for(i=0; i<N*2; i=i+2)
{
  X = MEM[i];
  MEM[i] = MEM[i] / MEM[i+1]; // quotient
  MEM[i+1] = X % MEM[i+1];    // rest
}
```

The system must include a dual-port RAM module that implements the array **`MEM[ ]`**. For a first implementation, the memory must be configured as 1024 x 32 bit. One memory port (**`address, datain, dataout, wenable`**) must be connected to appropriate output and input ports of module ioports, to allow writing/reading data to/from the memory through the serial interface. The second memory port must be used by the application circuit to implement the function referred above. The number of elements to process (N) and any additional control signals should also be provided by output ports of the block ioports. The arithmetic division must be implemented by the sequential divider already integrated in the circuit.

The controller must be built as a finite-state machine and perform the following sequence of tasks (consider each "task" may require one or more clock cycles):

1. wait for a signal to start the computation (this signal may be one bit from an output port of module ioports - call it **startvd**);
2. load zero to register ADDR, driving the memory address bus;
3. load register X with the data in the memory output data bus, connected to the input "dividend" of the sequential divisor;
4. increment register ADDR (memory address);

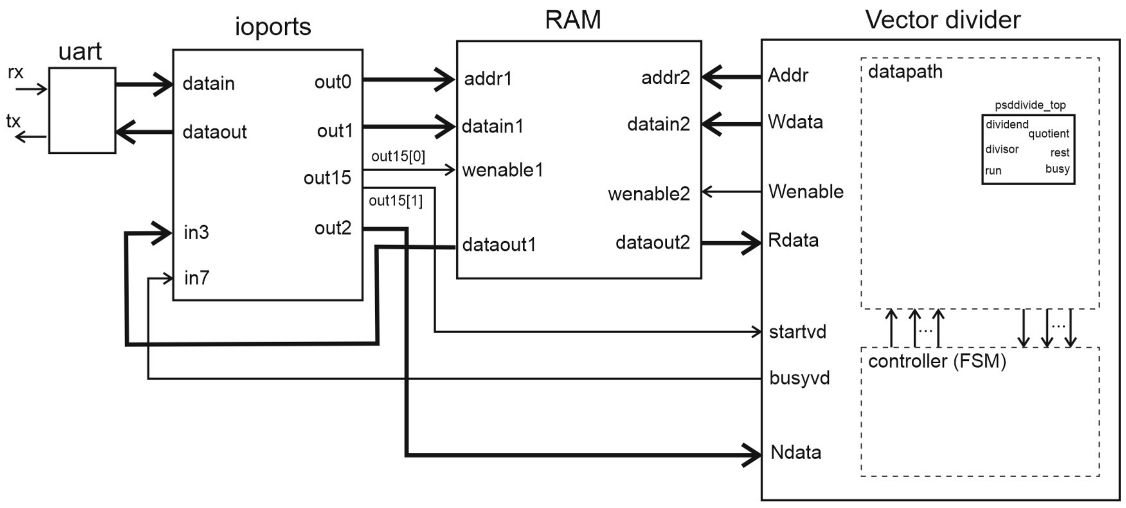
5. load register Y with the data in the memory output data bus, connected to the input “divisor” of the sequential divisor;
6. set to 1 the “run” control input of the sequential divisor (start the division process)
7. wait for the end of the division process (“busy” going down);
8. write the “rest” result to memory, using the current address;
9. decrement register ADDR;
10. write the “quotient” result to memory, using the current address;
11. add 2 to register ADDR;
12. compare register ADDR with the number of elements N multiplied by 2 (N must be one output port of module ioports); if equal, finish the process and return to the initial idle state; if different, go to line 3 (continue the process);

Possible improvements:

a. Consider now that the memory is reorganized to 512 x 64 bit, storing the pairs of operands in the same memory location (dividend in the high 32 bits and divisor in the low 32 bits). Show how to simply the finite-state machine by exploiting this new memory organization.
b. While the sequential divisor is running a division (remember this requires 34 clock cycles), the system may read from the memory a new pair of input data for the next division, write to the memory the result of the last division, and also do the necessary increments and decrements of the memory address register. Propose a new solution that exploits this, either for the initial memory organization (1024 x 32 bit) and for the shorter and wider memory (512 x 64 bits). Calculate the improvement in speed, measured as the number of clock cycles, with respect to both previous solutions.
c. Consider now the utilization of two sequential divisors and the 512 x 32 bit memory. Show how to further improve the performance of the system.

Top-level block diagram

Figure below shows a top-level block diagram of the complete system. The input/output interface (modules “uart” and “ioports”) is the same used in the laboratory exercise 3 and the “RAM” block (dual-port RAM) is implemented by the module “myram.v”. Your design is only the block “Vector divider” and it should use the same sequential divisor included in the laboratory exercise 3.



How to build a simple testbench

The simulation of your vector division module can be done with a very simple testbench, based on the testbench given for the RAM module. Note that your module will only interact with one RAM access ports and the other port is connected to module “ioports” for writing/reading the memory through the serial port. For simulation of your vector division module, the testbench can instantiate the RAM and connect it to your circuit, and do the following sequence of tasks:

1. load at the start of simulation (within an ”initial” statement) the RAM block with a known vector of data – this mechanism is already included in the Verilog module of the dual-port RAM - see myram.v and myram_tb.v
2. set the number of elements to process (input “Ndata”)
3. start the vector division module by activating the “startvd” control input
4. wait for the process to complete – wait until “busyvd” goes to zero
5. dump the RAM contents and compare the results stored in RAM with the expected results (use task DumpRam, included in myram_tb.v)