Parallel Data Processing in Reconfigurable Systems

Lab nº 2

Objectives

- Preparation of test data for experiments
- Design of parallel data sorters
- Introduction to high-level synthesis
- High-level synthesis and optimization of a data sorter
- Test in FPGA and experiments

Part I – Design of a parallel data sorter

1. Create a new project in Vivado for the FPGA of the Nexys-4 board.
2. Use the integrated Vivado IP generator to construct a dual-port ROM for storing 16 8-bit words (constructed from embedded Block RAM). Initialize the ROM with data specified in the text file "Test.coe" (shown below).

```
memory_initialization_radix = 16;
memory_initialization_vector =      FF EE DD CC BB AA 99 88
                                    77 66 55 44 33 22 11 08;
```

3. Design a circuit that would sort the data from the ROM using an iterative even/odd sorting network. First of all, copy the data to a 16×8-bit register and then work over the register. The register is organized as follows:

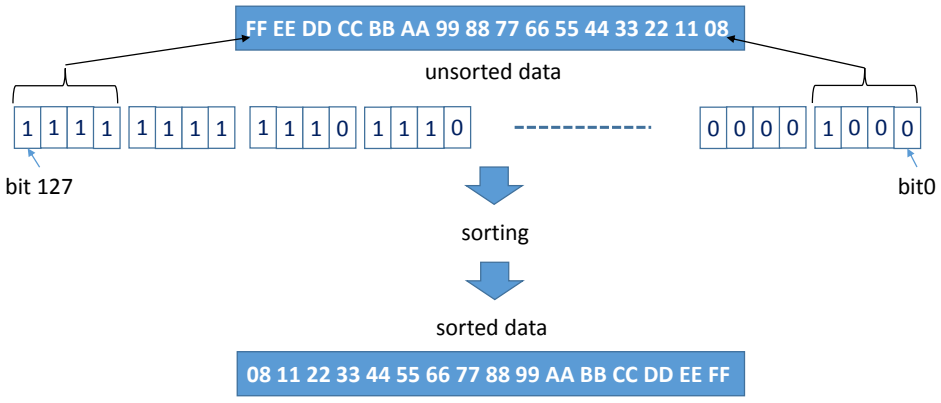

Figure 1 – Organization of the register for data sort.

4. Show the sorted data on 7-segments displays (4 data at a time). Select the data to show with switches (1..0). When sw(1..0) = "00" show the smallest sorted data (08 11 22 33 for the example above); when sw(1..0)="11" show the biggest sorted data (CC DD EE FF for the example above).

5. Measure how many clock cycles are required to sort the data and show the result on LEDs.
6. The simplified circuit block diagram is illustrated in Fig. 2. Use the following I/O components to interact with the circuit. Do not forget to modify the file "Nexys4_Master.xdc" accordingly.
   - *eight 7-segment displays – show 4 sorted words*
   - sw(1..0) – *select which 4 of 16 available sorted data items to show*
   - btnC – *reset*
   - *btnU – start sorting*
   - *led – number of clock cycles used by the sorter*
7. Synthesize and implement the project and test it on the board.
8. Write down the resources occupied by the circuit and the number of clock cycles required for sorting.

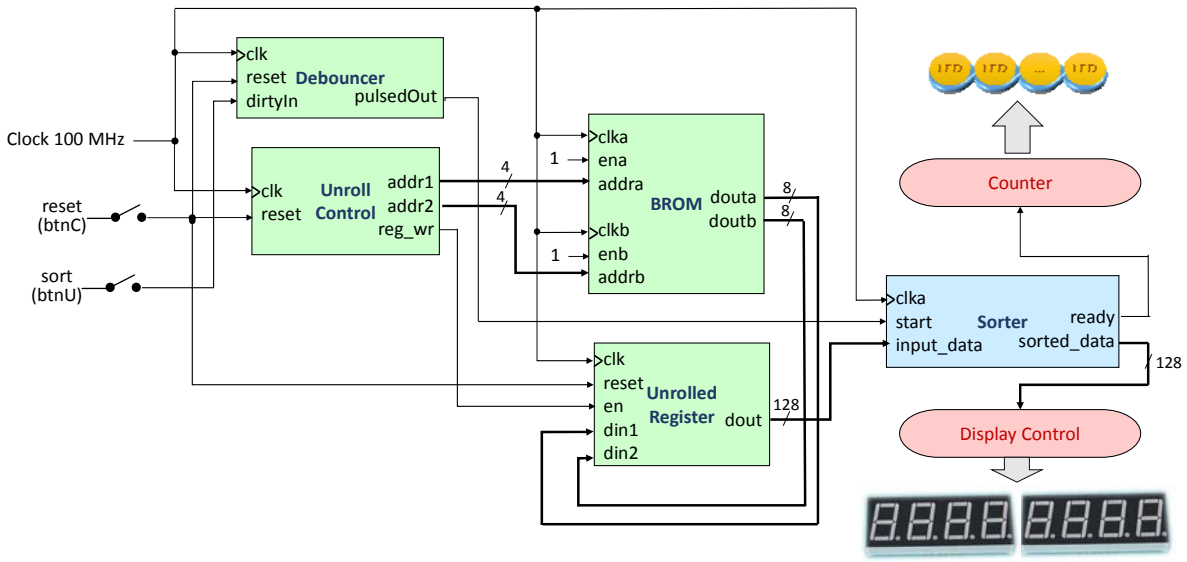

Figure 2 – The block diagram of the sorting circuit.

Part II – Design of a parallel data sorter with high-level synthesis

1. Create a new project in Vivado HLS.
2. Using either C or C++ programming language write a function that sorts N M-bit data items using an iterative even/odd sorting network. The function declaration should be the following:

```
ap_uint<N*M> EvenOddIterSorter (ap_uint<N*M> input_data);
```

3. Include in the project the file "EvenOddIterSorter.h" with the following contents:

```
#include <ap_int.h>
const unsigned int M = 8; //number of bits in each data item
const unsigned int N = 16; //number of data items
ap_uint<N*M> EvenOddIterSorter(ap_uint<N*M> input_data);
```

4. Test you function with the testbench “SorterTb.cpp” that would permit to test the top-level function **`EvenOddIterSorter`**:

```
#include <iostream>
#include <fstream>
#include "EvenOddIterSorter.h"

//error codes
const int SORT_ERROR = 200;
const int OK = 0;
using namespace std;

int main ()
{	ap_uint<N*M> input_data, sorted_data;
	ap_uint<M> item;

	ifstream unsorted_data_stream("input_data.dat"); // open for reading
	ofstream sorted_data_stream("sorted_data.dat"); // open for writing

	//get the input data
	cout << "Reading input data" << endl;
	for (unsigned i = 0; i < N; i++)
	{
		unsorted_data_stream >> item;
		cout << item << endl;
		input_data <<= M;   // shift left M bits
		input_data |= item; // write M LSBs
	}

	//perform sorting
	sorted_data = EvenOddIterSorter(input_data);

	//save the result
	ap_uint<M> mask = ~0;
	for (unsigned i = 0; i < N; i++)
	{	// extract M LSBs
		sorted_data_stream << (sorted_data & mask) << endl;
		sorted_data >>= M; // shift right M bits
	}

	cout << "Checking the result" << endl;
	if (system("diff -w sorted_data.dat sorted_data.gold.dat"))
	{
		cout << "FAIL: Output DOES NOT match the golden output." << endl;
		return SORT_ERROR;
	}
	else
	{
		cout << "PASS: The output MATCHES the golden output!" << endl;
		return OK;
	}
}
```

The testbench reads test data from a file, such as “input_data.dat”:

```
1 168 2 167 3 166 4 165
5 164 6 163 7 162 8 161
```

The result is written to the file “sorted_data.dat” and is compared with the known golden results stored in the file “sorted_data.gold.dat”:

```
168
167
166
165
164
163
162
161
8
7
6
5
4
3
2
1
```

5. Execute C simulation and debug your code if necessary.
6. Execute C Synthesis and check the reports. Optimize your design with synthesis directives so as to reduce the sorter’s latency.
7. Run C/RTL Cosimulation.
8. Once your design is optimized and functional, return to the Vivado project from the part I and replace the sorting circuit (block Sorter in Fig. 2). The synthesized RTL VHDL code (“EvenOddIterSorter.vhd”) is available in the HLS project folder /solution1/syn/vhdl/EvenOddIterSorter.vhd. Include this file as a source and provide for the required connections with the rest of the blocks shown in Fig. 2.
9. Synthesize and implement the project and test it on the board.
10. Write down the resources occupied by the circuit and the number of clock cycles required for sorting and compare these results with those from the part I.