A parallel kNN classification coprocessor

Semester project for the Neurocomputing course, WS2009/2010

by Ivan Shcherbakov

Overview

1. Introduction
   - *Problem statement and motivation*
2. The algorithm
   - *Algorithm description*
3. Evaluation
   - *Recognition accuracy evaluation tool*
   - *Evaluation results*
4. Hardware design
   - *Coprocessor testbed*
   - *Coprocessor design*
5. Tools
   - *ROM generator*
   - *Tools summary*
6. Conclusion

Problem statement/motivation

- kNN algorithm classifies objects based on training data

- The algorithm has great capacity for parallelism

- Dedicated hardware implementation can be faster than software on modern CPUs

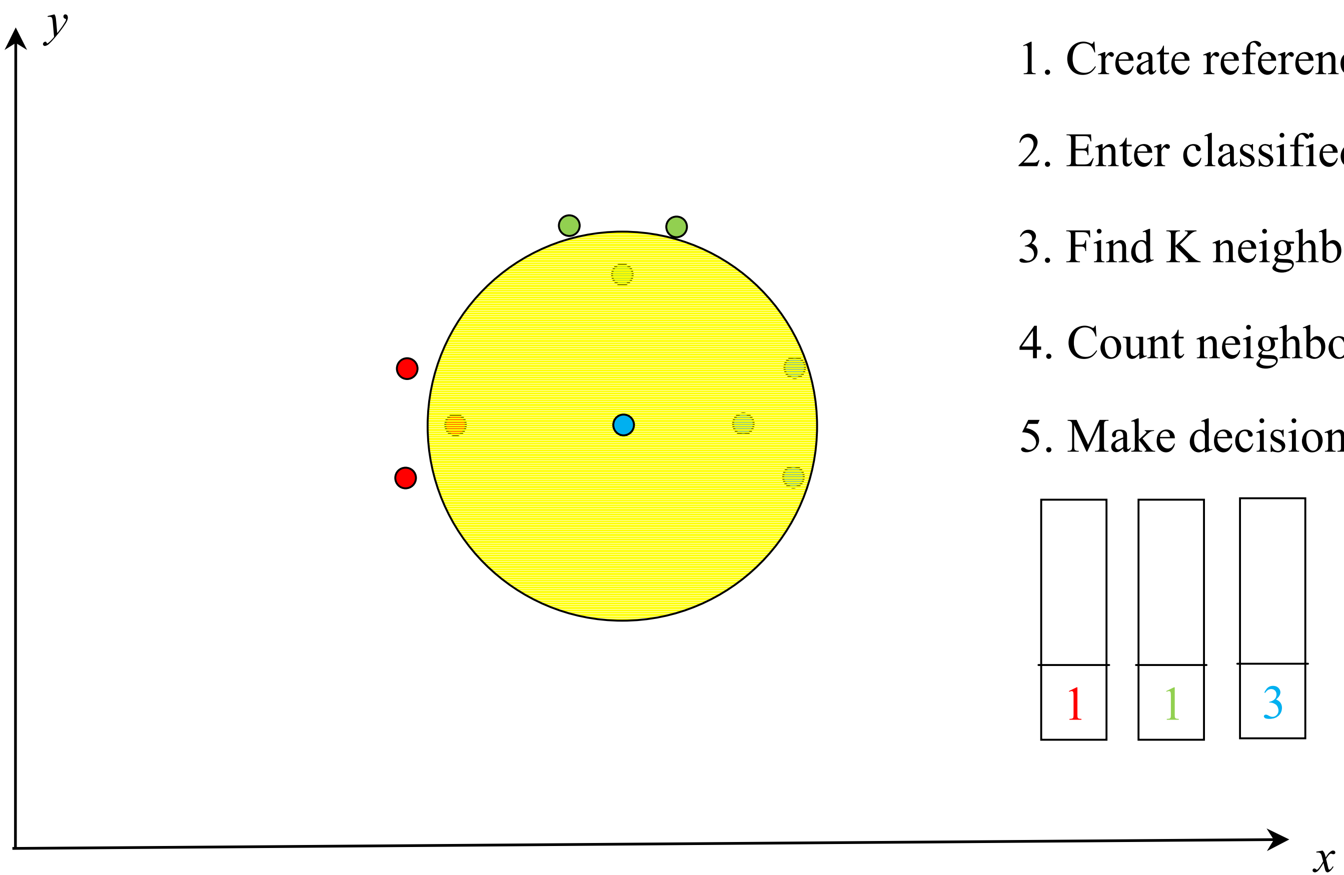
Overview

1. Introduction
   - *Problem statement and motivation*
2. The algorithm
   - *Algorithm description*
3. Evaluation
   - *Recognition accuracy evaluation tool*
   - *Evaluation results*
4. Hardware design
   - *Coprocessor testbed*
   - *Coprocessor design*
5. Tools
   - *ROM generator*
   - *Tools summary*
6. Conclusion

Algorithm description

- Inputs:
  - Features given as **coordinates** of a point in N-dimensional vector space
  - M "reference points" in the same vector space
  - Each reference point is associated with one of C classes
- Output:
  - Most widespread class among K nearest neighbors of the point.

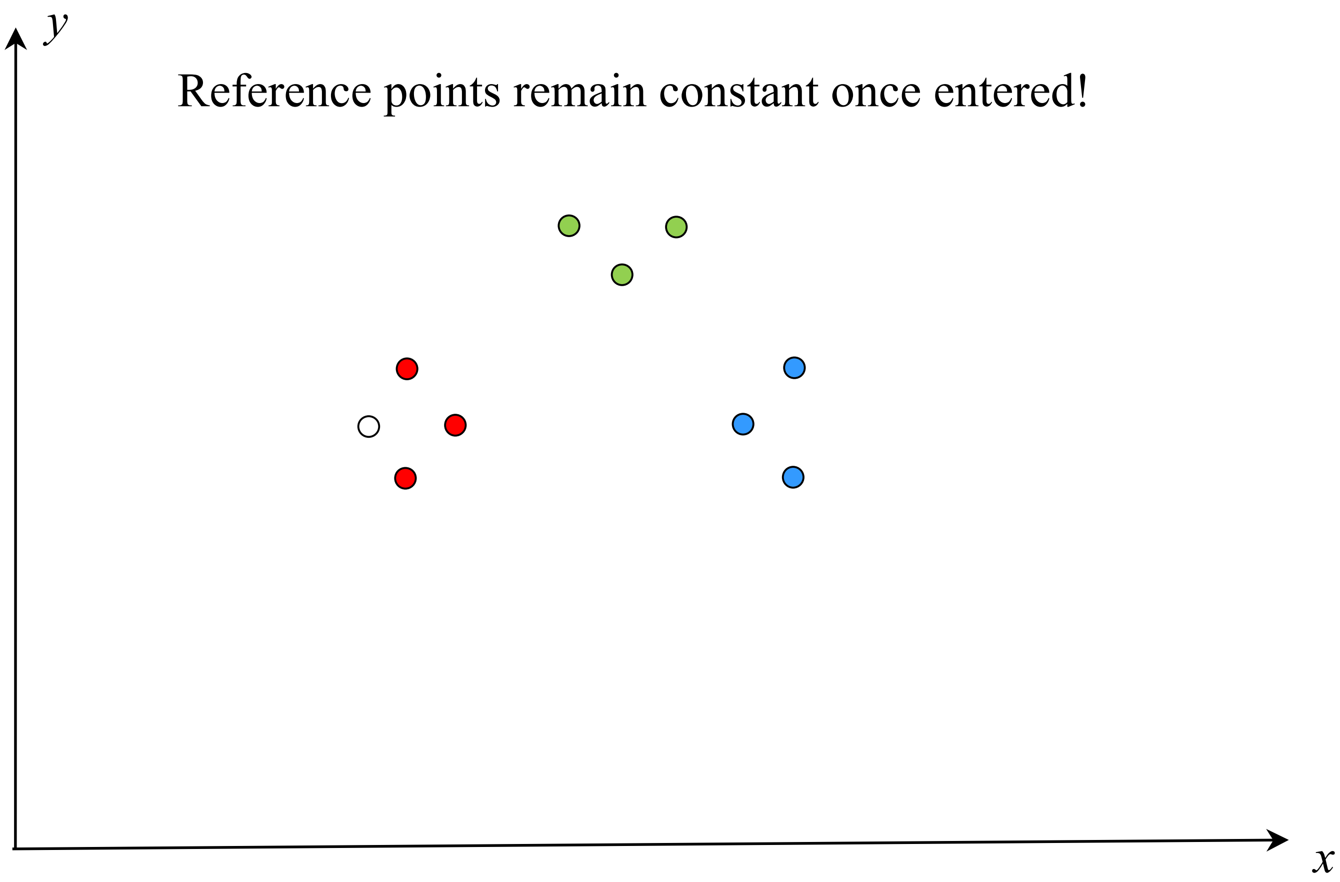
Algorithm description



1. Create reference points
2. Enter classified point
3. Find K neighbors (K=5)
4. Count neighbors
5. Make decision

Algorithm description

Reference points remain constant once entered!



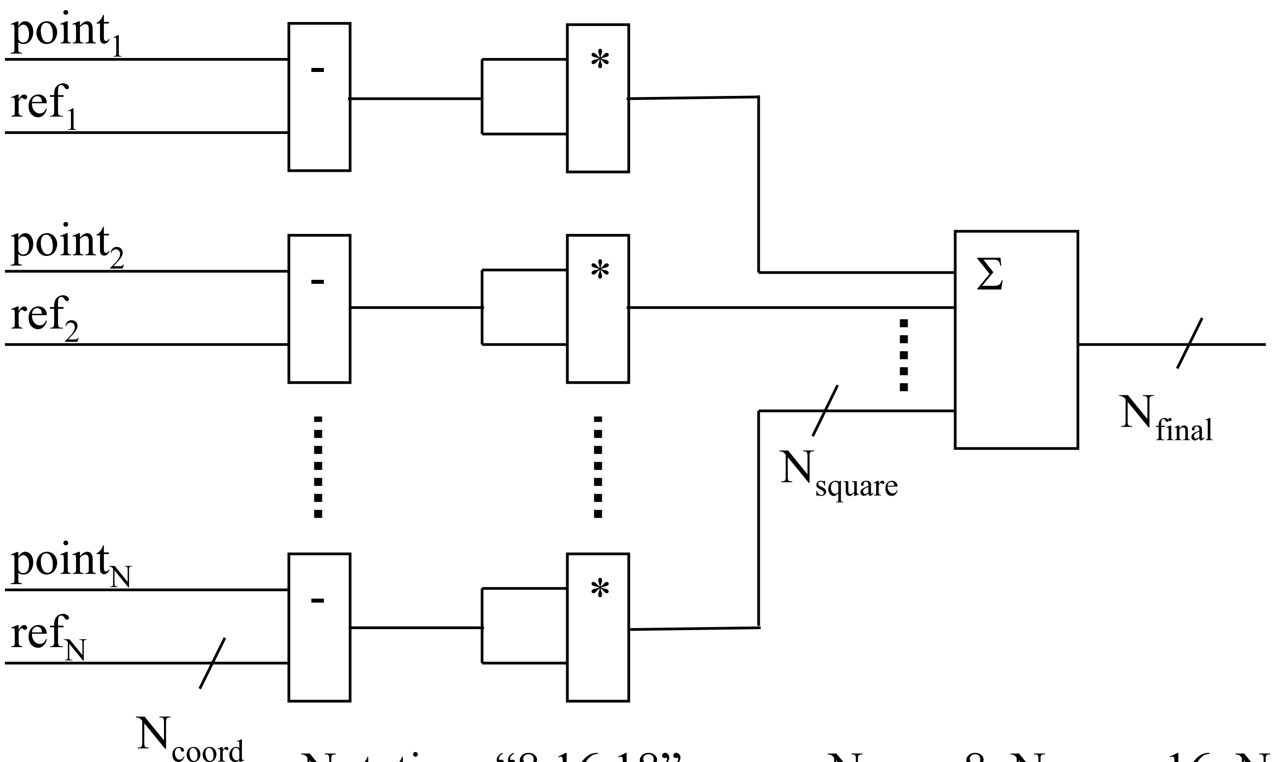
Overview

Data quantization

- Training/testing data:
  - http://archive.ics.uci.edu/ml/datasets.html
  - Iris, Vowel data, Sonar data repositories
- Problem:
  - Coordinates are represented as real numbers
  - Efficient hardware requires integer representation
  - Translation involves rounding (multiplying & clipping)
- Goal:
  - Assess recognition accuracy for different quantization levels.

Data quantization: details

- DFG of a "distance neuron":



Notation: "8,16,18" means N_{coord}=8, N_{square} =16, N_{final} =18

Accuracy assessment program

- C++ program comparing reference floating point implementation with “custom integer” implementations.

```
template <unsigned _Bits> class CustomInteger
template<...> AddCustomUnsignedIntegers()
template<...> GetAbsoluteDifferenceOfUnsigned()
template<...> SquareSignedAndGetMSB
```

- Easy API for comparing implementations:

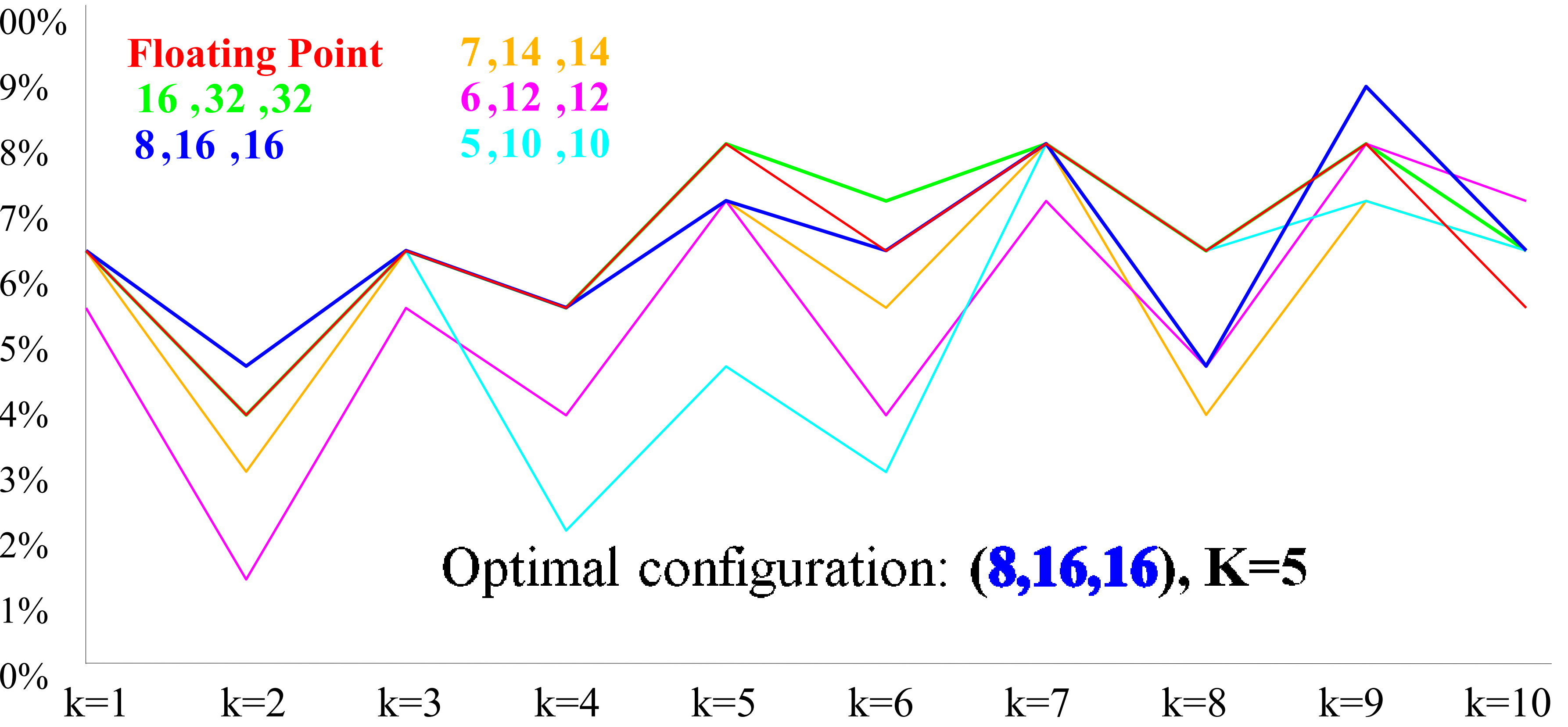
```
tester.Test("FP", new ReferenceKNN(0), ...);
tester.TestInteger<16, 32, 32>, ...);
tester.TestInteger<7, 14, 14>, ...);
...
```

Accuracy assessment program

2 modes of operation:

- Single pass with fixed training/testing border – for verifying hardware design (comparing exact results).
- **P-fold cross-validation – for assessing accuracy:**
  - Split points from every category in P groups of same size
  - For i between 1 and P do:
    - Use i-th group as testing data
    - Use all other groups as training data
  - **All testing performed using 10-fold cross validation**

Recognition accuracy assessment – Iris data



Recognition accuracy assessment - summary

Data set	Dimensions	Bit formula	Neighbors	Efficiency
Iris	4	(8,16,16)	5	98,0%
Vowel	10	(8,16,16)	3	95,4%
Sonar	60	(8,16,16)	3	83,1%

Overview

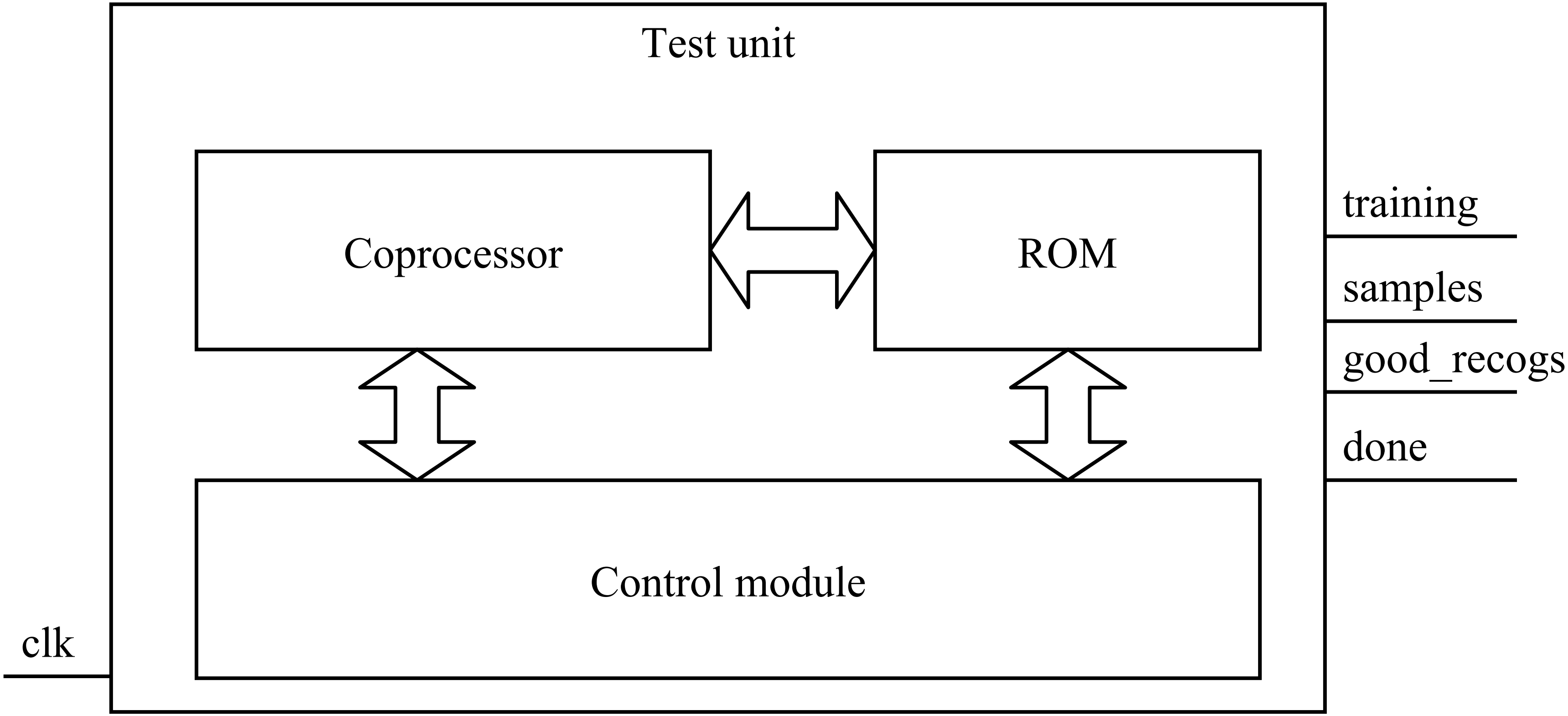
1. Introduction
   - *Problem statement and motivation*
2. The algorithm
   - *Algorithm description*
3. Evaluation
   - *Recognition accuracy evaluation tool*
   - *Evaluation results*
4. Hardware design
   - *Coprocessor testbed*
   - *Coprocessor design*
5. Tools
   - *ROM generator*
   - *Tools summary*
6. Conclusion

Board summary

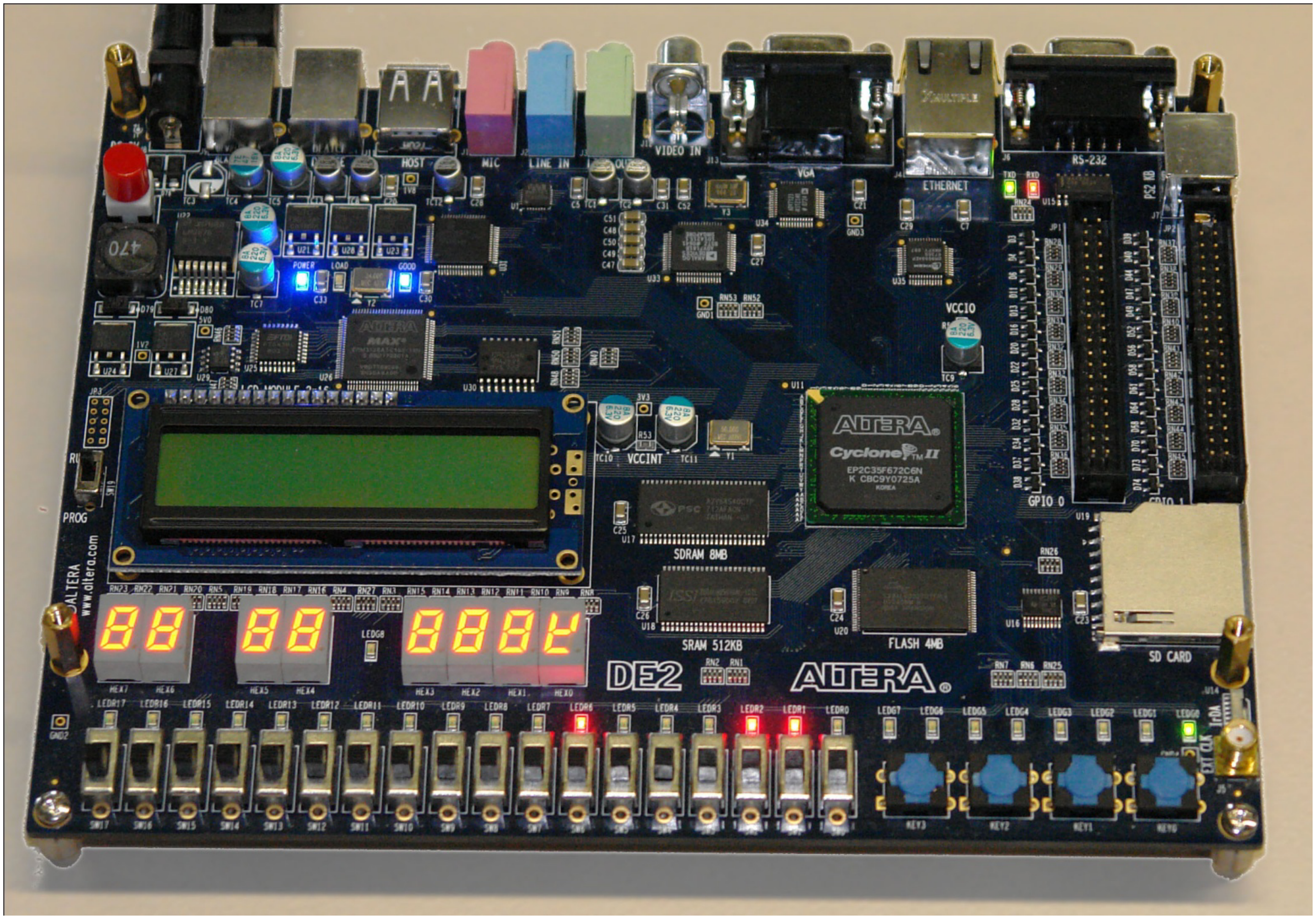
- No external I/O interface:
  - ➢ Testing/training data stored in ROM
  - ➢ Successful recognition count displayed by LED array
  - ➢ All debugging/profiling done in simulator!

- Results are coded as binary numbers:
  - E.g. “1000110” means “70 successful patterns”
  - Overall number of patterns (75) known at compilation
  - That way, “100110” means 93.3% (single run)

Coprocessor testbed



Classifier in action



Testbed interface

Port	Number of bits	Meaning
SamplesDone	16	Amount of samples processed
TotalSuccessfulSamples	16	Total successful recognitions
TrainingMode	1	Training mode active
Done	1	All ROM processed
clk	1	Clock

- No external inputs except clock are required
- Progress and results are easily observed on any FPGA board with 34+ GPIO ports.
- For small test sets less GPIO ports are required.

ROM interface

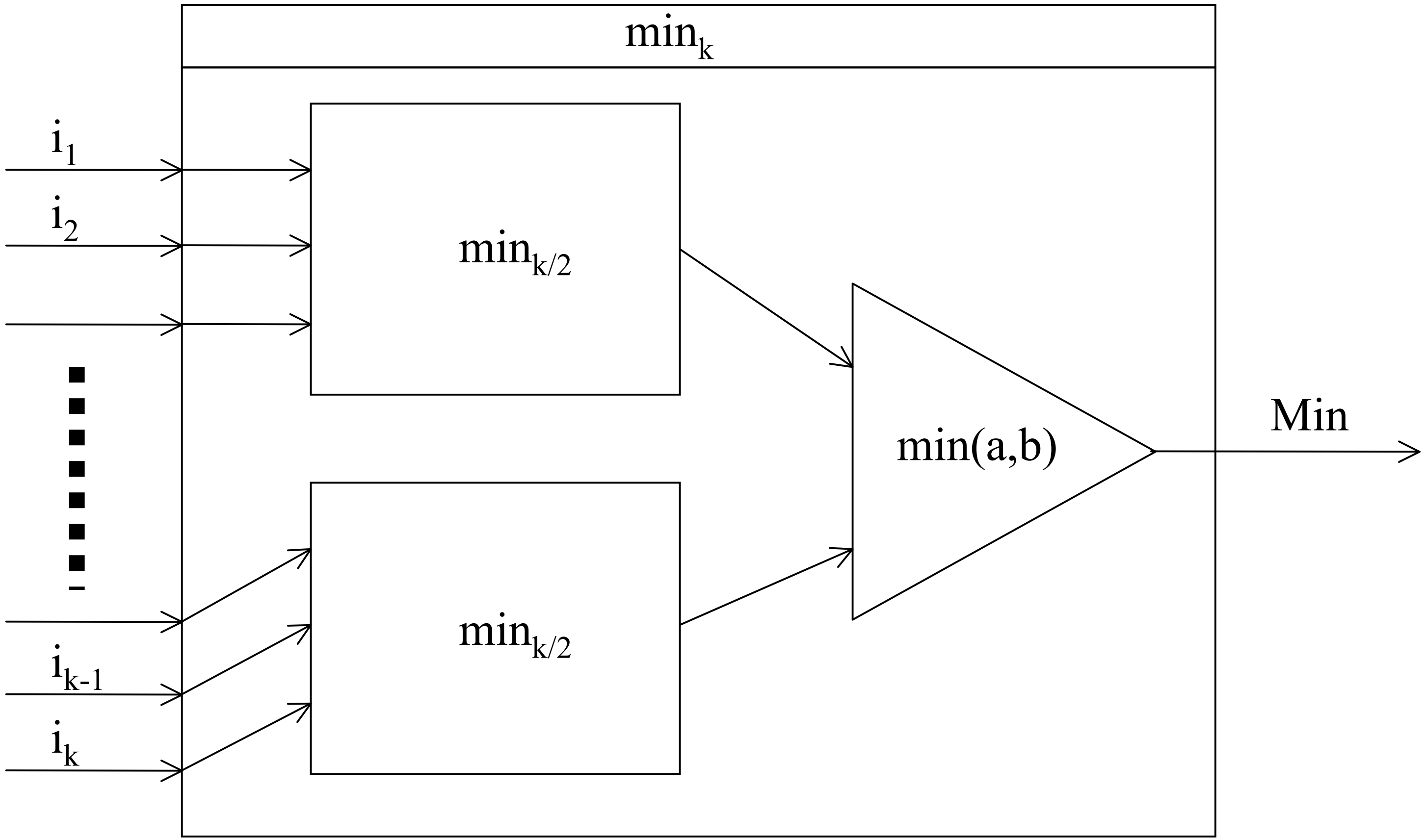
Port	Number of bits	Meaning
SampleNumber	16	Sample number to read
CoordinateNumber	log2(kNumDimensions + 1)	Coordinate number
UseTrainingSet	1	Use training samples
EndOfDataSet	1	No more samples available

- ROM contains normalized sample data.
- UseTrainingSet selects between training and testing data
- When SampleNumber is greater than last valid sample in ROM, **EndOfDataSet** is set to 1 and control logic finishes training/testing.

Coprocessor design

- Parallelism choices:
  - Parallel w.r.t. number of reference points
  - Serial w.r.t. number of dimensions
  - Serial w.r.t. number of neighbors
- Simplifications/assumptions:
  - $N_{square} = N_{final}$
  - Dimension 0 of a point stores category number

Parallel minimum finder



Parallel minimum finder

- Features:
  - Logarithmic depth and calculation time
  - Polynomial complexity (design size)
  - Simple, recursive definition using VHDL generics

- **Real implementation also returns INDEX of minimal element!**

Coprocessor interface

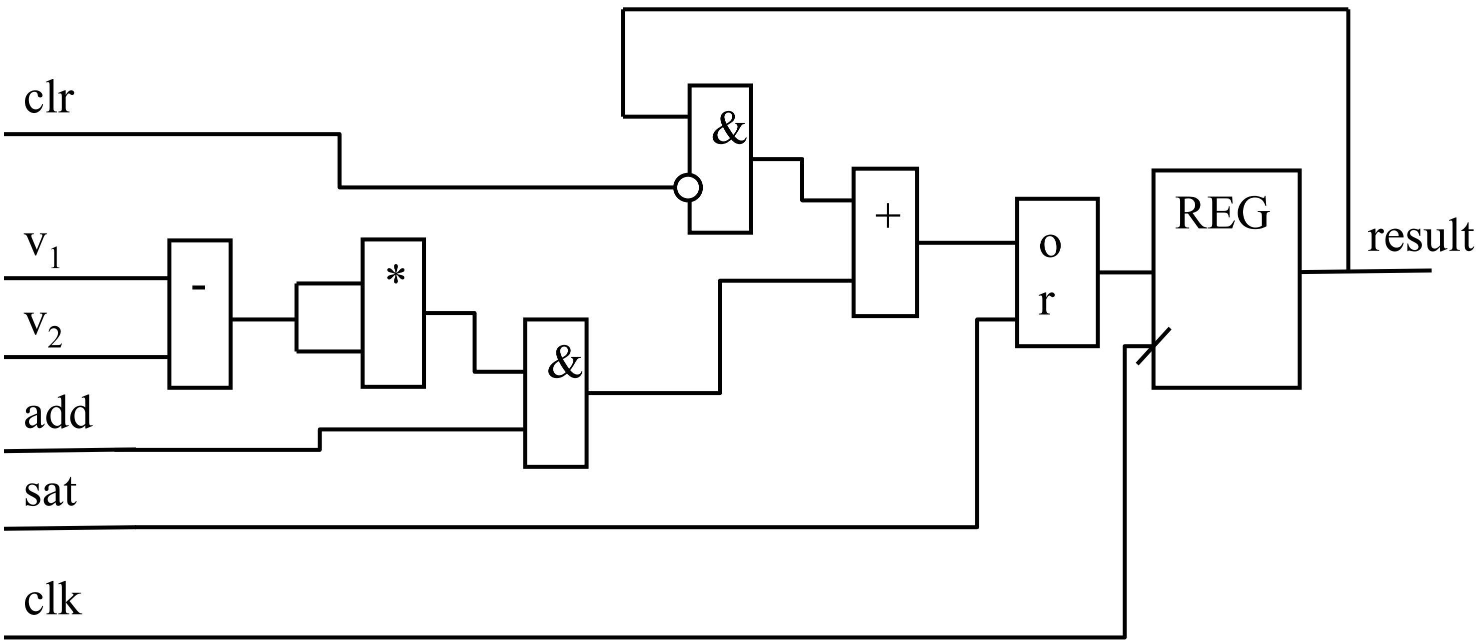
Coprocessor interface is synchronous and supports 3 modes:

- Training mode – refpoint coordinates are entered
- Preparation mode – coordinates for recognized point are entered
- Recognition mode – the coprocessor performs recognition during several cycles and sets RECOG_DONE when done.

Coprocessor interface

Port	Number of bits	Meaning
refpoint_num	log2(kNumRefPoints)	Current refpoint (train mode)
dimension_num	log2(kNumDimensions + 1)	Current dimension
coordinate_value	kBitsPerDimension	Current coordinate
TRAIN_ENABLE	1	Enable training mode
RECOG_PREPARE	1	Enable preparation mode
Reset	1	Reset internal state
RECOG_START	1	Start recognition mode
RECOG_DONE	1	"Recognition done" output
recognized_category	kBitsPerDimension	Recognized category output
clk	1	Clock

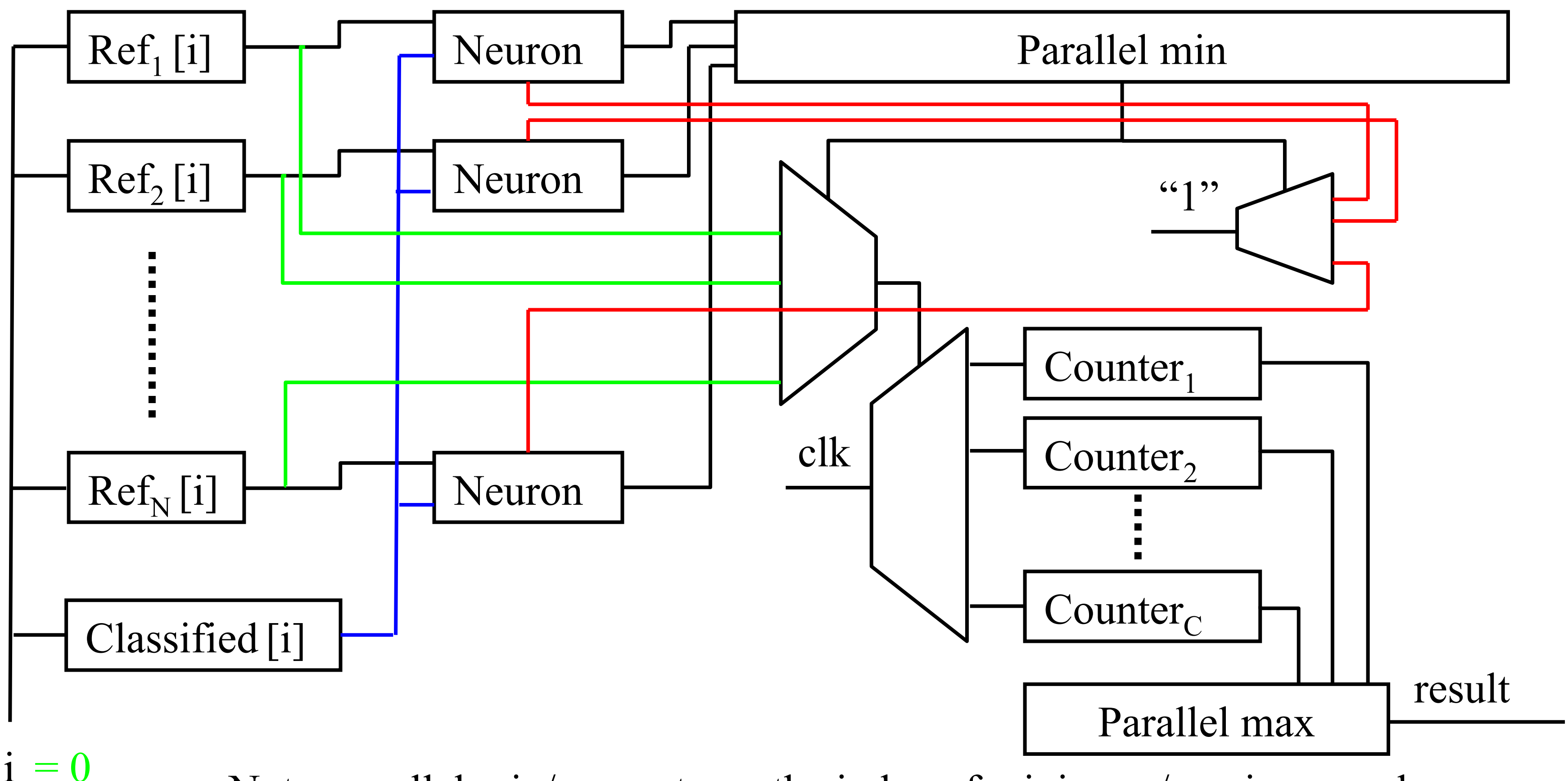
Serial accumulating neuron



Coprocessor control logic

- During recognition the coprocessor goes through a set of phases maintained by control logic:
  - **rpInactive** – no action is performed
  - **rpDistanceCalculation** – serial neurons are adding
  - **rpNeighbourCounting** – neighbor counters are active
  - **rpVoting** – maximum score is detected
  - **rpDone** – the result is forwarded to output

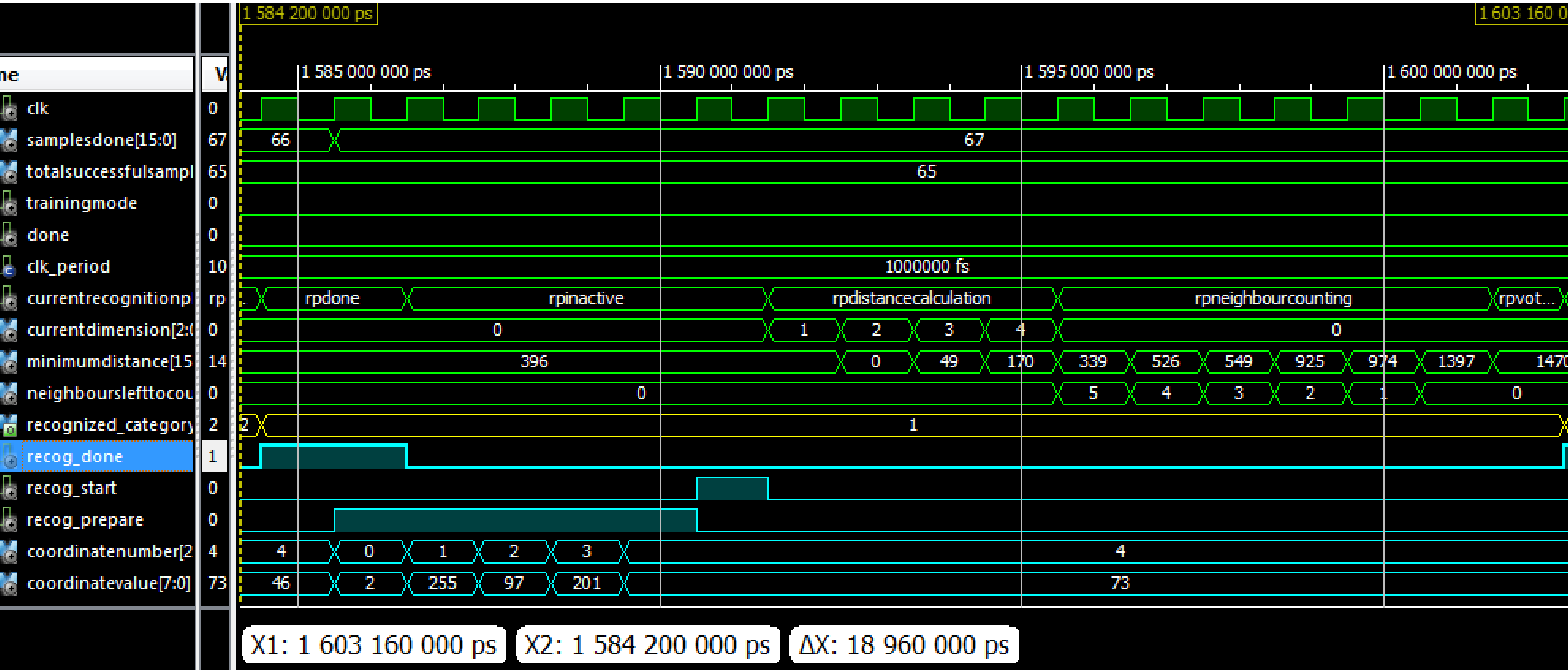
Coprocessor data path



Note: parallel min/max returns the index of minimum/maximum value

Coprocessor performance

Performance estimation using Xilinx ISE Webpack:



Performance evaluation

- Cycles required for single classification:
  - $T = N_{dimensions} + N_{neighbours} + 3$
  - Iris sample: T = 4 + 5 + 3 = 12 cycles
  - 50 MHz clock yields ~ 4 million recognitions/sec
  - 5 neighbours ⇔ 20 million connections/sec
  - 4000 times faster, than classifier.exe on T7300 CPU
- **But!:**
  - Classifier.exe is not optimized for performance
  - However, 50 MHz is not the limit!

Overview

1. Introduction
   - *Problem statement and motivation*
2. The algorithm
   - *Algorithm description*
3. Evaluation
   - *Recognition accuracy evaluation tool*
   - *Evaluation results*
4. Hardware design
   - *Coprocessor testbed*
   - *Coprocessor design*
5. Tools
   - *ROM generator*
   - *Tools summary*
6. Conclusion

ROM generator

- **romgen.pl** generates **TestROM.vhd** from any pair of training and testing files.
- All constants (data widths, refpoint count, etc.) are updated automatically.
- No manual work required to convert arbitrary train/test set into hardware.

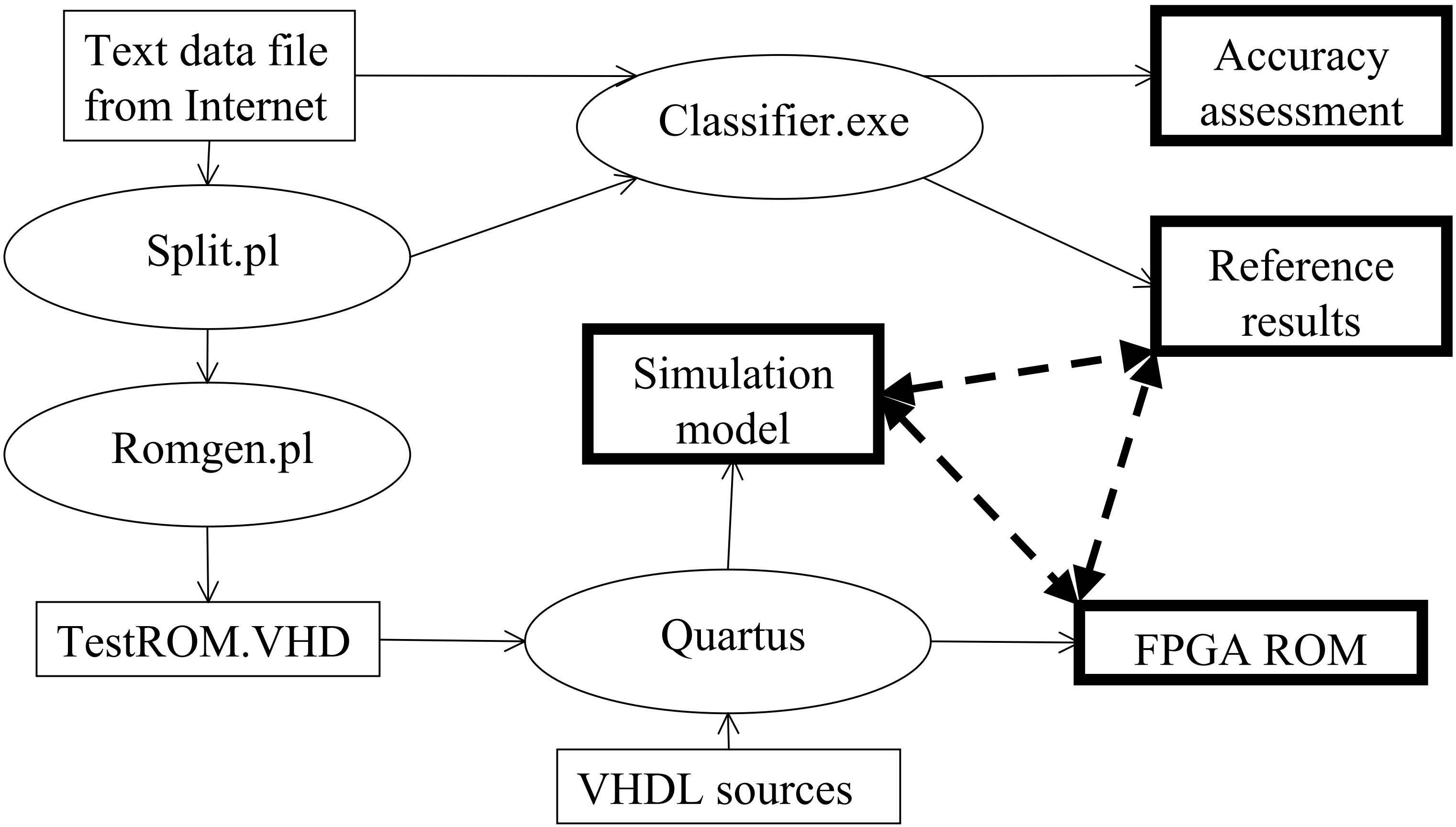
Constant summary

Constant	File	Default value
kBitsPerDimension	TestROM.vhd	8
kBitsAfterSquaring	TestROM.vhd	16
kNumDimensions	TestROM.vhd	AUTOMATIC
kNumCategories	TestROM.vhd	AUTOMATIC
kNumRefPoints	TestROM.vhd	AUTOMATIC
kNumNeighbours	TestROM.vhd	5

Tools summary

- **romgen.pl** generates **TestROM.vhd** from any pair of training and testing files.
- **split.pl** splits data file into training and testing part.
- **scanranges.pl** detects the amount of bits for lossless quantization.
- **classifier.exe** (C++) compares different values of K and different quantization modes producing a text summary and a chart.

Tools summary



Conclusion

- Splitting the work into C++, PERL and VHDL parts allowed:
  - Quickly testing/comparing quantization options
  - Having reference software design to test hardware against
  - Automatically analyzing/converting data files
  - Generating ROM files from data files
- Parallel VHDL implementation exploits parallelism in:
  - Logarithmic-depth parallel min/max
  - Parallel array of neurons
- Results of VHDL design on Altera board matched C++ simulation completely.

Future work

The performance can be significantly improved by:

- Analyzing time-accurate post-synthesis model:
  - Finding maximum frequency
  - Splitting complex operations over several cycles (e.g. parallel max5 is twice faster than min75)
- Exploiting additional parallelism:
  - Parallel computation w.r.t. dimension count
- Introducing pipelining:
  - Running **rpDistanceCalculation, rpNeighborCounting, rpVoting** in parallel.

Thank You!