FPGA design for Intrusion Detection/Prevention applications

PRESENTATION

MICHAEL DYRMOSE

DTU Fotonik
Department of Photonics Engineering

PRESENTATION OVERVIEW

- Introduction
- Background
- Implementation
- Results
- Conclusion

INTRODUCTION

INTRODUCTION

- Internet connectivity on many new platforms
  - New possibilities for end-users
    - Higher bandwidth demands
  - Increase in number of attacks
- Attack detection and prevention
  - Network traffic monitoring
    - Identify potential threats and attacks
    - Gather forensic evidence of successful attacks

INTRODUCTION

- Network traffic monitoring
  - Intrusion Detection System, e.g. Snort
    - Issues
      - Higher network speeds → More resources needed!

        (Packets are ignored if the flow is too high to handle!)
  - Offloading on dedicated hardware
    - FPGA (Field Programmable Gate Array)
      - Implement intrusion detection rules
      - Parallel execution of rules

BACKGROUND

BACKGROUND

- Snort
  - Popular open-source security package
  - Rules based on known attack patterns
    - Payload inspection
      - Simple string search
        - » abbbabccccccd
      - Regular expressions (RegEx)
        - » ab+abc*d

- Goal:

  Offload RegEx-based rules to dedicated hardware

BACKGROUND

- FPGA (Field Programmable Gate Array)
  - Programmable
    - Can be configured to behave like a specific HW design
  - Used in design phase of hardware production
    - Evaluate design before mass-production
    - VHDL source code

BACKGROUND

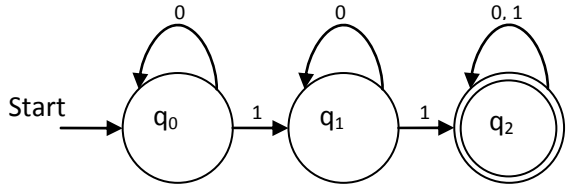
- Regular Expressions
  - String identifying patterns
  - Concise and flexible
    - Example: (a|b)cd matches ‘acd’ and ‘bcd’

BACKGROUND

- Regular Expressions
  - Operators
    - Alternation (|)
      - a|b matches {'a', 'b'}
    - Quantification (?, +, *, {x,y})
      - $a^{?}b$ matches {'b', 'ab'}
      - $a^{+}b$ matches {'ab', 'a…ab'}
      - $a^{*}b$ matches {'b', 'ab', 'a…ab'}
      - a{2,4}b matches {'aab', 'aaab', 'aaaab'}
    - Grouping ( )

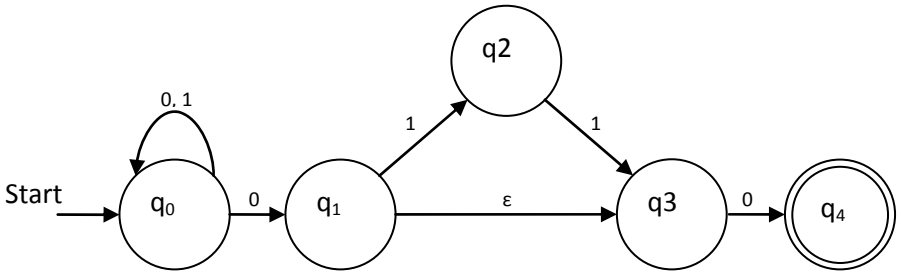
BACKGROUND

- Finite automata
  - Used to describe regular expressions
  - Directed graph
    - Nodes (circles)
    - Transitions (arrows)
    - Run on a string of input symbols
      - Input transfers current state to next node
  - Deterministic/Non-deterministic



BACKGROUND

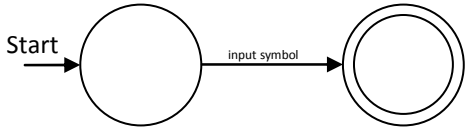
- Finite automata
  - Non-deterministic finite automata (NFA)
    - Multiple current states
      - The same input symbol can occur at multiple transitions from the same node
      - ε-transitions transferring the state at no input
    - Convert regular expressions to NFA



BACKGROUND

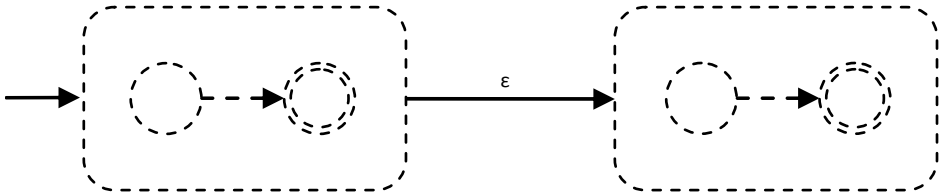
- NFAs are built by using construction rules
  - Each part of a regular expression is an expression itself

  – Basic rule
    - Single input symbol

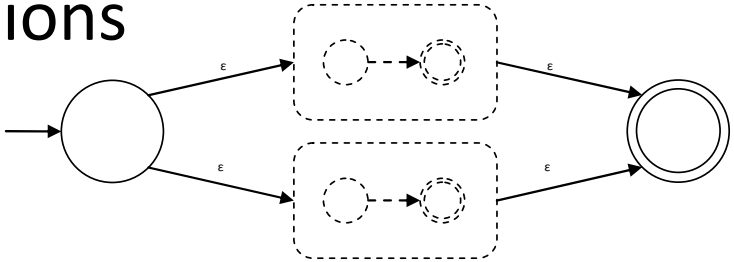


BACKGROUND

- Concatenation
  - Combine a sequence of expressions using ε-transition

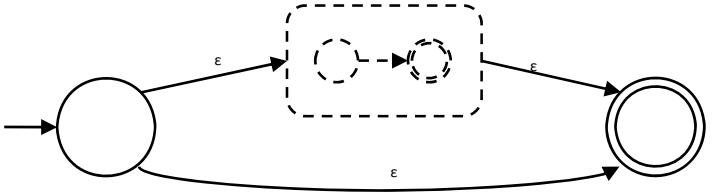


- Alternation (|)
  - Alternate between expressions

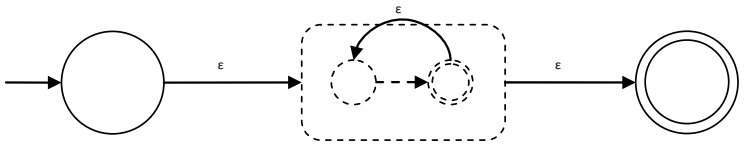


BACKGROUND

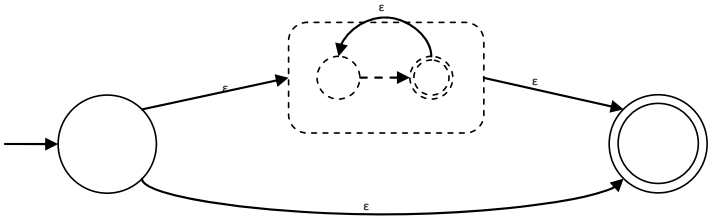
- ?-quantifier
  - Matches 0 or 1 occurrence



- +-quantifier
  - Matches at least one occurrence



- *-quantifier
  - Matches any number of occurrences



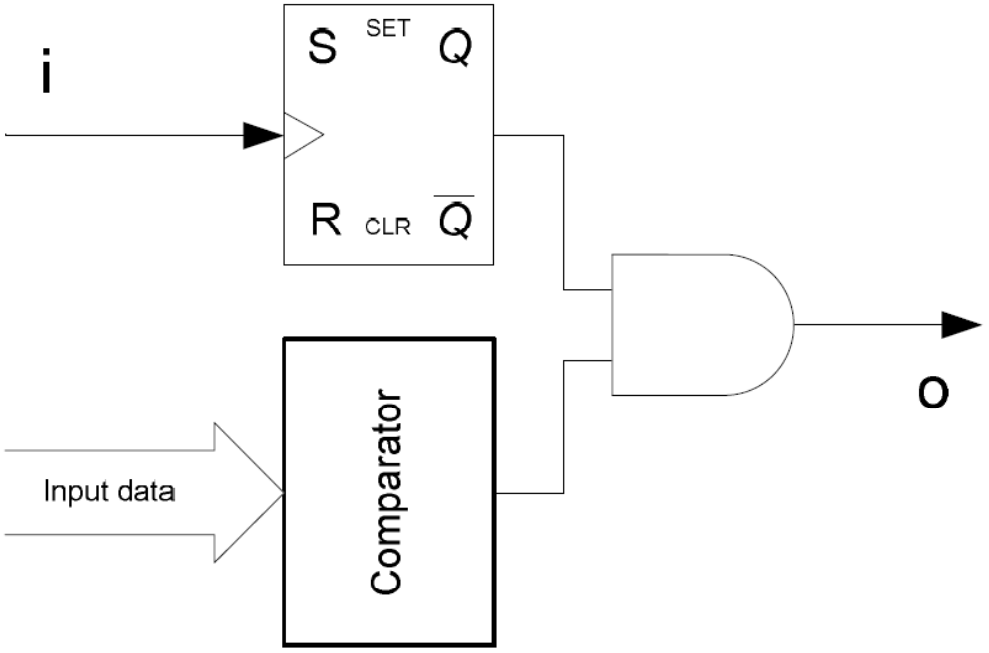
IMPLEMENTATION

IMPLEMENTATION

- Constructing regular expressions in hardware
  - Building FPGA designs corresponding to NFAs
    - Logical structures matches construction rules
    - Designs in VHDL source-code
  - SNORT2VHDL
    - Automatically convert RegEx-based Snort-rules to VHDL source-code

IMPLEMENTATION

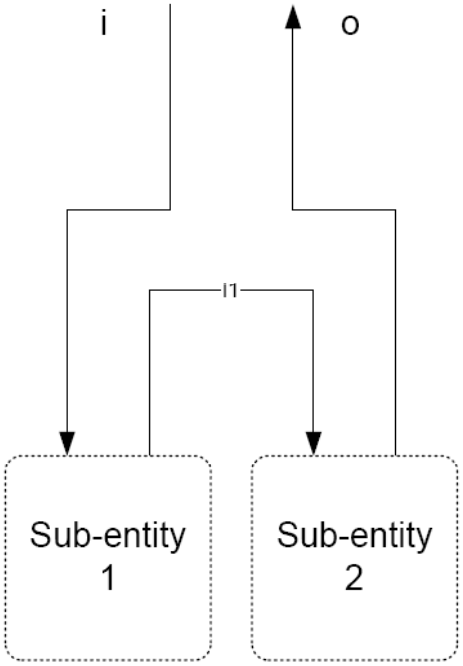
- Logical structures
  - Comparators



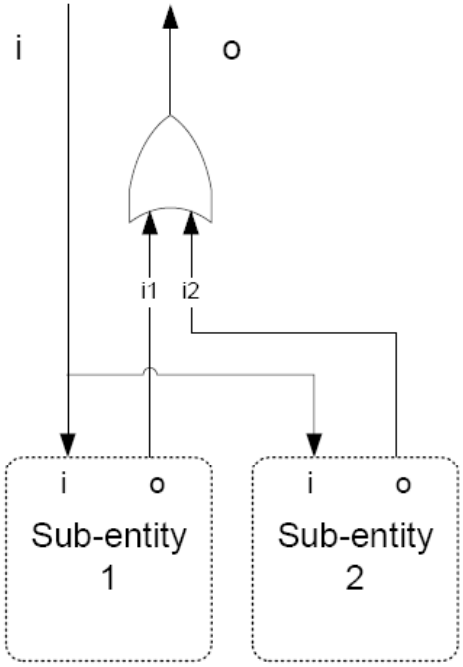
IMPLEMENTATION

- Logical structures

Concatenation

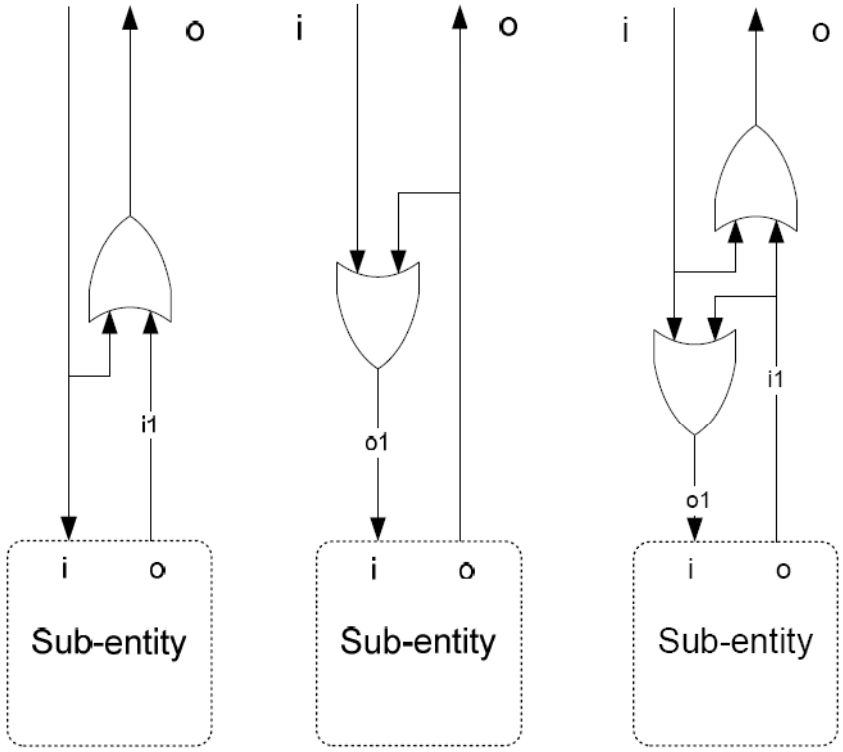


OR



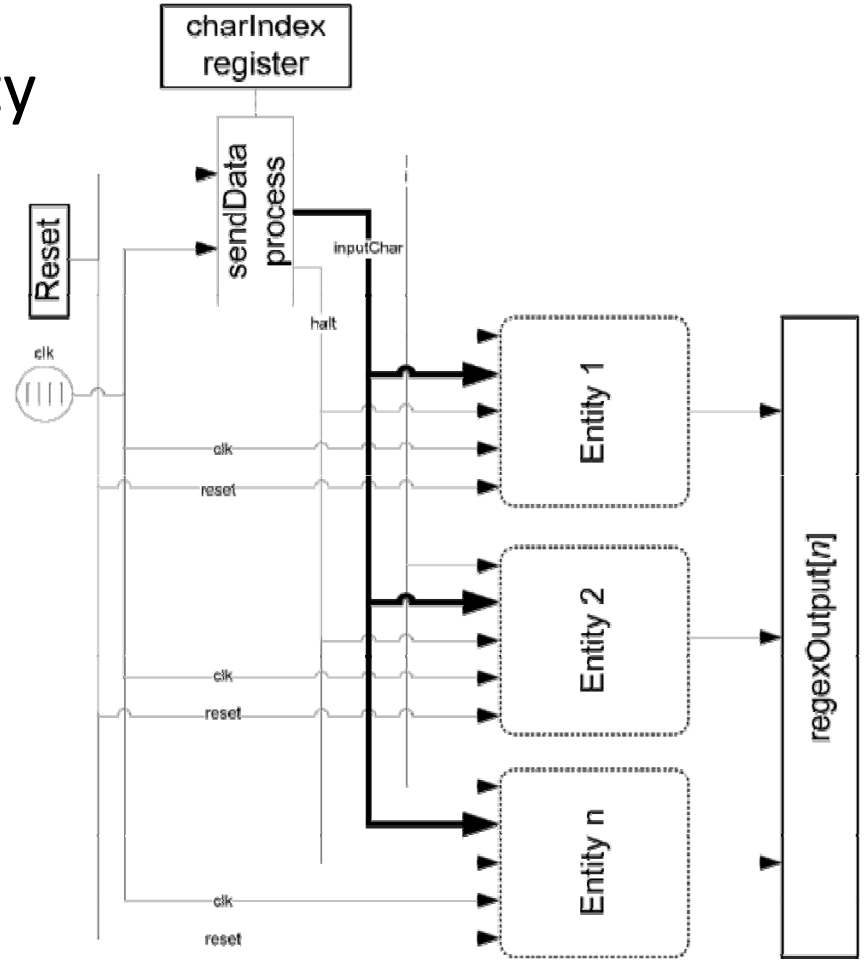
IMPLEMENTATION

- Logical structures
  - Basic quantifiers
    - ?-quantifier
    - +-quantifier
    - *-quantifier



IMPLEMENTATION

- Logical structures
  - Top-level control entity

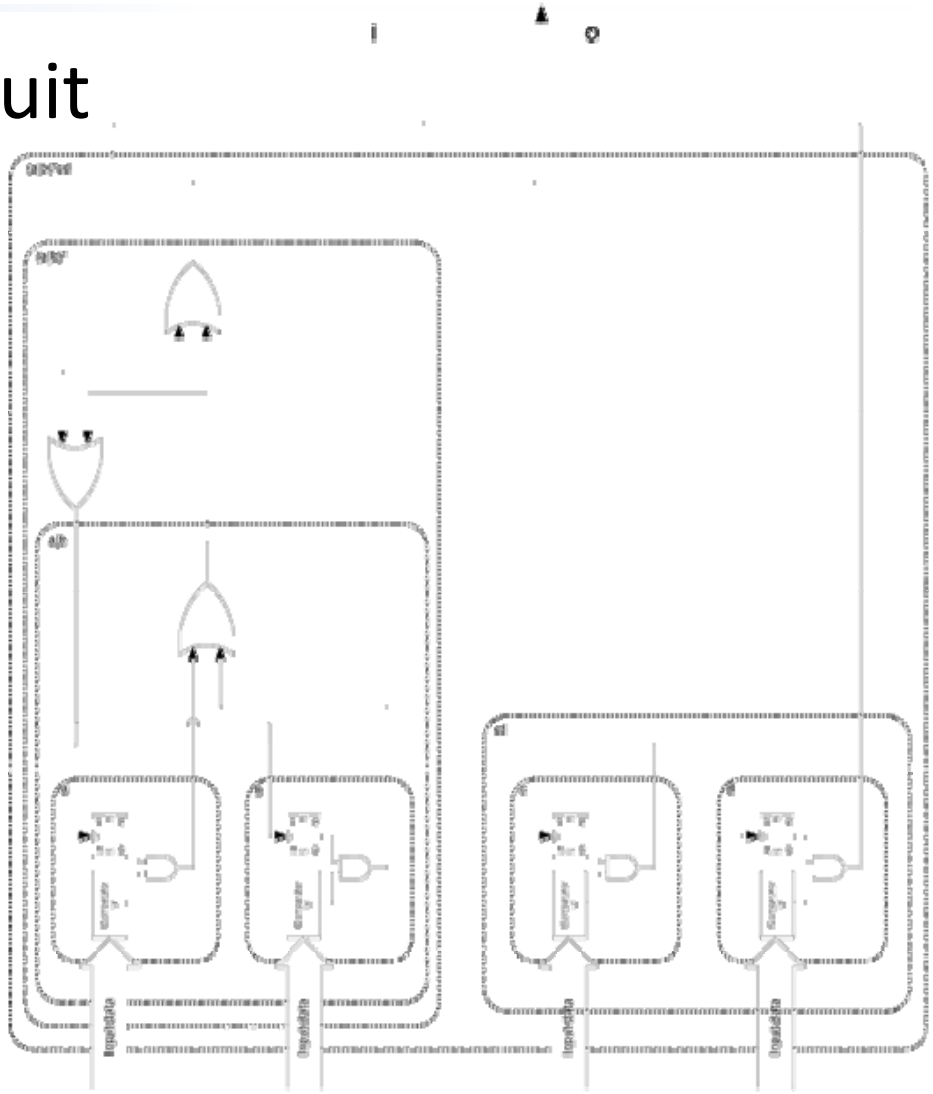


IMPLEMENTATION

- Complete logical circuit

  (a|b)*cd

  Shown without top-level entity



RESULTS

RESULTS

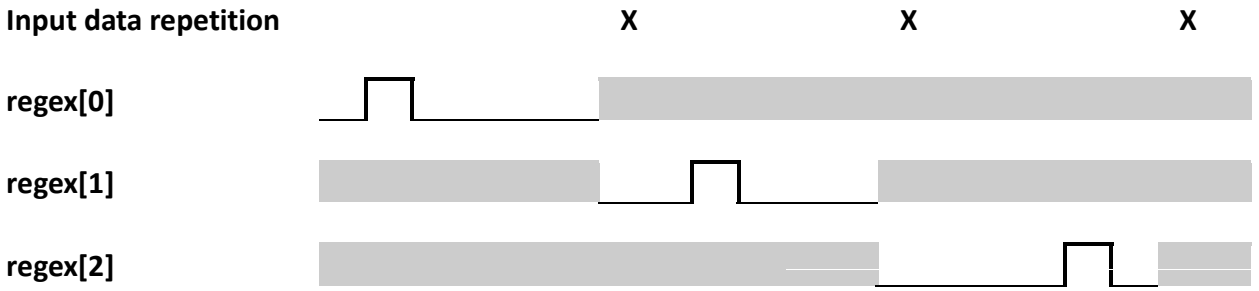
- Evaluation is two-fold
  - Functionality
    - Prove correct implementation
  - Potential
    - How much can be stored on the provided board
    - What can be achieved by using a larger board

RESULTS

- Functionality
  - Prove correct implementation of RegEx based rules
    - > 5000 regular expressions in Snort rule-set
    - 10 rules selected for the evaluation
      - Make up a small rule-set that uses all implemented operators
      - User-generated input data matching all rules
    - Compile rules with SNORT2VHDL and observe output
      - Rules matched?
      - Parallel implementation?

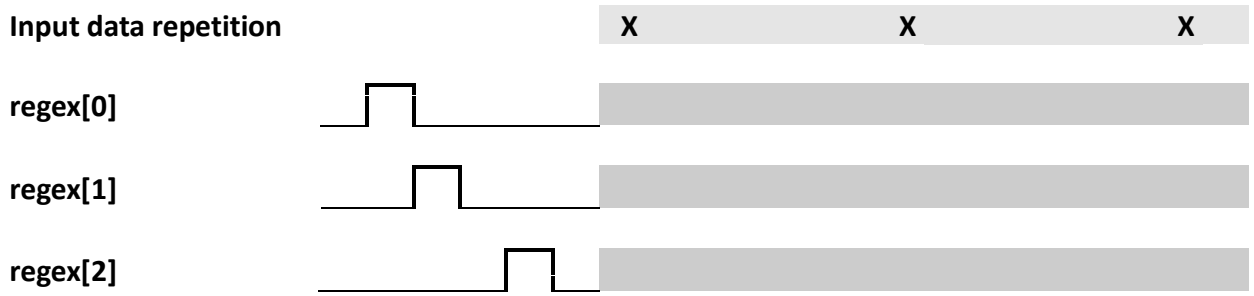
RESULTS

- Evaluating functionality
  - Standard Snort setup – single thread
    - The input data has to be repeated for each regular expression



RESULTS

- Evaluating functionality
  - FPGA implementation – parallel execution
    - The input data is parsed by each regular expression simultaneously

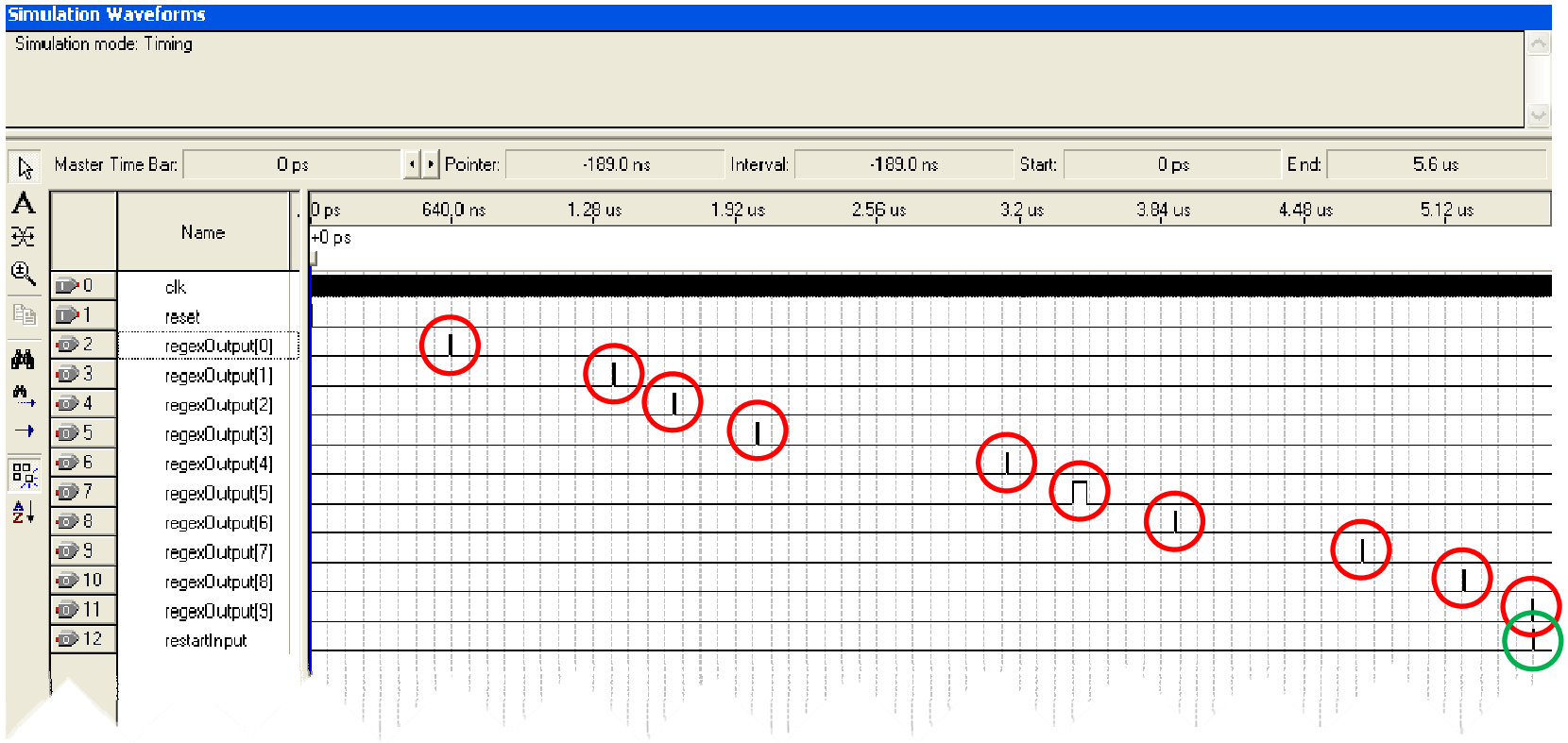


RESULTS

- Evaluation criteria
  1. The output signal from each implemented expression should be high at least once during simulation
  2. All expression has to provide high output before input data is repeated

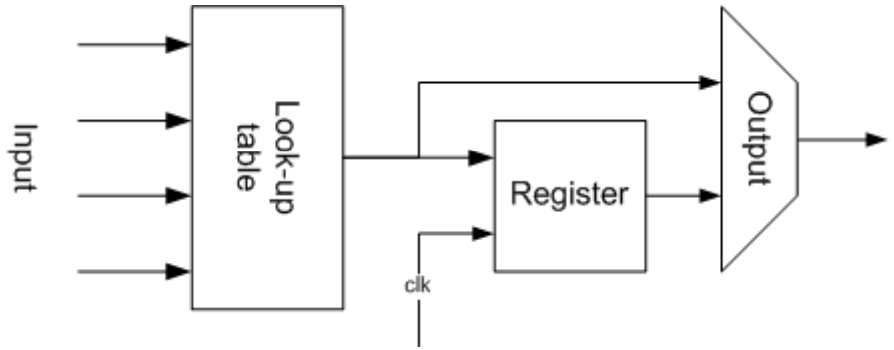
RESULTS

- Simulation output should match parallel scenario



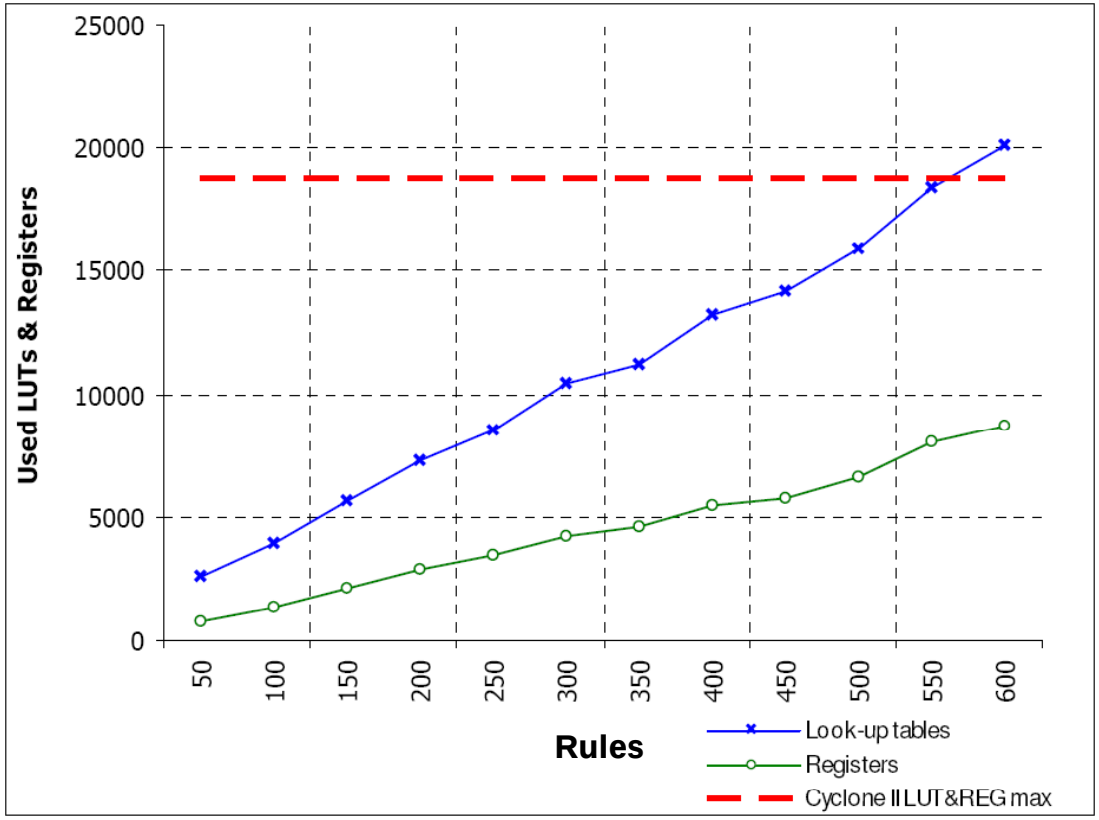
RESULTS

- Potential
  - Finite number of logic elements
  - Standard FPGA logic block implementation



RESULTS

- Resource consumption (Cyclone II)



RESULTS

- Cyclone II is too small to fit all rules
  - Available LUTs & Registers: ca 19000
    - Maximum: < 600
- Stratix II – high end board!
  - Available LUTs & Registers: ca 150000
    - Recompilation necessary due to more advanced architecture

RESULTS

- Resource consumption (Stratix II)

RESULTS

- Stratix II resource usage
  - Resource usage at 600 rules

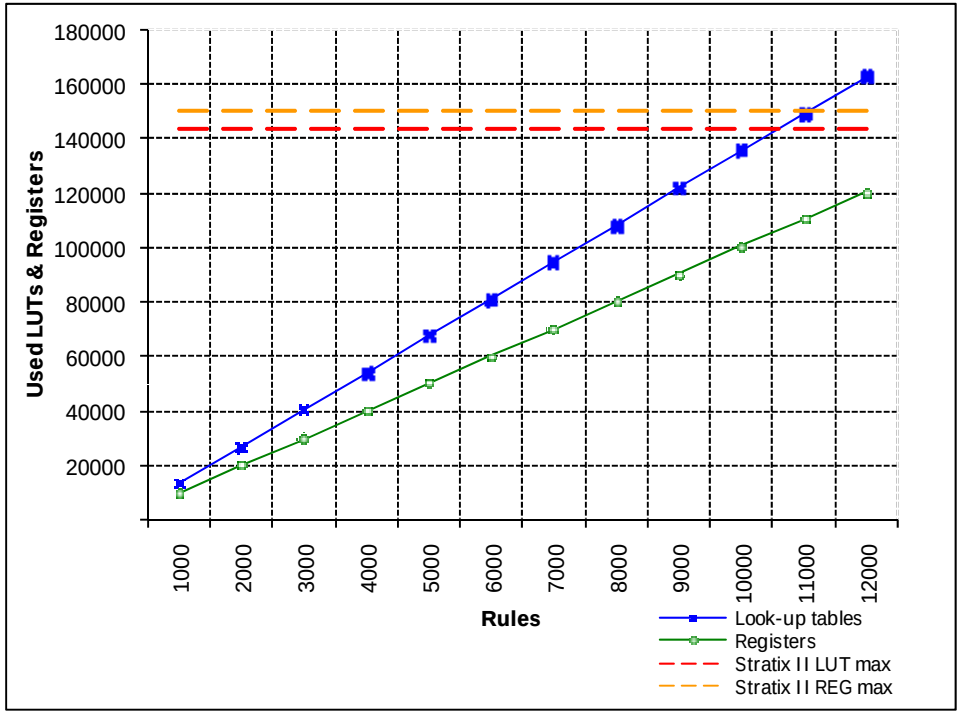
$$\frac{7464\ LUTs}{143520\ LUTs} = 5.2\ \%$$

$$\frac{5910\ Registers}{150336\ Registers} = 3.9\ \%$$

  - Projected resource usage
    - Assume linear expansion
    - Average number of LUTs/rule: ~ 13

RESULTS

- Estimated resource consumption (Stratix II)



RESULTS

- Performance
  - Stratix II board synthesizes the design at 151.35 MHz
    - Compilation result at 200 implemented rules
  - 1 input symbol (8 bits) pr. clock cycle
    - 151.35 MHz * 8 bits = 1210.8 Mbit/s
- Related research
  - Workstation performance (dec. '07)
    - 36 Mbit/s @ 200 rules
    - Performance increase:

» 1210.8 Mbit/s / 36 Mbit/s = 33.6

RESULTS

- Performance
  - Estimated PC performance at 10000 rules
    - 36Mbit/s x (200 rules/10000 rules) = 0.72 Mbit/s
  - Assuming FPGA speed is independent of number of rules
    - Estimated performance increase:

      1210.8 Mbit pr. sec / 0.72 Mbit pr. sec = 1681

CONCLUSION

CONCLUSION

- Using dedicated hardware in IDS
  - Parallel execution of rules
  - Performance independent of number of rules
  - High performance increase
    - Performance increase by a factor of over 1000

END OF PRESENTATION

WWW.MITSPECIALE.DK