Breaking The GSM A5/1 Cryptography Algorithm with Rainbow Tables and High-end FPGAs

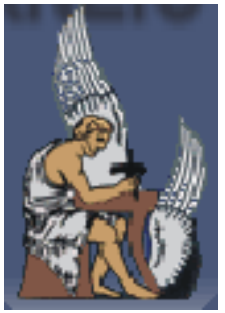
Maria Kalenderi[1], Dionisios Pnevmatikatos[1],

Ioannis Papaefstathiou[1,2], Charalampos Manifavas[3]

[1]ECE Department, Technical University of Crete

[2]Synelixis Solutions Ltd

[3]Applied Informatics & Multimedia Department, Technological Educational Institute of Crete



Presentation Overview

- Motivation
- A5/1 Cryptographic algorithm
- Crypto-attack with Rainbow Tables
- Our Rainbow Table Creation Engine
- Conclusions

Motivation: Why the A5/1 Alg?

- Cell-phone privacy is obviously important
- A5/1 is used in mobile cell phones (GSM) for the encryption of the exchanged information (voice/SMS) between mobile and base stations
- It is a *stream* cipher
- Optimized for efficient and cheap hardware implementation
- A5/1 used only for encryption, authentication in GSM phones is handled with other algorithm

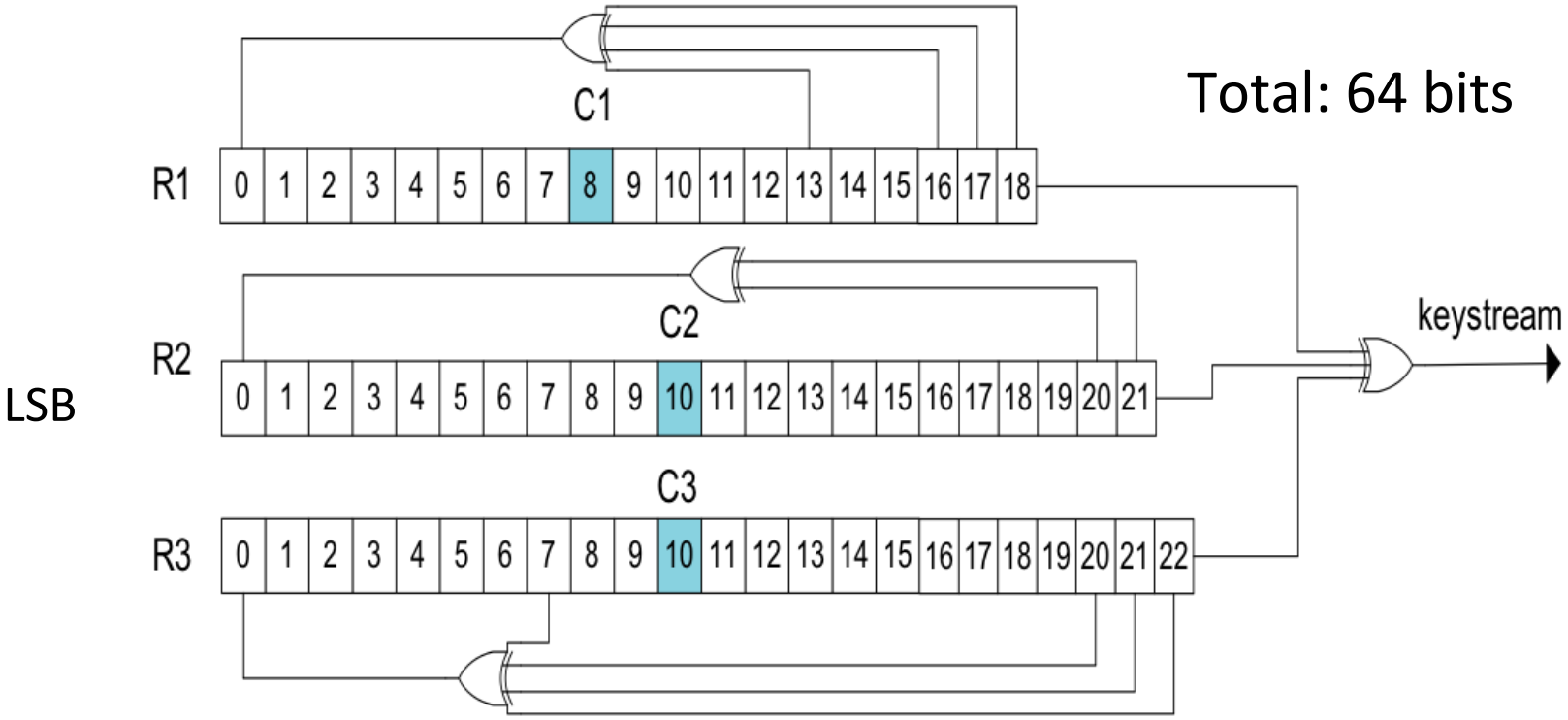
Motivation: Why Rainbow Tables?

- Rainbow tables are one of the most efficient methods for cracking passwords, encrypted by different cryptographic algorithms
- They trade ***online*** computation for one-time ***offline*** computation + ***storage***
- One-time computation is still expensive! We exploit parallelism and high-end FPGAs to construct Rainbow tables up to ***thousand*** of times faster than single threaded software.

A5/1 structure & operation

- 64 bit state, 3 LFSRs, 3 designated "clock" bits. Their majority generates a "clock" signal
- Process:
  - Initialize all 3 LFSRs to zero
  - Serially put in the encryption key (64 bits)
  - Serially put in the Frame Number (22 bits)
  - Now at **"Initial State"**
  - Clock 100 cycles and discard output
  - Next 228 clocks produce two 114 bit values to be used encrypting the uplink and decrypting the downlink (reverse use in the Base station)
  - XOR data and key to produce output

A5/1 structure (LFSRs)



"Clock" Rule = Majority ($C1_8$, $C2_{10}$, $C3_{10}$)
Controls when keystream bits are generated

Crypto-attack approaches

- Cryptography is used to protect sensitive information
- Since the introduction of cryptography, we have attack efforts
- Crypto attack: reverse the mapping of the key to the ciphertext. For *n*-bit functions we can:
  - Use an ***exhaustive search*** computing an average of 2^{n-1} values until the target is reached
  - ***Precompute and store*** a table of 2^n input/output pairs. Then, to invert a particular value, we only need a *single* table lookup

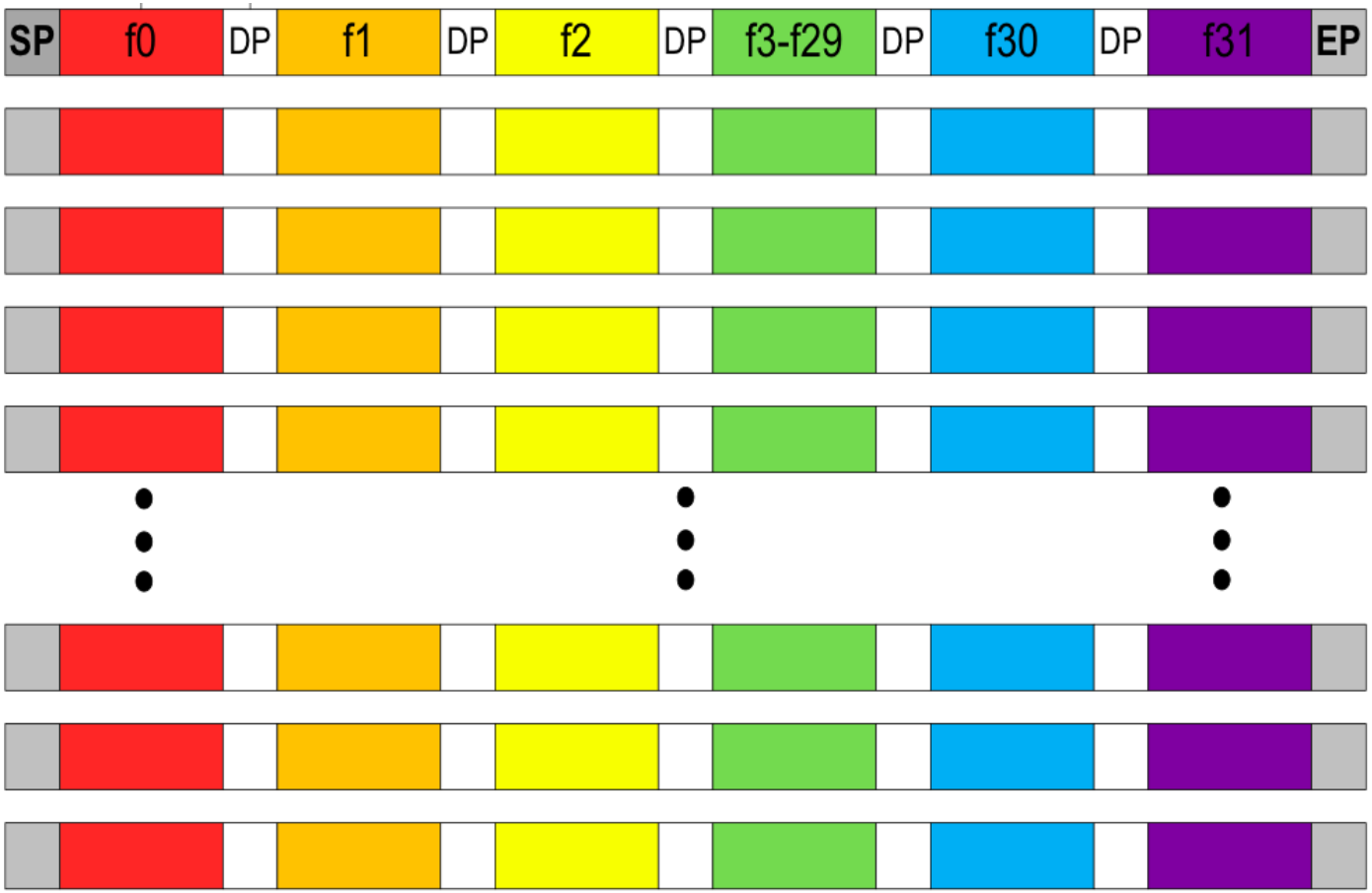
Space-Time tradeoff

- Lies in-between those two previous options (Originally proposed by Hellman)
- Idea: Do not store the entire table, but starting points from which we can re-compute at run-time several other points (for a ***chain***)
- Precomputation time of this approach is still in the order of 2^n
- Memory complexity is $2^{2n/3}$ and the inversion of a single value requires only $2^{2n/3}$ function evaluations.

Rainbow Tables

- Oechslin improved Hellman's approach (in 2003 & 2008)
- More compact representation (***less memory!!!***)
- Use a reduction function after each hash as the input for the next chain link (not needed in this work)
- The initial and final passwords of the chain comprise a rainbow table entry and they are called Startpoint (SP) and Endpoints (EP) respectively.
- Chains contain only ***Distinguished Points*** **(DPs)** to further improve efficiency (only keep results with a specific property)
- Used in many attacks (Nohl used it for breaking A5/1 using GPUs)

Rainbow Table Illustration

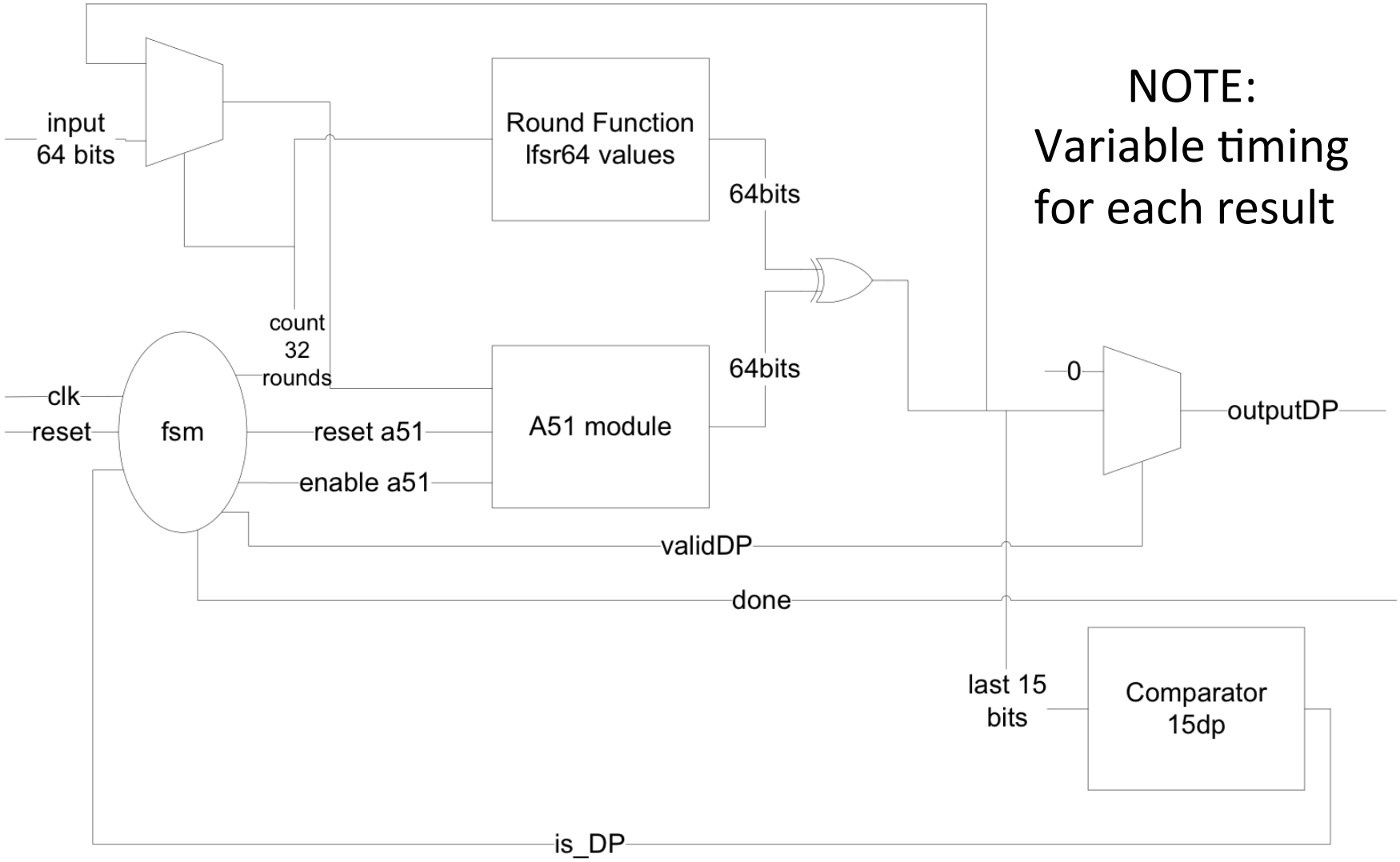


{SP,EP} pairs stored, the intermediate entries are re-computed!

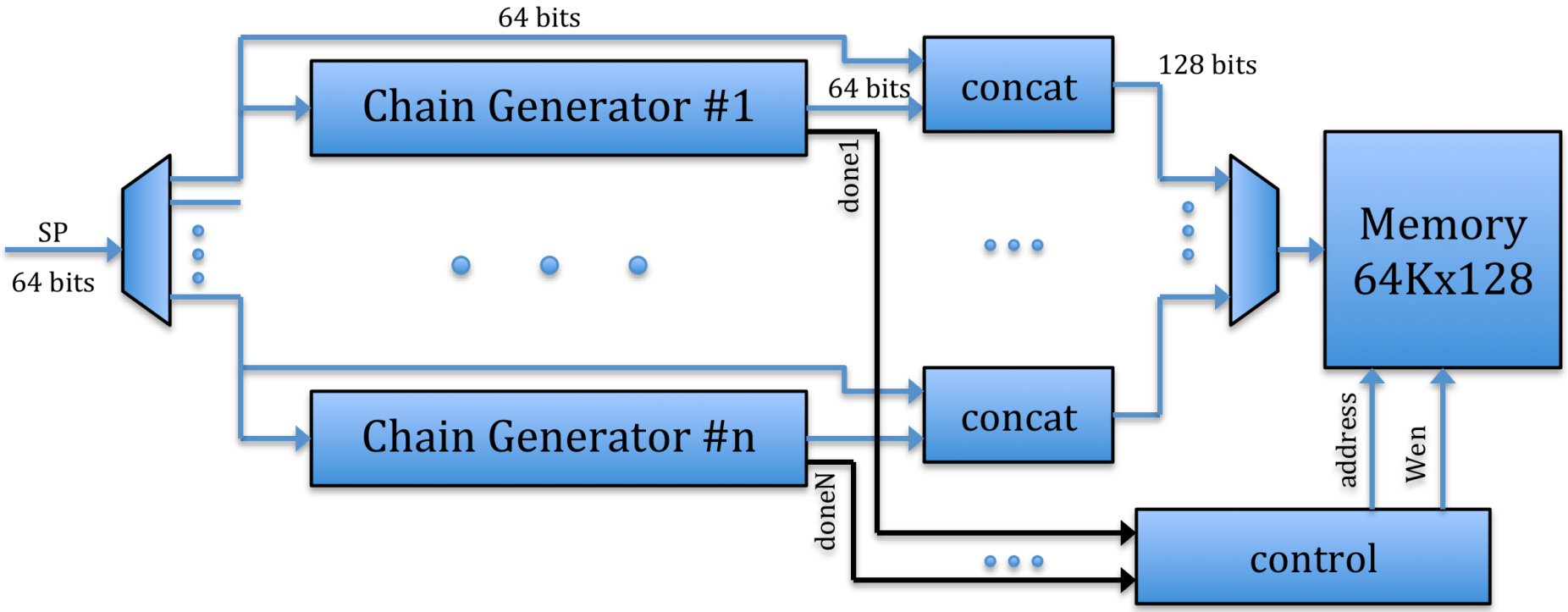
Our Rainbow Table creation approach

- Use a hardware implementation of A5/1
- Sequential but small!
- Exploit parallelism in creating different chains
  - Start with ***random*** Start-Points (A5/1 initial state)
  - Produce 32-entry chains (count only DPs)
  - Use as many ***parallel chain engines*** as can fit in FPGA
  - Deal with memory access problem (write results)
    Compare with S/W running the same algorithm
- Extrapolate/compare w/ published GPU results

A5/1 Chain creation engine



Connecting Multiple Parallel Engines



Mem size: 64Kx128bit (SP + EP). Controller produces addresses and coordinates Writes

Results

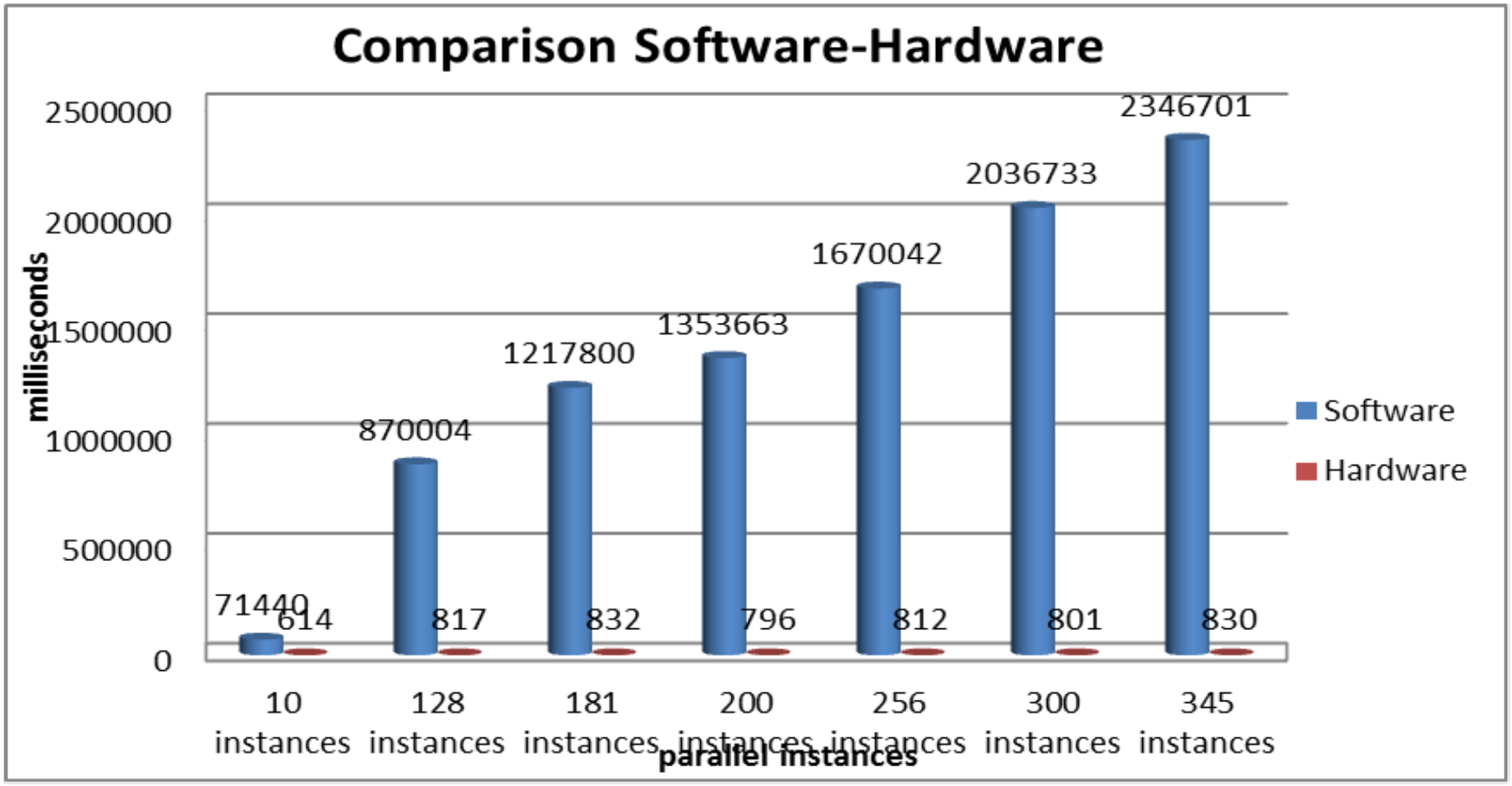
- Used a Virtex5 LX330T

# Engines	Frequency	Slice Registers (207360)	Slice LUTs (207360)	Block RAM (324)
1	178	488 (0%)	507 (0%)	2 (0%)
250	150	122802 (59%)	154103 (74%)	228 (70%)
345	146	165386 (79%)	206911 (**99%**)	228 (70%)

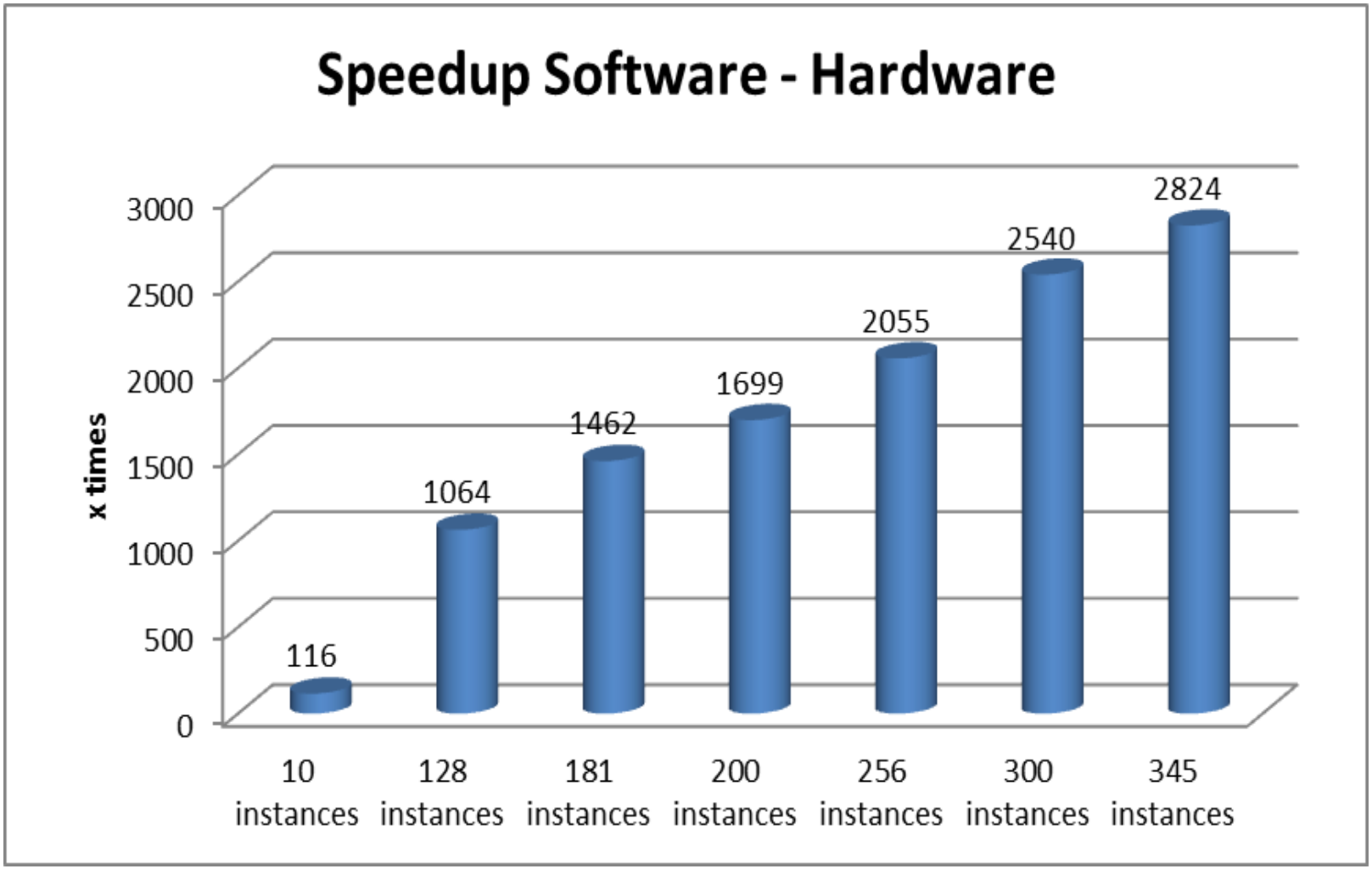
- Max throughput (345 engines): 415 chains/sec
- Power consumption 4.2 Watts
- ***S/W*** measured with Vtune on a P4@3GHz: 0.14 chains/sec

  NOTE: each chain takes different amount of processing

Execution Times (FPGA/SW)



Speedup FPGA/SW



Comparison w/ GPUs

- Published GPU results:
  - 162 chains/sec for GTX260
  - ~500 chains/sec for GTX280
  - Result quality is lower for GPU (cannot adjust)
- Performance:
  - 2.5x over GTX260, slightly slower than GTX280 (-17%)
- Power consumption:
  - GPU ~250W, FPGA ~4.2W (~60 times better)
- Estimate adjusting result quality:
  - 150x energy efficiency for roughly similar performance

Improvements/Extensions

- Current design is not very pipelined => we can further improve the operating frequency
- Use larger FPGA (V6, Kintex7, V7)
- Use an array of devices for more throughput
  - Optimize the cost/performance and/or power efficiency
  - Have to coordinate the devices
  - Have to measure the impact on result quality

Conclusions

- We exploit parallelism to create Rainbow table entries for A5/1
- The proposed system is quite simple and scalable to larger FPGA devices (or to multiple ones)
- We are almost 3,000 times faster than single threaded S/W running on a P4@3GHz
- Speed is roughly similar to a GTS280 GPU but at 60 times lower power and giving better quality results
- FPGAs prove both quite fast and extremely power efficient for this type of application