FPGA based hardware accelerator for Dash mining

Dries Truyens

Thesis submitted for the degree of
Master of Science in
Electrical Engineering, option
Electronics and Integrated Circuits

Thesis supervisor:
Prof. dr. ir. I. Verbauwhede

Assessors:
Prof. dr. ir. B. Preneel
Prof. dr. ir. M. Verhelst

Mentors:
Ir. P. Maene
Ir. B. Yang
Dr. ir. F. Vercauteren

Academic year 2015 – 2016
Preface

I would like to thank everybody who kept me busy the last year, especially my promotor and my assistants. I would also like to thank the jury for reading the text. My sincere gratitude also goes to my girlfriend and the rest of my family who supported me throughout the year.

Dries Truyens
# Contents

- Preface i
- Abstract iv
- List of Figures and Tables v
- List of Abbreviations vii

1 Introduction 1
  1.1 Overview 1

2 Bitcoin 3
  2.1 Network 3
  2.2 Transactions 5
  2.3 Blockchain 7
  2.4 Mining 9
  2.5 Future 12
  2.6 Conclusion 12

3 Dash 13
  3.1 Short history 13
  3.2 Masternodes 14
  3.3 Mining 15
  3.4 Attacks 16
  3.5 Conclusion 17

4 X11 profiling 19
  4.1 Miner selection 19
  4.2 Program selection 19
  4.3 Profiling results 19
  4.4 Comparison with references 21
  4.5 Conclusion 22

5 Design 23
  5.1 Board selection 23
  5.2 Overview 23
  5.3 Alternative design 25
  5.4 Conclusion 26

6 Algorithm implementation 27
## Contents

6.1 Skein ................................................................. 27
6.2 JH ........................................................................ 31
6.3 Keccak ..................................................................... 35
6.4 Hardware synthesis ................................................ 41
6.5 Conclusion ............................................................. 41

7 Nederlandstalige samenvatting 43
    7.1 Introductie ........................................................... 43
    7.2 Bitcoin ................................................................. 44
    7.3 Dash .................................................................. 45
    7.4 Profiling ............................................................... 45
    7.5 Design ................................................................ 45
    7.6 Implementatie van de algoritmes ......................... 46
    7.7 Conclusie ............................................................. 47

Bibliography ................................................................. 49
Abstract

Cryptocurrencies are gaining more popularity all over the world, with Bitcoin as the biggest player. This puts the Bitcoin network under a lot of pressure, and with the increase of nodes in the network, the limitations of Bitcoin surface. A new cryptocurrency will arise in the future, one that solves many issues that haunt Bitcoin. This could be Dash. Dash solves the main issue of Bitcoin, being the slow acceptance of transactions. In Bitcoin, it can take up to an hour for a transaction to be fully accepted. Dash solves this by adding a new node to the network, a masternode. These masternodes allow for almost instant transaction validation, and add a layer of anonymity to the network. This results in a fast, anonymous, and future proof cryptocurrency. Just like Bitcoin, Dash also needs to be mined. Dash uses X11 as their proof-of-work function. The mining evolves over time, eventually reaching FPGA and ASIC miners. This work describes the transition into the FPGA domain of Dash mining, by implementing some of the algorithms of X11 in hardware. For the selection of the algorithms, the original CPU-miner code is profiled. Profiling gives an indication of the distribution of the CPU time of each algorithm. Three different profiling methods are conducted, and the slower algorithms are selected to be implemented in hardware. The selected algorithms are Skein, JH and Keccak. The implementation of these is done in VHDL, a popular hardware descriptive language. The overall design decisions are explained next. First, a memory mapped design is suggested, but due to problems with the compilation for ARM the design is discarded, and an alternative design is proposed. How each of the algorithms work, and how they were implemented, is explained in the final chapter. This chapter also includes synthesis results on the timing, logic and area usage of the algorithms individually, and when integrated in the full system. At the end of the chapter the results are summarised and discussed.
List of Figures and Tables

List of Figures

2.1 Centralised versus decentralised (Bitcoin network) ...................... 4
2.2 Two linked transactions with transaction fees ............................. 5
2.3 Usage of private and public keys for signing transactions ............... 6
2.4 Pseudo-code for the mining process ...................................... 10
2.5 The evolution of the hashrate and difficulty over nine months of 2016 .. 11
2.6 Prediction of the Bitcoin reward ......................................... 12

3.1 Dash price chart ............................................................... 13
3.2 Number of Dash masternodes since November 2014 ........................ 14

4.1 Profiling of original miner code ......................................... 20
4.2 Profiling of the stripped miner code ..................................... 21
4.3 Profiling done in project eBASH ......................................... 22

5.1 Overview of the use of the ZedBoard .................................... 24

6.1 The MIX function of Threefish ........................................... 28
6.2 Four rounds of Threefish .................................................. 28
6.3 Unique block iteration on a 166-byte message ........................... 29
6.4 Datapath of the round function of Skein ................................. 30
6.5 Two of the 42 rounds used in JH .......................................... 32
6.6 S-boxes used in JH ........................................................... 32
6.7 Linear transformation in bit-wise computation .......................... 33
6.8 Permutation in JH .............................................................. 33
6.9 Full implementation of JH .................................................. 34
6.10 Round function of JH ....................................................... 34
6.11 Theta function of Keccak .................................................. 36
6.12 Rho function of Keccak ................................................... 36
6.13 Pi function of Keccak ...................................................... 37
6.14 Chi function of Keccak .................................................... 38
6.15 The new Chi function with lane complementing transform ............. 39
6.16 Full implementation of Keccak ........................................... 40
List of Figures and Tables

List of Tables

2.1 Contents of a transaction. ................................. 6
2.2 Contents of a block. ....................................... 8
2.3 Contents of the blockheader. ............................. 8

3.1 Comparison between GPU and ASIC mining for X11. .............. 16

5.1 Use of the communication shared register ....................... 25

6.1 Results after synthesis .................................... 41
6.2 Comparison of hardware and software ......................... 42
List of Abbreviations

Abbreviations

P2P  Peer-to-peer
SPV  Simple payment verification
UTXO Unspent transaction output
P2PKH Pay-to-public-key-hash
P2SH Pay-to-script-hash
POW  Proof of work
CPU  Central processing unit
GPU  Graphics processing unit
FPGA Field-programmable gate array
HDL  Hardware descriptive language
ASIC Application specific integrated circuit
SHA  Secure hash algorithm
DoS  Denial of Service
ARM  Acorn RISC machine
RISC Reduced instruction set computer
SSE2 Streaming SIMD extensions 2
SIMD Single instruction, multiple data
NIST National institute of standards and technology
eBASH ECRYPT benchmarking of cryptographic systems
ARX Addition, rotation and XOR
XOR  Exclusive-OR
PS   Processing system
PL   Programmable logic
LUT  Look-up table
DSP  Digital signal processor
RAM  Random-access memory
UBI  Unique block iteration
IV   Initial value
DMA  Direct memory access
Chapter 1

Introduction

In an era in which the current banking systems are under pressure, the cryptocurrencies are catching up. And in a distant future, might even be able to overtake them. The rise in popularity of Bitcoin is a clear example of this trend. But as more and more people join the Bitcoin network, its limitations become visible. A new cryptocurrency might be required in the near future. This could be Dash. Dash solves many problems from which Bitcoin suffers. It is a relatively new cryptocurrency, introduced in the beginning of 2014. Dash has increased immensely in value over the last year and is steadily becoming a respected cryptocurrency. As every cryptocurrency, Dash also needs to be mined. This generates the Dash-coins themselves and lets the network process incoming transactions.

The act of mining is similar to opening a digital lock by using brute-force. This can be done in a few ways, by using: CPU, GPU, FPGA or ASIC. The main objective is to get the mining speed as high as possible. For this, the miner code evolves over time, becoming more and more efficient.

1.1 Overview

This work covers the hardware implementation of a Dash CPU-miner. The main goal of this work is to increase the hash rate of the default CPU-miner using hardware acceleration. The report starts by explaining how Bitcoin works. It is very similar to Dash in many ways, but can be considered to be more basic. This makes for a good starting point to explain the working principles of Dash. In the following chapter, Dash is compared to Bitcoin. Chapter four discusses the profiling of the miner code and the selection of the algorithms that will be implemented in hardware. The fifth chapter gives an overview of the architecture and selection of the development board. The final chapter deals with the algorithms themselves. How every algorithm works is discussed briefly. The implementation choices are explained, and the hardware is compared to its software counterpart. The results are summarised and discussed at the end of the chapter. The last chapter includes a short Dutch summary of the paper.
Chapter 2

Bitcoin

Bitcoin is currency just like the Euro, but is completely digital. There are no physical notes or coins, just a software wallet that runs on your PC or phone that contains your Bitcoins. Bitcoin has not intrinsic value, its value only depends on how much the community thinks it’s worth. It all started in 2009, when Satoshi Nakamoto launched Bitcoin and made the very first donation of 50 BTC (BTC is the unit of Bitcoin) to the first person that was able to claim it. Bitcoin is now steadily becoming a more trusted currency. Recently the first Bitcoin terminals arrived, where one can exchange every day money for Bitcoins.

2.1 Network

This section covers the organisation of the network and its users. Since Bitcoin was meant to be a decentralised currency, it is unable to have a regulating entity that controls and monitors the transactions and everything else that happens in the network. The chosen alternative is to work with a peer-to-peer network in which there are no central nodes, but every device that is connected (i.e., a node) has the same rights and can fulfil the same duties.[2]

2.1.1 Peer-to-peer

A peer-to-peer network (or P2P) is a network of devices in which all of the nodes have equal rights and most are interconnected. There is no central node. This type of network became popular with the public thanks to file-sharing services like Napster, which was released in 1999. The main advantage of decentralised networks is the lack of a governing entity that holds the entire network together. When this central node were to fail, the entire network would collapse and all of its functionalities would be lost. For example, one can think of a malfunctioning bank not being able to provide services to its customers. However there are also some disadvantages to this type of network. Since a node can not be sure of the proper functioning of another node, it cannot trust other nodes. The network must find a way to reach consensus between
all nodes. The timing within the network is another issue, i.e., how to determine which of the multiple conflicting messages was broadcast first on the network.

2.1.2 Possible functions of peers (nodes)

Every node has certain rights and tasks to perform within the network. The main tasks within the Bitcoin network are listed below.

- **Routing**
  This allows for the node to be connected to the network, validating the blocks and their propagation.

- **Mining**
  The act of solving a mathematical problem that allows for transactions to be added to the blockchain. Some nodes are specialized at doing this specific task since being able to solve problem gives a significant reward.

- **Wallets**
  This keeps track of the private (and public) keys, and all of the addresses used by the user. Most applications on smartphones only implement this feature and 'routing'.

- **Blockchain**
  To be able to verify the validity of a transaction, the node needs to have knowledge of the complete history of all transactions within the network. This is done by saving the complete blockchain (a list of all previous transactions) on every node. This file is pretty hefty at 20GB, so most mobile nodes are unable to save this amount of data. They use a different type of blockchain and are called SPV (Simplified payment verification) nodes. They do not check the validity of a transaction themselves, but let the rest of the network decide.
2.2 Transactions

Transactions are used to trade Bitcoins across the network. They are small messages containing references to previous unspent transactions, to prove the ownership of the required Bitcoins. These are called UTXOs or Unspent Transaction Outputs. Once a transaction is accepted, it can be selected by miners to be in the next block of the blockchain. To persuade the miners, there is often offered a small reward for the miner. This is called a fee and is often based on the size of the transaction in kilobytes. A normal fee is about 0.00001 BTC/kB. The fee is calculated from the difference between the inputs and the outputs, any change that might be wanted can be done by making an extra output directed to oneself. Because of the lack of a central ledger containing everyone’s funds, every transaction input has to be traced back to the beginning of the blockchain to check its validity. Figure 2.2 shows two transactions, one spending the UTXO of the first one.

2.2.1 Contents

A transaction consists of numerous elements, which are listed in table 2.1. The timestamp is interpreted differently according to its value. When its value is 0, the transaction is immediately put in the next block (when chosen). If it’s between 0 and 500,000,000, the transaction is only to be processed after the block, with number equal to the timestamp value, has been made. Lastly, if the value is greater than 500,000,000, the transaction is to be accepted only when more seconds have passed (since January 1st 1970) than the value of the timestamp.

2.2.2 Keys

A digital signature is used to authenticate the creator of the transaction. This way it can be proven that the one spending the funds is the same person that made the transaction. It is thus impossible to spend the funds of others. The working principle is one commonly used in cryptography. The transaction message and the private key of the user are processed to produce a digital signature. This will then be sent over the network together with the transaction itself. Every peer can now verify that the
2. **Bitcoin**

<table>
<thead>
<tr>
<th>Amount of data (bytes)</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>4</td>
</tr>
<tr>
<td>Input counter</td>
<td>1 to 9</td>
</tr>
<tr>
<td>Inputs</td>
<td>unknown</td>
</tr>
<tr>
<td>Output counter</td>
<td>1 to 9</td>
</tr>
<tr>
<td>Outputs</td>
<td>unknown</td>
</tr>
<tr>
<td>Locktime</td>
<td>4</td>
</tr>
</tbody>
</table>

**Table 2.1:** Contents of a transaction.

![Diagram of private and public keys for signing transactions](image)

**Figure 2.3:** Usage of private and public keys for signing transactions

correct person issued the transaction using the public key of the owner, the digital signature, and the original transaction. A graphical representation of this can be found in Figure 2.3. The private key must be kept secret, because otherwise anyone would be able to sign a transaction using your identity.

### 2.2.3 Types of transactions

There are multiple ways to use transactions, but the standard way is to pay to the address of a node. But there are many more possibilities.

- **Pay-to-public-key-Hash (P2PKH)**
  This is the standard use of transactions. Here, the address is used to pinpoint the receiving peer. The name originates from the way the addresses are made. They are the output of a hash of the corresponding public key.

- **Pay-to-public-key**
  Directly paying to the public key itself is also an option.
2.3. Blockchain

- **Multisignature**
  Mostly used when decisions need to be taken by multiple people. E.g., when an employee has to make a transaction that needs to be verified by his/her superiors. This transaction needs \( N \) signatures out of a list \( M (>N) \) signatures to be released. This type of transaction produces larger transactions due to the multiple signatures. They will thus require a higher fee to be processed quickly.

- **Data-output**
  Using the address field to write data that can be seen by everyone and has a timestamp on it. This way one can store data in the blockchain, with no chance of it getting altered or removed afterwards. These transactions always have a value of 0 BTC, because it would be very difficult to retrieve the funds.

- **Pay-to-Script-Hash (P2SH)**
  This type is based on the multisignature model but solves the issue of the increased data usage. It relies on a compression of the transaction data. The compressed value uses much less data than the original script and has the same function. This also shifts the burden of paying the big fees to the receiver of the payment (fees are based on the amount of data in the transaction).

2.2.4 Pools

The maximal size of a block (containing transactions) is limited, so almost no issued transactions will be processed immediately. This requires a buffer for the transactions to be stored in, waiting to be added to a block. These buffers are called pools. Before a transaction is to be added to the transaction pool, it needs to be verified first.

There is also an orphan pool. This pool contains transactions that use inputs that are not yet available as UTXOs. When a parent transaction (producing the required UTXOs) is added, the corresponding orphan is also added to the main transaction pool.

2.2.5 Transaction chaining

For privacy reasons, multiple payments are sometimes made to different addresses one owns, so it becomes more difficult to trace the funds. The necessary payment happens last and is often called the child transaction. This child transaction can be issued even before the other payments were done. This causes the transaction to be put in the orphan pool awaiting the other payments to be fulfilled first.

2.3 Blockchain

The blockchain is a list of blocks that dictates the order in which transactions were executed. All transactions in the same block are executed at the same moment. It functions as the backbone of the verification method used in Bitcoin. Blocks are added to the blockchain by the miners. They solve a mathematical problem similar
to breaking a numerical padlock. This is further explained in the section about mining.

### 2.3.1 Contents

Every block in the blockchain consists out of the same elements, which are listed in Table 2.2. The block header is a very important part since it contains the information required for checking the validity of the block. Its composition can be found in Table 2.3. The Merkle tree is a binary tree containing all transactions of that block. It is used to summarise the big chunk of data that is the list of transactions. It also contains a counter (4 bytes) that is incremented once the nonce has reached its maximal value. Blocks are identified by two main characteristics: the value of their blockheader and their position in the blockchain.

<table>
<thead>
<tr>
<th>Amount of data (bytes)</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block size</td>
<td>4</td>
</tr>
<tr>
<td>Block header</td>
<td>80</td>
</tr>
<tr>
<td>Transaction counter</td>
<td>1 to 9</td>
</tr>
<tr>
<td>Transactions</td>
<td>unknown</td>
</tr>
</tbody>
</table>

**Table 2.2: Contents of a block.**

<table>
<thead>
<tr>
<th>Amount of data (bytes)</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>4</td>
</tr>
<tr>
<td>HashPrevBlock</td>
<td>32</td>
</tr>
<tr>
<td>HashMerkleRoot</td>
<td>32</td>
</tr>
<tr>
<td>Time</td>
<td>4</td>
</tr>
<tr>
<td>Bits</td>
<td>4</td>
</tr>
<tr>
<td>Nonce</td>
<td>4</td>
</tr>
</tbody>
</table>

**Table 2.3: Contents of the blockheader.**

### 2.3.2 Possible issues

Trusting miners to add blocks to the blockchain might pose some issues. The miners themselves might benefit from cheating or unlikely events might happen every now and then. The Bitcoin network has some intelligent ways to prevent this.

- **Simultaneous mined blocks**
  
  When two or more miners find a block at the same time, the network is split up in multiple versions of the blockchain depending on which block arrived first at the node. This phenomenon is called a fork and is often solved by simply
waiting for the next block to arrive. The winning miner then decides which of the branches should be followed. In the Bitcoin network, the longest branch is the correct one, the other is revoked and its transactions are put back in the transaction pool.

• Pre-computing
Miners might benefit from being able to pre-compute blocks. This way, they can easily dispose of another branch in the blockchain. This changes the course of transactions, which may cause a transaction of the miner to become invalid. This way the miner would be able to cheat and revoke his own transaction after it got accepted. Bitcoin has a solution for this. One of the things that are needed in the computation for the next is the hash of the previous blockheader. This is unknown at the time of computation of the first block since the nonce increments constantly.

2.4 Mining

Miners are the ones adding new transactions to the blocks, which in turn get added to the blockchain. Mining is what secures the network against fraudulent transactions. A new block is mined every 10 minutes on average.

The process of mining is done by applying a mathematical function to the blockheader of the new block. This process is repeated until the output of the function is below a certain value, represented by the difficulty. Every time the hash is calculated and the value is not low enough, the nonce inside the blockheader is incremented. The process of mining is displayed in the pseudo-code in Figure 2.4.

When eventually someone finds a suited nonce for his/her block, they broadcast the block over the network. Every node checks its validity by calculating the hash with the given nonce themselves. This is called the proof-of-work or POW.

2.4.1 History of mining

As every other cryptocurrency, Bitcoin started off with only CPU (Central Processing Unit) miners. The code for these is the easiest to write, and since they are present in all computers, this is a logical first step.

The first GPU-based (Graphics Processing Unit) miners appeared later on. A GPU is made for processing video and is normally used for video editing or gaming. A CPU and GPU can be best compared as the "thinker" and the "worker" respectively. The CPU is made for complex tasks and decision-making based on the running software. This makes it very suited for changing tasks at any point in time. This in contrast to the GPU, which is made for repeating the same task over and over again, and being very efficient at it. This makes it much faster at mining (hashing) than a CPU, leading it to be far more suitable at mining Bitcoins. GPUs will be able to work about twenty times faster than CPU’s.[30]

In a next step the development on FPGA (Field Programmable Gate Array) began.
These are integrated circuits to be programmed by the customer. This allows for the user to make custom-made hardware. Since the hardware on the CPU and GPU is general-purpose, it is not optimised for mining. However it is possible to achieve this with an FPGA. The hardware is to be written in a special hardware descriptive language (HDL). The standard in the industry are VHDL and Verilog. Many FPGAs are available for purchase, such as Icarus.[22] FPGAs will give approximately a factor twenty increase over GPU-based miners.[29]

The last step in the mining history is marked by the advent of the ASIC (Application-Specific Integrated Circuit) miners. ASICs are circuits specifically made for one purpose only. They are not reconfigurable like the FPGAs and are not able to perform any other task than mining. They are very expensive to develop, in contrast to FPGAs, but are much cheaper to mass-produce since they don’t allow any flexibility. They are the fastest alternative when it comes to Bitcoin mining, being up to a factor 1000 quicker than their FPGA counterparts.[28]

2.4.2 Hashing

The definition of hash functions as described by "Handbook of Applied Cryptography"[26]: Hash functions take a message as input and produce an output referred to as a hash-code, hash-result, hash-value, or simply hash. More precisely, a hash function h maps bit-strings of arbitrary finite length to strings of fixed length, say n bits. The function is many-to-one, implying that the existence of collisions (pairs of inputs with identical output) is unavoidable. The basic idea of cryptographic hash functions
is that a hash-value serves as a compact representative image (sometimes called an imprint, digital fingerprint, or message digest) of an input string, and can be used as if it were uniquely identifiable with that string. In Bitcoin the input of this hashing function is changed after every calculation by incrementing the nonce. This gives a completely new hash in the next iteration, completely independent of the previous hash. This way a correct nonce can not be predicted. Bitcoin uses SHA-256 as its hash function. It transforms every blockheader into a 256 bit string. The hashrate is a defining characteristic of the Bitcoin network. It varies through time and is directly linked to the difficulty of mining. This link exists because the Bitcoin network always strives to mine the next block in an average time of ten minutes. Adjusting the difficulty makes this possible. The current hashrate is about 1.6 exahash/s. Figure 2.5 shows the change in hashrate and difficulty over the course of nine months of 2016. [3]

2.4.3 Reward

There are two major incentives for the miners to hash new blocks: the block reward and the transaction fees. The first one is a big reward in the form of a transaction towards the owner of the winning miner node. The first blocks gave a 50 BTC reward. Nowadays this reward is 12.5 BTC and it will continue decreasing over time. Every 210,000 blocks, the reward halves. The 420,000th block was mined on the 9th of July, further lowering the reward to 12.5 BTC.[13] A halving of the reward is expected to happen every four years. It is estimated that almost all available Bitcoins will be mined by 2110-2140. Figure 2.6 shows an estimation of the reward over time, as predicted in 2009.[24]

Another source of reward for the miners are the transaction fees. These are added by the creators of the transactions themselves to make sure miners will choose their
2. **Bitcoin**

![Figure 2.6: Prediction of the Bitcoin reward](image)

*Figure 2.6: Prediction of the Bitcoin reward*

A transaction over others. This way, their transaction gets processed faster.

### 2.5 Future

As Bitcoin grows, more peers enter the network and more transactions are issued. Only one block can be processed every 10 minutes, containing a limited amount of transactions. This limits the acceptance rate of transactions. A possible solution for this would be to increase allowed blocksize. As of now, the maximum block size is limited to 1MB to prevent DoS attacks. This requires a new version of the software to arise, BitcoinXT. A permanent fork in the blockchain will appear, since the peers that have not updated to BitcoinXT will not accept the new, bigger blocks. A new cryptocurrency will be produced living next to Bitcoin, sharing the first part of the blockchain.

### 2.6 Conclusion

Bitcoin is the first cryptocurrency to be accepted by the general public and as time passes, it will become an even more trusted currency. Before Bitcoin can be useful as an alternative to present day banking, the rate at which transactions are accepted needs to be increased dramatically. This will require some changes to the Bitcoin network, or maybe even a completely new cryptocurrency.
Chapter 3

Dash

Dash is cryptocurrency very similar to Bitcoin, but with some major differences. It is also decentralised, open-source, and uses the same working principles as Bitcoin. However, in some areas it has some characteristic changes. It introduced masternodes, InstantX (a method for faster acceptance of transactions), Darksend (anonymity), its new POW function X11, and fungibility of coins are some examples. These differences are covered in this chapter. The last section gives a short summary of Dash, and compares Dash to Bitcoin.

3.1 Short history

Darkcoin was introduced by Evan Duffield in January 2014. It is a cryptocurrency much like Bitcoin. In March 2015 it got rebranded to Dash, which is a portmanteau word of ‘digital cash’. Dash has one of the most active communities of all cryptocurrencies and many volunteers that keep improving it further.[31] Dash has also been increasing in trading volume and in the last year its value has risen from about $2.5 (September) to $8 (July).[32] The price chart for last year can be found in Figure 3.1.

Figure 3.1: Dash price chart
3. Dash

3.2 Masternodes

One of the main differences with Bitcoin is that Dash uses masternodes. These are ordinary nodes that have more tasks and benefits than normal nodes. Masternodes are the main reason that Dash is faster and more anonymous than Bitcoin. In principle, everyone can turn his node into a masternode. The only requirements are a static IP-address and 1000 DASH (DASH is the unit of Dash) on that node. These requirements are necessary to overcome some possible attacks on the network. The main goals of masternodes is to provide instant transaction validation and anonymity. Controlling a masternode also has benefits to it. Every time a new block is mined, a masternode is selected and receives a part of the mined coins.[7] As of today there are about 4000 masternodes in the Dash network and the amount keeps rising steadily. In Figure 3.2 the number of masternodes since November 2014 can be found.[25]

3.2.1 Transactionlocking/InstantX

The first main goal of the masternodes is improved locking of transactions. Having slow confirmation is one of the main issues of Bitcoin and is specific to the currency itself. Bitcoin only uses the mining and the blockchain to know whether a transaction is valid or not. This causes unacceptable delays because one block takes approximately ten minutes to mine. And even after ten minutes, the transaction can still get revoked due to a longer arm of a possible fork. It’s common for bigger transactions to wait at least six consecutive blocks (one hour) before being accepted. The mechanism that solves this for Dash is called InstantX, and is based on locking the funds of a transaction.[16] The workings of InstantX are explained with a use-case.

Case: peer wants to lock funds he/she wants to spend
3.3 Mining

1. Peer issues locking command
2. Locking command propagates over complete network
3. Some masternodes are selected to reach consensus
4. Lock is sent over the network, when consensus over its validity is reached
5. Lock is accepted by every node
6. Transaction is accepted by everyone as valid

In step three, some masternodes are selected to participate in the locking of the funds of the transaction. This is done by the consensus of ten masternodes. This means that the ten selected masternodes should all agree on the validity of the transaction. This does introduce a possibility for attackers to get invalid transactions to be accepted by the network, but this attack will be discussed further on.

3.2.2 Darksend

Darksend is the feature of Dash that makes it anonymous. This is done by mixing funds for multiple nodes. Transactions can be formed by multiple nodes and made out to multiple nodes to merge funds together. This way the funds cannot be uncoupled afterwards and the transaction becomes anonymous. For adding even more anonymity, inputs and outputs are split up in slices of 100, 10, 1 and 0.1 DASH, so they cannot be traced by the amount that used.\cite{6} Darksend also makes the coin fungible. The mixing of the funds is done by a masternode. The user can choose how many rounds of mixing he/she wants. When using more rounds, the funds pass through multiple masternodes and every masternode mixes the funds with those of others. This makes it almost impossible for a third party to track the funds of the user. If someone still wanted to see what funds someone is spending, they can always control the selected masternode and ‘snoop’ the user’s funds as they pass through. However, since there are a lot of masternodes active, the big investment it brings for controlling one and the fact that most of the time more than one round is used for mixing makes it very unlikely and costly to do so. If there were only 1000 active masternodes and the attacker were to add 2000 controlled masternodes, it would cost 2.000.000 DASH (almost 16.000.000$) for a 19.75% chance of following a Darksend transaction with four mixing rounds.\cite{10}

3.3 Mining

The mining process is analogous to the one in Bitcoin, but Dash uses a different algorithm for its POW, X11. It has no affiliation with the X11 software for graphics. As the name suggests, this algorithm uses 11 hash functions. Namely: blake, bmw, grøstl, skein, JH, keccak, luffa, cubehash, shavite, simd, echo. All of these were participants in the SHA-3 competition that started in 2008.
An interesting feature of the X11 algorithm is that it’s supposed to be a lot more resistant to ASICs. This means that it is much harder for someone to implement X11 completely on an ASIC compared to the one-hash POW of Bitcoin. This is a very important feature because when ASICs arrive they outperform CPU, GPU and FPGA mining by a few orders of magnitude. The danger is that ASICs may centralize the hashing power of the network, making it vulnerable to 51% collusion attacks. These attacks allow the user, having 51% of the network’s power, to change the history of the longest chain in the blockchain by computationally orphaning blocks that were legitimately mined by other miners. This is done by introducing a longer chain of the fork than the current one.[8]

X11 is also more fair in the early stages of the evolution of mining. This is due to the fact that CPU and GPU mining give a similar return. People are not forced to invest in GPUs as fast as with Bitcoin. When switching to GPUs they would also benefit from the 30% less power usage of X11 in comparison to running Scrypt (a POW function for other cryptocurrencies).[8]

However, with the rise of the first ASICs for X11, most of the CPU and GPU miners are becoming obsolete. The first ASICs hit the market in the beginning of March 2016. The first one was the iBeLink DM384M X11 ASIC Miner[4]. Now we’ll chose the best performing GPU (as listed on the Dash website [1]), the AMD Radeon R9-290X [5], and compare it to the ASIC. The results can be found in Table 3.1.

<table>
<thead>
<tr>
<th></th>
<th>GPU</th>
<th>ASIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hashrate (MH/s)</td>
<td>2.6</td>
<td>384</td>
</tr>
<tr>
<td>Price ($)</td>
<td>(+/-) 500</td>
<td>2000</td>
</tr>
<tr>
<td>Power consumption (W)</td>
<td>90</td>
<td>715</td>
</tr>
</tbody>
</table>

Table 3.1: Comparison between GPU and ASIC mining for X11.

This shows that one would need about 150 GPU’s to get the same hashrate as an ASIC miner. The bigger cost of the ASIC has almost no effect on the comparison, since the GPU’s would still cost 30 times as much for the same hash rate. The ASIC miner is clearly superior to the GPU.

### 3.4 Attacks

With the introduction of the masternodes, some new vulnerabilities arise. These will be discussed in this section. Some vulnerabilities which could still pose problems in other currencies will also be included.[10]

#### 3.4.1 Sybil attack

This is an attack towards the masternodes in which a user tries to get control of all of the selected masternodes. This would give him the opportunity to validate an
incorrect transaction by putting a lock on an amount of DASH that have already been spent. However, the Dash network mitigates this by the large amount of masternodes in use.

For example:

Let's say that there are only 1000 masternodes available in the network. (In reality, there are almost 4000 active today.) An attacker would have to add 2000 extra masternodes before he even has a 1.72% chance of control over a single transaction lock. Considering that a masternode should have 1000 DASH in its wallet at all times, this would cost the attacker 2,000,000 DASH. Since the current price of one DASH is $7.90 [14/07/2016] and that there are only 6.6 million mined, makes it very unlikely someone would attempt such an attack and get make a profit.

3.4.2 Finney attack

In this type of attack, the malicious user mines a block containing a transaction in which he sends coins back to himself, but does not broadcast the block on the network. First, he spends his coins with a merchant. Immediately after the merchant has fulfilled his part of the bargain the attacker broadcasts his block, rendering the transaction to the merchant invalid.

To solve this, the network needs to be able to recognize a transaction that violates the existing transaction locks. Since the transaction inside the newly mined block was never locked, the new transaction to the merchant will be selected. The new block will then be rejected by the network.

3.4.3 Locking race attack

An attacker submits two conflicting locks to the network, one sending funds to himself and one sending funds to a merchant. This causes the network to be split up between two valid transactions until the elected masternodes propagate the correct lock. During this time, the merchant is tricked, thinking his transaction will be selected. When the lock arrives, he knows for sure whether his transaction is selected or not. The problem with this attack is that the time between the validating and locking is relatively short in Dash, just a matter of seconds. If the merchant were to wait a moment, the lock will be set and the merchant can’t be fooled by the attacker.

3.5 Conclusion

Dash is similar to Bitcoin, but it has some unique improvements to it. The usage of masternodes allows Dash to solve one of the main issues with Bitcoin, the acceptance speed of transactions. It also enables anonymity. The new POW algorithm, X11, also has some advantages over SHA-256 of Bitcoin. It has a smaller variation in CPU/GPU performance, and has an increased resistance to ASICs. However, two years after X11’s release, the first ASIC miners appeared. In short, Dash fixes many problems that are present in Bitcoin and will only grow in popularity in the years to come.
Chapter 4

X11 profiling

The goal of this report is to improve the overall hash rate of a Dash CPU miner by implementing the slower algorithms in hardware. To find these slower algorithms, the miner code needs to be profiled. In short, profiling shows how much processing time each part of your code is consuming.

4.1 Miner selection

Selecting a suited miner implementation is very important because it must also be possible to run it on the new platform. In our case, this new platform uses ARM cores. This limits our possibilities because the ARM processor is not able to use the SSE2 instructions, which its part of the x86 instruction set. Also the code should be made for CPU mining. The board has no graphics card available. This leaves very little options. The best fit is found in the `xcoin-cpuminer` by `ig0tik3d`.\[17\]

4.2 Program selection

Profiling is generally done by adding some compilation flags and running the compiled program afterwards, or by running the program within the profiling tool. In this work, the first option is used. The profiler of choice is `Gprof`, which is designed for Unix applications.\[21\] It uses a mix of instrumentation and sampling techniques for its performance analysis. Instrumentation techniques insert code into the program during compilation by using the `-pg` option of gcc. This tracks when a function is called. Sampling techniques probe the target’s program counter at fixed intervals using operating system interrupts. The result of this method is not exact but is rather a statistical approximation. Time spent in kernel mode can not be traced by Gprof.

4.3 Profiling results

Two main options are available to the profiling of the CPU miner code. Either by profiling the original miner code when running as usual (online), or by stripping
4. **X11 profiling**

4.1.1 Original miner code

This is the most straightforward approach of the two options. It simply profiles the original code of the miner while working in online mode. This means that it receives its inputs from the Dash network and calculates the hashes as if it is running without the profiler. This strategy does have some disadvantages. Since its working online, the block headers still need padding, and printing in terminal is also still included. The large number of files used for the miner, its confusing structure and the use of multiple threads also make it a lot more difficult to profile. The results can be found in Figure 4.1.

At 44.4% and 19.3% respectively, Echo and Keccak are the most time consuming algorithms. For Keccak, this might be correct. Keccak is known for also being very efficient in hardware implementations, while being rather slow in software. This makes it an excellent pick for the first algorithm to be implemented in hardware. For Echo, the reason might be found in the fact that it is the last algorithm in the chain. It thus needs to check the validity of the hash, by comparing it to the required difficulty. It also has the task of outputting some information to the terminal from time to time. This leads us to believe that the time consumption of Echo might be a lot lower than indicated by the profiling.
4.4 Comparison with references

4.3.2 Stripped miner code

The profiling of the original miner code has some issues. This method tries to solve them by stripping out all of the algorithms, and placing them in a separate 'offline' file. It will be initiated by a 512-bit fixed input and will feed the output of the last algorithm back to the first. This will loop for a large number of cycles. The advantages of this technique are numerous. The file is a lot clearer to read, there is no networking or any issues concerning that matter. Printing can be easily separated from the rest of the code, and multithreading can be avoided. The results of this profiling can be found in Figure 4.2

In this profiling Echo no longer consumes almost 50% of the computation time. Its shares dropped significantly, as expected, because now the networking is removed and no more unnecessary printing is done. Now Simd, Cubehash, JH, and Keccak are the biggest time consumers.

4.4 Comparison with references

Performance analysis has also been done in the NIST SHA-3 competition.[18] This was done by ECRYPT’s VAMPIRE lab, which measured the performance of the hashing functions in the competition. The project is called eBASH.[20] The project lists the required amount of cycles/byte for every hash function on a wide variety of platforms. The processor that was used for the profiling in the profiling part above is not listed, so the closest alternative is used. The results for 64-byte messages (512 bit) for that particular processor can be found in Figure 4.3.

The biggest consumers in this profiling are Simd, Groestl and Echo. This conforms
4. X11 profiling

Figure 4.3: Profiling done in project eBASH

to the rest of the NIST paper in which they state that ARX-based algorithms, such as Blake and Skein perform very well in software, and that Groestl and JH are considerably slower than others.[18] These results confirm the suspicion of the large time consumption by Echo. Even though it still holds a significant share, it is much less than the result of first profiling. The networking will have influenced the result of the first profiling considerably.

4.5 Conclusion

For the selection of the algorithms, all the profiling results are taken into account. Accelerating the miner code also implies introducing as little overhead as possible. Every time a transition between hardware and software is made, time gets lost. This forces the chosen algorithms to be consecutive in the X11 chain. As a first choice Keccak is selected. Keccak is chosen because of its relatively large time consumption in all profilings. Another good reason is because internally Keccak uses bit-oriented operations and is also said to have very good performance in hardware by NIST. To further increase the acceleration, two more algorithms are chosen alongside Keccak. In profiling of the stripped miner code, JH appeared to be one of the biggest time consumers. In the eBASH profiling, JH was an average consumer. JH is thus selected to be the second algorithm. For the last algorithm Skein is chosen, because it scored pretty high in the stripped code profiling and is the algorithm before JH. The chosen algorithms are thus, in order: Skein, JH and Keccak.
Chapter 5

Design

Many things are to be done before a working miner can be built on FPGA. This chapter explains these, and some other design choices that had to be made in the process.

5.1 Board selection

For the selection of the board, two major issues need to be considered. First, the board should have a hardcore to minimize the overhead between the hardware and the software part. Second, there should be enough programmable logic on the FPGA to be able to implement all necessary components. The Zynq boards are suited for this application. They are SoCs (System on Chip) with a hardcore and programmable logic. For the selection of the board there are two main options: the Zybo[12] and the ZedBoard[11]. In many ways both boards are the same, both use the same processor and have the same amount of on-chip memory. The major difference between both is the amount of programmable logic. The ZedBoard really comes out on top in this field. This comes at a cost though. The Zybo costs $189 and the ZedBoard $495. The ZedBoard is selected for this work even though the cost of the ZedBoard is much higher, the increase in programmable logic is of more importance. The ZedBoard has its own 32-bit ARM cortex A9 dual core processor, and is able to run an embedded variant of Linux. This will be used to run our cross-compiled miner code later on.

5.2 Overview

Now the board is selected, the different parts of the work can be described. The ZedBoard is split up in two areas. One is called the ’processing system’ (PS), it takes care of the software part and contains the processor. The other is the ’programmable logic’ (PL) which contains all of the logic cells, LUT’s, flip-flops, DSP slices, etc. All of the configurable hardware is situated in the PL. Since the miner is partially situated in both areas, an interface is needed and some sort of communication protocol. The interface consists of 16 32-bit shared registers, for a total of 512 bit. This is the amount of bits that the intermediary results of the Dash miner use. For
5. Design

Figure 5.1: Overview of the use of the ZedBoard

the communication, a 2-bit shared register is sufficient.
The processor is running a version of Linux, and on top of that the miner code can be run. The only issue is that the miner code first needs to be adapted to the ARM processor. This process is called cross-compilation and will be explained later on.
The algorithms are implemented in VHDL and then programmed onto the PL, together with a buffer to take care of the 32-bit transition from the PS (32-bit processor).
The complete scheme can be found in Figure 5.1.

5.2.1 Cross-compilation

To adapt the original miner code to the ARM processor it needs to be cross-compiled. This means that the code is compiled on a architecture that is not the same as the machine the created executable will be run on afterwards. This process has two main variables: host and target. The host is the machine that is compiling the code. The target is the one that will run the executable after compilation.[19]
Cross-compilation brings a lot of issues with it. Libraries used by the program are often not included on the other machine and need to be cross-compiled themselves.
The configure scripts often only test for endianness or page-size of the host machine, which might be different for the target machine. Memory faults that do no appear on the host machine, may appear on the target machine.
An alternative is to run the compilation on the target machine itself, however on the ZedBoard this is not recommended since the memory is volatile.
5.2.2  Communication

The communication between the PL and the PS is done through a two bit shared register. This ensures that both parts know when the data is valid. Since the data register is shared between both parts, the data can be altered and read by both. This can easily lead to incorrect readouts without a proper communication. The usage of this register can be found in Table 5.1.

<table>
<thead>
<tr>
<th>Code</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>From PS to PL, data not valid</td>
</tr>
<tr>
<td>01</td>
<td>From PS to PL, data is valid</td>
</tr>
<tr>
<td>10</td>
<td>From PL to PS, data not valid</td>
</tr>
<tr>
<td>11</td>
<td>From PL to PS, data is valid</td>
</tr>
</tbody>
</table>

Table 5.1: Use of the communication shared register

5.2.3  Buffer

In between the data register and the algorithms there needs to be a buffer to transform the 32 bit inputs to 512 bit data blocks. The algorithm part is made to be easily expanded to more algorithms and therefore takes only 512 bit inputs. The new algorithm blocks can then be easily added to the chain. This buffer is also made in VHDL.

5.2.4  Algorithms

The three chosen algorithms are implemented in hardware, so each has an input and output of 512 bit. This allows for an easier expansion of the work in the future. The algorithms are written in VHDL. The correct functioning is proven by simulation in the end. All of this is done in the Xilinx Vivado Design Suite.

5.2.5  Problems

As mentioned above, many problems can arise while trying to cross-compile code. This work was no exception. Due to incompatibility of many libraries, the lack of a debugging tool on the board, persistent memory faults and so on, the initial design was discarded.

5.3  Alternative design

This design no longer uses a specific shared register to transport data between the PL and PS. For this, the DMA (Direct Memory Access) is now used. Making use the AXI-Stream interfaces of Xilinx, the 512-bit values can be transported over
32-bit lines. The AXI-Stream interface allows for high-speed streaming of data, exactly what is needed for our design. The AXI-Stream also contains control signals (data_valid, last_block, etc.), the small 2-bit register is no longer needed. The algorithms do need to be altered to be compatible with the new interface. Skein and Keccak need to participate correctly in the handshaking procedure, and need to react appropriately to the signal marking the last block of data. This design also uses the 512-bit outputs of the hash functions rather than the 32-bit Stream interface outputs. This is done, because every time a function has to output its value to a 32-bit wide bus, a small amount of overhead is added. The software part now runs from a connected PC, and not on the Linux OS of the ZedBoard. The cross-compilation of the code is also no longer needed, as the software will now run as a bare-metal application.

5.3.1 Overhead

The new design uses a DMA for its data transfer. Since the bus is only 32-bit wide, not all required data will be able to travel to the algorithms within a clock cycle. This causes overhead, an extra delay due to the communication of data. This delay happens twice for every calculation using the hardware algorithms. Once while writing the data to the algorithms input ports and once when reading the outputs. Because the output data is 512-bit, 16 cycles will be required for the data to be fully transferred once.

5.4 Conclusion

In this chapter, the major design decisions were made and the most suitable board for the project was selected. Now, with the use of Xilinx Vivado the algorithms can be implemented in VHDL.
Chapter 6

Algorithm implementation

This chapter is about the chosen algorithms Skein, JH and Keccak. The way each algorithm functions, how it is implemented in hardware and how it compares to its software counterpart are described further in this chapter.

6.1 Skein

Skein is a family of hash functions with three different internal state sizes: 256, 512 and 1024 bit. Each of these can support any output size and is designed for a specific purpose. The 256-bit version is made for low memory applications such as smart cards and uses only 100 bytes of RAM. Skein-1024 is the most conservative variant and still runs fast in dedicated hardware implementations.[27] The 512 bit version is in between both extremes. This is also the version that is used in the X11 POW of Dash with a 512-bit output.

6.1.1 Functionality

Skein is built up from these three components:

- **Threefish**
  This is a tweakable block cipher and is defined for 256-, 512- and 1024-bit block sizes. The key that is used is the same size as the blocks. The idea behind Threefish is that its more secure to have multiple simple rounds than a few complex rounds. Threefish uses three operations: XOR, 64-bit addition and constant rotation. These operations are used in a MIX-function. A drawing of this function is shown in Figure 6.1. The upper-left operator computes the sum of two 64-bit words. The center-right one rotates its inputs. In the bottom-left, the XOR operator is found. The number of parallel MIX functions is determined by the size of the blocks. Each MIX function handles two 64-bit words. For 512-bit blocks, four parallel MIX functions are used.

The version used in Dash, Skein512, uses 72 rounds, each consisting of four parallel MIX functions and afterwards one permutation of the eight 64-bit
6. Algorithm Implementation

words. The permutation adds to the diffusion and the rotation constants are repeated every eight rounds. Every four rounds, a subkey is injected into the eight words. A subkey is a combination of the key words and the tweak words. An illustration of four Threefish rounds can be found in Figure 6.2.

• Unique Block Iteration (UBI)
This allows Skein to have an arbitrary amount of input bits. It uses chained blocks of Threefish and an XOR to compute the final hash. It splits up the

Figure 6.1: The MIX function of Threefish

Figure 6.2: Four rounds of Threefish
message into blocks of (in this case) 64 bytes and feeds this with a tweak and the previous output to the Threefish block. When the last message block does not contain the maximal amount of bits the block is padded. The tweak takes into account the block number and whether its the first or last block. In the general case, the input 'G' is a configuration string. In the case of a hash function, the configuration string can be precomputed as an IV (initial value). In Figure 6.3 this is done for a 166-byte input.

![Figure 6.3: Unique block iteration on a 166-byte message](image)

- **Optional Argument System**
  Skein is a very flexible algorithm, which allows users to add optional inputs. These include: configuration settings, keys, personalised strings, nonces, etc. The chosen options are added to the beginning of the UBI chain and get their own specific tweak value. This allows for expansion of the options at any time.

### 6.1.2 Implementation

For the hardware implementation in this work, the 72 rounds are split up in nine blocks each containing eight rounds. The rotation distances can now be fixed in the MIX-function of Threefish, since the rotation distances are repeated every eight rounds. The permutations are also fixed within the eight-round-block, enabling a simplified and compact implementation. The addKey functions are also included in this eight-round-block. This leaves only one addKey function to be added in the last round. The tweak and the initial value, used by the addKey function can be precomputed in the case of X11, since the input will always have the same length. Also the IV is constant for every first computation. They are thus hard coded in the design. For the 512 bit implementation that Dash uses, the input block is split up in eight 64 bit blocks that can be processed independently. This allows to add parallelism. Four MIX functions can be computed in parallel, increasing its throughput. X11 requires two full computations of Skein for one 512-bit output. For the first computation, the message is used with an initial value for the previous hash (G in Figure 6.3). The second computation uses the output of the previous calculation as its previous hash. For its message, it uses a 512-bit string of zeroes. The tweak values are also changed to a new set of fixed values for the second computation. The
result of this computation is the required output of the JH algorithm. The datapath of the round function of Skein can be found in Figure 6.4. The critical path is also added.

![Datapath of the round function of Skein](image)

**Figure 6.4: Datapath of the round function of Skein**

There are however possibilities for an increased clocking speed. The current implementation makes the critical path contain three addKey functions, eight Threefish operations and one multiplexer. The critical path is the longest path signals need to travel between two registers, in one clock cycle. It is critical parameter that limits the maximal clock frequency. The critical path could be easily split up by adding a register after the fourth round. This would also allow for pipelining of two independent hashes. The second input could enter the first halve of the round-block while the first input is starting in the second halve. The pipelining could be expanded even further. In this design, all eight rounds could be split up, and be able to hash eight inputs simultaneously. This is however not useful for the X11 application. The value of the next intermediary hash won’t be known yet at that moment, so pipelining would not be helpful.
6.1.3 Comparison with software

For the comparison, the code for the Skein algorithm is stripped from the miner code and cross-compiled for ARM. The program will run the Skein algorithm 5,000,000 times with its previous output as its new input, starting off with a fixed input. The program is repeated five times and all results are close to one another. The average computation time was 98.71s for 5,000,000 Skein computations. This gives an average computation time of 19.0252 µs for one Skein computation in software.

For the hardware, the time required for one full computation is measured. A bare-metal software application inputs data into the DMA, which feeds this to the input of the JH algorithm. The timer starts when the first data leaves the DMA, and is stopped when the last 32-bit block of data arrives at the DMA. The time to reach a correct result is measured in cycles. It takes 1.76µs for one Skein computation to complete, with a clock of 30MHz. This includes writing to and reading from the DMA.

6.2 JH

JH is a hashing function with a constant input size and internal state size. It has four variants differing in output size: JH-244, JH-256, JH-384 and JH-512. Every variant has the same internal operations and uses the same amount of rounds.[33] Dash uses the JH-512 version for X11.

6.2.1 Functionality

The main function of JH can be split up in three parts. In a first step, the XOR is calculated of the 512-bit input block with the first 512 bits of the previous hash. In the case of the first block, the previous hash is replaced by an initial value. In step two, the compression is done. The 1024 bit internal state is split up in 256 4-bit blocks. The new internal state runs through 42 rounds, each consisting of an S-box operation, a linear transformation and a permutation. Two rounds for a reduced internal state can be found in Figure 6.5. Afterwards, the 1024-bit internal state is reassembled. To finish, the XOR is calculated of the last 512 bit of the internal state with the input.[9]
6. Algorithm implementation

- **S-box**
  JH uses two 4-bit to 4-bit S-boxes. An S-box maps an input to an output based on the input bits. The selection of the correct S-box is done by one of the round_constant bits. The values of the round_constant are updated every round. Each S-box requires four input bits and maps these on a 4-bit output value. The S-boxes can be found in Figure 6.6.

<table>
<thead>
<tr>
<th>$x$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0(x)$</td>
<td>9</td>
<td>0</td>
<td>4</td>
<td>11</td>
<td>13</td>
<td>12</td>
<td>3</td>
<td>15</td>
<td>1</td>
<td>10</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>8</td>
<td>14</td>
</tr>
<tr>
<td>$S_1(x)$</td>
<td>3</td>
<td>12</td>
<td>6</td>
<td>13</td>
<td>5</td>
<td>7</td>
<td>1</td>
<td>9</td>
<td>15</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>11</td>
<td>10</td>
<td>14</td>
<td>8</td>
</tr>
</tbody>
</table>

**Figure 6.6**: S-boxes used in JH

- **Linear transformation**
  The linear transformation implements a 'maximum distance separable' function. It uses a rather complicated computation using binary polynomials. For an easier understanding, it can also be done with bit-wise operations, which can be found in Figure 6.7. A and B are the two (4-bit) input blocks that are transformed into C and D. The number in superscript denotes the number of the bit within the block, zero referring to the most significant bit.
6.2. JH

\[
\begin{align*}
D^0 &= B^0 \oplus A^1; \\
D^1 &= B^1 \oplus A^2; \\
D^2 &= B^2 \oplus A^3 \oplus A^0; \\
D^3 &= B^3 \oplus A^0; \\
C^0 &= A^0 \oplus D^1; \\
C^1 &= A^1 \oplus D^2; \\
C^2 &= A^2 \oplus D^3 \oplus D^0; \\
C^3 &= A^3 \oplus D^0.
\end{align*}
\]

Figure 6.7: Linear transformation in bit-wise computation

- **Permutation**
  The last step of every round is a permutation consisting of three basic permutations itself. These are called $\phi_d$, $P'_d$ and $\pi_d$. They allow to describe the total permutation more clearly. The basic permutations themselves will not be discussed here, only the final result of the permutation can be found in Figure 6.8. To improve readability, the internal state is reduced in the Figure.

![Figure 6.8: Permutation in JH](image)

6.2.2 Implementation

The computation of JH only needs one constant in every round, so it is efficient to calculate one round in every cycle. This allows for a high clock frequency since now the critical path will only contain one multiplexer, one S-box, one linear operation and one permutation. This also requires less programmable logic, since the same components can be reused for every round. The correct S-box is selected by a round constant. In this implementation these constants are hard coded. For the linear operation, a polynomial multiplication is described in the specifications of JH.[33] In this implementation a bit-wise computation is used. This version only uses bit-wise XOR operations. The limited use of components in this implementation does not allow pipelining. This is no problem since pipelining could not be applied to the implementation of JH for X11, due to the unavailability of the next input. X11 needs two hash computations of JH to calculate the correct output. The first computation uses a fixed value for its previous hash and the message. The result is then used by the following computation with a fixed message. The result of this last computation is the actual output of the JH algorithm. Figure 6.9 shows the full JH implementation. A closer look into the round algorithm can be found in Figure 6.10.
6. Algorithm implementation

Figure 6.9: Full implementation of JH

Figure 6.10: Round function of JH
6.2.3 Comparison with software

To compare the hardware implementation with its software counterpart, the JH code is stripped from the original miner. The code is altered so it runs the JH code 5,000,000 times consecutively. Every time the output of the last iteration is fed back to the input, the first iteration gets its input from a 512-bit fixed value. The results of five runs of this code averaged at 366.41s. The average time for one JH computation is 73.282 µs on the ZedBoard.

For the hardware, the time required for one full computation is measured. A bare-metal software application inputs data into the DMA, which feeds this to the input of the JH algorithm. The timer starts when the first data leaves the DMA, and is stopped when the last 32-bit block of data arrives at the DMA. The time to reach a correct result is measured in cycles. It takes 3.96 µs for one JH computation to complete, with a clock of 30MHz. This includes writing to and reading from the DMA.

6.3 Keccak

Keccak is a family of sponge functions. These are special hash functions that have a finite internal state, take as input a bit stream of any length and output a bit stream of any desired length. Keccak has many variants, denoted by the 'b' in KECCAK-f[b]. There are seven permutations with \( b = 25, 50, 100, 200, 400, 800, 1600 \). The value of \( b \) represents the size of the internal state. The smaller the internal state, the more lightweight the implementation can be made. In Dash, KECCAK-f[1600] is used. This is also the version of Keccak that was selected in the SHA-3 competition by NIST.[23] Keccak also takes two parameters: \( r \) and \( c \). These divide the internal state in two parts. One part is called the 'rate', which produces the output in the end. The first \( r \) (or less) bits of the internal state will be the output at the end. The other part is called "capacity", and defines the security level. The capacity will never be made public. \( R \) and \( c \) added up has to be equal to \( b \). Dash uses for these parameters \( r = 576 \) and \( c = 1024 \).[14]

6.3.1 Functionality

The internal state of the KECCAK-f[1600] function can be seen as a huge 5x5x64 3D binary matrix, with a total of 1600 bits. The choice of the permutation also fixes the number of rounds \( n_r = 12 + 2l \), with \( l = \log_2(b/25) \). In the case of KECCAK-f[1600], the number of rounds is equal to 24. Every round invokes five functions: theta, rho, pi, chi, and iota. These consist of only AND, XOR, NOT operations and rotations.

- **Theta**

  This function calculates the parity of every column of the internal state and saves this in a 5x64 bit matrix (sum_sheet). The bitwise XOR of two of its entries and a bit of the internal state is computed and overwritten in the internal state matrix. With the correct indices, this becomes:
6. Algorithm implementation

\[
\text{internal\_state}(row)(col)(i) = \text{internal\_state}(row)(col)(i) \text{ XOR} \\
\text{sum\_sheet}(col-1)(i) \text{ XOR sum\_sheet}(col+1)(i-1) \\
\text{for col, row : [0 to 4]; for i : [0 to 63]}
\]

All results for i are calculated \text{mod}64. Row and col are calculated \text{mod}5. Figure 6.11 displays this for one bit in the internal state.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{theta.png}
\caption{Theta function of Keccak}
\end{figure}

- **Rho**
The rho function rotates every 64 bit word over a fixed distance. These distances are set in specifications of the Keccak algorithm. A representation of the rho function can be found in Figure 6.12

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{rho.png}
\caption{Rho function of Keccak}
\end{figure}

- **Pi**
This function permutes the elements of the internal state within every slice. For this it uses fixed patterns described in the specifications. The patterns
can be summarised in the formula below. The effects of the pi function are visualised in Figure 6.13.

\[
\text{internal\_state}(row)(col)(i) = \text{internal\_state}(col)((\text{col} + 3 \times \text{row}) \mod 5)(i)
\]
for \( \text{col, row : [0 to 4]} \); \( \text{for } i : [0 \text{ to } 63] \)

\[\text{internal\_state}(row)(col)(i) = \text{internal\_state}(row)(col)(i) \oplus \left[ \lnot \text{internal\_state}(row)(\text{col} + 1)(i) \land \text{internal\_state}(row)(\text{col} + 2)(i) \right]\]
for \( \text{col, row : [0 to 4]} \); \( \text{for } i : [0 \text{ to } 63] \)

**Chi**

The Chi function adds non-linearity to the round by combining row elements, using three bitwise operations: AND, NOT and XOR. The result overwrites the row in use. The required logic can be found in Figure 6.14.
6. Algorithm implementation

- **Iota**
  The last function has the job of breaking up any symmetry caused by the previous functions. It does this by XOR’ing the 64 bit word in the first row and first column of the internal state with a constant. This constant is fixed by the specifications and determined by the round number.

### 6.3.2 Implementation

For the implementation of Keccak512-f[1600], the reference implementation of Keccak is used as a starting point. This needs to be altered significantly to become compatible with X11. Because now the input is a 512-bit value, the rate of the reference implementation had to be altered, and a new 64-bit word "keczword" is added. This is a constant that is added to the rate. The lane complementing transform is used in this implementation.[15] This transformation allows to remove many NOT operators by using the law of De Morgan, which states the following:

\[
(NOT a) \ AND \ (NOT b) = NOT(a \ OR \ b)
\]

This law can be applied to the Chi function of Keccak. As can be seen in Figure 6.14, the mapping done by Chi, takes five AND, five XOR and five NOT operations for the calculation of five lanes. Using the lane complementing transform the number of NOT operations can be reduced to one by replacing some AND operations by OR operations, and using the complement of six 64-bit lanes at the input. At the end, the output needs to be transformed accordingly by using the complement of two 64-bit lanes. The lane complementing transform allows to remove 80% of the NOT operations. The new Chi function can be found in Figure 6.15. In this figure, a2 and a4 were inverted at the input. They do not need an extra NOT operator.
In the round calculations of Keccak only one unique constant is required every round. This makes it beneficial to calculate one round in every cycle. The number of necessary components is reduced, and it allows for a high clock frequency, since now the critical path only consists of one run through the round function and one multiplexer. All rotation distances and round constants are hard coded in the design. This design does not allow pipelining. This is not an issue, since pipelining can not be used for the application of X11 anyway.

The full implementation can be found in Figure 6.16. The critical path is also added to the figure.
6.3.3 Comparison with software

For the comparison with the software implementation, the Keccak function is taken out of the miner code and cross-compiled for use on the ZedBoard. The code iterates 5,000,000 times over the Keccak function. In the first iteration, the input is a fixed 512-bit vector. Afterwards, the output of the last iteration is fed back to the input. The code is run five times and all results are similar, on average it takes 307.13s for 5,000,000 Keccak computations to complete. This results in an average time of 61.426 µs for one Keccak computation.

The hardware is benchmarked in a similar way. The time required for one full computation is measured. A bare-metal software application inputs data into the DMA, which feeds this to the input of the Keccak algorithm. The timer starts when the first data leaves the DMA, and is stopped when the last 32-bit block of data arrives at the DMA. The time to reach a correct result is measured in cycles. It takes 1.93µs for one Keccak computation to complete, with a clock of 30MHz. This includes writing to and reading from the DMA.
6.4 Hardware synthesis

With the algorithms implemented in hardware, their designs can be synthesised. For this, the Vivado environment has a built-in tool. Synthesis shows how much logic and memory the functions will need, once written to the ZedBoard. The achieved values are listed in Table 6.1. The numbers in parentheses displays the percentage of the available logic that is used by that particular function. The complete design is also included, this contains all hash functions, DMA, interconnects and many other blocks required in the design.

<table>
<thead>
<tr>
<th></th>
<th>Skein</th>
<th>JH</th>
<th>Keccak</th>
<th>Complete design</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUTs</td>
<td>8586 (16.14%)</td>
<td>3951 (7.43%)</td>
<td>3980 (7.48%)</td>
<td>20718 (38.94%)</td>
</tr>
<tr>
<td>Registers</td>
<td>2133 (2.00%)</td>
<td>2076 (1.95%)</td>
<td>2129 (2.00%)</td>
<td>10737 (10.09%)</td>
</tr>
<tr>
<td>Area (slices)</td>
<td>12868</td>
<td>6979</td>
<td>6463</td>
<td>36220</td>
</tr>
</tbody>
</table>

**Table 6.1:** Results after synthesis

The results are in line with the implementations themselves. Skein was expected to use significantly more LUTs then the other functions, due to the fact that each round function consists of eight rounds. For Keccak and JH, the results are similar. Both only compute one round per cycle. JH and Keccak will require less logic, but a similar amount of registers compared to Skein. Due to the extra components in the complete design, the amount of logic, memory and area differ from the sum of respective elements of the algorithms.

Synthesis also allows to check the timing of the design. The results of the timing analysis will determine the maximal clock frequency of the design. In this design, the overall critical path is situated in the Skein function. This was to be expected. Every function uses rounds to calculate the output hash. A register is thus needed at the end of every round function. This breaks a possible critical path across multiple functions, and causes the critical path of the slowest function to become the overall critical path. In this case, the Skein function has the longest critical path. It can be found in Figure 6.4. The critical path has a 29.654ns delay. This allows for a maximal clock frequency of 33.7MHz.

6.5 Conclusion

Some great results have been achieved with the use of hardware acceleration. With the addition of the hardware, the time consumption of the algorithms is reduced to a fraction of its original value. The CPU-miner is now able to achieve a higher hash rate. The results of this chapter are summarised in Table 6.2. The improvement line shows how many times the hardware is faster than its software counterpart. In this category Skein performs the worst. This is due to the fact that Skein already has an efficient software implementation in comparison to the other functions. The results
for the complete design, with all three algorithms chained, are also included. The complete design has less communication overhead, since there only needs to be one write transaction to the input of the Skein function, and one read transaction from the output of the Keccak function.

This work could be extended further, implementing more algorithms on the FPGA would further increase the acceleration, whilst not adding any extra overhead. For this work, only a limited amount of algorithms was selected, to prove the increased performance by using a hardware accelerator.

<table>
<thead>
<tr>
<th></th>
<th>Skein</th>
<th>JH</th>
<th>Keccak</th>
<th>Complete design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software (µs)</td>
<td>19.025</td>
<td>73.820</td>
<td>61.426</td>
<td>166.04</td>
</tr>
<tr>
<td>Hardware (µs)</td>
<td>1.76</td>
<td>3.96</td>
<td>1.92</td>
<td>5.27</td>
</tr>
<tr>
<td>Improvement</td>
<td>10.8</td>
<td>18.6</td>
<td>32.0</td>
<td>31.5</td>
</tr>
</tbody>
</table>

Table 6.2: Comparison of hardware and software
Chapter 7

Nederlandstalige samenvatting

7.1 Introductie

In een tijdperk waarin de huidige banksystemen onder druk komen te staan, krijgen de crypto valuta meer en meer aandacht. En in een verre toekomst, zijn ze misschien zelfs in staat om de gekende valuta te vervangen. De stijging in de populariteit van Bitcoin is een duidelijk voorbeeld van deze trend. Maar naarmate meer en meer mensen toetreden tot het Bitcoin-netwerk, worden de beperkingen van Bitcoin zichtbaar. Een nieuwe crypto valuta zou in de nabije toekomst nodig zijn. Dit kan Dash zijn.


Het 'minen' is vergelijkbaar met het openen van een digitaal cijferslot door elke mogelijke combinatie uit te proberen. Dit kan op een aantal manieren, met: CPU, GPU, FPGA of ASIC. De mining code evolueert doorheen de tijd, om steeds sneller en sneller te worden.

Dit werk gaat over de hardware-implementatie van een Dash CPU-miner. Het belangrijkste doel van dit werk is om de hashsnelheid van de standaard CPU-miner, met behulp van hardware-acceleratie te verhogen. Het rapport begint met uit te leggen hoe Bitcoin werkt. Het is vergelijkbaar met Dash op vele manieren, maar kan worden beschouwd als meer basic. Dit zorgt voor een goed uitgangspunt om de werkingsprincipes van Dash uit te leggen. In het volgende deel wordt Dash vergeleken met Bitcoin. Sectie drie bespreekt het 'profiel' van de miner code en de selectie van de algorithmes die in hardware zullen worden geïmplementeerd. Sectie vier geeft een overzicht van de architectuur en de keuze van de FPGA. De laatste sectie gaat over de algorithmes zelf. Hoe elk algoritme werkt wordt kort besproken. De implementatiekeuzes worden toegelicht, en de hardware wordt vergeleken met zijn tegenhanger in software. Voor een meer uitgebreide uitleg wordt verwezen naar de Engelstalige Paper.
7. Nederlandstalige samenvatting

7.2 Bitcoin

Bitcoin is een valuta net zoals de Euro, maar is volledig digitaal. Er zijn geen fysieke biljetten of munten. Het is enkel een stukje software dat jouw Bitcoins bevat. Bitcoin is ontstaan in 2009 en is vandaag de populairste crypto valuta. Voor zijn netwerk gebruikt Bitcoin een gedecentraliseerd systeem waarin er geen centrale entiteit is die alles regelt.[2]

Transacties worden gebruikt om Bitcoins te verhandelen. Het zijn kleine berichten met verwijzingen naar eerdere ongebruikte transactie outputs, om de eigendom van de benodigde Bitcoins te bewijzen. Dit zijn de zogenaamde UTXOs of Onbestede Transactie Outputs. Als de transactie eenmaal goedgekeurd is, kan deze door de miner worden gekozen om in het volgende blok van het blockchain opgenomen te worden. Om de miner te overtuigen de transactie te selecteren, wordt er vaak een kleine beloning voor de miner aangeboden.

De blockchain is een lijst van blokken die de volgorde waarin transacties werden uitgevoerd dicteert. Alle transacties in hetzelfde blok worden uitgevoerd op hetzelfde ogenblik. Het functioneert als de ruggengraat van de verificatiemethode in Bitcoin. Blokken worden toegevoegd aan de blockchain door de miners. Ze lossen een wiskundig probleem op dat vergelijkbaar is met het breken van een cijferslot.

Miners zijn degenen die de nieuwe transacties toevoegen aan de blokken, die op hun beurt worden toegevoegd aan de blockchain. Mining is hetgeen dat het netwerk beveiligt tegen frauduleuze transacties, want een miner mag geen transacties aan de blok toevoegen als de transactie niet geverifieerd is. Een nieuw blok wordt elke 10 minuten gemined, gemiddeld gezien. Het proces van mining wordt uitgevoerd door een mathematische functie op de blockheader van de nieuwe blok toe te passen. Dit proces wordt herhaald totdat de uitkomst van de functie onder een bepaalde waarde komt. Elke keer als de uitkomst wordt berekend en de waarde niet laag genoeg is, wordt een teller in de blok verhoogd. Deze functie wordt een hash functie genoemd en wordt veel gebruikt in de cryptografie.[26] Elke blok die gemined wordt levert de eerste miner een bedrag Bitcoins op.

Met het groeiend aantal gebruikers van Bitcoin worden ook meer en meer transacties verstuurd. Slechts een blok kan elke tien minuten verwerkt worden. Deze blokken kunnen ook maar een beperkt aantal transacties bevatten. Dit is het grootste probleem van Bitcoin en zal er uiteindelijk voor zorgen dat Bitcoin nooit de huidige banksystemen zal kunnen vervangen.
7.3 Dash

Dash is een crypto valuta net zoals Bitcoin, maar heeft toch grote verschillen op bepaalde vlakken. Het grootste verschil van Dash met Bitcoin is het gebruik van masternodes in Dash.[10] Dit zijn normale nodes met een uitgebreid takenpakket. Ze zorgen ervoor dat Dash sneller transacties kan verwerken en dat er een vorm van anonimiteit is. Hiervoor gebruikt het twee nieuwe mechanismes: InstantX en Darksend respectievelijk.

InstantX is gebaseerd op het voortijdig locken van de inputs van een transactie.[16] Hierdoor moet de transactie nog niet opgenomen worden in een blok om toch als geaccepteerd te kunnen worden beschouwd.

Darksend is een mechanisme dat zorgt voor het mixen van de coins binnenin de transactie.[6] Zo worden ze ontraceerbaar en is de transactie anoniem.


7.4 Profiling

Om een degelijke acceleratie te kunnen halen moeten de traagste hash functies uit de miner code gehaald worden. Hiervoor moet deze eerst worden geprofileerd. Dit laat toe om de vertraging door elk algoritme apart te kunnen uitlezen. Hiervoor zijn er drie verschillende profilings uitgevoerd: op de originele code, op een versie zonder networking en met de referentiedata van de NIST SHA-3 competitie.[18] Uit de resultaten blijkt dat Skein, JH en Keccak de traagste algoritmes zijn.

7.5 Design

De FPGA die voor deze toepassing kan gebruikt worden moet voorzien in een grote hoeveelheid aan programmeerbare logica en kan best ook een ingebouwde processor hebben. Hierdoor is de ZedBoard[11] met geïntegreerde ARM core gekozen geweest. Voor het design zijn er twee grote alternatieven. Een ontwerp dat een gedeeld register gebruikt samen met een kleiner register voor de communicatie tussen de software en hardware, of een ander systeem dat een DMA (Direct Memory Access) gebruikt om de gegevens te transporteren. Het eerste systeem nodt heeft aan een gecompilteerde code voor de ARM core en het maken van deze bracht veel problemen met zich mee.

Het tweede ontwerp is uiteindelijk gekozen. De functies moeten nu wel aangepast worden aan de nieuwe interface die de DMA gebruikt. Tussenin de functies wordt wel nog de 512-bit waarde gebruikt, zodat de overhead zo klein mogelijk blijft. De software wordt nu verzorgd door een bare-metal applicatie, die geen nood heeft aan een besturingssysteem.
7. Nederlandstalige samenvatting

7.6 Implementatie van de algoritmes

7.6.1 Skein

Skein behandelt zijn inputs in blokken van 64 bytes en verwerkt elke blok in 72 rounds. Elk van deze rounds bestaat uit een MIX functie en een permutatie functie. Deze round wordt ook wel Threefish genoemd. Na de verwerking van een blok, wordt de XOR berekend met de input van die blok.[27]

Voor de implementatie van Skein worden de 72 rounds opgesplitst in negen keer acht rounds. Dit heeft als voordeel dat nu de rotatie afstanden vastliggen in elk van de MIX functies. Ook de eerste input hash kan voorberekend worden aangezien Dash steeds hetzelfde aantal inputbits nodig heeft. Het hierbij horende datapad en kritisch pad kan gevonden worden in Figuur 6.4. X11 laat geen pipelining toe, hoewel er in deze implementatie wel de optie toe is. Er kan namelijk een extra register worden geplaatst na de vierde round. Dit zou het kritische pad halveren en laat pipelining toe.

7.6.2 JH

Het hashen met JH verloopt in drie grote stappen. In een eerste stap wordt de XOR berekend van de input en een deel van de vorige output. Dit wordt door 42 rounds gestuurd, elk bestaande uit een S-box, een lineaire transformatie en een permutatie. Figuur 6.5 illustreert deze round functies. Op het einde wordt nogmaals de XOR berekend van de input met het overige deel van de output.[33]

Elke round is er nood aan een nieuwe round constante, er is dus geen reden om verschillende rounds samen te verwerken. Elke round wordt dus apart in een cycle verwerkt. Voor de lineaire transformatie wordt ook gebruik gemaakt van een alternatieve methode enkel bitsgewijze XORs nodig heeft. De volledige implementatie, met aanduiding van het kritische pad is te vinden in Figuur 6.9. Deze implementatie laat geen pipelining toe, dit is geen probleem aangezien X11 toch de volgende input niet vroeger aan de ingang kan afleveren.

7.6.3 Keccak

Keccak is de functie met de grootste interne state. Het gebruikt 1600 bits in de berekening van zijn 24 rounds. Deze 1600 bits worden in een 3D matrix (5x5x64 bit) geplaatst. Elke round van Keccak bestaat uit vijf functies: theta, rho, pi, chi, and iota. Deze bestaan enkel uit AND, XOR, NOT operatoren en rotaties.

Voor de implementatie is de referentie implementatie van Keccak gebruikt als startpunt. Vele aanpassingen waren nodig om deze functie compatibel te maken met X11. Een optimalisatie is uitgevoerd op de Keccak implementatie, de 'lane complementing transform'.[15] Deze zorgt ervoor dat de Chi functie van Keccak 80% minder NOT poorten gebruikt. Het enige benodigde hiervoor is dat sommige bits (eenmaal) geïnverteerd moeten worden aan de input en dat sommige ANDs door
7.6.4 Hardware synthese
De synthese van de designs laat toe om de benodigde logica en geheugen te kennen, ook een timing analyse wordt hierbij uitgevoerd. De resultaten voor de synthese zijn te vinden in Figuur 6.1. De timing analyse wordt uitgevoerd op het gehele systeem en geeft de onder andere de vertraging van het kritische pad. In het geïmplementeerde design is dit 29.654ns. Dit zorgt ervoor dat de maximale klokfrequentie voor het gehele systeem begrensd wordt op 33.7MHz. Voor de benchmarking van de hardware wordt een veiligheidsmarge ingebracht. Het benchmarken van de hardware zal gebeuren op 30MHz.

7.6.5 Vergelijking met software
Na de implementatie kan elk algoritme individueel worden vergeleken met zijn software versie. Hiervoor worden 5,000,000 hashes uitgerekend in software. Voor de hardware wordt de tijd gemeten (in cycles) die er verstrijkt tussen de eerste data blok die de DMA verlaat en de laatste data blok die aankomt aan de DMA. De tijd dat het kost om dan een hash uit te rekenen, wordt dan vergeleken. De resultaten zijn terug te vinden in Tabel 6.2.

7.7 Conclusie
De resultaten uit de voorgaande sectie bewijzen dat hardware acceleratie toelaat om enorme verbeteringen te maken in de CPU-miner van Dash. Drie algoritmes werden vervangen door een equivalente hardware implementatie. De algoritmes waren na implementatie 10.8 tot 32 maal sneller dan hun software versies. Verdergaande op dit werk, zouden meer algoritmes geïmplementeerd kunnen worden in hardware. Dit zou toelaten om de hash snelheid nog verder te verhogen zonder extra overhead te veroorzaken.
Bibliography


[5] Koen Crijns. 3x amd radeon r9 290 / 2x amd radeon r9 290x review: Asus, gigabyte en sapphire. URL: https://be.hardware.info/reviews/5204/14/3x- amd-radeon-r9-290--2x-amd-radeon-r9-290x-review-asus-gigabyte- en-sapphire-stroomverbruik, last checked on 2016-07-14.


[17] ig0tik3d. xcoin-cpuminer. URL: https://github.com/ig0tik3d/xcoin-cpuminer, last checked on 2016-02-25.


[27] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, Jesse Walker. The skein hash function family. Submission to NIST, round 3, October 2010.


Master thesis filing card

Student: Dries Truyens

Title: FPGA based hardware accelerator for Dash mining

UDC: 621.3

Abstract:
This work describes the partial hardware implementation of a Dash CPU-miner. The main goal of this work is to increase the hash rate of the default CPU-miner using hardware acceleration. Firstly, the way Bitcoin works is explained to get a better understanding of the overall principles behind cryptocurrencies. Then, Dash is compared to Bitcoin. The changes Dash made to the original Bitcoin network and the new vulnerabilities are explained. The code for the Dash miner is examined using profiling tools, which reveal the amount of time each algorithm consumes. The slower algorithms are then implemented in hardware. At the end, the hardware and their software counterparts are compared, and the results are discussed.

Thesis submitted for the degree of Master of Science in Electrical Engineering, option Electronics and Integrated Circuits

Thesis supervisor: Prof. dr. ir. I. Verbeauwhede

Assessors: Prof. dr. ir. B. Preneel
          Prof. dr. ir. M. Verhelst

Mentors: Ir. P. Maene
          Ir. B. Yang
          Dr. ir. F. Vercauteren