CHAPTER 4
DESIGN AND TESTING

4.1 Test Software

There are two software that will be used to run this test:

4.1.1 Bitcoin Core v0.8.0

Proof-of-Work Algorithm: SHA256

https://github.com/bitcoin/bitcoin/releases/tag/v0.8.0

4.1.2 Litecoin Core v0.8.7.5

Proof-of-Work Algorithm: Scrypt

https://github.com/litecoin-project/litecoin/releases/tag/v0.8.7.5

4.1.3 Required Dependencies:

- git
- build-essential libtool autotools-dev automake pkg-config libssl-dev libevent-dev bsdmainutils
- libboost-system-dev libboost-filesystem-dev libboost-chrono-dev libboost-program-options-dev libboost-test-dev libboost-thread-dev
- libboost-all-dev
- software-properties-common
- repository ppa:bitcoin/bitcoin
- libdb4.8-dev libdb4.8++-dev
- libminiupnpc-dev
- libzmq3-dev
- libqt5gui5 libqt5core5a libqt5dbus5 qttools5-dev qttools5-dev-tools libprotobuf-dev protobuf-compiler
- libqt4-dev libprotobuf-dev protobuf-compiler

4.2 Test System

4.2.1 There are specification and configuration used for Test System 1 :

Table 4.1: Test System 1 specification table

Processor Name	Intel Pentium G4560
Code Name	Kaby Lake
Technology	14nm
Specification	Intel Pentium G4560 CPU @ 3.50GHz
L3 Cache	3 MB
Number of Cores	2
Number of Threads	4
Model	MSI Z270 Gaming M5
Chipset	Intel Z270, Kaby Lake
Max Bandwidth	DDR4-2133 (1066 MHz)
Size	8192 MBytes
Channels #	Dual
DRAM Frequency	1066.5 MHz
CAS# Latency (CL)	15.0 clocks
Model	Samsung SSD 960 EVO 250GB
Firmware Version	3B7QCXE7
Interface	PCIe Gen. 3 x 4
Total Capacity	233GB
OS Family	Linux
Distribution	Ubuntu Desktop
OS Version	16.04 LTS
Kernel Version	4.4

4.2.2 There are specification and configuration used for Test System 2 :

Table 4.2: Test System 2 specification table

Processor Name	AMD Ryzen 3 1300X
Code Name	Summit Ridge
Technology	14nm
Specification	AMD Ryzen 3 1300X CPU @ 3.5GHz
L3 Cache	8 MB
Number of Cores	4
Number of Threads	4
Model	Gigabyte A320M-DS2
Chipset	AMD A320, Summit Ridge
Max Bandwidth	DDR4-2133 (1066 MHz)
Size	8192 MBytes
Channels #	Single
DRAM Frequency	1066.5 MHz
CAS# Latency (CL)	15.0 clocks
Model	Samsung SSD 850 EVO 250GB
Firmware Version	EMT03B6Q
Interface	SATA 6.0Gb/s
Total Capacity	233GB
OS Family	Linux
Distribution	Ubuntu Desktop
OS Version	16.04 LTS
Kernel Version	4.4

4.2.3 There are specification and configuration used for Test System 3 :

Table 4.3: Test System 3 specification table

Processor Name	Intel Core i5 4200H
Code Name	Haswell
Technology	22 nm
Specification	Intel Core i5 4200H CPU @ 2.80GHz
L3 Cache	3MB
Number of Cores	2
Number of Threads	4
Model	ASUS Intel® HM86 Express Chipset
Chipset	Intel HM86, Haswell
Max Bandwidth	DDR3-1600 (800 MHz)
Size	8192 MBytes
Channels #	Single
DRAM Frequency	800 MHz
CAS# Latency (CL)	11.0 clocks
Model	Samsung SSD 850 EVO 250GB
Firmware Version	EMT02B6Q
Interface	SATA 6.0Gb/s
Total Capacity	233GB
OS Family	Linux
Distribution	Ubuntu Desktop
OS Version	16.04 LTS
Kernel Version	4.4

4.2.4 There are specification and configuration used for Test System 4 :

Table 4.4: Test System 4 specification table

Processor Name	AMD Ryzen 7 1800X
Code Name	Summit Ridge
Technology	14nm
Specification	AMD Ryzen 7 1800X CPU @ 3.6GHz
L3 Cache	16MB
Number of Cores	8
Number of Threads	16
Model	MSI X370 Gaming Pro Carbon
Chipset	AMD X370, Summit Ridge
Max Bandwidth	DDR4-2133 (1066 MHz)
Size	8192 MBytes
Channels #	Single
DRAM Frequency	1066.5 MHz
CAS# Latency (CL)	15.0 clocks
Model	Samsung SSD 850 EVO 250GB
Firmware Version	EMT03B6Q
Interface	SATA 6.0Gb/s
Total Capacity	233GB
OS Family	Linux
Distribution	Ubuntu Desktop
OS Version	16.04 LTS
Kernel Version	4.4

4.3 Test Scenario

The author need to create new cryptocurrency to run the test and it must not connect to the Bitcoin or Litecoin blockchain, to create new cryptocurrency using bitcoin core there are many step:

1. Installing all required dependencies on the test system, you can follow these steps:
   - Open the Terminal on Ubuntu and type these commands
   - sudo apt-get install git
   - sudo apt-get install build-essential libtool autotools-dev automake pkg-config libssl-dev libevent-dev bsdmainutils
   - sudo apt-get install libboost-system-dev libboost-filesystem-dev libboost-chrono-dev libboost-program-options-dev libboost-test-dev libboost-thread-dev
   - sudo apt-get install libboost-all-dev
   - sudo apt-get install software-properties-common
   - sudo add-apt-repository ppa:bitcoin/bitcoin
   - sudo apt-get update
   - sudo apt-get install libdb4.8-dev libdb4.8++-dev
   - sudo apt-get install libminiupnpc-dev
   - sudo apt-get install libzmq3-dev
   - sudo apt-get install libqt5gui5 libqt5core5a libqt5dbus5 qttools5-dev qttools5-dev-tools libprotobuf-dev protobuf-compiler
   - sudo apt-get install libqt4-dev libprotobuf-dev protobuf-compiler
2. On this step you need to copy the coin project from github into your test system, you can do that by typing this command in your terminal:
   - git clone -b 0.8 https://github.com/litecoin-project/litecoin.git
3. Once the coin forked, you need to change the name of your coin project (in this case I'll name the coin as Unikacoin. to change the coin name into your own, you can type this following commands:

- find . -type f -print0 | xargs -0 sed -i 's/litecoin/unikacoin/g'
- find . -type f -print0 | xargs -0 sed -i 's/Litecoin/Unikacoin/g'
- find . -type f -print0 | xargs -0 sed -i 's/LiteCoin/UnikaCoin/g'
- find . -type f -print0 | xargs -0 sed -i 's/LITECOIN/UNIKACOIN/g'
- find . -type f -print0 | xargs -0 sed -i 's/LTC/UNQ/g'

4. After changing the coin name you should change the network parameters, firstly you need to change up the port that used by your coin network:
   - find . -type f -print0 | xargs -0 sed -i 's/9333/2333/g'
   - find . -type f -print0 | xargs -0 sed -i 's/9332/2332/g'

5. Now you need to change the public-address identifier, as you know all the Litecoin public address started with capital L. so you going to make your own by following this step:
   - open your coin directory and go to `src` folder and open base58.h
   - search class CBitcoinAddress (fyi it's on the line number 275)
   - now change the value of variable PUBKEY_ADDRESS
   - you'll notice it was INT value, so you need to open this page to know the INT code for each character by looking at bitcoin address prefixes table.
   - https://en.bitcoin.it/wiki/List_of_address_prefixes
   - change the value, then save.

6. After that you going to generate the alert key and genesis hash, you can do that by typing this command into your terminal:
   - openssl ecparam -genkey -name secp256k1 -out alertkey.pem
   - openssl ec -in alertkey.pem -text > alertkey.hex
   - openssl ecparam -genkey -name secp256k1 -out testnetalert.pem
   - openssl ec -in testnetalert.pem -text > testnetalert.hex
   - openssl ecparam -genkey -name secp256k1 -out genesiscoinbase.pem
   - openssl ec -in testnetalert.pem -text > genesiscoinbase.hex

7. Now you have to change the alert key of the coin to your alert key:
   - in `src` directory open file alert.cpp and find pszMainKey value to your alert key value
   - to see your alert key, type `cat alertkey.hex` on the terminal, and copy the

value of public address

- replace them, if you want to use the testnet you can replace them too using `cat testnetalert.hex` copy the value and replace
- after that you need to change genesis hex value, `cat genesiscoinbase.hex` and open main.cpp file and go to line number 2788, change parsehex("value"); into your genesis hex value.

8. Now you going to handling the pure magic, in this step you need to change the `magic number`, it's very important because it's going to prevent your coin from connecting to another network (for example Litecoin network). And you're probably notice it's all hexadecimal number and it's really hard to understand, so you can just change them randomly and leave the 0x digit the way they are.
   - you can change the testnet one on main.cpp line number 2745.
   - you can change the main one on main.cpp line number 3082. and the pure magic is done, now save the change.
9. After that you should delete all the hardcoded network information that exist in the Litecoin network. There are two things you going to delete, the first is sign node and the second is hardcoded ip addresses.
   - open file net.cpp and line number 1178. You can just delete all and leave the NULL value.
   - you can do the same thing for the testnet. And you need to scroll down a little bit, and find the array pnSeed[], actually it's the hardcoded ip addresses you can just delete that and leave 0x0. And save.
10. And the next part is optional which is changing the blockchain parameter. If you just need to test the coin you don't need to do this. But if you want to change the blockchain information you can look around on main.cpp
11. To change the initial mining block code which determine how many coin you get if you find the block.
    - open main.cpp and go to line 1090. You will find nSubsidy variable, you can change the value to your own. From this step you that the amount of the coin is doesn't mean the number of the block. So, block is not the coin. You also need to change nValue on line number 2787. Then back to line 1090.
    - scroll down a little bit and you will see nTargetTimespan, nTargetSpacing. It's

basically a variable that control the block that going to be mined if a bunch of miner connect to the network.

12. If you want to change something like maximal money variable or coin base maturity you can go to file main.h and just change the value. And save.
13. On main.cpp line number 2782 its customary too, you can change the pszTimestamp value to your own.
14. After all, you need to change the nTime variable on main.cpp, you'll notice it was an epoch time, to get current epoch time you can type on terminal `date +%s` and copy it. And don't forget to delete the hashMerkleRoot value. Leave it "0x" and save. And compile your coin by typing `make -f makefile.unix`
15. If the compilation has finished, you can try to run by typing `./unikacoind` but it must be failed. Because the merkle root = "0x" but anyway this is good. Now you can plug your merkle hash from the debug.log, it's located on your home directory named .unikacoin and its hidden folder. Ctrl + H to show all your hidden file. Open debug.log and copy the last value on the debug.log without the timestamp, it's your valid merklehash. Copy it to your main.cpp right over the "0x" on your merklehash value (line number 2809). And save.
16. Now right below the testnet, before the debug print (line 2804). You need to paste this code:

```
if (true && block.GetHash() != hashGenesisBlock)
        {
          printf("Searching for genesis block...\n");
          // This will figure out a valid hash and Nonce if you're
          // creating a different genesis block:
          uint256 hashTarget = CBigNum().SetCompact(block.nBits).getuint256();
          uint256 thash;
          char scratchpad[SCRYPT_SCRATCHPAD_SIZE];

          loop
```

```
        {
#if defined(USE_SSE2)
            // Detection would work, but in cases where we KNOW it always has SSE2,
            // it is faster to use directly than to use a function pointer or conditional.
#if defined(_M_X64) || defined(__x86_64__) || defined(_M_AMD64) || (defined(MAC_OSX) && defined(__i386__))
            // Always SSE2: x86_64 or Intel MacOS X
            scrypt_1024_1_1_256_sp_sse2(BEGIN(block.nVersion), BEGIN(thash), scratchpad);
#else
            // Detect SSE2: 32bit x86 Linux or Windows
            scrypt_1024_1_1_256_sp(BEGIN(block.nVersion), BEGIN(thash), scratchpad);
#endif
#else
            // Generic scrypt
            scrypt_1024_1_1_256_sp_generic(BEGIN(block.nVersion), BEGIN(thash), scratchpad);
#endif
            if (thash <= hashTarget)
                break;
            if ((block.nNonce & 0xFFF) == 0)
            {
                printf("nonce %08X: hash = %s (target = %s)\n", block.nNonce, thash.ToString().c_str(), hashTarget.ToString().c_str());
            }
            ++block.nNonce;
```

```
        if (block.nNonce == 0)
        {
            printf("NONCE WRAPPED, incrementing time\n");
            ++block.nTime;
        }
    }
    printf("block.nTime = %u \n", block.nTime);
    printf("block.nNonce = %u \n", block.nNonce);
    printf("block.GetHash = %s\n", block.GetHash().ToString().c_str());
}
```

17. Now delete the testnet genesis hash (line 2749) leave it “0x”, and the main net one (line 38) leave it “0x” and save. Then compile and run it again. To run the testnet you can type `./unikacoind -testnet` and the main `./unikacoind`
18. After that you can open the debug.log for testnet and the main net on your coin configuration directory just like before. Go to very end, copy the nTime, nNonce, GetHash value to main.cpp with same variable name you can just replace it. And save.
19. And one more thing you have to do, you have to delete all the checkpoints on the file checkpoints.cpp just delete all the checkpoints and leave it (0, uint256(“0x*replace this with your genesis hash*”)); and for the timestamp correct it with your own. Total number of transaction is going to be 0, and for the estimated transaction you can leave it 1.0 or something else. Do the same thing for the testnet. And save.
20. Now you can compile the graphical interface, to do this you need to go into your coin directory and typing `qmake` after that simply typing `make` and now your coin was created.
21. You can do the same thing using the Bitcoin core or the Litecoin core since the Litecoin core is forked from Bitcoin core too.

22. Before running the test you need to open your coin configuration folder that hidden on your home directory, and make a file named `unikacoin.conf` and write this code: addnode=*your peer ip address* (example: addnode=192.168.1.2) and do the same thing on the other computers since you don’t have DNSSeed.

23. To run the test open the terminal you can go to your coin directory and type `./unikacoin-qt`

After the graphical interface application opened and get synchronized you can open the console and type `setgenerate true` to mine your block. Since it’s a blockchain author need more than 1 computer to run the tests.