REPORT ATTIVIT À PROGETTUALE

SISTEMI DIGITALI

FPGA-Petalinux sha256D miner

Dragoș Mihăiță Iftimie

Abbozzo di un introduzione al mining di cryptovalute

Attualmente, le cryptovalute esistenti vengono implementate utilizzando la tecnologia Blockchain. [Forse non vero]
Essa non e' altro che una catena di blocchi condivisa e negoziata dai peer in maniera decentralizzata e disentermediata.
Nel nostro caso specifico la modalita' di negoziazione usata e' quella del ProofOfWork.
Per far si che la rete accetti un nuovo blocco, chi e' intenzionato ad aggiungerne uno, dati il blocco precedente e i dati contenuti in quello che si vuole aggiungere, e' necessario risolvere un problema computazionalmente difficile.
Piu' nel dettaglio, il ruolo del miner e' quello di raccogliere le transazioni dai peer della rete per creare un blocco e (grazie alla liberta' di modificare 64 bit del blocco, nonce ed extranonce) far si che il suo hash sia minore di un intero chiamato target, deciso dalla rete.
In questo caso il problema e' "difficile" perche' l'unico modo per trovare uno degli hash validi e' scorrere tutto lo spazio di ricerca.

Progetto

Nel file "Project.pdf"[1] e' possibile trovare il project charter da me creato per il corso IT project management; esso e' stato una guida durante l'implementazione e un utile strumento per avere un feedback su come e' andato il progetto, consultandolo a cose fatte.
Il file contiene, oltre che la pianificazione del progetto, anche un quality assurance e risk management plan.
Leggendo introduzione, project goals e project scope [1] e' possibile capire quali sono il contesto, gli obiettivi e i confini del progetto.

In poche parole bisogna capire se e quanto e' conveniente utilizzare una Zedboard7000 per il mining delle seguenti cryptovalute: Garlicoin, Dogecoin, Groeslcoin, Bitcoin

Work breakdown structure

Di seguito e' possibile vedere la scomposizione dell'intero progetto nei sotto-progetti necessari alla sua implementazione.

- **HLS**
  A partire da codice C, sintetizzare una soluzione e un'implementazione mappabile su FPGA. Progettare un modulo che svolga il lavoro piu' "pesante" del miner;

- **Vivado**
  creare un progetto per configurare fino al livello board.
  Indicare l'implementazione HLS da usare, ottenerne l'indirizzo base, quello delle porte di controllo, input e output; configurare le altre risorse della board;

- **Petalinux**
  Configurare il sistema operativo, modificandolo e compilandolo in base alle necessita' applicative;

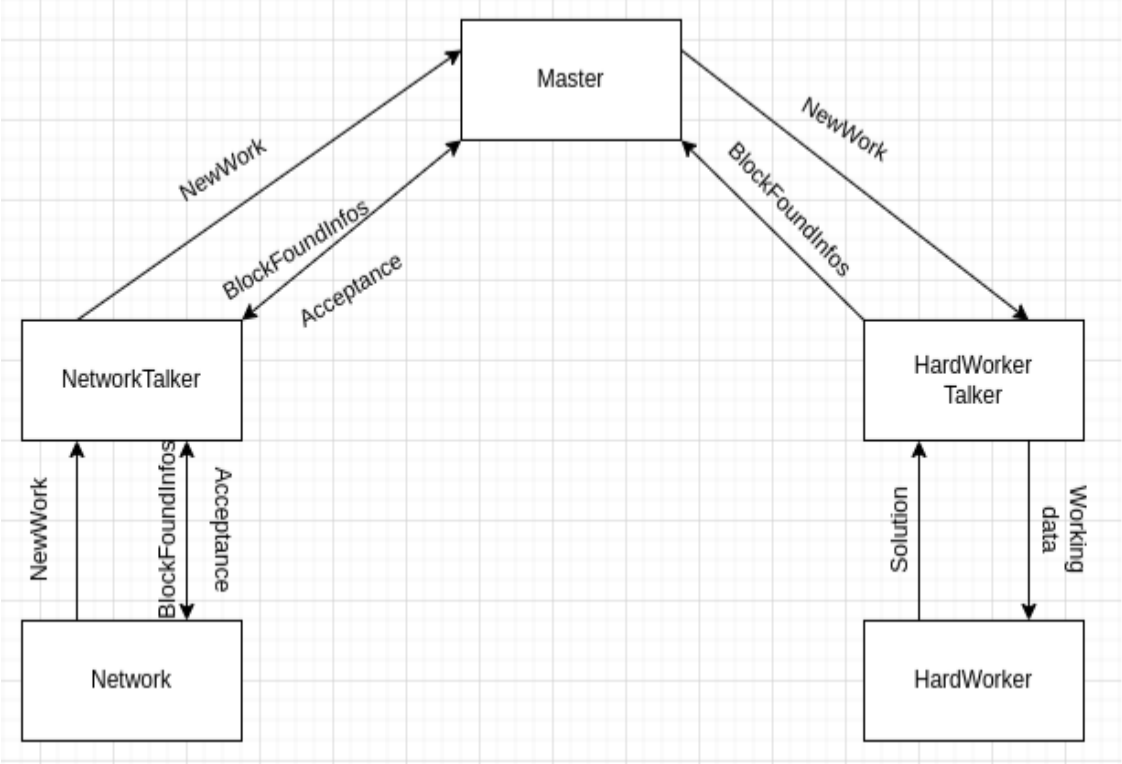
- **Miner**

Sviluppare un software che verra' eseguito su Petalinux e che si occupera' del management del processo di mining;

- **API**
  Sviluppare delle API invocabili dal software per permettergli di interagire con l'FPGA;

- **Test**
  Testare l'intera infrastuttura interagendo con blockchain reali per ottenere i dati che stiamo che stiamo cercando (potenza di calcolo, consumo).

Design del miner



Il design da me proposto cerca di essere il meno vincolante possibile tramite una netta separazione dei compiti e delle responsabilita' dei vari componenti coinvolti.
Cio' va a scapito delle performance, che un design piu' monolitico avrebbe offerto.
Tra le liberta' concesse vi sono:
- Indipendenza spaziale dei vari componenti (componenti tra loro remoti? Locali?);
- Indipendenza dal quale e quante cryptovalute si vogliono minare (Garlicoin?Bitcoin?...);
- Indipendenza da chi e quanti computano (FPGA? PC? Cellulare?).

Hard worker – HLS

Garlicoin:
Nella cartella "alliumHLScod" e' possibile trovare il codice da cui avviene la sintetizzazione; esso e' stato testato via software tramite il file testbench.cpp.

Il codice C viene dall'implementazione ufficiale di allium ed e' stato adattato a un paradigma descrittivo per poter essere utilizzato in HLS.

Dogecoin:
Sfortunatamente, per via della mia malagestione, il codice, preso e adattato dall'implementazione ufficiale di scrypt, e' stato perso.
In ogni caso e' stato testato positivamente via software.

Groestlcoin:
Nella cartella "groestlHLScod" e' possibile trovare il codice poi sintetizzato; esso e' stato testato positivamente via software; il codice e' stato adattato dall'implementazione ufficiale di groestl.

Bitcoin:
Nella cartella "sha256dHLScod" e' possibile trovare il codice poi sintetizzato.
Anche esso e' stato testato positivamente via software.
La top-function (scan_sha256d.cpp) implementa un modulo che, tramite axi-lite, ottiene in input il target e la parte di header pre computata dall'HardWorkerTalker, per offrire in output l'ultima soluzione trovata, il goldenNonce.
Per motivi di debug, il modulo offre anche il nonce attuale e l'hash attuale.
L'implementazione di sha256 usata e' la seguenta:
https://opensource.apple.com/source/clamav/clamav-158/clamav.Bin/clamav-0.98/libclamav/sha256.c.auto.html.

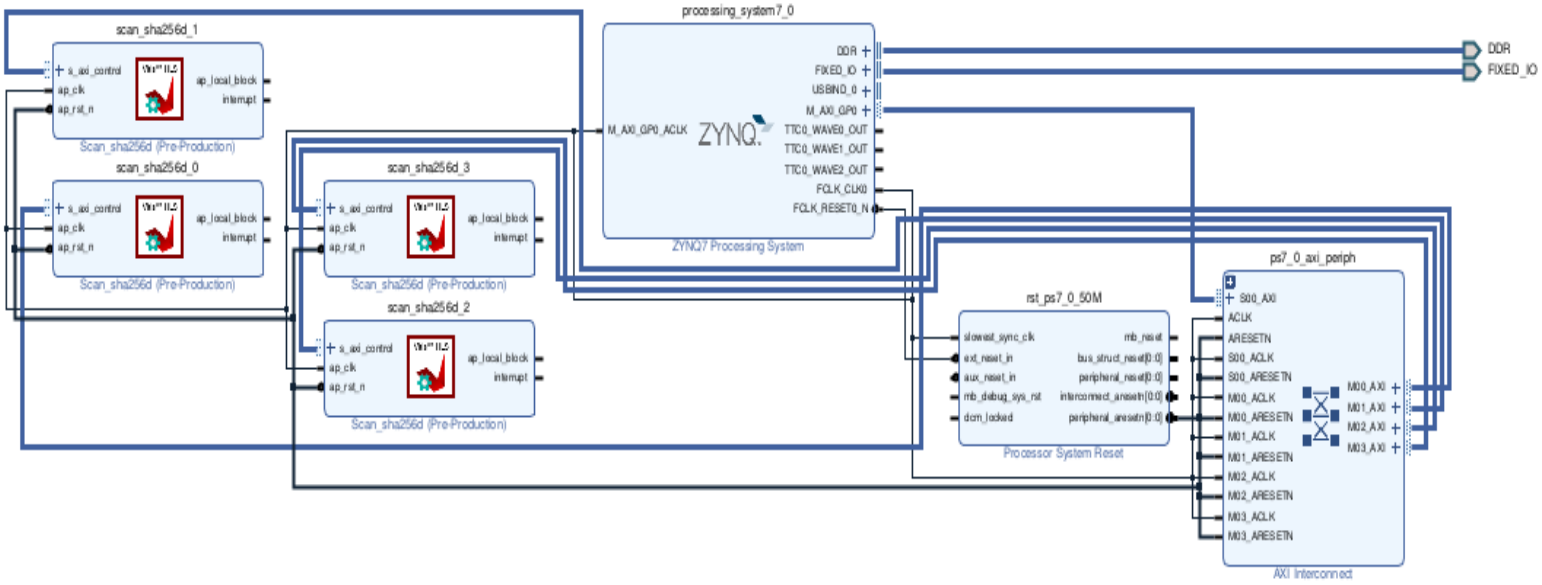
Oltre alle ottimizzazioni da me aggiunte per la copia degli input/output, l'implementazione di sha256 contiene dei flag definibili a tempo di compilazione che ne permettono l'ottimizzazione; questo mi ha permesso di applicare diversi livelli di ottimizzazione, che mi hanno portato a delle conclusioni.

Seguono due esempi di questi flag:

```
    for (i = 0; i < SHA256_HASH_WORDS; i++) {
#ifdef SHA256_FAST_COPY
      *((unsigned int *) hash) = BYTESWAP(sc->hash[i]);
#else /* SHA256_FAST_COPY */
      hash[0] = (unsigned char) (sc->hash[i] >> 24);
      hash[1] = (unsigned char) (sc->hash[i] >> 16);
      hash[2] = (unsigned char) (sc->hash[i] >> 8);
      hash[3] = (unsigned char) sc->hash[i];
#endif /* SHA256_FAST_COPY */
```

```
#if SHA256_UNROLL == 1
  for (i = 63; i >= 0; i--)
    DO_ROUND();
#elif SHA256_UNROLL == 2
  for (i = 31; i >= 0; i--) {
    DO_ROUND(); DO_ROUND();
  }
#elif SHA256_UNROLL == 4
  for (i = 15; i >= 0; i--) {
    DO_ROUND(); DO_ROUND(); DO_ROUND(); DO_ROUND();
  }
```

Vivado



A partire dal modulo sintetizzato in HLS, e' stato possibile produrre un design Vivado con 4 workers in parallelo.
Il design ha un consumo elettrico stimato di 1.956 W e l'utilizzo di LUT/FF/BRAM del 78/58/17 %
Successivi test hanno dimostrato che il modulo comandato via software produce l'hash che ci si aspetta e che trova le giuste soluzioni.

Garlicoin e Dogecoin:
Sfortunatamente, entrambi i moduli non possono essere mappati sull'FPGA per via delle risorse insufficenti.
La ricerca per queste due valute si ferma qui.

Groestlcoin:
Il design Vivado utilizzando il modulo sintetizzato ha un consumo stimato di 1.715W e utilizzo di LUT/FF/BRAM del 23.94 14.92 19.6.
Durante i test reali, il modulo non produceva in output l'hash che ci si aspettava.
Per via delle insufficienti risorse assegnate al progetto (tempo), la ricerca per questa valuta e' stata interrotta piuttosto che tornare alla fase HLS e risolvere.

Da ora in poi ci si riferira' soltanto al branch del progetto che si occupa del Bitcoin.

Petalinux

Questa fase non ha richiesto uno sforzo concettuale troppo oneroso, pero' e' stata estremeamente time-consuming per via dei continui errori da dover risolvere.
L'aver utilizzato Arch Linux non ha aiutato.

```
WARNING: Host distribution "arch" has not been validated with this version of the build system; you may possibly experience unexpected failures. It is recommended that you use a tested distribution.
```

Per andare incontro alle necessita' applicative, il rootfs e' stato configurato per includere i programmi necessari al funzionamento (ssh, python…);
Inoltre il kernel e' stato configurato per permettere a mmap di accedere direttamente agli indirizzi in memoria fisica.
Oltre a cio', sono stati anche generati il first stage boot loader, il pacchetto contenente tutti gli artefatti e il bitstream, da dare in pasto successivamente alla zedboard.

HardWorkerTalker

Questo componente si occupa di ricevere lavoro dal master, manipolarne le informazioni e assegnare a ogni worker un header in maniera tale che ognuno di essi lavori su uno spazio di ricerca diverso.
La comunicazione con il master avviene tramite code locali mentre per comunicare con i lavoratori sono state sviluppate delle primitive (codice nella cartella minerSoftware, scanShaPrimitives.py) per permettere di leggere/scrivere gli argomenti e interagire a basso livello con il modulo hardware.
Grazie a vivado e al file generato drivers.h, e' stato possibile conoscere gli indirizzi di memoria dei 4 lavoratori e dei loro controlli; le primitive scrivono direttamente in memoria utilizzando mmap.
Per ricevere soluzioni dai lavoratori, il componente, tramite polling, controlla se il golden nonce di ogni lavoratore e' cambiato e, in caso affermativo, lo manda al master come soluzione.

Una possibile miglioria sarebbe quella di sviluppare dei driver e far segnalare al lavoratore, tramite interrupt, che ha trovato una soluzione.

NetworkTalker

Il seguente componente implementa un client Stratum v1 (protocollo di comunicazione molto usato per comunicare con i server pool mining).
Esso e' stato sviluppato per il corso Component Based Software Engeneering e la sua documentazione puo' essere trovata nel file "StratumComponentDocumentation.pdf".

Master

Il master si occupa di mantenere in maniera centralizzata le informazioni necessarie all'intero processo (per esempio credenziali, ip del server che assegna lavori, portafoglio su cui si riceveranno i "frutti" del mining…) e log.

Test

L'intera infrastuttura e' stata fatta lavorare per 3 giorni utilizzando il modulo per Bitcoin, accettando lavori reali.
Sfortunatamente nessuna soluzione e' stata trovata.
Dalle misurazioni e' emerso che ognuno dei 4 lavoratori hardware esegue circa 50k hashes/second.
Il consumo effettivo non e' stato misurato e ci si e' limitati alla stima fornita da Vivado.

Conclusioni e commenti

Bitcoin:
Utilizzando un profitability estimator, con questa potenza di calcolo, e' stato stimato un guadagno di 0.00000453 euro l'anno.
E' troppo tardi per fare profitti con FPGA.

Giocando con le ottimizzazioni di cui si e' parlato nella parte HLS, non e' stato difficile capire che per ottenere di piu' e' conveniente sintetizzare un lavoratore piu' lento in termini di hashes/second,

se questo significa avere un lavoratore che occupa meno area sull'FPGA, in quanto cosi' e' possibile farne lavorare di piu' in parallelo.
Infatti, cercando di avere un lavoratore il piu' ottimizzato possibile (in termini di velocita'), ho potuto mappare un solo lavoratore capace di calcolare 57k hashe/second.

Garlicoin e Dogecoin:
Non c'e' molto da aggiungere, era prevedibile che i loro algoritmi di hashing non fossero mappabili in quanto ASIC-resistant.
Al tempo speravo comunque di riuscirci, ma ora ho capito che se le cose hanno dei nomi, e' molto probabile che quel nome descriva correttamente le cose.

Groestlcoin:
e' un peccato essermi fermato, riguardando ora le cose, questa era la criptovaluta piu' promettente.

Design:

Il design da me proposto, grazie alla sua flessibilita', si e' dimostrato estremamente utile.
Infatti oltre a permettere il mining di diverse crypto, il disaccoppiamento fornito dall'HardWorkerTalker mi ha permesso di realizzare molto velocemente l'estensione al progetto, che ha dei presupposti diversi da quello originale.

Progetto intero:
e' stato un lungo viaggio con molte svolte e senza un obiettivo preciso e dettagliato pre-determinato (il report e' un tentativo postumo di rendere il tutto coerente, grazie all'aiuto del project plan, che pero' e' stato sviluppato a meta' progetto in base alla sua evoluzione).
Ho capito l'importanza di avere un punto di partenza e di fine il piu' dettagliato possibile e che i tempi stimati per lo svolgimento di un progetto e' meglio siano pessimistici.

Estensione

Siccome il progetto originario non ha ottenuto nulla di "reale", ho avuto il bisogno di estenderlo.
L'estensione rilascia l'assunto "l'hard worker sara' su FPGA", prevedendo un lavoratore CPU.
Per questo ho sviluppato un HardWorkerTalker capace anche di cercare soluzioni.
Questa scelta va contro l'approccio non-monolita del design, ma e' stata fatta per ragioni di tempo.
Il talker, utilizzando ctypes, esegue allium, il cui codice e' in C.

Nella cartella GarlicoinSoftwareMiner e' possibile trovare il codice funzionante e qui e' possibile vedere il risultato di 2 ore di mining:

https://explorer.grlc.eu/get.php?q=GgCX4k45pJesNzJ6fkvqi5jzzq3YK68fJ3