Univerza v Ljubljani

Fakulteta za elektrotehniko

Gal Nadrag

Asinhrona sekvenčna digitalna vezja

Magistrsko delo

Magistrski študijski program druge stopnje Elektrotehnika

Mentor: Aleksander Sešek

Ljubljana, 2023

Zahvala

Hvala Sandi, car si. Karin hvala za smehe orehe. Družina, hvala de ste mi težil ko se mi ni dal. Pa še faks je tud dost kul. Pa folk na šihtu

Povzetek

V zadnjem desetletju smo bili priča prenehanju povečevanja taktnih frekvenc posameznih procesorskih jeder. A hkrati število tranzistorjev še vedno narašča. Trenutni trend je povečevanje števila jeder, vendar nekaterih problemov ni mogoče paralelizirati, zato ta pristop ne pomaga.

V tem delu predstavljamo metodo za načrtovanje asinhronih digitalnih vezij. Ta vezja ne potrebujejo taktnega signala in so hitrostno omejena le z omejitvami vrat in povezav. Predstavljamo tudi metodo za implementacijo takšnih vezij v FPGA. Implementacije osnovnih vezji doežejo efektivne frekvenco 35MHz.

Ključne besede: Asinhrona logika, Digitalna logika, FPGA

Abstract

In the last decade, we have witnessed the cessation of the increase in clock frequency of individual processor cores. At the same time, the number of transistors continues to grow. The current trend is to increase the number of cores, but some problems cannot be parallelized, so this approach does not help.

In this work, we present a method for designing asynchronous digital circuits. These circuits do not require a clock signal and are speed-limited only by gate and connection constraints. We also present a method for implementing such circuits in FPGA. Implementations of basic cirucits reach effective frequencies of 35MHz.

Key words: Asynchronous logic, Digital logic, FPGA

Vsebina

Seznam slik

Seznam tabel

Seznam uporabljenih simbolov in kratic

V pričujočem zaključnem delu so uporabljene naslednje veličine in simboli:

Kratica	**Pomen**
FPGA	Field programmable gate array
STG	State transition graph
DFS	Dataflow structure

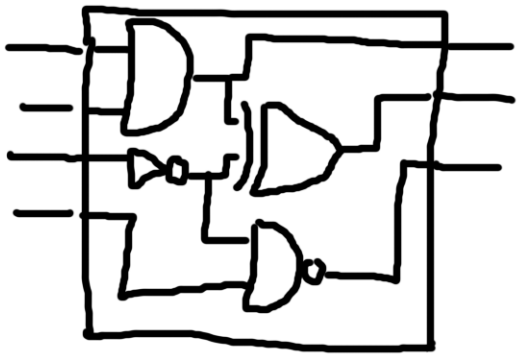
Veličina / oznaka		**Enota**	
Ime	Simbol	Ime	Simbol
čas	t	sekunda	s
frekvenca	f	Hertz	Hz
napetost	U	Volt	V
tok	I	Amper	A
kapacitivnost	C	Farad	F

obdelava

1 Uvod

Digitalna vezja uporabljamo za obdelavo podatkov. Z njimi implementiramo algoritme, ki iščejo praštevila, rešujejo enačbe itd.

Teoretično lahko poljuben algoritem implementiramo z ogromno tabelo. Glede na vhodne podatke preprosto najdemo pripadajočo rešitev. Prednost takega pristopa je, da ne potrebujemo nikakršnega spomina. Takšnim vezjem pravimo **kombinatorična vezja**. Kombinatorična vezja so pogosto uporabljena za osnovne operacije npr. logične funkcije(IN, ALI...), seštevanje, primerjava števil itd. ne pa za implementacijo celotnih algoritmov.

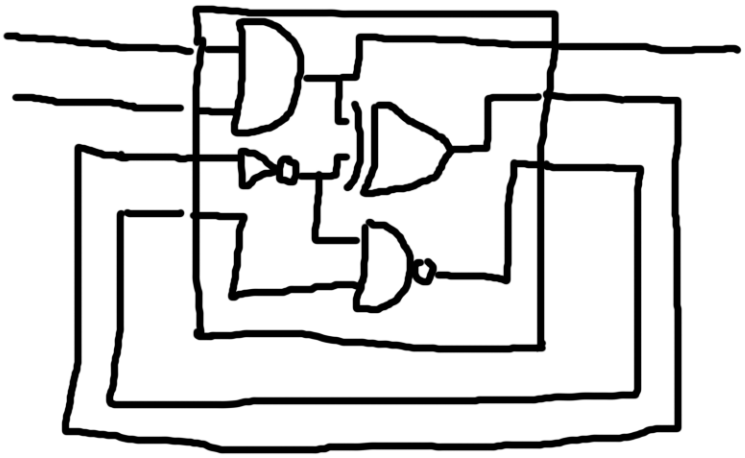


Slika 1.1:

Problem kombinatoričnih vezji je **skaliranje**. Ko raste število vhodov in

izhodov rastejo tudi površina, ki jo vezje porabi in čas, ki ga vezje porabi, da izračuna rezultat. Površina raste proporcionalno številu vhodov. Zakasnitev raste proporcionalno kvadratu številu vhodov. Iz teh razlogov, želimo omejiti velikost posameznih kombinatoričnih vezji.

Rešitev je, da uporabimo iterativne algoritme. Torej obdelujemo podatke v manjših kosih. To omeji velikost potrebnih kombinatoričnih vezji in dovoli večkratno uporabo istih kombinatoričnih vezji. To zahteva, da izhod kombinatoričnega vezja povežemo nazaj na vhod oz., da naredimo povratno vezavo. Izhod takih vezji ni več odvisen le od vhodov, temveč tudi od stanja povratne zanke, torej imajo taka vezja spomin, imenujejo se **sekvenčna vezja**.



Slika 1.2:

Ampak tu se pojavi nova težava. Predstavljajmo si seštevalnik, katerega izhod povežemo na enega izmed vhodov, na drugi vhod postavimo konstanto 1. Želeli bi si, da vezje šteje 1,2,3... Ampak to se ne zgodi, namesto tega na izhodu dobimo kaotične signale. Razlog je, da je v vezju ogromno število povratnih zank, vendar med seboj niso sinhronizirane. Vsaka povratna zanka ima rahlo drugačno zakasnitev, zato zanke počasi zlezejo iz faze. Rezultat je, da vezje ni stabilno in ne opravlja želene naloge.

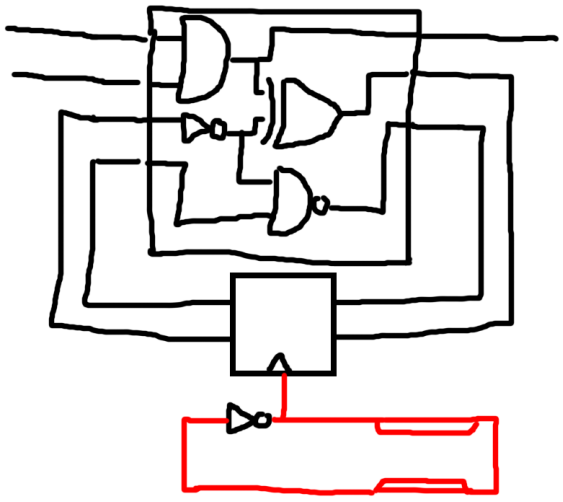
Nujno je torej vpeljati mehanizem s katerim periodično sinhroniziramo vse

povratne zanke. To storimo s spominskimi elementi, ki jih vstavimo v povratno vezavo, imenujemo jih **registri**. Registri shranijo podatke na svojem vhodu in jih oddajo na izhodu. Njihova naloga je, da ne prepustijo naprej novih podatkov, dokler niso stari podatki ponovno obdelani. Efektivno ponovno sinhronizirajo faze vseh povratnih zank.

Torej moramo izvedeti kdaj so podatki na vhodu registra obdelani. Takrat lahko register sprejme nove podatke. Podatki so obdelani, ko se spropagirajo skozi **najdaljšo** izmed vseh povratnih zank.

V splošnem imamo dva načina na katera lahko storimo:

- Globalna sinhronizacija(Taktni signal): V tej shemi uporabimo predpostavko, da obdelava podatkov v **vseh** vezjih med dvema zaporednima registroma ne traja dlje od neke konstante. Vse registre nato naenkrat sprožimo z to periodo. To storimo s taktnim signalom. [1]

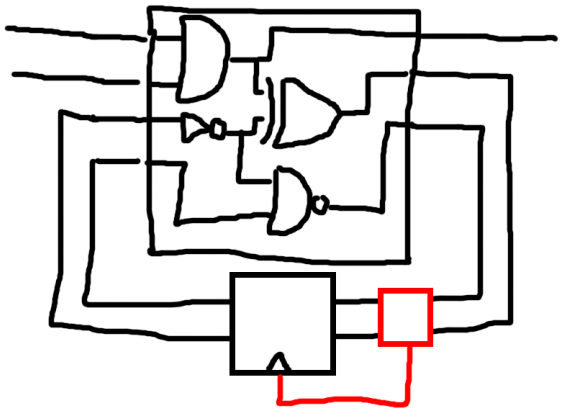


Slika 1.3:

- Lokalna sinhronizacija: V tej shemi sinhroniziramo vse povratne zanke na

[1]Taktni signal je ustvarjen prek kristalnega oscilatorja ali RC oscilatorja. Vse povratne zanke v vezju nato sinhronizirano z tem oscilatorjem, ki mora imeti najdaljšo periodo.

najdaljšo izmed povratnih zank. Vezje moramo zasnovati tako, da je ta zanka vedno najdaljša, neodvisno od variacij v vezju in vhodnih podatkih.



Slika 1.4:

2 Teoretične osnove

Da bomo lahko asinhrona vezja analizirali in zasnovali potrebujemo povzeti teoretične osnove s katerimi taka vezja modeliramo. Moramo se naučiti kako v takih vezjih signali potujejo in kako dosežemo, da potujejo urejeno.

2.1 Asinhroni koncepti

V sinhronih vezjih taktni signal določa časovne točke, kjer so signali v vezju veljavni. Ker v asinhronih vezjih nimamo taktnega signala, morajo biti vsi signali vedno veljavni. Zato proučujemo naša vezja na nivoju sprememb. Sprememba je prehod nekega signala iz stanja 1 v stanje 0 ali obratno. Potovanje sprememb skozi naše vezje določa njegovo delovanje. Da asinhron sistem deluje pravilno mora slediti naslednjim pravilom.

- Spremembe se ne pojavljajo iz nikoder.
- Spremembe ne izginjajo.

Da se spremembe ne smejo zgoditi brez razloga je jasno nujno, saj sicer vrednost podatkov ni nujno pravilna. Obstajajo vezja, ki delujejo v ekstremnih pogojih npr. v vesolju, kjer lahko šum inducira spremembe napetostnih nivojev v vezju. Taka vezja potrebujejo redundantne kopije, da zagotovijo pravilnost izračunov.

Konceptu, da spremembe ne smejo izginjati pravimo **indikacija**.

2.1.1 Indikacija

Indikacija je pomemben koncept pri asinhronih vezjih. Signal je indiciran, če ima vsaka njegova sprememba posledice. Prestavljajmo si ALI logična vrata. Recimo, da je stanje na vhodu

$$[0, 0] \mapsto 0$$

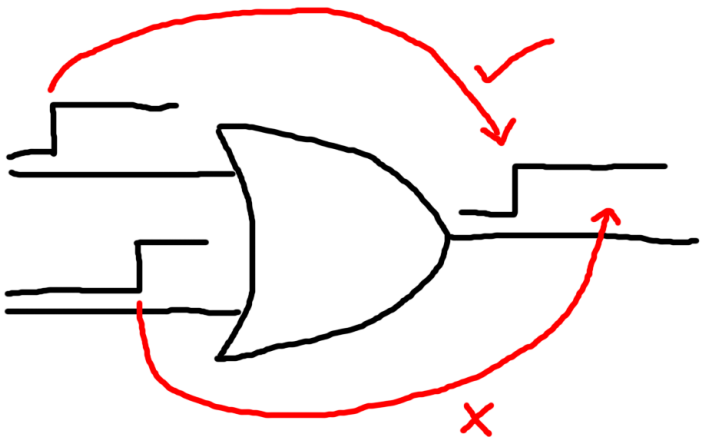
Če en signal preklopi, se spremeni stanje izhoda, torej ima sprememba posledice.

$$[0, 1] \mapsto 1$$

Če sedaj preklopi še drugi signal, se stanje na izhodu ne spremeni .

$$[1, 1] \mapsto 1$$

To pomeni, da ne vemo če se je drugi signal že spropagiral od vira. Po drugi strani, če originalni signal preklopi nazaj, se izhod ponovno spremeni. Če ne želimo uporabljati časovnih predpostavk, mora vsak preklop vsakega signala biti indiciran. To pomeni, da mora na vhodu ALI in IN vrat signal vedno preklopiti dvakrat zaporedno, na vhodu C elementa morata signala preklopiti eden za drugim itd.



Slika 2.1:

2.1.2 Hitrostna neodvisnost

Naredimo model asinhronih vezji, da bolje razumemo kako delujejo. Predstavljajmo si vezje vrat, povezanih med seboj. Vsaka vrata imajo vhodne in izhodne signale. Ko vhod vrat diktira spremembo izhoda postanejo vrata aktivna. Po neznanem časovnem zamiku se spremeni tudi izhod. Zanimivo je, kar se zgodi na izhodu vrat. Vrata ki imajo na vhodu izhodni signal naše originalnih vrat imajo tri opcije. Lahko sprememba vhoda ne spremeni izhoda. Lahko sprememba vhoda ne povzroči spremembe izhoda. Lahko pa so bila vrata aktivna, a zaradi spremembe izhoda sedaj niso več aktivna.

Asinhrona vezja delujejo pravilno, če se zadnji kriterij nikoli ne zgodi.

Obstajajo različni razredi asinhronih vezji, kjer je ta kriterij realiziran z različnimi predpostavkami.

2.1.3 Hitrostna neodvisnost

Hitrostno neodvisna vezja imajo neznane zakasnitve v vratih in nimajo zakasnitev v žicah. To žal ni zelo realno v današnjih vezjih, kjer so zakasnitve v žicah lahko velik del zakasnitve v logičnih vejah. Še posebej pa to ne velja v FPGA, zato ta vezja niso zelo zanesljiva.

2.1.4 Zakasnilna neodvisnost

Vezja, ki so neodvisna na zakasnitve imajo neznane zakasnitve v žicah in vratih, tu ne predpostavimo nič, ampak S takimi vezji ne moremo procesirati podatkov. Citat

2.1.5 Kvazi-zakasnilna neodvisnost

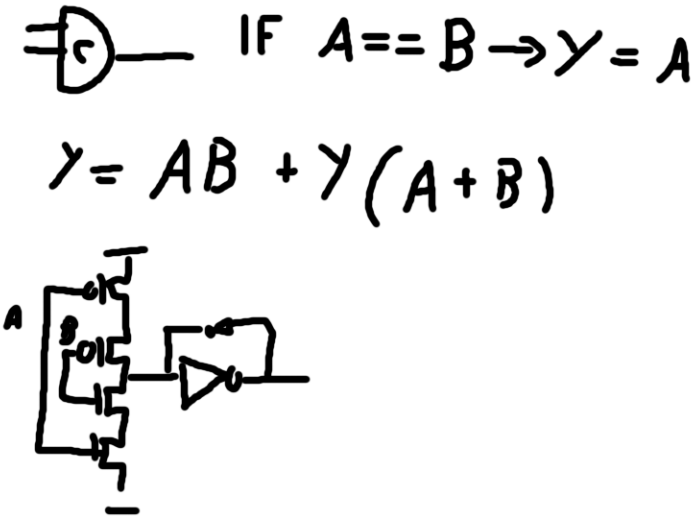
Srednja pot je kvazi zakasnitvena neodvisnost, kjer imamo določene predpostavke, o zakasnitvah v le določenih žicah.

2.2 Sinhronizacija

Sinhronizacija je pojav, kjer dva procesa delita informacije, da delita neko vrednost. V našem primeru je ta vrednost perioda.

2.2.1 Muller C element

Muller C element je osnovni element sinhronizacije dveh signalov. Ima dva vhoda, izhod se spremeni, ko sta oba vhoda enaka.



Slika 2.2:

2.3 Združevanje

Ko združimo poti dveh signalov, na izhodu dobimo izhod, če je na vhodu katerikoli od vhodnih signalov. Je ALI funkcija za signale.

2.4 Multiplekser

Izbira en signal šofira po dveh poteh. Izbiro poti lahko nadzira zunanji signal. To je multiplekser signalov

2.5 Razcepljanje

Razcepljanje je preprosta duplikacija izhodnih signalov.

2.6 Osnovna vezja

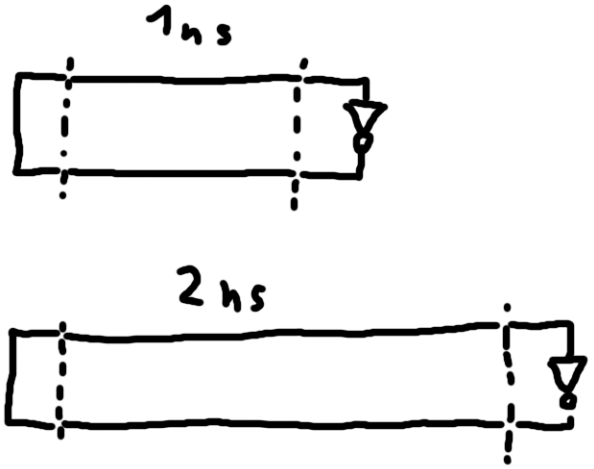
Najosnovnejše vezje, ki ima periodo je obročni oscilator. Obročni oscilator je narejen iz inverterja, ki povzroči negativno povratno zanko in zakasnilne linije, ki določa periodo oscilatorja.

Začnimo torej z sinhronizacijo dveh obročnih oscilatorjev. Za ta namen potrebujemo posebna logična vrata.

Poglejmo si kako tak element sinhronizira dva signala:

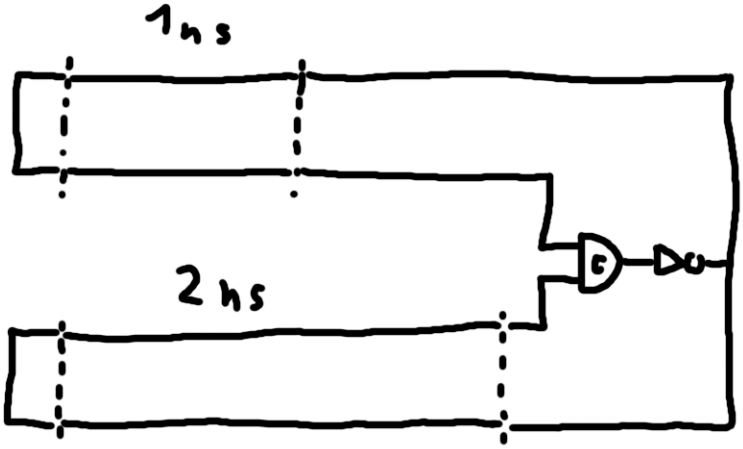
2.6.1 Vzporedna sinhronizacija

Imamo dve povratni zanki, vsaka z različno zakasnitvjo, kateri želimo sinhronizirati.



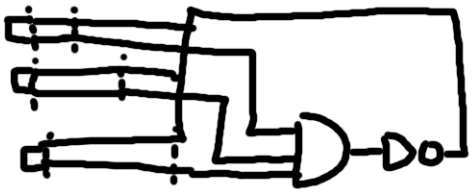
Slika 2.3:

Če ustvarimo prek C elementa skupno povratno zanko, mora hirejši vedno čakati počasnejšega, torej sta sinhronizirana.



Slika 2.4:

Trivialno lahko sinhroniziramo poljubno število povratnih zank.

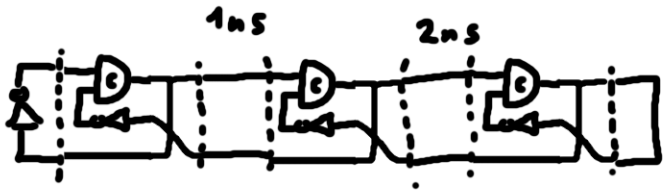


Slika 2.5:

To lahko razumemo kot vzporedno vezavo oscilatorjev, kjer je skupna perioda, perioda najdaljše povratne zanke.

2.6.2 Zaporedna sinhronizacija

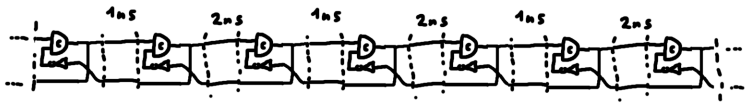
Lahko pa vežemo oscilatorje tudi zaporedno na sledeči način:



Slika 2.6:

Tu povratne zanke prepletemo med seboj. Vsako povratno zanko sprožijo njeni sosedje. V tem primeru prva zanka sproži drugo. Ko druga konča ponovno sporži prvo.

Tak vzorec lahko nadaljujemo poljubno dolgo.



Slika 2.7:

3 Zasnova asinhronih vezji

Če želimo zasnovati asinhrona vezja moramo zasnovati visoko nivojski pogled, podobno kot pri sinhronih vezjih. Pri sinhronih vezjih je osnovna dogma to, da imamo zaporedje registrov in logike med njimi. Na vsak pozitivni rob urinega takta, se podatki prenesejo za en register naprej. To nam dovoli, da poenostavimo zasnovo sinhronih digitalnih vezji na zaporedje matematičnih operacij.

Podobno lahko storimo tudi pri asinhronih vezjih, imamo dodatne omejitve. Še vedno imamo zaporedje registrov, ter logiko med njimi, vendar se sedaj podatki ne pretakajo periodično, zato moramo zagotoviti da ne pride do zastojev.

3.1 Protokoli

Najprej poglejmo kako podatke prenašamo skozi asinhrona vezja. Na sploh se protokoli deljio v dve skupini po dve skupini.

3.1.1 Faznost

- 2phase hitrejsi, ampak rabimo feedback da vemo kdaj smo done
- 4phase preprostejši več komunikacije

Pomislimo na mullerjev cevovod. Prenos podatkov po njem lahko gledamo na dva načina. Lahko gledamo na nivoje signalov. Cikel torej gre reqdata, ackdata, reqnull, acknull. V tem pogledu potujeta skozi cevovod dva valova. Prvi je po-

datkovni val, ki prenaša podatke. Drugi je zaključni val, ki zakjuči komunikacijo, in spravi kanal v prvotno stanje.

Lahko gledamo robove signalov. Torej gledamo prenos dogodkov (pozitivni in negativni rob imata enak pomen). Torej gre samo za podatkovne valove, ki sledijo eden drugemu.

3.1.2 Podatki

- bundled data: Podoben sinhronim vezjem, potrebuje zakasnitve, da zagotovimo časovne predpostavke.
- Dual rail: Dve žici na bit, manj predpostavk

Do sedaj smo gledali cevovod le kot 1 biten podatkovni kanal. Vendar za procesiranje podatkov želimo več-bitne podatke. Cevovod lahko posplošimo na dva načina.

Lahko signale, ki potujejo po cevovodu uporabimo, za odpiranje klasičnih registrov. To pomeni da lahko uporabimo klasične registre in klasično logiko. Vendar moramo paziti, da ohranimo pot skozi cevovod najdaljšo, torej potrebujemo zakasnilne elemente.

Lahko sklopimo več cevovodov skupaj. Med seboj moramo povezati povratne zanke in poskrbeti, za pravo kodiranje podatkov.

3.1.2.1 Kodiranje

Kot vemo, moramo poskrbeti, da povratno zanko sinhroniziramo na najpočasnejši signal v zanki. To lahko zagotovimo z pravilnim kodiranjem podatkov. Koda je neodvisna od zakasnitev ko nobena kodna beseda ni vključena v drugi kodni besedi. Želimo tudi da je konkatanacija dveh kodnih besed nova kodna beseda.

Imamo dva jasna kandidata

- One hot quad.
- Dual rail: One hot dual

One hot quad ima enako podatkovno gostoto kot dual rail, vendar ima pol manj preklopov, kar jo naredi bolj energisjko učinkovito. Vendar nismo še našli učinkovitih impelmentacij. Zato uporabimo Dual rail.

Glede na gornje izbire dobimo 4 načine implementacije asinhronih vezji. Poglejmo si osnove vseh ter pogledjmo prednosti in slabosti.

3.1.3 4-Phase bundled data

Ta stil je najbolj podoben klasični sinhroni logiki. Uporablja Mulerjev cevovod kot osnovno vodilo podatkovnega poteka. Na ta cevovod pripnemo navadne zapahe skozi katere pretekajo podatki. Med zapahe lahko vstavimo klasično logiko, a moramo paziti, da vsatvimo v Mullerjev cevovod zadostne zakasnitve, da zagotovimo dovolj časa, da se signali propagirajo skozi logiko. Ker je protokol

3.1.4 2-Phase bundled data

Ta stil je podoben 4-Phase bundled data, saj enako uporablja Mullerjev cevovod kot vodilo podatkov in klasično logiko. Ker uporablja 2 fazni protokol predstavljajo preklopi na cevovodu podatke o stanju cevovoda. Zato potrebujemo posebne zapahe, ki so odprti, ko sta stanja trenutne in naslednje stopnje cevovoda različni, sicer pa je zaprt. Tak protokol je hitrejši in bolj energetsko učikovit, prednost je tudi, da ne potrebujemo asimetričnih zakasnitev. Slabost so tudi posebni zapahi, ki niso del večine knjižnjic.

3.1.5 4-Phase dual rail

Ta stil uporablja dual rail komunikacijo, kar prinese veliko overhead saj moramo uporabiti dvakrat več linij za podatkovne povezave in moramo vgraditi spominske

celice v samo kombinatorično logiko. Druga stran je izjemna zanesljivost protokola, ki je dosežena popolnoma brez zakasnitev. Edina časovna predpostavka so nekatere isohronične veje, ki so zelo lahko izpolnjene

3.1.6 2-Phase dual rail

Ta magisterska naloga se nanaša na implementacijo tega stila komunikacije. Je podoben 4-Phase dual rail, ampak namesto uporablja 2 fazni protokol, torej pridobimo na hitrosti in energijski učinkovitosti. Slabost je, da je površina takšne izvedbe daleč največja. V nadaljnih poglavjih, se fokusiramo na prednosti in slabosti tega stila, ter njegovi detajli.

3.2 Cevovodi

Cevovodi so osnovni arhitekturni element podatkovnega prenosa. Sestavljeni so iz zaporedja spominskih elementov, ki posredujejo podatke iz prejšnjega v naslednjega.

Glavno pravilo cevovodov je sledeče: Register lahko shrani nov podatek od svojega prednjika, če je njegov naslednjik shranil podatek, ki ga trenutno vsebuje.

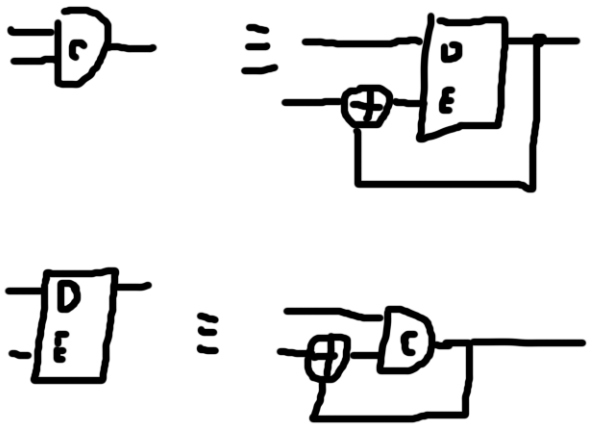
Osnovna implementacija enobitnega cevovoda v asinhronih vezjih je seveda Mullerjev cevovod. Če želimo cevovod posplošiti na več bitov lahko signalizacijo preprosto uporabimo za odpiranje navadnih registrov v bundled data stilu.

Poglejmo si bolj detajlno posplošitev za dual rail sisteme.

Po Mullerjevem ceovodu, potujejo dogodki, ki pa nimajo vrednosti. Poglejmo si, kako poslati eno bitno vrednost.

Za to uporabimo dva vzporedna Mullerjeva cevovoda. Po enem bo tekel podatek za vrednost 0, po drugem podatek za vrednost 1. Povratni zanki obeh sihnalov moramo združiti, saj dobimo vedno signal le po eni izmed vej.

Implementacija 4fazne signalizacije, je veliko bolj preprosta kot 2fazne. Če



Slika 3.1:

uporabimo nadaljno transformacijo, lahko vezje poenstavimo. Izhodno vezje, je večbitna verzija Mousetrap vezja.Cite Mousetrap.

Poglejmo si detejlno delovanje vezja. Sprememba pride po enem izmed cevovodov. Nato se sprememba propagira naprej, če je prostor in povratna informacija je poslana nazaj. V tej celotni verigi, Časovni zamiki niso važni za delovanje vezja. Vezje deluje pravilno po sami zasnovi (Seveda moramo biti pozorni na same časovne zahteve osnovnih elementov).

Tu spet vidimo veliko slabost dual rail, saj je potrebno več kot 2x več logike, za navaden cevovod.

Če želimo poslati večbiten podatek, uporabimo več enobitnih cevovodov. Povratne zanke skupaj vežemo s C elemnti, saj moramo tokrat sinhronizirati povratne zanke vseh enobitnih cevovodov.

3.3 Logika

Logika je osnovni element podatkovnega procesiranja. Kot vhod ima dva ali več podatkov, kot izhod pa en podatek. Kot vemo lahko vsa logična vezja implementiramo z NAND vrati, zato si poglejmo izdelavo NAND logičnih vrat.

V bundled data stilu, so NAND vrata implemntirana v klasičnem načinu npr CMOS. Poglejmo si implementacijo v dual rail stilu.

Osnovna ideja je podobna, kot implementacija klasičnih kombinatornih vezij z ALI/IN matriko. Najprej imamo detektor za vsako možno vhodno kombinacijo(IN), nato izhode teh detektorjev kombiniramo(ALI) skladno z našo pravilnostno tabelo.

Pri 4 phase stilu to implementiramo z uporabo C elementov za detektorje vhodne kombinacije in ali vrat za kombinacijo teh izhodov. Tu se je pomembno zavedati, da niso vse možne vhodne kombinacije veljavne in zato zaznavamo le kombinacije, ki so veljavne.

Pri 2-faznem stilu uporabimo podobno implementacijo, vendar moramo vhode C elementov sami ponastaviti, da ne ostanejo nekateri detektorji le delno aktivirani. To storimo z XOR vrati, ki jih postavimo za vhode. Ker vemo točno katera kombinacija vhodov je bila detektirana, lahko ponastavimo le tiste vhode, ki so bili aktivirani.

Za 4 fazna vezja obstaja tudi način optimizacije logičnih vezji citeNCL. V NCL logiki gledamo na C elemente in ALI vrata kot poseben primer N od M prestopnih vrat. S tem posplošenim naborom vrat lahko načrtamo kombinacijska vezja z veliko manjšim številom tranzistorjev oziroma v našem primeru, manjšim številom celic.

3.4 Podatkovni potek

Premikanje podatkov povezju je podatkovni potek. Za obravnavo podatkovnega poteka, nas zanimajo le registri, ter povezave med njimi, saj so logični elementi

prozorni (podatke obdelajo, a ne vplivajo na njihovo pot skozi vezje).

V sekvenčnih vezjih se podatki premikajo iz enega zapaha v naslednjega. Footnote: Registri so narejeni iz para zapahov. Lahko uporabimo tudi samo zapahe in jih prožimo z alternirajočo fazo ure. Iz tega je razvidno, da za sinhrona vezja potrebijemo v ciklu vedno sodo število zapahov. Za pravilno delovanje mora vezje zagotavljati naslednjim zahtevam:

- Podatki ne izginjajo
- Podatki se ne pojavljajo iz nikoder
- Zaporedje podatkov je konsistentno.

Podatkovni pretok je pri asinhronih vezjih podoben kot pri sinhronih vezjih vendar obstajajo pomembne razlike. V sinhronih vezjih so v registrih vedno veljavni podatki in lahko se zanašamo na hitrost pretoka podatkov po vezju. Vsi podatki se ob urinem ciklu premaknejo za en register naprej. V asinhronih vezjih potujejo podatki kot kot valovi. V eno smer potujejo valovi podatkov, v nasprotno smer pa potujejo valovi potrdil, ki signalizirajo sprembo podatkov.

Število teh valov v vezju je pomembno, saj niso vse konfiguracije ustrezne. Če so na primer vsi zapahi zasedeni z podatki, se ne more noben podatek premakniti, kar ustavi celotno vezje. Če ni nobenega podatka, vezje jasno ne more delovati. Poglejmo si pravila ki jim vezja morajo slediti in pravila po katerih se podatki obnašajo v vezjih.

3.4.1 Gradniki

Logični potek vezji gradimo iz majhne skupine gradnikov, ki zagotavljajo možnost zasnove velike večine vezji.

3.4.1.1 Registri, izvori in ponori

Registri so glavni del vezja, ki diktirajo potek podatkov. Linearnem zaporedju registrov rečemo cevovod. Cevovodi služijo obdelavi podatkov v večih stopnjah tako, da med njih damo logične bloke, ali pomnilniku, da naredimo pretok podatkov skozi vezje bolj fleksibilno. Zaporedju registrov, kjer je zadnji register v zaporedju povezan z prvim pravimo obroč. Take strukture so uporabne za iterativne izračune.

Izvori so viri podatkov. Ko dobijo potrdiev generirajo podatek. Ponori so nasprotje, vhodne podatke zavžejo in oddajo potrditev. Izvori in ponori pogosto prestavljajo mejo med vezjem in okoljem a ne vedno. Izvori predstavljajo tudi npr. konstante.

3.4.1.2 Podvojitve in sinhronizacija

Podvojitve so deli vezja, kjer je podatek dupliciran in posredovan dvema registroma. Potrditve slednjih registrov sta združeni, torej potrebujemo potrditev obeh nadaljnih pretokov. Sinhraonizacija je nasprotna operacija, kjer se dva podatka združita v enega. Tu čakamo, da so prisotni vsi vhodni podatki, preden nadaljujemo. Tu so ponavadi logični bloki.

3.4.1.3 Razcepi in sponi

Razcepi so del vezji, kjer en podatkovni tok narekuje pot drugemu podatkovnemu toku. Je podoben multiplekserju, vendar multiplekser odda na vseh izhodih podatke (na ne izbranih izhodih je podatek 0) medtem, ko razcep posreduje podatek na le en izhod.

Spon je obratna operacija, kjer se dva pretoka podatkov združita. Tu ne čakamo vseh virov, vsak podatek preprosto posredujemo naprej.

3.4.2 2phase

V dvofaznih protokolih, se v vezju pretakajo le podatkov. Vsak cikel mora imeti vsakj dva zapaha. Enega za podatkovni val in drugega, v katerega se podatkovni val premakne.

3.4.3 4phase

Štirifazni protokoli so sicer identični dvofaznim, z razliko, da se po vezju pretakajo pari podatkov in ponastavitev. Iz tega razloga potrebuje vsak cikel vsaj 3 registre. Enega za podatek, enega za ponastavitev in enega, ki je prazen, da se vanj premakne podatek oz. ponastavitev.

3.4.4 Inicializacija

Inicializacija vezja je vredna bližjega pogleda. Če v inicaliziranem vezju ni podatkovnih valov lahko vezje preprosto inicializiramo tako, da vse spominske elemente postavimo na 0.

Če pa imamo v inicaliziranem vezju podatkovne valove, moramo poskrbeti za pravilno inicializacijo zapahov in povratnih zank. TODO add specifics maybe or cut

4 Impementacija asinhronih vezji

Implementacija vezji je proces pretvarjanja načrta za vezje v dejansko vezje, ki obdeluje vhodne podatke in oddaja izhodne podatke. Obstajajo različni načini implementacije digitalnih vezji

- Simulacija
- FPGA
- Integrirana vezja

4.0.1 Simulacija

Simulacija je najpreprostejši način implementacije. Slaba stran simulacije je, da niso upoštevane časovne lastnosti realnih implementacij, kar je v primeru asinhronih vezji zelo pomembne. Slabost je tudi dejstvo, da simulirano vezje ni uporabno za samo obdelavo podatkov.

4.0.2 Integrirana vezja

Integrirana vezja so končna faza implementacije elektronskih vezji. Pri izdelavi integriranih vezji imamo največ nadzora, zato lahko zagotovimo nekaterim časovnim predpostavkam, ki jim sicer ne moramo. Slaba stran je visoka cena izdelava.

4.0.3 FPGA

Implementacija v FPGA je privlačna, saj nima visoke cene in hkrati dobimo dejansko vezje na katerem lahko pomerimo časovne in ostale karakteristike. Slabosti FPGA implementacije pa je, da nimamo preciznega nadzora nad časovnimi karakteristikami in primitivi, ki so nam na voljo.

Za to magistersko nalogo smo se odločili za implementacijo v vezju FPGA, saj predstavlja srečno srednjo pot.

4.1 Arhitektura FPGA

FPGA so vezja za implementacijo specializirane logike.

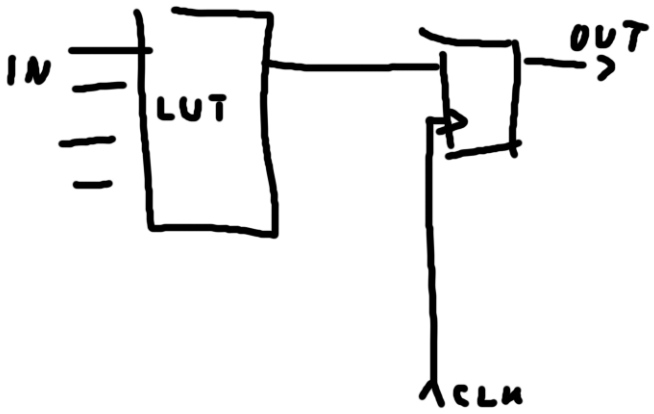
Sestavljena so iz množice logičnih celic, ki jih povezuje matrika programibilnih povezav.

4.1.1 Logične celice

Logične celice so kosi ligitalne logike v FPGA vezjih, ki opravljajo funkcijo obdelave podatkov. Pogosto so narejene iz kombinatoričnega in spominskega dela. Kombinatorični del je zadolžen za samo obdelvo podatkov, spominski del pa za urejen pretok samih podatkov.

4.1.1.1 Kombinatorični del

Kombinatorični del je implementiran kot iskalne tabele. To so kosi spomina, ki so sprogramirani ob zagonu vezja. V njih je sprogramirana pravilnostna tabela kombinatoričnega vezja.



Slika 4.1:

4.1.1.2 Spominski del

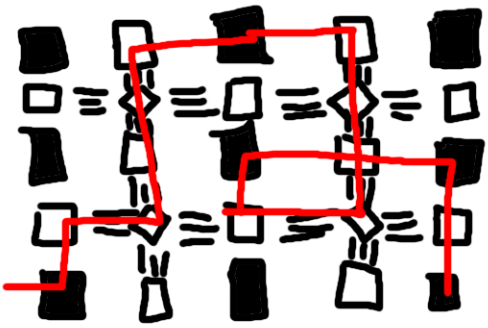
Spominski del je implementiran kot konfigurabilna flip-flop celica, kateri lahko nastavimo polariteto, ali je zapah ali register ima reset in preset itd.

4.1.2 Povezave

Povezave zagotovijo možnost da se katera koli celica poveže z katero koli drugo celico v FPGA vezju. Matrika povezav je narejena iz mreže programibilnih križišč, ki so povezana s svojimi sosedi. Te lahko povežejo arbitrarna križišča med seboj. Na vsako križišče je povezana tudi logična celica. Taka arhitektura nam dovoli izjemno fleksibilnost a ne pride brez slabosti. Ker so povezave programibilne, pomeni, da imajo relativno veliko kapacitivnost, kar upočasni hitrost vezja.

4.1.3 Vgrajeni bloki

Pogosto imajo FPGA vezja tudi dodatne funkcionalnosti, ki pohitrijo pogoste primere uporabe.



Slika 4.2:

- **Prenosne verige**: hitre povezave med logičnimi bloki, uporabljene za ustvarjanje logike z velikim številom vhodov npr. seštevalniki
- **Globalne linije**: Povezave za uro in reset so ponavadi globalne, zato imajo nekatrea FPGA vezja posebne povezave za take signale. Urni signal potrebuje posebno pozornost, saj mora vse celice zadeti z čim manjšo razliko v zakasnitvah.
- **Specializirani bloki**: Pogosti gradniki digitalih vezji, ki zasedejo veliko prostora kot npr. množilniki in spominski bloki so ponavadi vgrajeni direktno v FPGA, da prihranimo logične celice. Nekatra FPGA vezja imajo tudi celotna procesorska jedra direktno vgrajena.

4.1.4 Vhodi in izhodi

Pomemben del FPGA vezji so tudi izhodi in vhodi. Ti so povezani na posebne bloke ki ojačajo signale, tako da so primerni za zunanje okolje. ZAgotovijo tudi sposobnost stanja visoke impedance. Pomembno je tudi, da se signali ob vstopu v vezje sinhronizirajo na notranju urni takt. Večina izhodov in vhodov so navadne digitalne povezave. Nekatera FPGA vezja imajo tudi posebne izhode in vhode za

posebne protokole kot npr. PCI-E, DDR itd.

4.2 Implementacija

Za implentacijo vezji v FPGA je potrebno spisati opis vezja v enem izmed jezikov za opis vezji(HDL). To je ponavadi (system)verilog ali VHDL. Nato programsko orodje prevede ta opis vezja v konkreten načrt tega vezja v FPGA vezju.

Cilj tega magisterskega dela je izdelava asinhronih vezji v FPGA vezjih. Problem je, da so FPGA vezja namenjena izdelavi sinhronih vezji. Velikokrat povezave med logičnimi elementi niso narejene kakor bi pričakovali, ne moramo se zanašati na zakasnitve v vezju. Večina implementacij asinhronih vezji v FPGA zato eksplicitno določi postavitev logičnih elementov in njihovih povezav v vezju. Taksen pristop deluje za manjša vezja vendar, ko vezja postanejo večja postane zelo časovno potraten.

Namesto da ročno diktiramo povezave, da zagotovimo predpostavkam, raje uporabimo asinhroni stil, kjer so predpostavke dovolj šibke, da se lahko zanesemo, da bodo izpolnjene brez dodatnih navodil. Zato se odločimo za dual-rail asinhroni stil.

4.2.1 SystemVerilog

Odločili smo se za implementacijo v jeziku SystemVerilog. Systemverilog je jezik, ki se uporablja za zasnovo in simulacijo digitalnih elektronskih vezji. Uporablja se za zasnovo vezji, ki so implementirana v FPGA.

4.2.2 Knjižnica

Izdelali smo knjižnico v jeziku systemverilog, za izdelavo asinhronih vezji v FPGA. Knjižnica je sestavljena hierarhično

1. Osnovni gradniki

2. Funkcionalne enote

3. Vezja

Implementirali smo knjižnico za 4 fazen in 2 fazen stil. Kljub tem, da imajo elementi indentične vhode in izhode moramo biti pazljivi, saj štirifazna vezja ne podpirajo ciklov krajših od 3 elementov medtem, ko so lahko v dvofaznih vezjih tudi obroči z dvema elementoma.

4.2.3 4-Phase dual rail

Štirifazna vezja implementiramo z NCL logiko. Za zapahe uporabimo mousetrap registre, ker so lepše implementirani v FPGA vezjih.

NCL logika uporablja nivojsko logiko, za izdelavo kombinatoričnih vezji. Za pravilno ponastavitev poskrbi histerezni karakter, ki poskrbi da se ničelna vrednost pravilno propagira skozi vezje.

4.2.3.1 Nivojska logika

TODO

4.2.4 2-Phase dual rail

Dvofazna vezja tudi uporabljajo mousetrap zapahe, vendar za kombinatorična vezja uporablja metodo bližjo naivni logični sintezi

Princip je, da zaznamo vse možne kombinacije vhodnih preklopov. Nato resetiramo te vhode in pošlenmo izhode TODO bolj natančno

4.2.5 Workcraft

Workraft je programsko orodje za delo z grafnimi modeli. Uporabno je za zasnovo in sintezo gradnikov asinhronih vezji, kot tudi za simulacijo arhitekture raznih asinhronih vezji.

4.2.5.1 Sinteza gradnikov

Željeno vezje opišemo z uporabo grafa signalnih preklopov(STG). V tem grafu določimo zaporedje preklopov vhodnih in izhodnih signalov, ki določajo delovanje vezja. Za C element na primer, določimo da preklopu obeh vhodov sledi preklop izhoda. Iz tega grafa orodje sestavi vezje, ki se obnaša tako kot graf, ki ga opisuje.

4.2.5.2 Simulacija arhitekture asinhronih vezji

Za simulacijo vezji uporabimo graf signalnih preklopov(STG) oz. graf podatkovnega poteka(DFS) odvisno od stila implementacije. Za dvofazna vezja, je bolj primeren STG, saj modelira preklope, medtem ko je za štirifazna vezja bolj primeren DFS, saj upošteva ponastavitvani val, ki sledi podatkovnem valu. Te simulacije so uporabne, saj z njimi lahko preverimo, da se podatki po vezju pretakajo kot smo si to zamislisi.

4.2.6 Implementacija cellic

Knjižnica implementira vse potrebne primitive za izdelavo poljubnih asinhronih vezji.

4.2.6.1 Primitivi

Imamo 2 Primitiva. Primitivi so implementirani direktno z FPGA primitivi, da zagotovimo, da delujejo pravilno.

- Muller C element. Klasični element implementiran z pravilnostno tableo.TODO test portable
- Nivojska vrata. Implemetirana direktno z pravilnostno tabelo. TODO test portable

4.2.6.2 Osnovne celice

Osnovne celice so implementirane direktno v SystemVerilog kodi, in so prenosne na kateri koli FPGA. So najnižja stopnja v hierarhiji, zato lahko vsebujejo le druge osnovne celice.

- TOGGLE. Element za dvofazna vezja. Sintetiziran s pomočjo programa workcraft.

4.2.6.3 Funkcionalne celice

Funkcionalne celice so sestavljene iz osnovnih celic in dodatne povezovalne logike. To so celice, ki imajo neko konkretno funkcijo kot npr. zapahi, splošni C elementi itd.

- Registri.

4.2.6.4 Logika

To so končani kombinatorični sklopi, ki jih lahko vmestimo direktno med registre

- AND,OR.
- FULL ADDER.
- ect.

4.2.6.5 Vezja

Končana vezja, ki opravljajo določeno nalogo. Te se razlikujejo med dvofanimi in štirifaznimi implementacijamo

- PIPELINE.
- COUNTER.
- FIBBONACCI.
- GCD.

5 Rezultati

5.1 PIPELINE

Frekvenca, zanesljivost, meritve, ect

5.2 CNT

Frekvenca, zanesljivost, meritve, ect

5.3 FIB

Frekvenca, zanesljivost, meritve, ect

5.4 GCD

Zamik, zanesljivost

5.5 RISCV

Bomo vidl

6 Zaključek

Načrtali smo Asinhrona vezja in jih implementirali v FPGA. Kljub Suboptimalni arhitekturi FPGA smo dosegli delujoče rezultate in ogrodje za nadalnje delo. Hitrosti vezji so nižje kot bi upali.

Literatura

[1] T. Oetiker, H. Partl, I. Hyna in E. Schlegl, *Ne najkrajši uvod v LaTeX 2ε, The not so short introduction to LaTeX 2ε*. Elektronska verzija dostopna na http://www-lp.fmf.uni-lj.si/plestenjak/vaje/latex/lshort.pdf, 2006. Bor Plestenjak, Slovenski prevod in priredba.