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. Dru?ina, 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 frekvenco 35MHz kar je jakoma počasi. Tako pride.

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.

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
LEDR	Level-Encoded Dual-Rail

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

1 Uvod

Digitalna vezja uporabljamo za obdelavo podatkov. Osnovna digitalna vezja glede na vhodne podatke naredijo izhodne podatke, torej nimajo spomina. Taka vezja imenujemo **kombinatorična** vezja. Z kombinatoričnimi vezji lahko izračunamo kar koli želimo a imamo težavo:

Velikost: velikost kombinatoričnega vezja je v splošnem proporcionalna kvadratu števila vhodov in proporcionalna številu izhodov. Ker želimo obdelovati gigabajte podatkov in več je to jasno nesprejemljivo.

Rešitev je, da kose vezja uporabimo večkrat, torej 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.

Ampak tu se pojavi nova težava. Predstavljajmo si seštevalnik, katerega izhod povežemo na enega izmed vhodov, Na drugi vhod damo 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. Zato zanke počasi zlezejo iz faze in na izhodu dobimo šum.

Nujno je torej vpeljati mehanizem s katerim periodično sinhroniziramo povratne zanke. To storimo s spominskimi elementi, ki jih vstavimo v povratno vezavo, rečemo jim **registri**. Registri ne smejo prepustiti naprej novih podatkov, dokler niso stari podatki popolnoma obdelani. Efektivno ponovno sinhronizirajo faze vseh povratnih zank.

Torej moramo izvedeti kdaj so podatki na vhodu registra obdelani, in podatki

trenutno v registru obdelani. Takrat lahko register sprejme nove podatke.

V splošnem imamo dva načina na katera lahko storimo:

-Globalna sinhronizacija(Urin takt): V tej shemi uporabimo predpostavko, da obdelva podatkov v **vseh** vezjih med dvema zaporednima registroma ne traja dlje od neke konstante. Vse registre nato naenktrat sinhroniziramo z to periodo. To naredimo z urinim taktom. Footnote: Urin takt je generiran prek kristalnega oscilatorja ali RC oscilatorja. Vse povratne zanke v vezju nato sinhronizirano z tem oscilatorjem, ki mora imeti najdaljšo periodo.

-Lokalna sinhronizacija V tej shemi sinhroniziramo vse povratne zanke na en signal. Zagotoviti moramo, da ima ta signal **najdaljšo** periodo izmed vseh povratnih zank.

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. Najosnovnejša metoda ureditve signalov je sinhronizacija.

2.1 Sinhronizacija

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

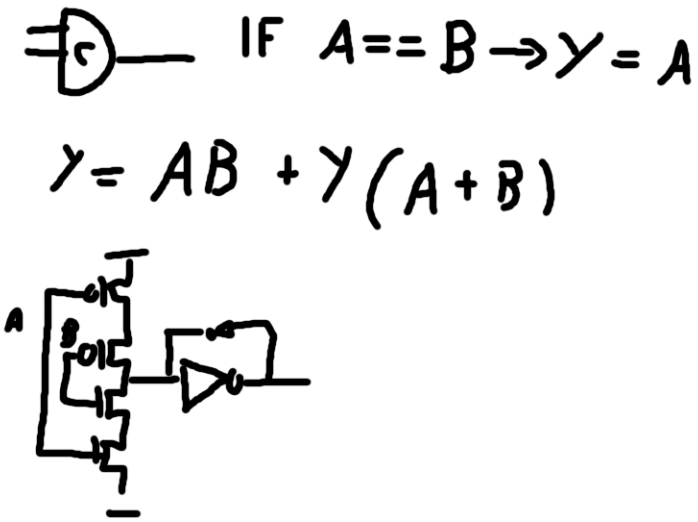
2.1.1 Osnovna vezja

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.

2.1.1.1 Muller C element

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



Slika 2.1:

Poglejmo si kako tak element sinhronizira dva signala:

2.1.1.2 Vzporedna sinhronizacija

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

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

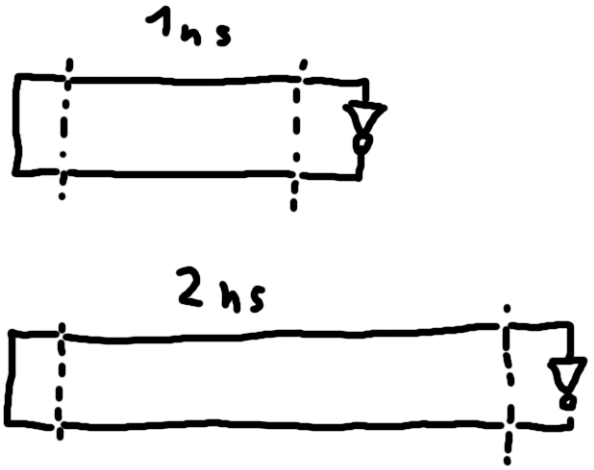
Trivialno lahko sinhroniziramo poljubno število povratnih zank.

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

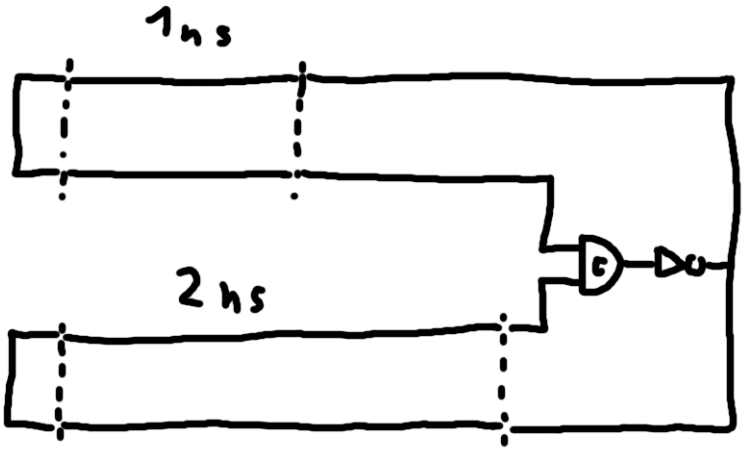
2.1.1.3 Zaporedna sinhronizacija

Lahko pa vežemo oscilatorje tudi zaporedno na sledeči način:

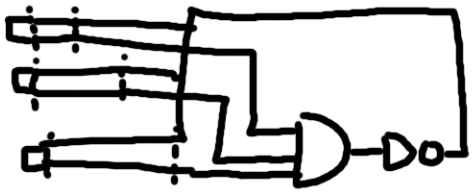
Tu povratne zanke prepletemo med seboj. Vsako povratno zanko sprožijo



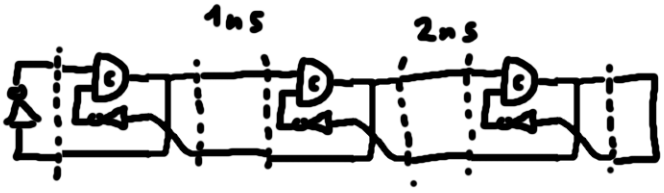
Slika 2.2:



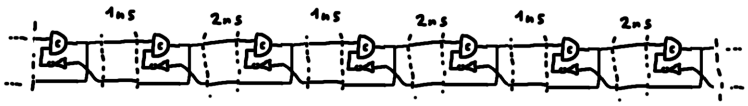
Slika 2.3:



Slika 2.4:



Slika 2.5:



Slika 2.6:

njeni sosedje. V tem primeru prva zanka sproži drugo. Ko druga konča ponovno sporži prvo.

Tak vzorec lahko nadaljujemo poljubno dolgo.

2.2 Teoretične osnove

Poglejmo si sedaj teoretične osnove asinhronih vezji. Naredili bomo kriterijje, katerim naše vezje mora slediti, da bo delovalo zanesljivo.

2.2.1 Indikacija

Indikacija je pomemeben koncept pri asinhronih vezjih. Signal je indiciran, če ima vsaka njegova sprememba posledice. Prestaavljajmo si ALI logična vrata. Recimo, da je stanje na vhodu 0,0. Če en signal preklopi, se spremeni stanje izhoda, torej ima sprememba posledice. Če sedaj preklopi še drugi signal, se stanje na izhodu ne spremeni. 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.

2.2.2 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 izhodini 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.

2.2.3 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 zakasnive v logičnih vejah. Še posebaj pa to ne velja v FPGA, zato ta vezja niso zelo zanesljiva.

2.2.4 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

2.2.5 Kvazi-zakasnilna neodvisnost

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

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.

Poglejmo vse 4 kombinacije

3.1.3 4-Phase dual rail

Karl Fant, dve žici na bit ect. Zelo zanesljivo, zelo velika površina.

3.1.4 4-bundled data

klasična logika, ampak potrebujemo zakasnitve (lahko predvidimo zakasnitve)

3.1.5 2-Phase dual rail

Ta magisterska

3.1.6 2-bundled data

Micropipelines ect.

3.2 Cevovodi

Linija iz knjige: A register may input and store a new data token from its predecessor if its successor has input and stored the data token that the register was pre- viously holding.

Če pogledamo dano vezje lahko vidimo, da deluje kot 1-biten FIFO. Podatki se nalagajo noter in črpajo ven. Temu konstruktu se reče Mulerjev cevovod, in je osnova vseh asinhronih vezji.

Lahko naredimo tudi obroče, Ali več sklenjenih obročov...

Osnovna celica cevovoda je..., če pogledamo logične funkcije: ..., To lahko izvedemo tudi na druge načine.

3.3 Logika

Sedaj imamo cevovode, želimo med stopnjami cevovoda procesirati podatke. Dva načina:

- vzporedno imamo default logiko - Bundlede data
- Vzporedni sklopljeni cevovodi

Vzporedno logika, timing assumptions, delays kako prožit latche, samo pol jih ima naenkrat noter data, isto kot v sinhronih vezjih.

Vzporedni cevovdi, kako sklopit, kako procesirat podatke ect...

3.4 Podatkovni potek

Podatki se pretakajo po asinhronih vezjih kot valovi. v eno smer gre data, v drugo gre ACK. Vedno moramo kontrolirati število podatkovnih valov v vezju ect. To lahko gledamo kot graf po katerem se pretekajo tokeni.

Osnova igre ect:

Drugače za 2 phase kot za 4 phase:

3.4.1 2phase

Najmanj 2 v ciklu poljubno število, inicializacija. Vrsta grafa, za to

3.4.2 4phase

Najmanj 3 v ciklu sodo število, inicializacija. Vrsta grafa, za to

4 Impementacija asinhronih vezji

4.1 Arhitektura FPGA

FPGA so vezja za implementacijo specializirane logike.

Narejena so iz množice spominskih in kombinatoričnih celic. Kombinatorične celice so mali kosi spomina, s katerimi definiramo pravilnostno tabelo naše funkcije. Spominske celice so navadni DFF. Pomembna stvar so povezave med celicami, ki so konfigurabilne. To pomeni, da mi določimo pravilnostne tabele in povezave med njimi.

Obstajajo tudi prenosne verige, reset in clk linije in vgrajeni spomin, množilniki CPU ect.

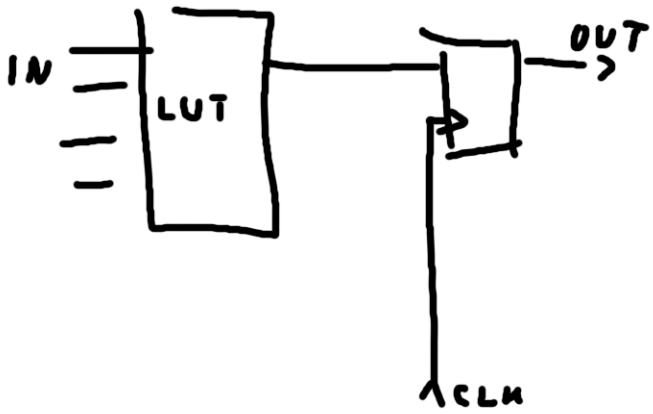
Ima možnost povratne vezave, tako da lahko gledamo kot max preprost turing machine???, na sploh lohk iz teh blokov zelo lahko delamo vezja po kopitu register,logika,register...ect

4.1.1 FPGA Celice

FPGA Celica zgleda pribljižno takole:

4.1.1.1 LUT

Lut je mehn košček spomina. Notr ma par bitou, direk implemenira tabelo



Slika 4.1:

4.1.1.2 FF

FF je preprost FF, ima puno option bitov ect

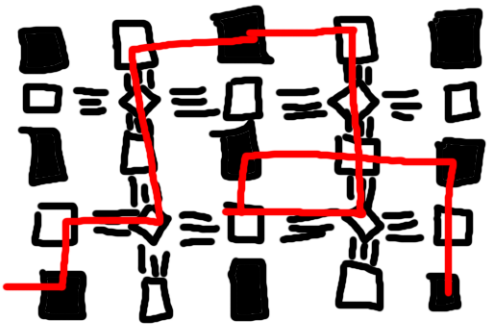
4.1.2 FPGA povezave

Povezave so večino površine, grejo med celicami, so programibilne, torej imajo puno tranzistorjev, torej zakasnitve...

4.2 Implementacija

4.2.1 Verilog

Jezik za opis procesov, uporablja se za programiranje FPGA



Slika 4.2:

4.2.2 Knjižnica

Primitive, funkcionalne enote, sestava

4.2.3 4-Phase dual rail

Za impelementacijo tega protokola potrebujemo posplošene C elemente, Logika je narejena iz spominskih celic, data/ null waves ect

4.2.4 2-Phase dual rail

Podobno, kot pri 4 Phase ampak uporabljamo robove, potrebujemo več tranzistorjev za logiko, saj potrebujemo povratno informacijo, od izhodov

4.2.5 Implementacija cellic

4.2.5.1 Primitivi

Imamo 5 Primitivov, iz katerih sestavimo ostalo funkcionalnost vezja:

C muller C, razne implementacije, rezultati P za TP C splošno, drevesna struktura ect

4.2.5.2 Funkcionalne celice

injektorji pričnejo delovanje cikla cmpl det preveri če je stanje validno adder sestevalnik mem reg spominska celica

4.2.5.3 Vezja

FIB fibbonacci counter, 1,2,3,5,8,13 toy example predstavi inicalizacijo

GCD, ciklična komputacija, preprostejša inicializacija

RISCV čim bolj preprost processor, proof of concept

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.