# Teorija

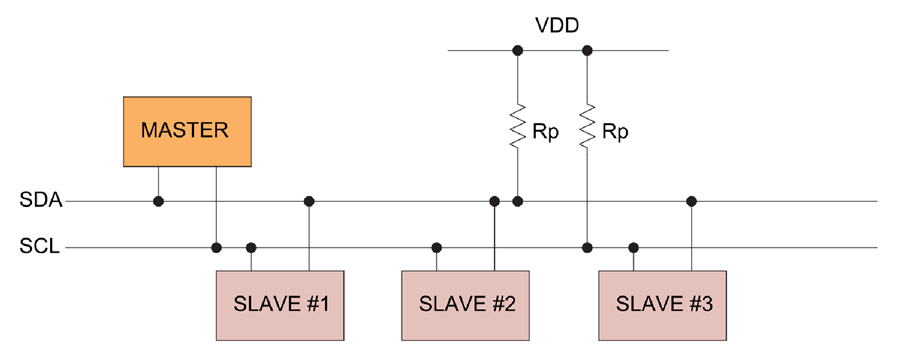
I2C tas ir sinhrona, 8 bitu, secīga saskarne starp integrālam shēmām, kuri ir tuvu viens otram. Tas ir 2 vadu kopne. Ātrumi [100 kbps : 5 Mbps]. Saskarne ir *open drain* veidā, tāpēc *pull up* rezistori ir nepieciešami. *SDA – bidirectional* , *SCL – unidirectional (*no *master* puses*)*, neskaitot *clock stretch* momentu. Stipras puses:

1. 2 vadi
2. *Acknowlegement* bits protokolā
3. Vairāk nekā viens masteris (kursa darba ne tiek ievērots)

Vājas puses :

1. Ātrums
2. *Pull-up* rezistori
3. Protokola sarežģītība (cmp to *SPI, UART*)

Ģenerāla struktūra izskatās sekojoši:

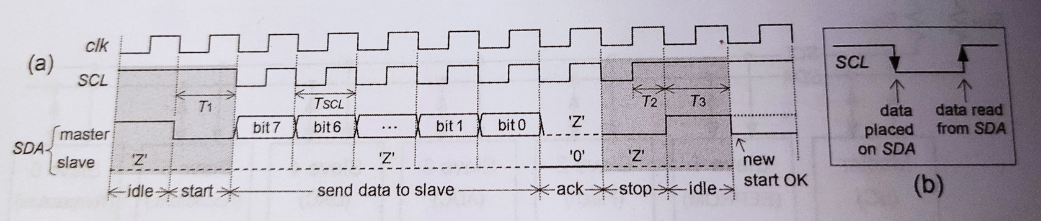


Protokola apraksts:

Dati tiek sūtīti pa 8 bitu grupām. Pēc katras *ack* bits ir vadīts pie ‘0’ no *slave* puses. Datu parsutišana (var but gan viens baits , gan, protams, vairakie) ir starp starta un beigu secībam. Abiem piemit ipasiba ka *SCL* jābut “1” . Tomēr Starta secībai – *SDA* jābūt ‘0’, bet Beigu secībai – *SDA* jābūt “1”. No tam izriet *idle* stāvoklis, kamēr gan *SCL* gan *SDA* ir “1”. Protams no *master* puses.

Par katra baita parsutīšanu. MSB iet pirmais. Kamēr masteris sūt kaut-ko slave ir Z stāvokļi. Pēc 8. bita masteris aizej Z stāvokļī lai slave varētu atsūtit ack bitu ‘0’ (flow control). Lasišanas procedurā masteris atsut ack ‘0’ pēc katra baita. Pēc pedeja baita ir jāatsūt *no-ack* bitu ‘1’. Dati tiks paradīti uz SDA linijas kritošā taktēšanas frontē un tiks lasitas augošā.

Svarīgi ir arī pieminēt par vertībam kuri ir noradīti attēlā , kā *T1, T2, T3.* Tas vertības ir noradītas konkrētas ierices datu lapā un tos vajag respektēt.



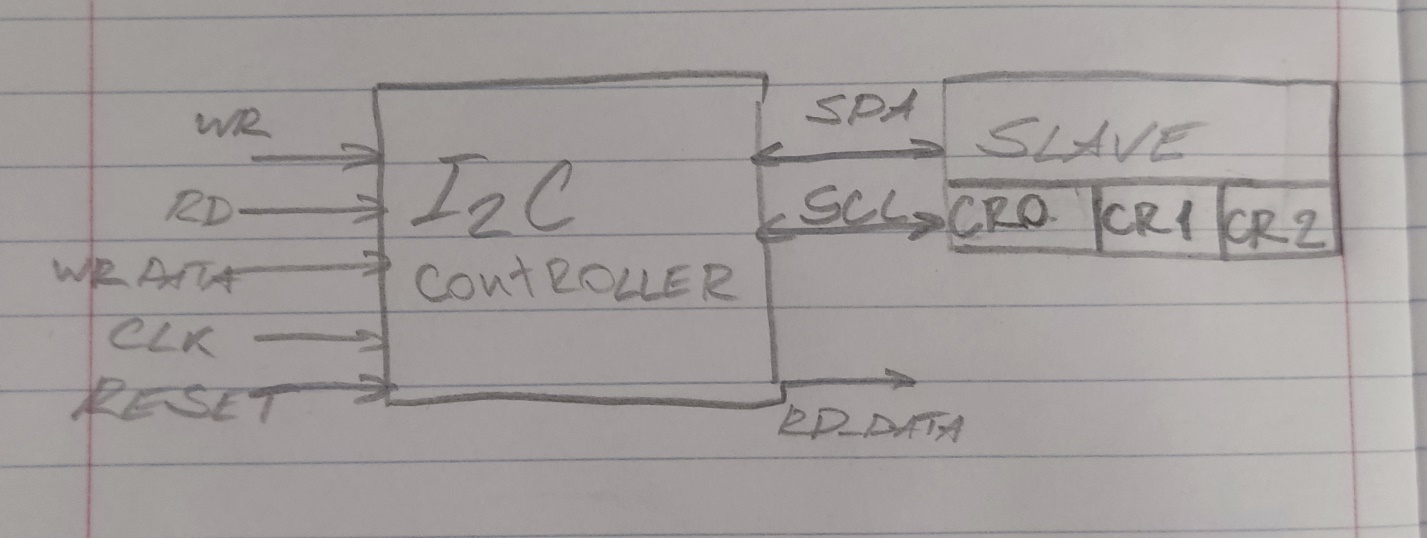
Protokola likumi ir skaidri redzeti uz zimējumā. Protokola likumi zema abs limeni vienmer ir identiski , tomēr augstas abs limena I2C protokola izmantosana varetu atkariba no ierices.

Informācijas daudzums ir pietiekams lai sākt projektēšanas posmu.

# Augstas abstrakcijas plānošana, implementēšanas soļu apraksts un koda piemēri

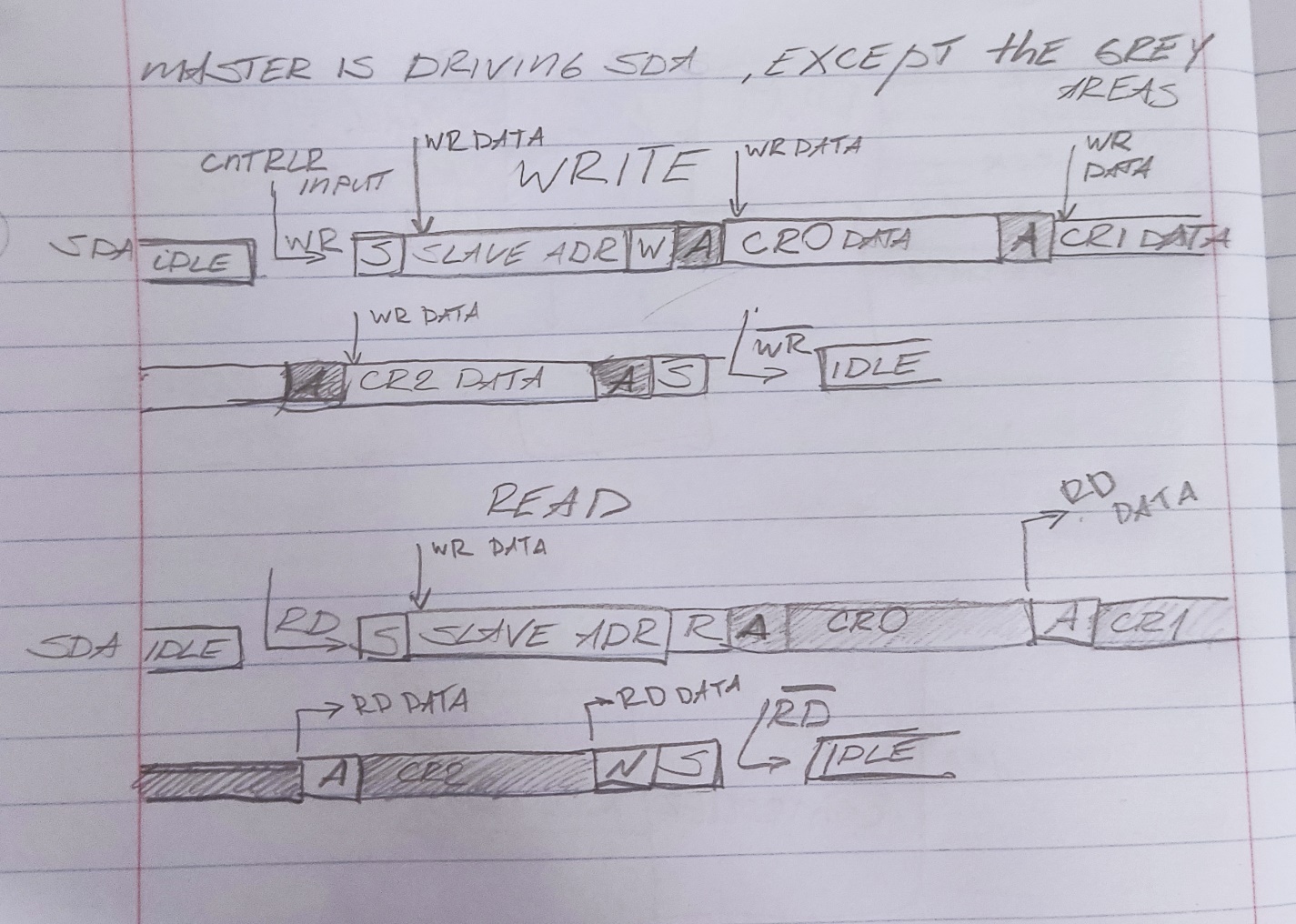
Zinot galvenās interfeisa īpašības ir iespējams sākt plānot sistēmu no augstas abstrakcijas līmeņa un strukturāla skata.

Lai uzsākt projektēšanu no *testbench* apraksta ir jāizdomā vai jāpaņem kaut-kādu reālu ierīci ar reālo adresi un iespējamam konfigurācijām. Izdomāsim abstrakto kanonisko ierīci ar adresi SLAVE\_ADDRES - 0110101 un lai tai ierīcei būs trīs konfigurācijas reģistri – CR0, CR1 un CR2, kurus jāsakonfigurē pēc kārtas, secīgi. Konkretizēsim no *master* puses - uz ierīci var atsūtīt tas reģistru vērtības – paradot uz SDA adresi, pēc tam *write* komandu un pec tam 3 reģistru vērtības. No ierīces var ari nolasīt tas reģistrus – paradot uz SDA adresi un *read* komandu. Tātad I2C kontrollerim jāspēj ierakstīt un nolasīt no ierīces 3 reģistrus pa I2C līnijām. Varam apzīmēt I2c kontrolleri un *emulated slave* no augstas abstrakcijas līmeņa sekojoši



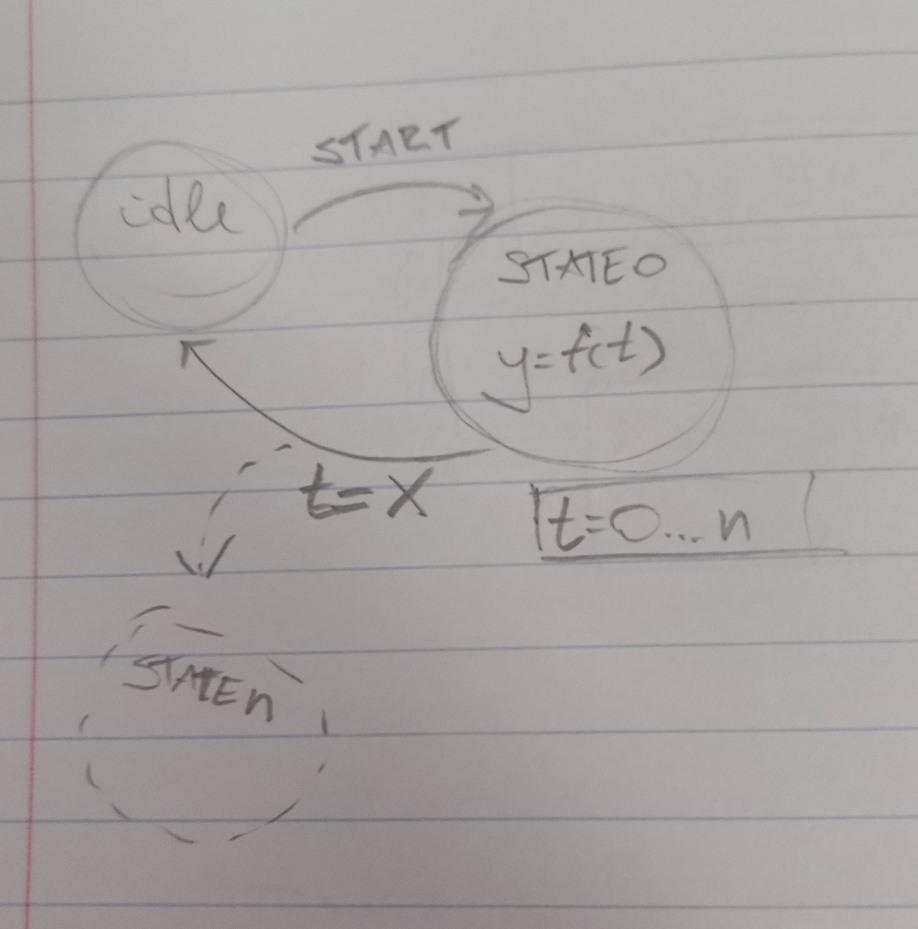
1. Pic. I2C and emulated slave interaction interfaces

Lai labāk saprast *testbench* interfeisa signālus varam shematiski attēlot SDA līnijas stāvokļus atkarība no kontrollera ieejas signāliem

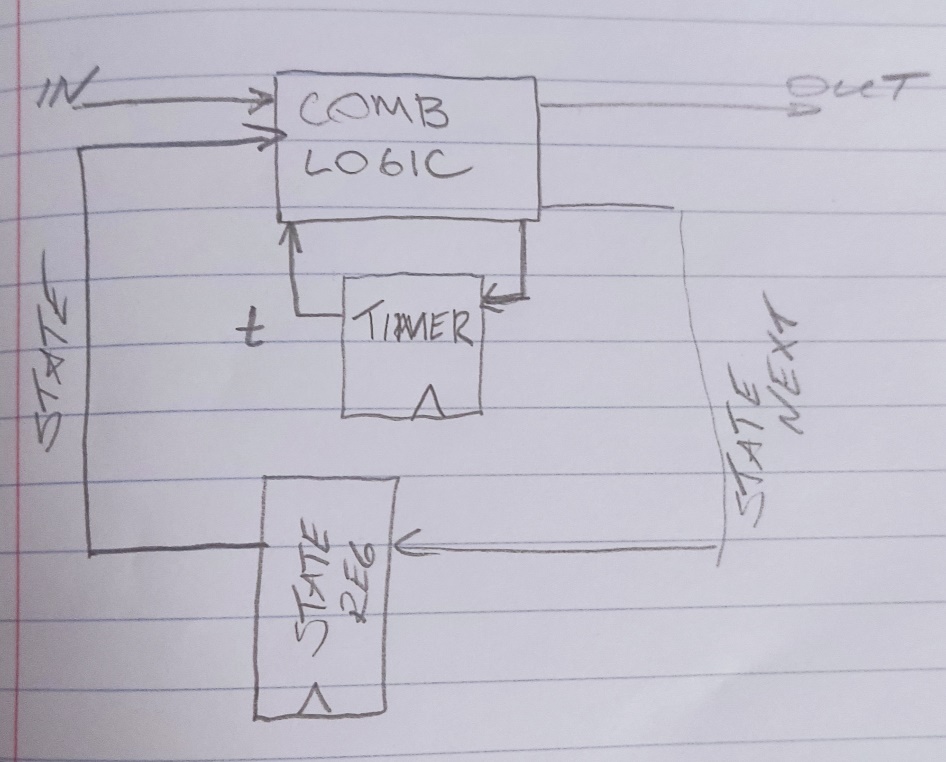


1. SDA linijas stavokļi atkarība no ieejas signāliem

Ņemot vēra to, ka interfeiss ir secīgs, ar atkārtojamo raksturu, un vairākiem līdzīgiem vai vienādiem(bitu pārsūtīšana) posmiem ir ērti uzprojektēt to veidā, kur specials, konkrētai ierīcei domāts *pointer* kontroles SDA un SCL līniju stāvokļus. *Pointer* secīgi aizies starp visam iespējam līnijas stāvokļiem kamēr pārējais VHDL kods nodrošinās *SDA* un *SCL* vajadzīgus stāvokļus. *Pointer* var tikt uzprojektēts ar rekonfigurācijas iespējam , tad viņam būs iespēja kontrolēt *SDA* un *SCL* līnijas jebkurai vai gandrīz jebkurai ierīcei. Šeit autors neaizies ārpus izdomātas ierīces piemēra laika trūkuma dēļ. Tātad, shematiski jau no strukturāla skata *I2C* kontrolleris varētu izskatīties sekojoši:



1. Ideja FSM planošanai ir sekojoša



1. Strukturala ideja ir sekojoša

Izeja ir atkarīga ne tikai no stavokļa bet arī no *Timer* vertības.

1. FSM *pointer* I2C kontrollerim

*Testbench* planošana.

Lai konkrēti definēt I2C kontrollera interfeisu definētos nosacījumos, jāuzraksta *testbench* vīdi. *Testbench* pats pa sevi ir kaut-kāda moduļa interfeisa pārbaude. Ja interfeiss modulim strādā tā, kā domāts var secināt ka iekšēji definēts funkcionāls (gan uzvedības, gan strukturāla skatā) pārbaudāmam modulim ir pareizs. Tāda veida var pārbaudīt jebkādu sistēmu jebkāda abstrakcijas līmeņi.

*Risunok testbencha iz uchebnika*

*Stimuli generator* tiek definēts no plānota kontrollera ieejas interfeisa.

Pirmkārt, *reset* un *clk* signāli būs definēti triviāla veidā.

*screen*

Otrkārt, *Wr* un *Rd* signālu sēcība būs definēta tā, ka no sakuma ir jāieraksta *CR0, CR1, CR2* vērtības, pēc tam jāizlasa tas. Tātad, no sakuma *Wr* ir ‘1’, pēc stingri definēta laika (takts periodu) *Rd* ir ‘1’.

*screen*

Treškārt, ierakstu dati būs masīvs no 3 vērtībām kuras *I2C* kontrollerim jāatsūt uz abstrakto ierīci. Uztaisīsim tos stingri definēta veidā.

*Screent*

Mūsu gadījuma *DUV* ir I2C kontrolleris.

*Template generator* ir daļa kurā tiek saģenerētas sagaidāmas signālu vērtības pēc *DUV.* Mūsu gadījumā tas varētu būt tieši lasīšanas dati.

Komparators salīdzinās lasīšanas datus ar ierakstīšanas datiem, tas darbības rezultāts dos galīgo iespēju secināt cik pareizi strādā *DUV.*

# Kods

# Projekta faili un hierarhija