# Teorija

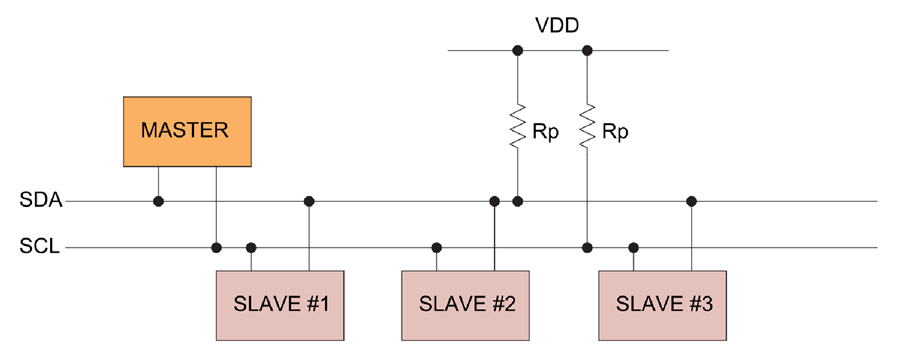
I2C tas ir sinhrona, 8 bitu, secīga saskarne starp integrālam shēmām, kuri ir tuvu viens otram. Tas ir 2 vadu kopne. Ātrumi [100 kbps : 5 Mbps]. Saskarne ir *open drain* veidā, tāpēc *pull up* rezistori ir nepieciešami. *SDA – bidirectional* , *SCL – unidirectional (*no *master* puses*)*, neskaitot *clock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

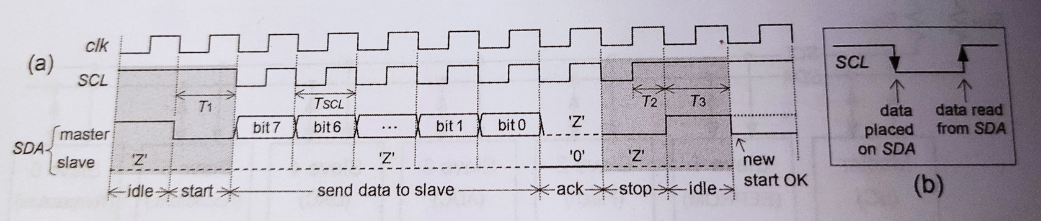


Protokola apraksts:

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

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 no augstas abstr līmeņa sekojoši

*Risunok I2c kontrollera*

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:

*Struktura i2c kontrollera*

// risunok testbencha iz uchebnika

# Implementēšanas soļu apraksts

// kod testbencha po sledam risunka.

// kod uzstrojstva

# Projekta faili un hierarhija