FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Návrh a verifikace integrovaného obvodu pro testování pamětí typu SRAM
Student:	Bc. Šimon Branda
Vedoucí:	Ing. Tomáš Novák
Studijní program:	Informatika
Studijní obor:	Návrh a programování vestavných systémů
Katedra:	Katedra číslicového návrhu
Platnost zadání:	Do konce zimního semestru 2021/22

Pokyny pro vypracování

- Seznámit se s fungováním pamětí typu SRAM, s jejich časováním a s možnými způsoby jejich testování.
- Analyzovat různé možnosti návrhu testovacího čipu a provést systémový návrh.
- Naimplementovat zvolený systém v jazyce VHDL, který bude ovládat paměť typu SRAM a pomocí volitelných parametrů umožní charakterizaci jejího časování a kvalifikaci, zda SRAM paměť splňuje specifikaci.
- Komunikace s integrovaným obvodem pomocí rozhraní SPI
- Provést verifikaci správnosti testovacího čipu jak na úrovni RTL tak i na gate level netlistu realizovaném v cílové technologii (180um)
- Syntetizovat vytvořený design do určené technologie a provést statickou časovou analýzu implementovaného designu.
- Vytvořit vektorové testy pro reálné testování výsledného čipu.
- Shrnout a diskutovat výsledky návrhu, implementace a verifikace.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Hana Kubátová, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 8. září 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Návrh a verifikace integrovaného obvodu pro testování pamětí typu SRAM

Bc. Šimon Branda

Katedra číslicového návrhu
Vedoucí práce: Ing. Tomáš Novák

7. ledna 2021

Poděkování

Chtěl bych moc poděkovat za veškeré předané znalosti, poznatky a informace vedoucímu této práce Ing. Tomáši Novákovi a také mým kolegům z ASICentra, kteří mi od prvního dne ve firmě pomáhají. Dále bych rád poděkoval mé přítelkyni, která mi pomáhala v těch nejtěžších chvílích a vždy mi byla oporou.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel dohodu, na jejímž základě se ČVUT vzdalo práva na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ustanovení § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 7. ledna 2021 . . . . . .

Odkaz na tuto práci

Branda, Šimon. *Návrh a verifikace integrovaného obvodu pro testování pamětí typu SRAM*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

V této práci je rozebrán návrh testovacího čipu pamětí typu SRAM. Nejdříve jsou paměti rozdělené dle jejich vlastností, poté jsou analyzovány paměti SRAM a je vysvětlen RTL návrh testovacího čipu, který byl napsán v jazyce VHDL. Návrh je rozdělen dle jednotlivých funkčních bloků, ve kterých je vysvětlena jejich funkce a co bylo nejnáročnější na implementaci. Poté je vysvětlen průběh syntézy a také statické časové analýzy. V poslední kapitole je řešena verifikace RTL designu a také post-layout netlistu. V závěru jsou vysvětleny vektorové testy a jejich funkce u tohoto testovacího čipu.

Klíčová slova testovací čip, paměť SRAM, RTL design, syntéza, RTL verifikace, post-layout verifikace, vektorové testy, VHDL

Abstract

The main topic of this thesis is the design of the test chip of SRAM memories. At first the memories are divided by their properties, then the SRAM memories are analyzed. In the second chapter there is an overview of the RTL design of the test chip which was written in the VHDL language. The overview is divided by each of the functional blocks where it's told about their functioning and the most challenging parts when being designed. Then the synthesis and static time analysis is described. In the last chapter there is described the verification, both RTL and post-layout verification. There is also explained the vector (pattern) tests and for what they are used in this test chip.

Keywords test chip, SRAM memory, RTL design, synthesis, RTL verification, post-layout verification, vector tests, VHDL

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Od vynálezu prvního tranzistoru uběhla již řada let. Stále se ale technologické hranice tohoto komponetu posouvají a každý rok můžeme vidět něco, co by si lidé nedokázali pár let zpátky ani představit. Díky těmto pokrokům se posouvají možnosti počítačů a také zařízeních s nízkou spotřebou („low power“). Tyto zařízení vyrábí například firma EM Microelectronic, mateřská firma české firmy ASICentrum, ve které byla tato práce zadána.

Jejich produkty se specializují na výše zmíněné „low power“ a bateriemi napájené zařízení. Pro potřebu jejich produktů si EM Microelectronic vyrábí i různé druhy pamětí, které jsou potřebné pro správný chod mikroprocesorů. Tyto paměti se musí během svého vývoje změřit a otestovat, zda splňují všechny své parametry, aby mohli být použity do finálních zařízení. Z tohoto důvodu vznikla tato práce, která má za úkol vytvořit testovací obvod, který bude kontrolovat správnou funkcionalitu a časové parametry pamětí typu SRAM (Static Random Access Memory).

Možná vyvstává dotaz, proč je nutné vytvářet testovací čip a proč se nemůže paměť otestovat přímo skrz příslušné vstupní a výstupní „PADy“ (místa na plošném spoji sloužící k propojení s periferiemi). Pokud se porovná zpoždění signálu, které vzniká průchodem právě zmiňovaným „PADem“ (jednotky i desítky nanosekund), s jemností časování pamětí SRAM(desetiny nanosekund) je jasné, že se tato volba nemůže použít. Je nutné vytvořit testovací čip, který má paměť SRAM umístěnou přímo na čipu, co nejblíže řídící testovací jednotce, která byla v této práci navržena.

V první kapitole se tato práce věnuje analýze paměti SRAM, jak funguje a jak se může ovládat. Poté jsou zde analyzovány funkce testovacího čipu, co musí obsahovat a jak se musí chovat. Ve druhé kapitole je popsána samotná implementace testovacího čipu, dále jsou vysvětleny implementační postupy, jak jednotlivé bloky čipu fungují a jak jsou bloky mezi sebou propojeny. V této kapitole je také projednávána fyzická implementace a její kroky – syntézní postup, „Place and Route“ metoda a také Statická časová analýza (STA – Static

Time Analysis). Poslední kapitola je věnována verifikaci jak RTL (Register-Transfer Logic) designu, tak i verifikaci po „Place and Route“. Dále jsou zde popisována jednotlivá úskalí, která se při verifikaci objevila a na závěr je zde popsána tvorba vektorových testů a k čemu u tohoto testovacího čipu slouží.

Kapitola 1

Analýza návrhu

Tato kapitola se zabývá prvotními kroky ve tvorbě testovacího čipu. Nejdříve je třeba nastudovat testovaný typ paměti, jaké má parametry a jaké důležité závislosti budou muset být otestovány. Poté se musí zanalyzovat celkový návrh testovacího čipu, jak bude strukturován, jak se budou jednotlivé bloky chovat a jak budou mezi sebou propojeny.

1.1 Pamět typu SRAM

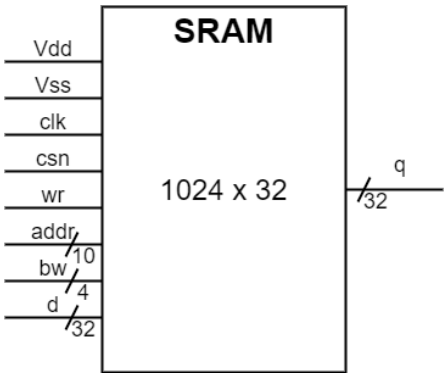
Existuje spousta druhů pamětí, které se dělí podle různých odlišností. Paměti typu SRAM jsou energeticky závislé, to znamená, že pro udržení zapsané hodnoty potřebují stálý přísun energie. Tím se liší od jiných pamětí, například typu NVM (Non-Volatile Memory), které mají svůj obsah napevno zapsaný a nepotřebují tedy stálé napětí.

Paměť SRAM je velice podobná paměti DRAM (Dynamic Random Access Memory). Obě tyto paměti jsou energeticky závislé (bez přísunu energie ztratí uložený obsah), ale liší se mezi sebou stavbou jednotlivých paměťových buněk. Jedna paměťová buňka SRAM paměti se skládá ze 4-6 tranzistorů, zatímco jedna buňka u paměti DRAM se skládá pouze z jednoho tranzistoru. Díky tomuto složení udržuje SRAM paměť zapsanou hodnotu neustále. Paměti DRAM s postupem času klesá nabití kondenzátorů a je tedy nutné pravidelně obnovovat její uložené hodnoty.

Kvůli své vnitřní stavbě, zabírá pamět DRAM mnohem menší plochu než pamět SRAM a díky její velikosti je také levnější. Jak bylo řečeno v předchozím odstavci, paměť DRAM ale potřebuje pravidelné obnovování svých hodnot. Tato operace zvyšuje aktuální spotřebu energie. Pravidelné obnovování u paměti DRAM také způsobuje pomalou přístupovou rychlost, která je až 4x menší než přístupová rychlost paměti SRAM. Zadavatel této práce si vybral k používání paměť SRAM především z důvodů úspory energie.

1.1.1 Specifikace použité SRAM paměti

Tato práce je tvořena pro specifickou paměť typu SRAM dodanou zadavatelskou firmou EM Microelectronic. Do paměti je možné uložit 1024 slov, kde každé slovo může být široké až 32 bitů. Obecný diagram paměti s jejími vstupy a výstupy je zobrazen na obrázku 1.1.



Obrázek 1.1: Diagram paměti SRAM a její vstupů a výstupů, které jsou popsané v tabulce 1.1.

Název signálu	Šířka	Výchozí hodnota[1]	Směr signálu	Popis
addr	10	X	Vstup	Adresní signál
bw	4	X	Vstup	Signál pro zápis určitých bytů vstupního slova
clk	1	0 V	Vstup	Hodinový signál pro čtení a zapisování
csn	1	V_{dd}	Vstup	Signál „Chip Select“ aktivní v nule – aktivace paměti SRAM
d	32	X	Vstup	Vstupní datový signál
q	32	X	Výstup	Výstupní datový signál
vdd	1	V_{dd}	Vstup	Napájení paměti
vss	1	0 V	Vstup	Uzemnění
wr	1	0 V	Vstup	Signál aktivující zápis do paměti (0 – čtení, 1 – zápis)

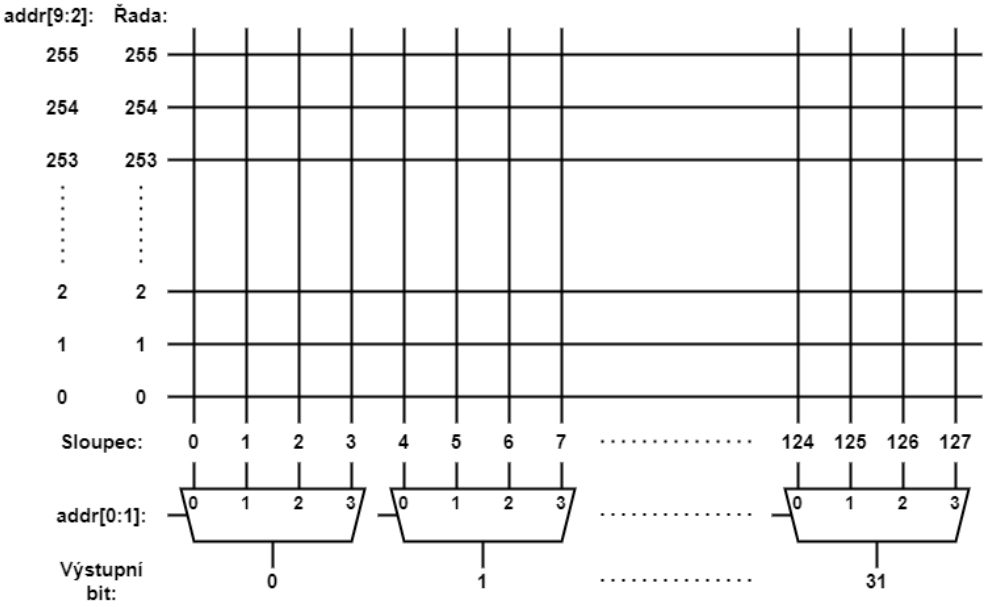
Tabulka 1.1: Tabulka vstupních a výstupních signálů.

[1]Možné výchozí hodnoty v tabulce: V_{dd} – hodnota vstupního napětí, 0 V – hodnota nulového napětí, X – jakákoli vstupní hodnota.

Pro aktivaci paměti z pohotovostního režimu slouží signál „Chip Select“. Pokud je tento signál neaktivní, jakákoli změna vstupních signálů se nezpropaguje do paměti. Pokud se ale změní do aktivní fáze, paměť reaguje přesně tak, jak je očekáváno. Zjednodušeně tedy funguje jako vypínač. Tento signál je aktivní v hodnotě logické nuly, a proto je jeho zkratka „CSN“ – *CS* je zkratka pro „Chip Select“, *N* ve zkratce znamená opačnou polaritu signálu.

Druhým důležitým signálem je operační signál sloužící pro zápis dat. Pokud je tento signál aktivní, pak s náběžnou hranou hodin vždy probíhá zápis dat na adresu, která je aktuálně na adresním vstupu. Naopak, pokud je signál neaktivní, probíhá vždy s hranou hodin čtení z vybrané adresy. Tento signál má obvyklé chování, jeho aktivní hodnota je tedy logická jednička, neaktivní hodnota je logická nula. Z tohoto důvodu má zkratku „WR“ bez žádných přípon.

Jak již bylo výše zmíněno, v této paměti může být uloženo až 1024 slov, které jsou 32 bitů široké. Paměť se tedy skládá z 32768 paměťových buněk. Ty jsou umístěny do čtvercové sítě o rozměrech $256 * 128$ bitů, která je znázorněna na obrázku 1.2. Z obrázku je patrné, na jednom řádku jsou vždy uložené čtyři slova. V prvních čtyřech sloupcích jsou uloženy nulté bity všech slov na řádku, v dalších čtyřech sloupcích jsou uloženy první bity všech slov na řádku a takto to pokračuje až k poslední sloupcům, ve kterých jsou uloženy 31. (poslední) bity všech slov na řádku.



Obrázek 1.2: Diagram rozložení paměťových buněk paměti SRAM.

Tato paměť má jednu adresu, která se skládá z 10 bitů. Po vstupu adresy do paměti se spodní 2 bity oddělí, jak je ukázáno na obrázku 1.2. Horních 8 bitů slouží k adresaci jednoho příslušného řádku ze všech 256 možných. Spodní dva bity adresy vybírají příslušné bity v řádku, které patří jednomu ze čtyř možných slov. S těmito vybranými paměťovými buňkami se poté provádí zápis nové hodnoty nebo přečtení té stávající.

Jeden z dalších vstupních signálů je signál „ByteWrite“. Tento vstup určuje do kterých bytů vybraného slova bude povolen zápis. Skládá se ze 4 bitů, proto je slovo z paměti rozděleno na čtyři části, které jsou ovládány jednotlivými bity této hodnoty. V tabulce 1.2 je ukázáno, jak hodnota „ByteWrite“ funguje a které bity ovládají příslušné části paměti.

Hodnota ByteWrite	data[31:24]	data[23:16]	data[15:8]	data[7:0]
0000’b	Nezapsáno	Nezapsáno	Nezapsáno	Nezapsáno
0001’b	Nezapsáno	Nezapsáno	Nezapsáno	Zapsáno
0010’b	Nezapsáno	Nezapsáno	Zapsáno	Nezapsáno
0011’b	Nezapsáno	Nezapsáno	Zapsáno	Zapsáno
0100’b	Nezapsáno	Zapsáno	Nezapsáno	Nezapsáno
0101’b	Nezapsáno	Zapsáno	Nezapsáno	Zapsáno
0110’b	Nezapsáno	Zapsáno	Zapsáno	Nezapsáno
0111’b	Nezapsáno	Zapsáno	Zapsáno	Zapsáno
1000’b	Zapsáno	Nezapsáno	Nezapsáno	Nezapsáno
1001’b	Zapsáno	Nezapsáno	Nezapsáno	Zapsáno
1010’b	Zapsáno	Nezapsáno	Zapsáno	Nezapsáno
1011’b	Zapsáno	Nezapsáno	Zapsáno	Zapsáno
1100’b	Zapsáno	Zapsáno	Nezapsáno	Nezapsáno
1101’b	Zapsáno	Zapsáno	Nezapsáno	Zapsáno
1110’b	Zapsáno	Zapsáno	Zapsáno	Nezapsáno
1111’b	Zapsáno	Zapsáno	Zapsáno	Zapsáno

Tabulka 1.2: Ukázka funkce signálu „ByteWrite“.

Je důležité také říct, že tato paměť potřebuje vnější hodinový signál. Nemá uvnitř žádný hodinový oscilátor, proto musí být hodinový signál přiveden z testovacího čipu.

Nejdůležitější vstupy a výstupy, které patří ke každé paměti, jsou ty datové. Tato paměť má velikost slova 32 bitů, proto i datové vstupy a výstupy mají takovýto počet bitů. Před zápisem vstupních dat do paměti jsou vždy vybrána platná data, která jsou určena hodnotou „Byte Write“. Na čtení z paměti nemá tato hodnota vliv a vždy je přečteno všech 32 bitů dat.

1.1.2 Operační módy paměti a jejich časové parametry

Tato paměť má tři možné operace. Rozlišují se podle hodnoty signálů „Chip Select“ a „Write“:

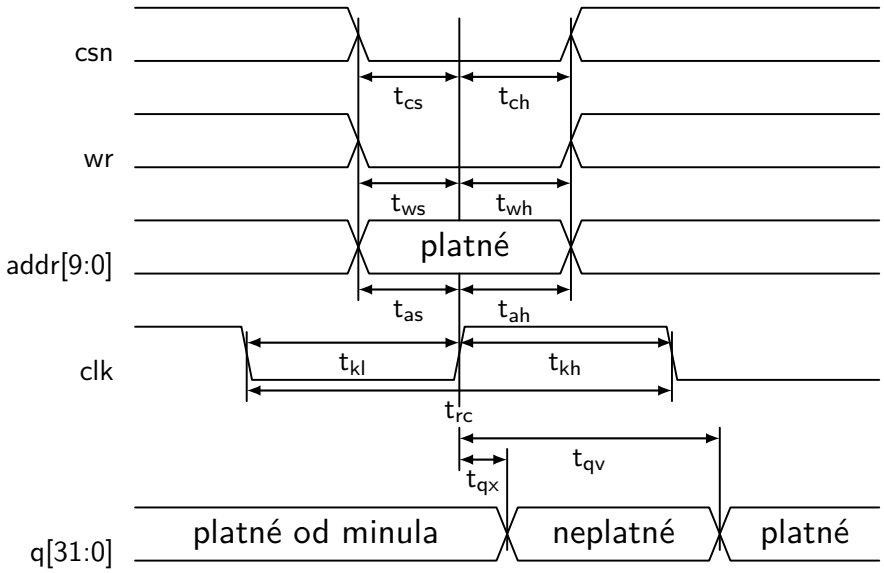
- „CSN“ = 1 – pohotovostní režim,
- „CSN“ = 0 a „WR“ = 0 – režim čtení a
- „CSN“ = 0 a „WR“ = 1 – režim zápisu.

Pohotovostní režim

Tento režim je spuštěný pouze při neaktivní hodnotě signálu „CSN“, ve kterém se na výstupu neustále udržuje poslední čtená hodnota. Je také zaručeno, že paměť bude odolná vůči jakékoli změně signálů na všech vstupech a tím pádem nemůže nastat zápis nebo čtení nechtěných dat. Je také výhodné, že má paměť nízkou spotřebu, pokud se nachází v tomto režimu.

Čtení z paměti

Režim čtení je k dispozici při aktivní hodnotě signálu „CSN“ a zároveň při neaktivní hodnotě signálu „WR“. Čtení probíhá dle diagramu na obrázku 1.3, ve kterém jsou znázorněny důležité parametry pro správný průběh této operace. Z tohoto obrázku je patrné, že všechny vstupní signály (adresní signál a signály „CSN“ a „WR“) musí přijít do paměti s předstihem před hodinovým signálem o velikosti t_{as} ,resp. t_{cs} a t_{ws}. Tyto časové parametry, i další potřebné parametry pro čtení a zápis, jsou vysvětleny v tabulce 1.3.



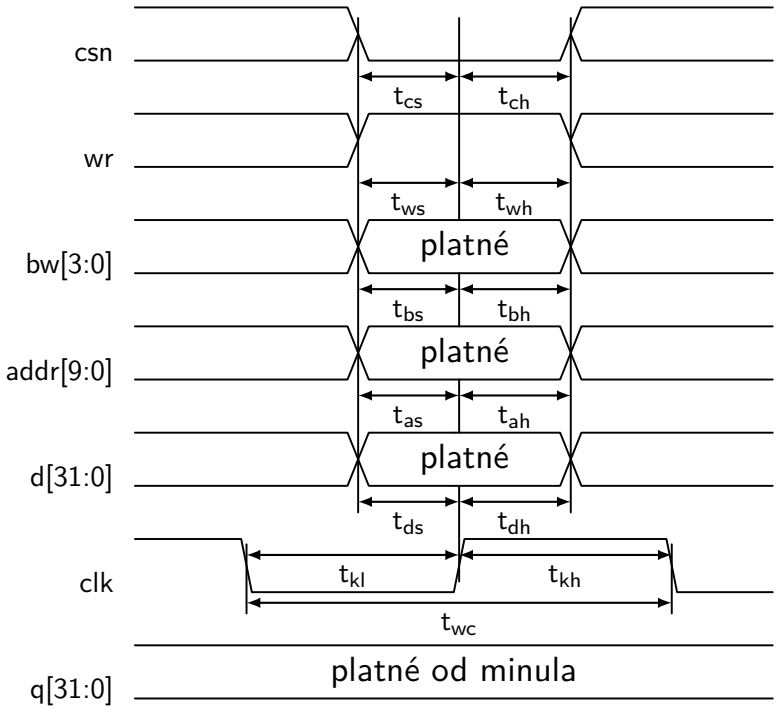
Obrázek 1.3: Diagram časových specifikací režimu čtení.

Pokud jsou splněny dané předpoklady a dodrženy časové podmínky spojené s přesahem od vzestupné hrany hodin (časové parametry t_{ah}, resp. t_{ch} a t_{wh}), začne se generovat správný výstup. Nejdříve je po uplynutí časového parametru t_{qx} zneplatněn výstup z paměti a následně je zpropagován správný výsledek, jež se objeví na výstupu po uplynutí doby t_{qv}, která se počítá od vzestupné hrany hodin.

Zápis do paměti

Režim zápisu funguje podobně jako čtecí režim, vysvětlený v předchozí části. Také potřebuje aktivní hodnotu signálu „CSN“, ale navíc je vyžadována aktivní hodnota signálu „WR“. Tyto signály se stejně jako ostatní signály musí řídit časováním, které je zobrazeno na obrázku 1.4 a vysvětleno v tabulce 1.3.

Pro zápis dat do paměti je třeba splnit požadavky nejen výše zmíněných signálů, ale také signálu „ByteWrite“, adresního signálu a také vstupních dat. Pokud se splní jejich předstih i přesah od vzestupné hrany hodin, tak se zapíší data ze vstupu na určenou adresu v paměti.



Obrázek 1.4: Diagram časových specifikací režimu zápisu.

Zkratka	Vysvětlivka
t_{qv}	Maximální čas na propagaci výsledku čtení
t_{rc}	Minimální čas čtecího cyklu

Zkratka	Vysvětlivka
t_{wc}	Minimální čas zapisovacího cyklu
t_{as}	Minimální předstih signálu addr před náběžnou hranou hodin
t_{ah}	Minimální přesah signálu addr po náběžné hraně hodin
t_{bs}	Minimální předstih signálu BW před náběžnou hranou hodin
t_{bh}	Minimální přesah signálu BW po náběžné hraně hodin
t_{cs}	Minimální předstih signálu csn před náběžnou hranou hodin
t_{ch}	Minimální přesah signálu csn po náběžné hraně hodin
t_{ds}	Minimální předstih datového signálu před náběžnou hranou hodin
t_{dh}	Minimální přesah datového signálu po náběžné hraně hodin
t_{ws}	Minimální předstih signálu wr před náběžnou hranou hodin
t_{wh}	Minimální přesah signálu wr po náběžné hraně hodin
t_{kh}	Minimální délka hodinového signálu v hodnotě '1'
t_{kl}	Minimální délka hodinového signálu v hodnotě '0'
t_{qx}	Minimální vzdálenost náběžné hrany hodin od změny výstupu

Tabulka 1.3: Jednotlivé časové specifikace čtecích a zapisovacích signálů.

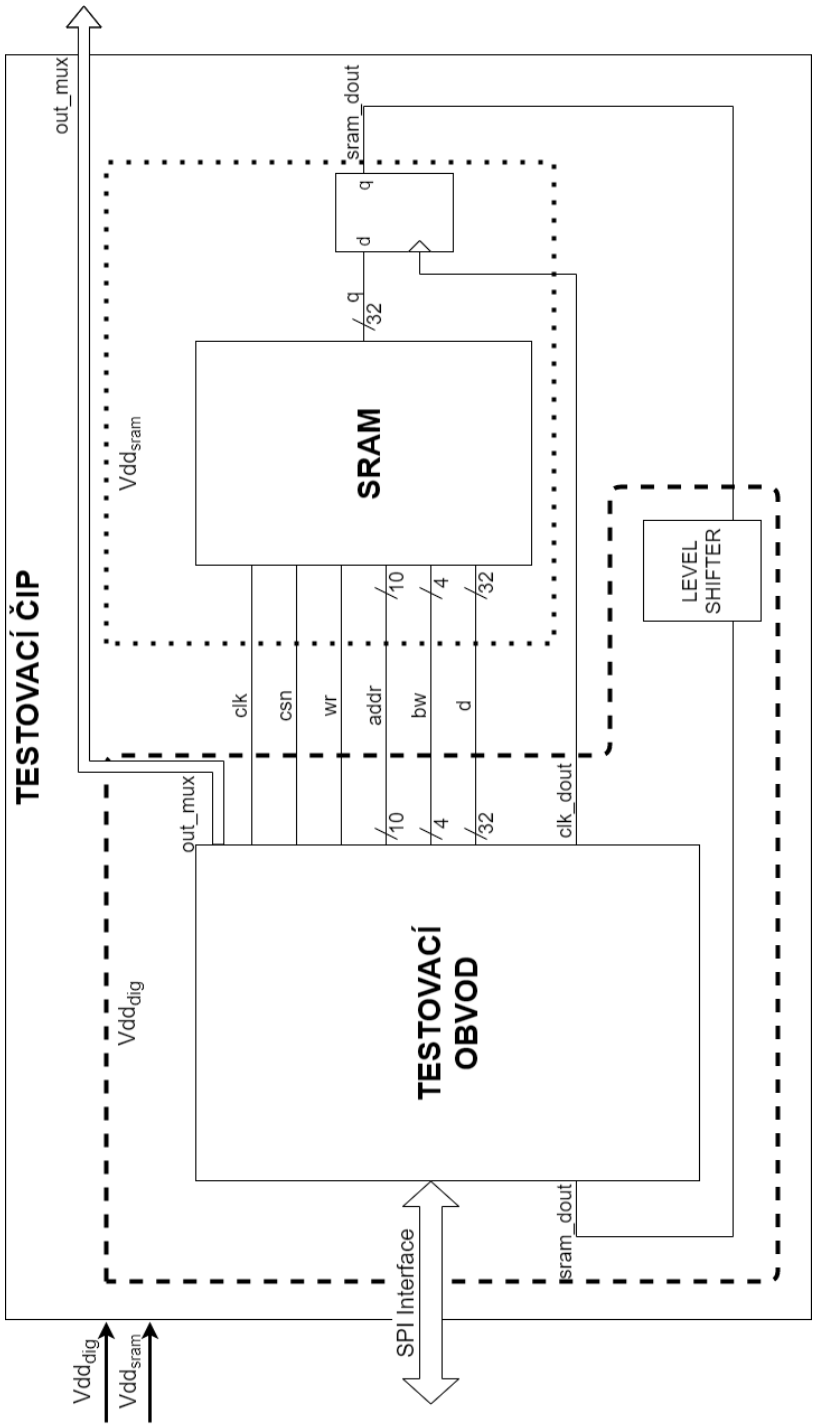
1.2 Blokové schéma paměti s testovacím obvodem

Z předchozí kapitoly 1.1.1 je jasné, jak vypadá paměť, kterou je potřeba otestovat. Je ale nutné si ukázat, jak bude výsledný testovací čip vypadat, co bude ovládat a jak s ním bude komunikovat uživatel.

Zjednodušené výsledné blokové schéma na obrázku 1.5 ukazuje, jak je celý systém provázaný. Systém se skládá ze dvou částí, které mají odlišné napájecí napětí. Pod jedním napětím je navrhovaný digitální blok, který ovládá pamět. Ta je napájená druhým napětím, které slouží výhradně paměti a také výstupním klopným obvodům, které jsou umístěny hned za jejím vstupem. Paměť SRAM bude mít vždy napětí nižší než testovací obvod z důvodu potřeby rychlejší propagace signálů v testovacím obvodu. Proto jsou měniče napětí (anglicky „Level Shifters“) umístěny pouze za výstupy z paměti. Měniče napětí slouží k bezchybnému přechodu mezi různými napěťovými doménami a jsou potřeba při změně z nižšího napětí na vyšší.

Výstupy testovací čipu jsou přímo propojené s testovanou pamětí. Výstup z paměti je načten klopným obvodem, který je umístěn co nejblíže paměti. Je to z toho důvodu, že při měření chceme co nejméně nepřesností. Rozdílnosti v časování SRAM paměti jsou v řádech desetin nanosekund, proto je nepřípustné přidávat k výstupním datům zpoždění způsobené dlouhou cestou signálu. Po načtení výstupního signálu projde signál skrz „Level Shifter“ a poté už jde přímo do testovacího obvodu.

Jak lze na obrázku 1.5 vidět, komunikace s čipem probíhá skrz SPI (Serial Peripheral Interface) rozhraní. Toto rozhraní se skládá ze čtyř signálů.



Obrázek 1.5: Blokové schéma testovacího čipu a paměti SRAM.

- CSB (Chip Select Bit) – Signál „Chip Select“ funguje stejně jako stejnojmenný signál v pamětích SRAM. V jeho aktivní pozici (logická nula) je SPI rozhraní připraveno přijímat, nebo odesílat vybraná data. Při neaktivní hodnotě SPI rozhraní nereaguje a nedokáže odeslat ani přijmout žádná data.
- SCK (Serial Clock) – Vstupní hodinový signál pro SPI transakce. Pokud je signál „CSB“ neaktivní, pak je hodinový vstup hradlovaný a nezpůsobí žádné transakce.
- SDI (Serial Data In) – Vstupní data pro SPI rozhraní. Bity jsou odesílány skrz jednu linku. Do testovacího čipu přichází od nejvýznamnějšího bitu, až po ten nejméně významný.
- SDO (Serial Data Out) – Výstupní data z SPI rozhraní. Bity jsou odesílány stejně jako vstupní data, tedy skrz jednu linku vždy významnějšími bity dřív.

Hlavní úlohou rozhraní SPI je zápis a čtení z registrové mapy. V registrové mapě jsou přístupné veškeré registry, které může uživatel ovládat. Může zde nastavit vstupní data, zdroj hodinového signálu a maximální možnou adresu. Také lze skrz SPI a registrovou mapu odstartovat operaci s pamětí. Skrz SPI může uživatel také z registrové mapy číst adresu, na které testovací obvod skončil, či může přečíst data, která testovací obvod naposledy načetl.

Úkolem testovacího čipu je testovat klasické i mezní chování paměti SRAM. Pro tuto potřebu je tedy nutné, aby testovací čip ovládal veškeré vstupní signály paměti. Tyto signály musí být řízeny a časovány tak, aby byly schopné otestovat všechny časové parametry paměti SRAM z kapitoly 1.1.2. Je třeba vědět, jak velký přesah od hodinové hrany mají vstupní signály paměti, také je důležité vědět zda se výstup generuje správně a zda se tento výstup generuje ve správnou chvíli.

K tomuto testování správnosti výstupu je tedy třeba ovládat signál, který ukládá hodnotu výstupu do klopného obvodu (zobrazen na diagramu 1.5 hned za výstupem paměti). Tento signál, i všechny vstupy paměti, musí testovací čip dokázat ovládat. Je potřeba mít možnost posunout každý ze zmíněných signálů dopředu nebo dozadu nezávisle na ostatních signálech. Je třeba mít možnost ovládat tento posun co nejjemněji (o desetiny nanosekund), aby se daly co nejpřesněji otestovat krajní meze paměti.

Pro kalibraci zpoždění jednotlivých ovládacích signálů, je třeba mít nad jejich posunem vizuální kontrolu. K tomuto účelu je třeba mít tyto signály vyvedené ven z čipu ke sledování. Protože se ale tento čip bude vyrábět pouze v malé sérii a tyto série se „bondují“ (připojování drátů k pouzdru čipu) ručně, je důležité mít co nejméně výstupů. Proto není možné, aby byly veškeré signály vyvedeny samostatně. Všechny kontrolní signály tedy budou vyvedeny pouze dvěma výstupními signály.

Ke korektnímu návrhu čipu je nutné vědět, jaké budou možnosti vstupních hodin a také to, jaká bude maximální frekvence těchto hodin. K tomuto čipu budou přivedeny dva hodinové zdroje.

- CLK_ATB (Automated Test Bench) – Tento hodinový signál slouží především jako hodinový signál pro komunikaci skrz SPI. Maximální možná frekvence tohoto signálu je 12.5 MHz.
- CLK_XTAL – Tento hodinový signál je určený pro fungování obvodu při generování vstupních signálů do paměti, tedy pro tvorbu systémových hodin. Je generován z krystalu, který dokáže generovat frekvence od 1 MHz do 100 MHz a poté je upraven pomocí PLL (Phase-Locked Loop).

1.3 Specifikace funkcí čipu

Důležitým úkolem je specifikovat jednotlivé funkcionality čipu. Je potřeba se řádně domluvit se zadavatelem, co od výsledného čipu požaduje. Dle domluvy lze poté vyspecifikovat detaily jednotlivých funkčních prvků.

1.3.1 Adresace

Jedním ze základních prvků testování pamětí je adresování slov určených k otestování. Čip musí zvládnout čtení nebo zápis na jakékoli místo v paměti. Proto musí mít adresa aspoň 10 bitů, které jsou potřeba k adresaci všech 1024 slov, která se nachází v paměti SRAM. S dodavatelem ale bylo dohodnuto, že bude mít testovací čip adresu širokou 12 bitů, z důvodu větší přízpůsobitelnosti a kompatibilitě s nově vytvořenými paměťmi.

V testovacím čipu je důležité mít možnost nastavit počet slov, která bude chtít uživatel přečíst nebo do nich zapsat. Kvůli této vlastnosti je třeba si dát pozor na možnost přetečení. Pokud chce uživatel přečíst více slov než jich je v paměti, čip to musí rozpoznat a po přečtení nejvyššího slova musí adresu změnit na tu nejnižší a pokračovat ve čtení.

1.3.2 Tvorba vstupních operačních signálů paměti

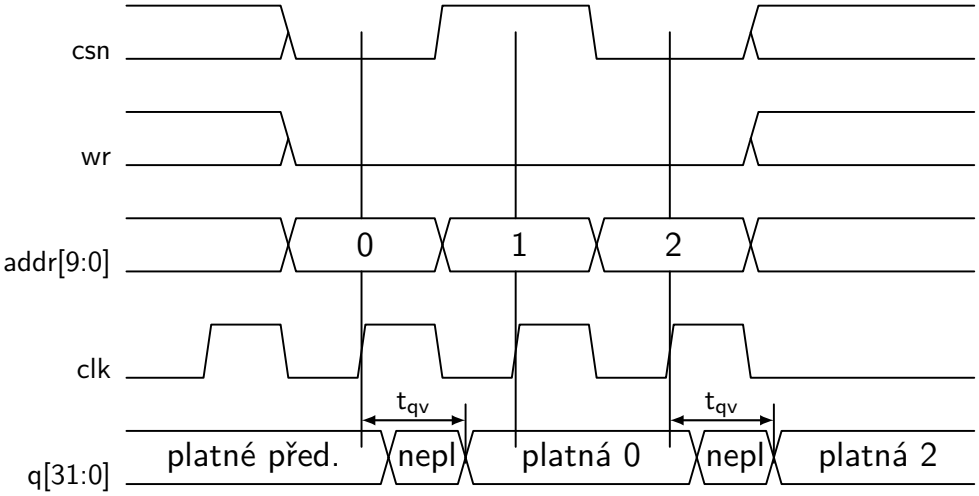
Pro správné testování paměti musí čip generovat vstupní signály paměti dle obrázků 1.3 a 1.4, tedy operační signály „CSN“, „WR“ a také hodinový signál paměti.

Tyto signály je potřeba tvořit vždy stejnou hodinovou hranou. Bylo dohodnuto, že veškeré signály budou generovány náběžnou hranou systémových hodin. Jejich správný předstih nebo přesah od hodinového signálu paměti bude poté řešený zpožďovacím stromem, popsaným později v podkapitole 1.3.4.

Pokud probíhá více operací za sebou (uživatel vybral dvě a více čtení nebo zápisů), signál „CSN“ zůstává od první do poslední operace v aktivní hodnotě.

Do neaktivní polohy se přesune až po dokončení všech operací. To samé platí o signálu „WR“, pokud je více operací zápisu za sebou.

Předchozí odstavec ale nemusí platit, pokud má některý ze signálů aktivní „přepínací režim“. Tento režim při aktivních operacích způsobuje, že s každou náběžnou hranou systémových hodin změní signál svou hodnotu na opačnou. Pomocí toho se může střídat vždy čtecí operace se zapisovací, nebo se může číst či zapisovat ob jednu adresu.



Obrázek 1.6: Diagram ukázky přepínacího režimu pro signál „CSN“.

Čtení ob jednu adresu je ukázáno na obrázku 1.6, kde se střídá signál „CSN“. Na obrázku můžeme vidět provedení tří čtení. To znamená, že první čtení je provedeno s hodnotou signálu „0“ (platná a provedená operace, kterou můžeme vidět na propagaci signálu „q“), ale následující čtení je provedeno s hodnotou signálu „1“ (operace se neprovede, žádná změna výstupu). Třetí čtení se poté provede stejně jako první, tedy úspěšně, a zpropagují se data z druhé adresy na výstup. Tento přepínací režim signálu „CSN“ se dá použít i při zápisu dat, kde způsobí stejné chování čipu.

Tento přepínací režim je také možné zapnout u signálu „WR“. Tímto režimem způsobíme střídání čtení a zápisů. Při tomto režimu bude jedna adresa přečtena a na následující adresu bude zapsán datový vstup paměti. Tento postup se bude opakovat tolikrát, kolikrát uživatel zvolil. Tímto testováním je navíc možné testovat u zmíněných signálů předstih nebo přesah od hodinového signálu paměti.

1.3.3 Tvorba datového vstupu

Stejně jako je nutná tvorba operačních signálů, tak je důležitá i tvorba datového vstupu paměti. Tento vstup bude vždy na náběžnou hranu systémových

hodin zpropagován z registrové mapy, do které byl načten pomocí SPI rozhraní. S dodavatelem paměti je dohodnuto, že tento vstup bude pro každou lichou adresu negován a pro každou sudou adresu zůstane přesně tak, jak je uložený v registrové mapě. Tento postup automaticky tvoří v celé paměti šachovnicový tvar jedniček a nul, který se obvykle používá při testování pamětí.

Stejně jako mají operační signály „přepínací režim“, tak datový signál má „přímý režim“. Tento režim způsobí to, že na každou adresu v paměti budou uložena ta stejná data, jako jsou v registrové mapě. Tím se bude v paměti tvořit souvislá plocha buď jedniček, nebo nul.

Pro správné testování paměti musí umět testovací čip ovládat všech 32 bitů vstupních dat. S dodavatelem bylo ale dohodnuto, stejně jako u adresního signálu, že testovací čip bude moct ovládat až 64 bitů vstupních dat do paměti. Tento krok je důležitý pro budoucí použití vytvořeného čipu, aby byl využitelný i pro paměti vyrobené v budoucnu.

1.3.4 Zpoždění jednotlivých signálů pro ovládání paměti

Jak bylo výše několikrát zmíněno, testovací čip potřebuje zpožďovat různé signálypro tvorbu předstihu nebo přesahu o libovolné velikosti od hrany hodin. Pro potřeby SRAM paměti je důležité, aby bylo možné signály zpozdit až o několik nanosekund. Je také důležité zpoždění změnšovat či zvětšovat po co nejmenších rozdílech.

Pro tuto potřebu bylo dohodnuto se zadavatelem, že jeden interval zpoždění se bude skládat ze dvou invertorů. Velikost tohoto zpoždění bude záviset na vstupním napětí a na teplotě prostředí, ale předpokládá se, že jeden krok bude maximálně do desetin nanosekund.

Takovýchto bloků bude mít každý signál až 64. Tímto počtem se dosáhne požadovaného maximálního zpoždění, aby se dokázaly ošetřit všechny časové parametry. Pro výběr jednoho z těchto zpoždění bude vytvořen stromový multiplexor. Ten se bude skládat z několika vrstev multiplexorů a všechny multiplexory budou dohromady tvořit symetrický strom. Tím se zaručí stejné propagační zpoždění pro různé konfigurace, protože k propagaci jakéhokoli signálu je třeba projít skrz stejný počet multiplexorů.

1.3.5 Snímaní výstupu z paměti

Protože zadavatel požaduje kontrolu časování výstupu z paměti, je tedy potřeba snímat výstup z paměti v určený čas. Pro tuto potřebu je nutné výstup z paměti snímat klopnými obvody. Cesta z paměti až do testovacího čipu by mohla být příliš dlouhá (a tím vytvářet neovladatelné zpoždění), proto je nutné dát klopné obvody co nejblíže paměťovému výstupu. Díky těmto klopným obvodům bude zaručeno snímání správného výstupu z paměti.

Pro toto snímání bude vyveden z čipu speciální signál (znázorněný na obrázku 1.5 jako „clk_dout“), který bude přiveden do klopných obvodů za

pamětí jako jejich hodinový signál. Tento signál bude generovaný systémovými hodinami a bude možné ho zpozdit o uživatelem zvolený čas, stejně jako operační signály, adresu nebo datový vstup.

Data vystupující z klopných obvodů poté budou muset projít skrz měniče napětí (ukázané na obrázku 1.5), které upraví napájení těchto signálů pro správné fungování čipu.

1.3.6 Porovnání a kontrola výstupu

Tento čip je primárně nastaven pro zápis hodnot v šachovnicovém tvaru (sudé adresy obsahují nuly a liché adresy obsahují jedničky). Proto vytvořený čip bude obsahovat dva registry pro porovnávání výstupu. Registry budou mít 64 bitů, aby bylo možné zkontrolovat jakoukoli v budoucnu připojenou paměť. Jeden registr bude odpovídat slovům, která budou přečtena ze sudých adres a druhý registr bude odpovídat slovům z lichých adres.

S dodavatelem bylo domluveno, že pokud se vyskytnou při čtení neočekávaná data, čip se zastaví. Při tomto zastavení by měl čip uložit chybnou adresu a přečtená chybná data do registrové mapy, aby měl uživatel možnost tyto data přečíst skrz SPI komunikaci a odhalit možnou příčinu.

1.3.7 Propagace pomocných signálů

Dříve již bylo zmíněno, že operační signály paměti bude možné vybrat do dvou specializovaných výstupů z čipu, které může uživatel pozorovat. Na čipu se budou nacházet ale i další dva výstupy, které budou obecnějšího charakteru (všechny tyto signály jsou na obrázku 1.5 pod názvem „out_mux“). Na nich bude moct uživatel pozorovat zvolené pomocné signály, které například signalizují probíhající operaci, nebo signalizují stav porovnávání výstupu s předpokládanou hodnotou.

Veškeré tyto výstupní signály, které jsou vybírány z více vnitřních signálů, prochází multiplexorovým stromem, podobně jako u zpožďování signálů. Tímto způsobem jsou všechny signály zpožděny o stejný čas.

1.4 Návrh architektury obvodu

Všechny výše popsané speficikace je nutné rozdělit do jednotlivých funkčních bloků, které se mezi s sebou propojí a utvoří celý fungující čip. Jeho blokové schéma můžeme vidět na obrázku 1.7.

Pro tento testovací čip byly tři bloky převzaty od zadavatelské firmy – z důvodu ušetření času a využití již implementovaných bloků. Byly převzaty bloky pro SPI komunikaci, pro výběr hodin a také synchronizační blok. Z části byl převzat blok s registrovou mapou, ta však byla celá upravena pro potřeby tohoto testovacího obvodu. Zbylé bloky byly navrhnuty přímo pro tento testovací obvod. Zde je souhrn všech funkčních bloků uvnitř testovacího čipu:

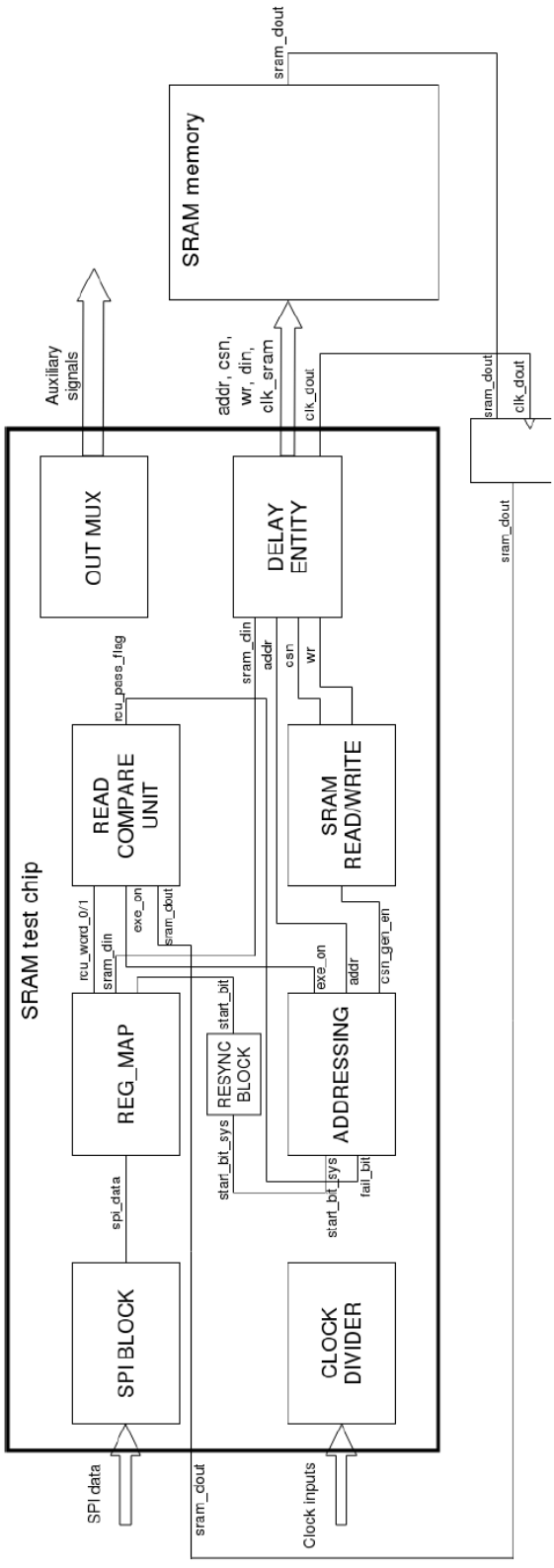
- adresní blok,
- blok pro čtění a zápis,
- porovnávací blok,
- zpožďovací blok,
- blok pro vnější propagaci signálů,
- SPI blok,
- blok pro výběr hodin,
- synchronizační blok a
- blok s registrovou mapou.

1.4.1 Synchronizace signálů

Systém bez synchronizačních prostředků by v asynchroním prostředí nebyl funkční. Protože jsou vnější signály asynchronní vůči vnitřním hodinám systému, je třeba ošetřit možné problémy, které můžou nastat.

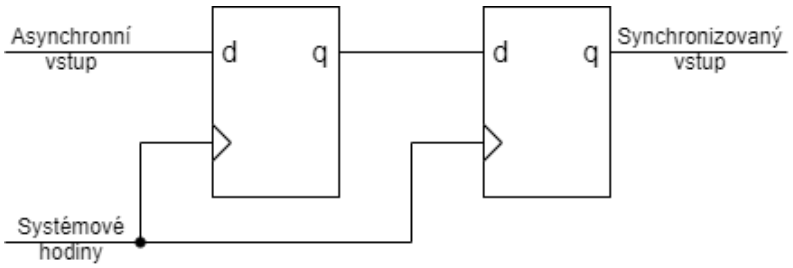
Synchronizační blok zajišťuje sjednocení signálů tak, aby právě k těmto problémům nedošlo. Pokud by signál změnil svou hodnotu ve stejnou dobu ve které je tento signál snímán klopným obvodem, dostal by se klopný obvod do metastabilního stavu – to je stav, ve kterém není jisté, zda je výstup klopného obvodu „nula“ nebo „jednička“ [1]. Metastabilita se sice po nějaké době ustálí, ale nikdy není jisté, jak dlouho to bude trvat. Tento stav by narušil chod celého systému a nebyl by funkční.

Synchronizovat ale není potřeba všechny signály, které jsou přivedeny asynchronně do čipu. Ve specifikaci tohoto čipu je uvedeno, že není možné zapisovat do registrů skrz SPI rozhraní při provádění jakékoli operace. Uživatel musí tyto zápisy provést před odstartováním čtení či zápisu.



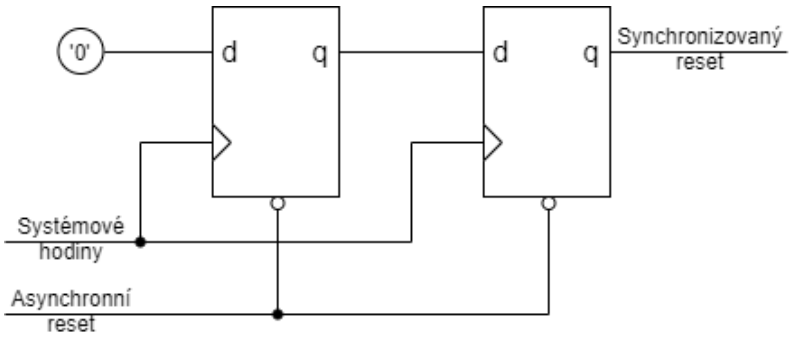
Obrázek 1.7: Detailní schéma zapojení testovacího čipu s pamětí.

Tímto krokem se zajistí stabilita veškerých asynchronních signálů, které byly přivedeny skrz SPI, a je jisté, že nedojde k žádné metastabilitě klopných obvodů.



Obrázek 1.8: Schéma synchronizace asynchronního signálu.

Synchronizační blok, který metastabilitě signálu předchází, se skládá ze dvou klopných obvodů spojených za sebou, které jsou zobrazeny na obrázku 1.8. Pokud by se náhodou první klopný obvod dostal do metastabilního stavu, tak má celou hodinovou periodu k ustálení a druhý klopný obvod má jistotu, že vždy bude snímat ustálenou hodnotu. Tento synchronizátor byl použit u signálu *start_bit* (můžeme ho vidět v diagramu na obrázku 1.7), který odstartovává všechny operace a je tedy nutné, aby byl synchronizovaný do systémové domény.



Obrázek 1.9: Schéma synchronizace asynchronního resetu (aktivní hodnota resetu je logická „jednička“).

Pokud chceme synchronizovat resetovací signál, musíme postupovat odlišně. Synchronizátor asynchronních resetů je znázorněn na obrázku 1.9. Asynchronní reset je přiveden na resetovací port klopných obvodů. Při aktivaci resetu jsou klopné obvody okamžitě přivedeny do resetovací polohy. Při uvolnění asynchronního resetu je nejprve načtena „stálá nula“ (v případě, že je aktivní hodnota resetu v jedničce) do prvního klopného obvodu a v dalším hodinovém taktu je načtena hodnota z prvního klopného obvodu. Tímto systémem se zabrání jakkékoli metastabilitě při manipulaci s asynchronním resetem[2].

Tímto synchronizátorem resetů jsou stabilizovány resety přicházející skrz SPI rozhraní. Tyto signály se nazývají *sram_reset*, *addr_reset*, *rcu_reset* a *flag_reset*. Jejich význam je blíže popsán v kapitole 2.1.2.

1.4.2 Adresní blok

V tomto bloku probíhá mimo inkrementaci adresy také inkrementace provedených operací. Protože se může paměť číst nebo zapisovat několikrát dokola, je třeba tento údaj inkrementovat odděleně od adresy.

Datový vstup paměti se také tvoří zde v adresním bloku. Protože se vstupní data mění v závislosti na lichosti a sudosti adresy, je výhodné mít generování vstupních dat spojené s adresací.

Do tohoto bloku vede také synchronizovaný startovací signál *start_bit_sys*, který můžeme vidět na obrázku 1.7. Tento signál odstartuje inkrementaci adresy a také generování dalších signálů mimo tento blok. Proto je z adresního bloku vyveden signál *exe_on*, který aktivuje další bloky testovacího čipu.

1.4.3 Blok pro čtení a zápis

Tento blok generuje signály pro čtení a zápis, tedy signály *csn* a *wr*. Po aktivaci signálu *exe_on*, který vede z adresního bloku, se začnou generovat jejich příslušné hodnoty.

Uživatel může skrz SPI rozhraní zapnout přepínací mód jednotlivých signálů. Pokud je jakýkoli mód aktivní, tak se vybraný signál neudržuje v aktivní hladině, ale s každým hodinovým pulsem mění svou hodnotu z aktivní do neaktivní a naopak.

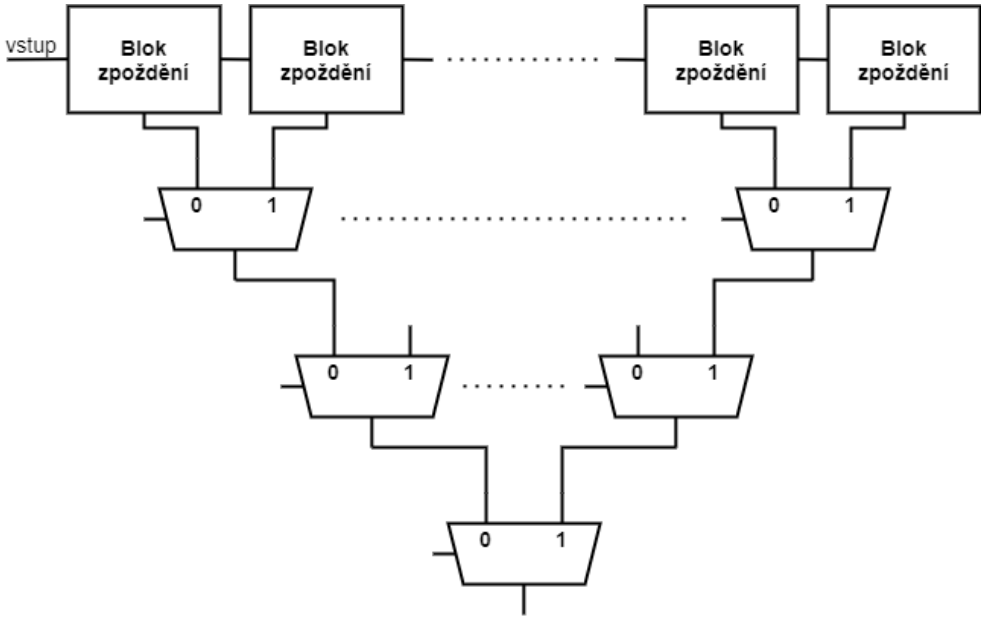
1.4.4 Porovnávací blok

Do porovnávacího bloku vstupují předpokládaná data s právě navzorkovanými daty z paměti a porovnávají se. Předpokládaná data jsou uložená v registrové mapě a dají se změnit skrz rozhraní SPI.

Z porovnání vzniknou dva signály. Jeden z výsledků je kombinační signál, který je aktivní (rovná se „1“), pokud je v daný moment porovnání správné. Pokud je porovnání špatné, tak je tento signál neaktivní (rovná se „0“).

Druhým výsledkem je sekvenční signál, který je resetován do aktivní pozice (rovná se „1“). Pokud je s náběžnou hranou systémových hodin porovnání špatné, signál se deaktivuje a v této poloze zůstane až do resetu celého systému. Pokud bude celé čtení správné, signál bude celou dobu aktivní.

Kombinační signál ukazuje uživateli aktuální hodnotu porovnávání, zatímco sekvenční signál ukazuje výsledek celého porovnávání od odstartování operace. Pokud je sekvenční signál aktivní, tak víme, že od začátku operace nenastala žádná chyba. Pokud v průběhu operace nějaká chyba nastala, sekvenční signál bude neaktivní.



Obrázek 1.10: Názorné schéma multiplexorů ve tvaru stromu a bloků na zpoždění.

1.4.5 Zpožďovací blok

Všechny signály vedoucí do paměti SRAM musí mít možnost být libovolně zpožděny. Do tohoto bloku vedou jednotlivé signály ovládající pamět SRAM (můžeme je vidět na obrázku 1.7). Ty jsou nezávisle na ostatních zpožděny dle nastavení od uživatele.

Zpoždění je vytvořeno řadou obsahující 64 zpožďovacích buněk, mezi kterými jsou vyvedené signály, které vstupují do multiplexorového stromu. Tento strom i s buňkami zpoždění můžete vidět na obrázku 1.10. Díky symetrické stavbě tohoto stromu je zaručeno, že každé zpoždění signálu bude mít totožnou délku průchodu tímto stromem a díky tomu se velikost daného zpoždění nezmění.

1.4.6 Blok pro vnější propagaci signálů

Uživatel, který bude pracovat s tímto testovacím čipem se potřebuje ujistit, zda čip funguje správně. K tomu potřebuje mít některé vnitřní signály vyvedené mimo čip, aby mohl sledovat, zda probíhá uživatelem požadovaná operace, nebo zda se například čte správné slovo v paměti.

Z testovacího čipu vedou celkově čtyři pomocné signály. Do dvou z nich vedou pouze signály určené pro ovládání paměti, jako např. *CSN*, adresa, datový vstup či zapisovací signál. Do dalších dvou vedou i vnitřní signály

z testovacího čipu. Uživatel se tedy může podívat na to, zda aktuálně nějaká operace probíhá, zda je dokončena, nebo jaký je výsledek porovnávání.

Uživatel si může vyvést stejný vnitřní signál dvěmi propagačními signály ven a může měřit odchylku šíření signálu uvnitř čipu. Také lze pomocí těchto venkovních signálů zjišťovat velikost jednotlivých kroků zpoždění a sledovat změnu této velikosti v závislosti na napájecím napětí. Tímto způsobem probíhá kalibrace zpoždění operačních signálů při reálném testování.

Kapitola 2

Implementace

Poté, co se potvrdily všechny potřebné funkcionality čipu, je třeba je naimplementovat. Rozvržení návrhu do jednotlivých bloků bylo upřesněno v minulé kapitole. Tyto informace je potřeba dodržovat při implementaci.

Implementace je psána v jazyce VHDL, což je zkratka pro VHSIC-HDL, kde VHSIC znamená Very High Speed Integrated Circuit („velmi rychlý integrovaný obvod“) a HDL je zkratka pro Hardware Description Language („jazyk pro popis hardware“). Tento jazyk je tedy určen pro popis hardware, kde může uživatel jednoduše popsat chování jednotlivých bloků. Díky tomuto jazyku lze vytvářet jednoduché a nezávislé funkční bloky, které se poté dají spojit do složitějších systémů[3]. Tento jazyk se dá využít jak pro popis hradel, popis RTL nebo i popis algoritmů. Proto je využitý v této práci nejenom na návrh integrovaného obvodu, ale také pro jeho testování[4].

2.1 RTL realizace

Prvním krokem implementace je zprovoznění jednotlivých funkčních bloků, ozkoušení jejich správnosti a jejich správné zapojení do celého bloku.

Každý funkční blok (v jazyce VHDL se blok nazývá entita) je implementován v separátním souboru pro zachování čitelnosti, pro snadnější odstraňování chyb a také to napomáhá lepší přenositelnosti jednotlivých entit do jiných projektů.

V první podkapitole se probere rozhraní testovacího čipu a v dalších podkapitolách se vysvětlí vytvořené funkce jednotlivých entit a v případě obtížích při návrhu se v nich rozebere řešení problému. Pro snažší pochopení problémů jsou v některých podkapitolách ukázány části zdrojových kódů.

2.1.1 Rozhraní testovacího čipu

Rozhraní bylo předem domluvené se zadavatelem této práce. Kompletní rozhraní se všemi signály, šířkou jednotlivých signálů a jejich popisem můžeme

vidět v tabulce 2.1. V první části tabulky jsou napájecí signály, následují signály pro komunikaci SPI a poté ve třetí části tabulky se nachází signály využívané k ovládání paměti SRAM.

SPI rozhraní se používá v zadavatelské firmě u více projektů a proto z důvodu usnadnění práce bylo převzato i do této práce. SPI protokol pro správné fungování potřebuje celkově čtyři signály, které poznáme v rozhraní dle zkratky „spi“ v jejich jméně. SPI potřebuje hodinový signál, který je dodáván signálem *pad_spi_clk*. Poté je nutný signál *pad_spi_csb*, který aktivuje odesílání nebo přijímání dat. Dále jsou potřeba datové signály dovnitř čipu (*pad_spi_din*) a také ven z čipu (*dig_spi_dout*).

Název signálu	Šířka	Směr signálu	Popis
Vdd	1	-	Kladné vstupní napětí
Vss	1	-	Nulové vstupní napětí (uzemnění)
pad_spi_din	1	Vstup	Datový vstup pro SPI rozhraní
dig_spi_dout	1	Výstup	Datový výstup z SPI rozhraní
pad_spi_clk	1	Vstup	Hodinový vstup pro SPI rozhraní
pad_spi_csb	1	Vstup	„Chip Select“ signál pro SPI rozhraní, aktivní v logické nule
pad_reset	1	Vstup	Hlavní externí resetovací signál
pad_clk_atb	1	Vstup	Hodinový vstup z ATB přístroje
pad_clk_xtal	1	Vstup	Hodinový vstup z externího krystalového oscilátoru
sram_msk_mo	64	Vstup	Externí volba šířky aktivních datových signálů z paměti SRAM
sram_addr_mo	12	Vstup	Externí volba šířky aktivních adresních signálů do paměti SRAM
dig_dout	64	Vstup	Výstupní data z paměti, které jsou načtena pomocí signálu *dig_clk_dout*
dig_addr	12	Výstup	Adresní signál vedoucí do paměti SRAM
dig_bw	4	Výstup	Signál „Byte Write“ vedoucí do paměti SRAM
dig_wr	1	Výstup	Zápisový signál vedoucí do paměti SRAM
dig_clk_sram	1	Výstup	Hodinový signál pro paměť SRAM
dig_csn	1	Výstup	„Chip Select“ signál pro paměť SRAM
dig_din	64	Výstup	Datový vstup paměti SRAM
dig_clk_dout	1	Výstup	Hodinový signál pro registry umístěné za pamětí SRAM
dig_out1	1	Výstup	První pomocný výstup z čipu
dig_out2	1	Výstup	Druhý pomocný výstup z čipu

Název signálu	Šířka	Směr signálu	Popis
dig_out3	1	Výstup	Třetí pomocný signál vyvádějící signály ovládající paměť SRAM
dig_out4	1	Výstup	Čtvrtý pomocný signál vyvádějící signály ovládající paměť SRAM
dig_gp_reg_0	8	Výstup	Registry rezervované pro případné budoucí využití
dig_gp_reg_1	8	Výstup	
dig_gp_reg_2	8	Výstup	
dig_gp_reg_3	8	Výstup	

Tabulka 2.1: Seznam vstupů a výstupů testovacího čipu.

Tento čip může být ovládán ze dvou zdrojů hodinového signálu. Jeden zdroj pochází z testovacího přístroje, z něhož se vytvořený čip testuje a validuje. Tento hodinový signál nedosahuje vysokých frekvencí, ale pro komunikaci skrz SPI rozhraní dostačuje. Druhý hodinový signál je tvořen z krystalového oscilátoru, který je umístěn na plošném spoji u testovacího čipu a dokáže vytvořit až frekvenci o hodnotě 100 MHz. Signál z tohoto oscilátoru bohatě dostačuje rychlostním potřebám tohoto testovacího čipu a proto není potřeba mít jiný zdroj hodinového signálu.

Vytvářený testovací čip je tvořen nejen pro aktuální využití, ale hlavně pro využití v budoucnu. Z tohoto důvodu je adresní i datová sběrnice větší než u používané paměti SRAM. K výběru velikosti paměti na daném testovacím obvodu slouží signál *sram_addr_mo*. Tento signál označuje počet aktivních adresních bitů. Jeho hodnota se mění pomocí takzvaného „Metal Option“, kde se jednotlivé bity signálu připojí napevno drátem v metalické vrstvě (proto „Metal Option“) do stálé jedničky nebo nuly. Tyto „Metal Option“ se pak dají jednoduše měnit jen změnou metalické vrstvy. Tímto způsobem funguje i signál *sram_msk_mo*, který určuje aktivní část datové sběrnice.

Další část rozhraní testovacího čipu se skládá přímo ze signálů pro paměť SRAM. Z paměti přichází do čipu datová sběrnice, která je sice široká 64 bitů, ale aktivní bity jsou rozeznány pomocí signálu *sram_msk_mo*. Další signály pro SRAM paměť jsou výstupní a jsou nutné k ovládání vybrané paměti. Mezi adresou, zápisovým signálem a hodinovým signálem je také signál pro navzorkování výstupních dat *dig_clk_dout*. Tento signál vede jako hodinový vstup do klopných obvodů umístěných za pamětí SRAM, jak lze vidět na obrázku 1.7.

V rozhraní nechybí ani již zmiňované čtyři pomocné signály pro lepší přehled uživatele nad operacemi testovacího čipu. Dva z těchto signálů (*dig_out1* a *dig_out2*) dokáží uživateli ukázat nejen signály vedoucí do paměti, ale také například signál ukazující správnost porovnávání výstupu, nebo třeba

systé-mový hodinový signál. Další dva z těchto signálů slouží výhradně pro propagaci operačních signálů pro paměť. Uživateli slouží pro kontrolu jednotlivých zpoždění na signálech vedoucích do paměti. Poslední čtyři výstupní signály (*dig_gp_reg_0-4*) z rozhraní čipu jsou pomocné registry pro možné budoucí použití, které prozatím nejsou využívány.

2.1.2 Významné registry v registrové mapě

Před popisem jednotlivých implementačních kroků je potřeba popsat nejdůležitější registry, do kterých uživatel může zapsat nebo z nich může číst uložené hodnoty. Všechny registry mají velikost jednoho Bytu, takže mají celkem 8 bitů, které se dají využít.

Nejdůležitějším a nejpoužívanějším registrem je registr *rw_cfg* – což je zkratka pro „Read-Write Config“ registr. Tento registr obsahuje pět signálů, do kterých může uživatel zapsat nebo z nich číst, a dva signály, ze kterých může pouze číst:

- *start_bit* – Zápisem do tohoto signálu uživatel spustí vybranou operaci. Tento signál vede do resynchronizačních klopných obvodů a poté odstartuje generování adresy a dalších signálů.

- *done_bit* – Z tohoto signálu je možné pouze číst a tento signál indikuje dokončenou operaci testovacího čipu. U zápisové operace vždy indikuje dokončený zápis všech požadovaných adres, u čtení může indikovat jak správně dokončené čtení, tak i předčasně dokončené čtení, kvůli chybě v porovnání.

- *direct_access* – Tímto signálem může uživatel nastavit způsob tvorby vstupních dat. Pokud je tento signál neaktivní (logická hodnota je „0“), tak se v každém zapisovacím cyklu střídají nezměněná vstupní data zregistrů *din_reg* s negovanými daty z těchto registrů. Pokud je signál *direct_access* aktivní, tak jsou vstupní data vždy nezměněná a do paměti se zapisují přímo z registru *din_reg*.

- *bw_mode* – Tento signál ovládá střídací režim signálu *bw*. Pokud je tento signál aktivní, každý druhý cyklus se zapisuje nebo čte s platnou hodnotou signálu *bw* a v dalších cyklu se zapisuje nebo čte s nulovou hodnotou tohoto signálu.

- *rcu_pass_flag* – Toto je druhý, čistě čtecí, signál v registru *rw_cfg*. Zde se ukazuje výsledek proběhnutých čtení z paměti SRAM. Pokud čtení proběhlo bez chyby, signál bude aktivní s hodnotou „1“. Pokud ale při čtení nastala chyba, tento signál bude deaktivován a uživatel přečte logickou nulu.

- *sram_csn_mode* – Pomocí tohoto signálu se spustí režim přepínání signálu *csn*. Pokud je tento signál neaktivní, tak se po celou dobu operace drží signál *csn* v jeho aktivní poloze, tedy v logické nule. Pokud je ale signál aktivní, tak se při každém hodinovém cyklu střídá hodnota signálu *csn* z aktivní do neaktivní polohy.
- *sram_read_mode* – Tento režim určuje obecný režim, který bude testovací čip provádět po odstartování uživatelem. Skládá se ze dvou bitů, které mají tyto pojmenování:
  - „00“ – Výchozí pozice, kdy neprobíhá žádná operace.
  - „01“ – Režim čtení, při kterém se aktivuje pouze signál *csn*, signál *wr* je po celou dobu neaktivní neaktivní.
  - „10“ – Zápisový režim, který aktivuje signály *csn* i *wr*.
  - „11“ – Střídavý režim, který v jednom taktu zapisuje a v druhém čte.

Tento registr se tedy používá jako poslední zapsaný registr do testovacího čipu před činností testovacího čipu. Pomocí něho uživatel nastaví příslušné operace, které mají probíhat v čipu, odstartuje testovací čip a také vyhodnotí skrz něj výsledek operace.

Dalšími potřebnými registry jsou registry sloužící k nastavení zpoždění na jednotlivých signálech. Každý ze vstupních signálů paměti (adresa, „Chip Select“ signál, zápisový signál, datový signál a hodinový signál) mají každý svůj registr pro zpoždění. Šestý registr pro zpoždění se využívá pro zpoždění signálu *clk_dout*, který má na starost snímání dat z výstupu paměti. Všechny tyto registry jsou složené z šesti bitů, takže mají 64 možností nastavení. Hodnota „0“ znamená nezpožděný signál a hodnota „63“ znamená nejvíce zpožděný signál.

Vstupní data do paměti se nastavují skrz registry *din_reg_0* až *din_reg_7*. Každý bit vstupních dat může být nastaven buď do jedničky nebo nuly, skrz příslušný registr. Data jsou uspořádány logicky, takže nultý bit v registru *din_reg_0* je na nulté pozici vstupních dat a sedmý bit v registru *din_reg_7* je na 63. pozici ve vstupních datech.

Pro zjištení, jaká data byla přečtená, jsou v registrové mapě registry *dout_reg_0* až *dout_reg_7*. Při čtecích operacích jsou výstupní data z paměti vždy uložena i do těchto registrů. Uživatel si tedy může po dokončení operace zjistit poslední přečtená data, která mohla být například chybná.

Při čtení se výstupní data porovnávají s předem předpokládanými hodnotami na lichých a sudých adresách. K těmto předpokládaným hodnotám slouží registry *rcu_word_0_0* až *rcu_word_0_7* pro sudá slova a *rcu_word_1_0* až *rcu_word_1_7* pro lichá slova.

Uživatel může před každou operací nastavit, které vnitřní či operační signály budou vyvedeny ven z čipu, aby je bylo možné sledovat. K tomu slouží dva registry *out_cfg* a *out2_cfg*. Spodní čtyři bity registru *out_cfg* slouží

k výběru výstupního signálu *dig_out1*, horní čtyři bit poté slouží pro výběr signálu *dig_out2*. Registr *out2_cfg* slouží pro výběr signálů *dig_out3* a *dig_out4*, které propagují pouze operační paměťové signály.

Resetovací signály

Uživatel má možnost resetovat určité části testovacího čipu nezávisle na jiných. K této potřebě slouží registr *reset_reg*. Tento registr obsahuje čtyři resetovací signály – *sram_reset*, *addr_reset*, *rcu_reset* a *flag_reset*. Všechny tyto bity jsou takzvané „one-shot" registry. Pokud některý z těchto bitů uživatel aktivuje SPI zápisem, tak tyto bity budou následující SPI operací deaktivovány. Tím je zaručeno, že tyto bity nebudou nastavené déle, než je potřeba. Tyto bity jsou poté synchronizovány a dále vedou do jednotlivých klopných obvodů, které resetují do výchozí pozice. Reset *sram_reset* resetuje operační signály paměti (například signály *csn* a *wr*), reset *addr_reset* je připojen ke klopným obvodům v adresním bloku, *rcu_reset* je přivedený do porovnávacího bloku a reset *flag_reset* je přiveden ke klopným obvodům signálů, které značí dokončené či započaté významné události nebo například výsledek porovnávání.

Implementace registrů

Registry jsou implementovány pomocí dvou signálů. Jeden signál s názvem *s_rw_cfg_decod* slouží k rozpoznání, zda byla z SPI přečtena jeho adresa. Pokud byla přečtena adresa vybraného registru, tento signál se aktivuje.

```
s_rw_cfg_decod<='1' WHEN spi_address=A_RW_CFG(6 DOWNTO 0) ELSE'0';
```

Zdrojový kód 2.1: Ukázka aktivace signálu *s_rw_cfg_decod*.

Druhý signál obstarává uložení hodnoty a jmenuje se *srw_cfg*. Tento signál je široký osm bitů a do tohoto signálu se ukládá hodnota dat, které se přečtou z SPI. Ve zdrojovém kódu 2.2 můžete vidět implementaci klopného obvodu sloužícího pro tento registr.

```
P_RW_CFG : PROCESS (spi_wr_data, reset)
BEGIN
  IF reset ='1'THEN
    srw_cfg    <= D_RW_CFG;
  ELSIF spi_wr_data'EVENT AND spi_wr_data='0' THEN
    IF (s_rw_cfg_decod = '1') THEN
      srw_cfg(7)          <= data_in(7);
      srw_cfg(5 DOWNTO 4) <= data_in(5 DOWNTO 4);
      srw_cfg(2 DOWNTO 0) <= data_in(2 DOWNTO 0);
    END IF;
  END IF;
END PROCESS;
```

Zdrojový kód 2.2: Ukázka klopného obvodu registru *srw_cfg*.

Ve zdrojovém kódu 2.2 můžeme vidět, že je tento registr asynchronně resetován. Tento klopný obvod je ovládaný hodinovým signálem z SPI, který

se tvoří po přečtení všech osmi bitů z SPI komunikace. Pokud je aktivován signál *s_rw_cfg_decod* (z SPI byla přečtena adresa tohoto registru), tak se do zadaných bitů zapíší data z SPI komunikace. Můžeme také vidět, že při zápisu dat jsou vynechány dva bity, šestý a třetí. Tyto bity slouží jen pro čtení, proto není možné do nich skrz SPI komunikaci zapsat.

Pokud chce uživatel data z registrů přečíst, vždy se hodnota registrů zapíše do výstupního signálu *data_out*, ze kterého je poté vytvořen sériový přenos dat skrz SPI. Tato situace je ukázána ve zdrojovém kódu 2.3. Můžeme vidět, že data z tohoto registru jsou vybrána pouze, když je aktivní signál *s_rw_cfg_decod*. Pokud ano, do signálu *data_out* se poskládají jednotlivé bity registru, tentokrát i s vloženými bity určenými pouze pro čtení. Lze vidět, že v registru *rw_cfg* se na šesté pozici nachází *done_bit* a na třetí pozici *rcu_pass_flag*.

```
data_out <= srw_cfg(7) & done_bit & srw_cfg(5 DOWNTO 4) &
            rcu_pass_flag & srw_cfg(2 DOWNTO 0)
            WHEN s_rw_cfg_decod =  '1' ELSE
```

Zdrojový kód 2.3: Ukázka čtení z registru *srw_cfg*.

2.1.3 Adresní blok

Uživatel může zahájit čtení nebo zápis pomocí aktivace signálu *start_bit* v registru *rw_cfg*. Po synchronizaci tohoto signálu (popsané v kapitole 1.4.1) se zaktivuje nejdříve adresní blok. Zde se zaktivuje sekvenční signál *exe_on*, který značí právě probíhající operaci. RTL návrh klopného obvodu tohoto signálu můžeme vidět na zdrojovém kódu 2.4. Tento signál zůstává aktivní až do té doby co je neaktivní signál *done_bit*. Tento signál se zaktivuje po provedení předepsaného počtu operací nebo pokud se při čtení špatně přečte aktuální slovo. V tu chvíli se testovací čip zastaví na chybné adrese a skončí.

```
p_exe_on_i : PROCESS (clk_sys, reset, addr_reset)
BEGIN
  IF (reset = '1' OR addr_reset = '1') THEN
    exe_on_i <= '0';
  ELSIF (clk_sys = '1' AND clk_sys'EVENT) THEN
    IF (start_bit = '1' AND done_i = '0') THEN
      exe_on_i <= '1';
    ELSE
      exe_on_i <= '0';
    END IF;
  END IF;
END PROCESS;
```

Zdrojový kód 2.4: Návrh klopného obvodu pro signál *exe_on*.

Zároveň se se signálem *exe_on* aktivuje kombinační signál *csn_gen_en*. Tento signál slouží k povolení generování signálů ve bloku pro čtení a zápis.

Adresní čítač je složen z klopných obvodů a jednoduchých kombinačních signálu, které můžeme vidět ve zdrojovém kódu 2.5. Klopné obvody jsou asynchronně resetované buď hlavním resetem vedoucím z „PADu“ nebo adresním resetem vedoucím z registrové mapy. Pokud je jeden z resetů aktivní, do adresních klopných obvodů se načte startovací adresa, která se dá nastavit skrz SPI rozhraní a je uložená v registrové mapě.

```
address_proc : PROCESS (clk_sys, reset, addr_reset, address_start)
BEGIN
  IF (reset = '1' OR addr_reset = '1') THEN
    address_counter_i <= address_start;
  ELSIF (clk_sys'EVENT AND clk_sys = '1') THEN
    IF (exe_on_i = '1' AND fail_bit_n_i = '1') THEN
      address_counter_i <= address_counter_next;
    END IF;
  END IF;
END PROCESS;

-- address combinational logic
-- incrementation & overflow when address is maximal
address_counter_next <= address_counter_i - address_max
                        WHEN address_counter_i = address_max ELSE
                        address_counter_i + "000000000001";
```

Zdrojový kód 2.5: Návrh adresního čítače.

Inkrementace adresy není podmíněna pouze náběžnou hranou hodin, ale také dvěma dalšími signály, které jsou uvedeny v kódu 2.5. Jedním z nich je *exe_on* a druhý je *fail_bit_n*. Tento sekvenční signál vychází z porovnávacího bloku. Je resetován vždy do aktivní polohy a deaktivuje se pouze při špatném porovnání přichozích dat a jejich předpokládané hodnoty.

Pokud jsou tedy signály *exe_on* a *fail_bit_n* aktivní, tak při náběžné hraně hodin se adresní registr inkrementuje. Inkrementační hodnota se tvoří v kombinační proměnné *address_counter_next*. Pokud je aktuální hodnota adresního registru rovna maximální možné adrese, tak se adresa vynuluje. V ostatních případech se adresa inkrementuje o jedničku.

S adresou je svázaný sekvenční signál *bw_sys* (signál „ByteWrite“, který je tvořen v systémové doméně, proto přípona *_sys*), který má stejné časovací požadavky jako adresní signál. Jeho hodnota je předem definovaná v registru *bw*, který je uložen v registrové mapě a je možné ho změnit skrz SPI rozhraní.

V testovacím čipu je speciální mód, který je ovládaný signálem pojmenovaným *bw_mode*. Pokud je tento signál nulový, hodnota *bw_sys* zůstává skrz celou operaci stejná a odpovídá hodnotě *bw* z registrové mapy. Pokud je ale *bw_mode* aktivní, v každém hodinovém taktu se hodnota *bw_sys* změní. V jednom taktu bude hodnota stejná, jako je uložená hodnota v registrové mapě, v druhém taktu bude ale hodnota *bw_sys* nulová. Tímto módem se tedy docílí zapsání nebo přečtení vždy každé druhé adresy.

Druhým inkrementačním signálem v adresním bloku je *noc_counter* („Number of Cycles“). Tento čítač (zobrazen ve zdrojovém kódu 2.6) je inkrementován pro kontrolu počtu provedených operací. Protože může uživatel provést různý počet operací, je potřeba tuto informaci kontrolovat odděleně od adresního čítače. Tento čítač je při resetování vynulovaný a vždy začíná z této pozice. Pokud je splněná podmínka inkrementace, tak se při náběžné hraně systémových hodin vždy zvýší tento čítač o jedničku.

```
p_noc_counter : PROCESS (clk_sys, reset, addr_reset)
BEGIN
  IF (reset = '1' OR addr_reset = '1') THEN
    noc_counter       <= (OTHERS => '0');
  ELSIF (clk_sys'EVENT AND clk_sys = '1') THEN
    IF (exe_on_i = '1' AND end_round = '0' AND
        done_i = '0' AND fail_bit_n_i = '1') THEN
      noc_counter       <= noc_counter_next;
    END IF;
  END IF;
END PROCESS;

-- noc counter combinational logic
noc_counter_next <= noc_counter + 1;

last_round_i <= '1' WHEN (noc_counter+1 = num_of_cycles OR
                          end_round = '1') ELSE '0';
end_round    <= '1' WHEN (noc_counter = num_of_cycles) ELSE '0';
```

Zdrojový kód 2.6: Návrh čítače pro kontrolu počtu provedených operací.

Podmínka inkrementace tohoto čítače je komplikovanější než u adresního čítače. *Noc_counter* potřebuje mít aktivní signály *exe_on* a *fail_bit_n* stejně jako adresní čítač. Také ale potřebuje mít neakitvní signál *end_round*, který značí že *noc_counter* vykonal všechny potřebné operace.

Poslední důležitou funkcí adresního bloku je tvorba datového vstupu paměti SRAM. Jeho všechny klopné obvody jsou při resetování vynulovány. Při náběžné hraně dochází poté k zapsání hodnot z registrové mapy do signálu *sram_din_sys*. Tento zápis má ale několik modifikací, které můžeme vidět ve zdrojovém kódu 2.7.

Pokud je uživatelem aktivovaný signál *direct_access*, tak zůstává signál *sram_din_sys* vždy stejný jako jsou data v registrové mapě. Při neaktivním signálu *direct_access* se před zahájením operace (signál *exe_on* je neaktivní) nastaví vstupní data opačně oproti probíhající operaci. Pokud je tedy adresa sudá, tak se data z registrové mapy invertují a při liché adrese zůstávají nezměněné. Ale pokud operace probíhá (signál *exe_on* je aktivní), tak se při sudé adrese propaguje nezměněný obsah z registrové mapy, naopak při liché adrese se vstup invertuje.

```
p_din : PROCESS (clk_sys, reset, addr_reset)
BEGIN
  IF (reset = '1' OR addr_reset = '1') THEN
    sram_din_sys_i <= (others => '0');
  ELSIF (clk_sys = '1' AND clk_sys'EVENT) THEN
    IF direct_access = '1' THEN
      sram_din_sys_i <= sram_din_spi;
    ELSIF exe_on_i = '0' THEN
      IF address_counter_i(0) = '0' THEN
        sram_din_sys_i <= NOT sram_din_spi;
      ELSE
        sram_din_sys_i <= sram_din_spi;
      END IF;
    ELSE
      IF address_counter_i(0) = '1' THEN
        sram_din_sys_i <= NOT sram_din_spi;
      ELSE
        sram_din_sys_i <= sram_din_spi;
      END IF;
    END IF;
  END IF;
END PROCESS;
```

Zdrojový kód 2.7: Návrh tvorby datového vstupu.

2.1.4 Blok pro vytváření operačních signálů

Blok nazvaný *sram_read_write* (zobrazený na obrázku 1.7) má za úkol generovat operační signály *csn* a *wr* pro paměť SRAM. Oba dva generované signály mají své klopné obvody, které jsou podobně konstruované. Ve zdrojovém kódu 2.8 vidíme pro ukázku generování signálu *wr*.

I tyto klopné obvody jsou resetovány asynchronně, buďto pomocí externího signálu *reset* nebo pomocí resetovacího signálu *sram_reset* z registrové mapy.

Generování operačních signálů je aktivováno signálem *csn_gen_en*, který byl popsán v minulé kapitole 2.1.3. Pokud je tento signál neaktivní, zůstávají signály ve výchozích pozicích. Při aktivním generování signálů potom záleží na operačním módu, který se volí pomocí signálu *sram_read_mode*. Tento signál byl popsán v předchozí kapitole 2.1.2.

Ve zdrojovém kódu 2.8 tedy vidíme, že pokud je při náběžné hraně hodin aktivní režim zápisu (hodnota signálu *sram_read_mode* je rovna „10“), tak se zaktivuje signál *dig_wr_i* a zůstane po celou dobu aktivní. Pokud je při náběžné hraně zapnutý střídavý režim, tak se hodnota zápisu invertuje a tím bude každou hodinovou periodu probíhat jiná operace.

Aktivace signálu *csn* se neřídí pomocí signálu *sram_read_mode*, ale pouze pomocí signálů *csn_gen_en* a *csn_mode*. Pokud je aktivní signál *csn_gen_en*, tak se při náběžné hraně hodin signál *csn* zaktivuje (pokud už samozřejmě aktivní nebyl). Při aktivním režimu *csn_mode* se ale při každé náběžné hraně

```
wr_proc : PROCESS (clk_sys, reset, sram_reset)
BEGIN
  IF (reset = '1' OR sram_reset = '1') THEN
    dig_wr_i <= '0';
  ELSIF (clk_sys = '1' AND clk_sys'EVENT) THEN
    IF (csn_gen_en = '1') THEN
      -- WRITE MODE
      IF (sram_read_mode = "10") THEN
        dig_wr_i <= '1';
      -- READ WRITE TOGGLING MODE
      ELSIF (sram_read_mode = "11") THEN
        dig_wr_i <= NOT dig_wr_i;
      ELSE
        dig_wr_i <= '0';
      END IF;
    ELSE
      dig_wr_i <= '0';
    END IF;
    ELSE
  END IF;
END PROCESS;
```

Zdrojový kód 2.8: Návrh klopného obvodu signálu *dig_wr*.

hodin zinvertuje signál *csn*. Tímto se střídá aktivita testovacího čipu, který jeden hodinový cyklus provádí operaci, ale v druhém cyklu čip paměť odpojí.

2.1.5 Zpožďovací blok

Zpoždění vstupních signálů paměti SRAM bylo popsáno teoreticky v kapitole 1.3.4, v této kapitole bude zpoždění rozebráno z hlediska implementačního. Podrobné blokové schéma je možné vidět na obrázku 2.1.

Zpoždění signálu probíhá dle schématu na obrázku 1.10. Každá zpožďovací linie má 63 zpožďovacích buněk, z jejich okolí vede 64 signálů s různým zpožděním (první signál je propagovaný bez zpoždění, poslední signál je zpožděn všemi 63 buňkami) a k nim je připojen multiplexorový strom, který má šest úrovní. Na konci tohoto stromu vychází signál s uživatelem zvoleným zpožděním. Uživatel si může skrz registrovou mapu nastavit libovolné zpoždění v uvedeném rozmezí.

Pokud máme ale více-bitové signály, zpoždění nejde jednodušše udělat skrz tuto cestu. Každý z bitů by mohl nabrat jiné zpoždění z fyzikálních důvodů a vytvářeli by se tak náhodné hodnoty, které v signálu nemůžou být. Proto byla pro všechny signály (i pro jednobitové signály) zvolena metoda, která je znázorněna ve zdrojovém kódu 2.9. Bylo to takto vytvořeno, aby všechny signály měli stejného zpoždění. Jsou dva signály, které nejsou snímány do klopného obvodu a to *dig_clk* a *dig_clk_dout* (můžeme vidět na obrázku 2.1). Oba jsou zpožděné z hodinového signálu a proto nemusí být

```
-- delay component for address clock
i_delay_selector_addr : delay_selector
  GENERIC MAP (
    dl_w        => dl_w
  )
  PORT MAP (
    clk_in      => clk_sys,
    mux_sel     => dl_addr,

    clk_out     => clk_addr
  );

-- SAMPLING process of address and ByteWrite with delayed clock
addr_proc : PROCESS (clk_addr, reset, addr_reset)
BEGIN
  IF (reset = '1' OR addr_reset = '1') THEN
    dig_addr_dff <= (OTHERS => '0');
    dig_bw_dff   <= (OTHERS => '1');
  ELSIF (clk_addr = '1' AND clk_addr'EVENT) THEN
    dig_addr_dff <= addr;
    dig_bw_dff   <= bw;
  END IF;
END PROCESS;
```

Zdrojový kód 2.9: RTL návrh pro tvorbu zpoždění na signálech *dig_addr* a *dig_bw*.

vedeny do klopného obvodu.

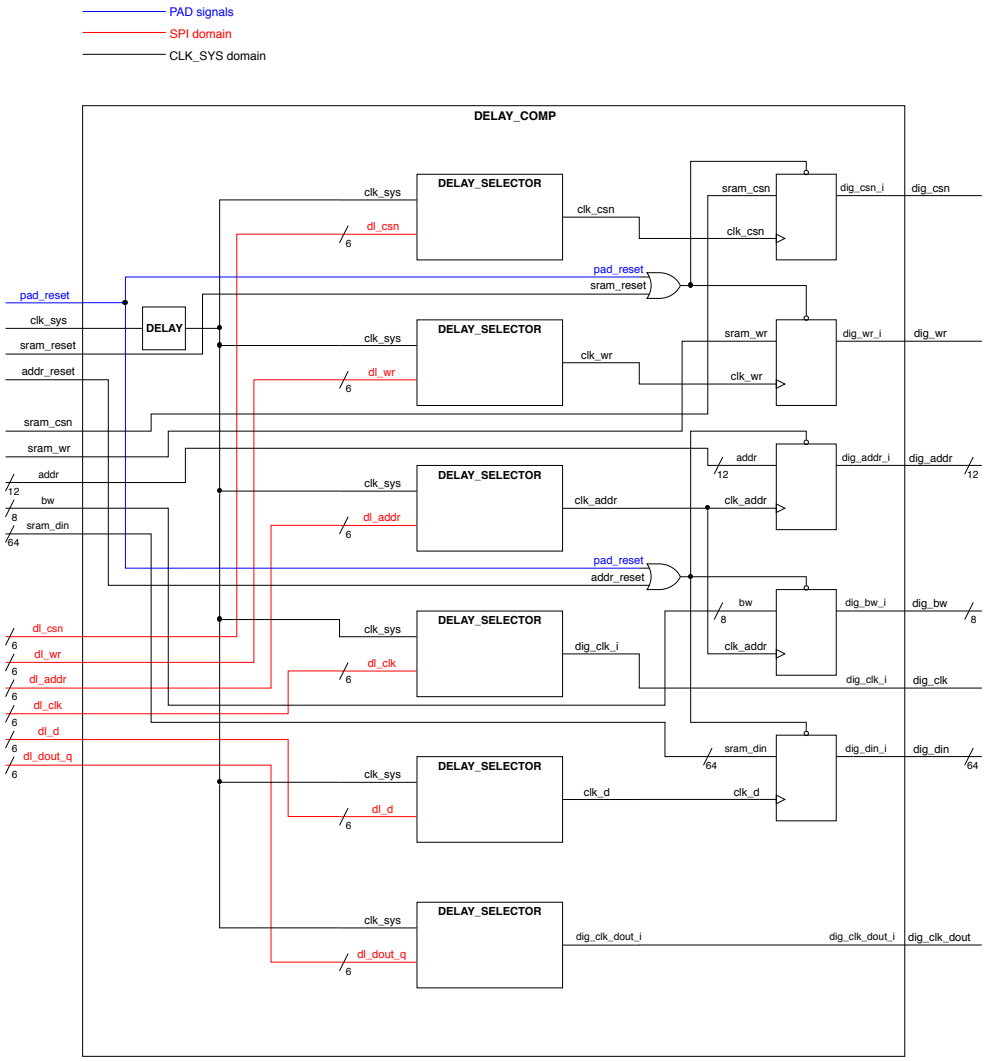
Zpožďování signálů tedy probíhá tak, že se zpozdí systémové hodiny a vytvoří se z nich specifické hodiny pro daný signál, v ukázce například zpožděné hodiny *clk_addr*, které se zpozdí pomocí nastavení registru *dl_addr*. Tuto hodnotu ovládá uživatel skrz SPI rozhraní. Tento hodinový signál poté vstupuje do klopného obvodu, který při jeho náběžné hraně navzorkuje systémový adresní signál *addr* a zpropaguje už tedy zpožděný adresní signál *dig_addr_dff*.

U tohoto návrhu klopného obvodu můžeme vidět, že signály *bw* a *addr* jsou vzájemně provázané a proto jsou ovládány pouze jedněmi zpožděnými hodinami. Pro ostatní signály byly vytvořeny oddělené hodiny, které se jmenují dle ovládaných signálů – *clk_csn*, *clk_wr* a *clk_din*.

2.1.6 Blok pro porovnávání dat

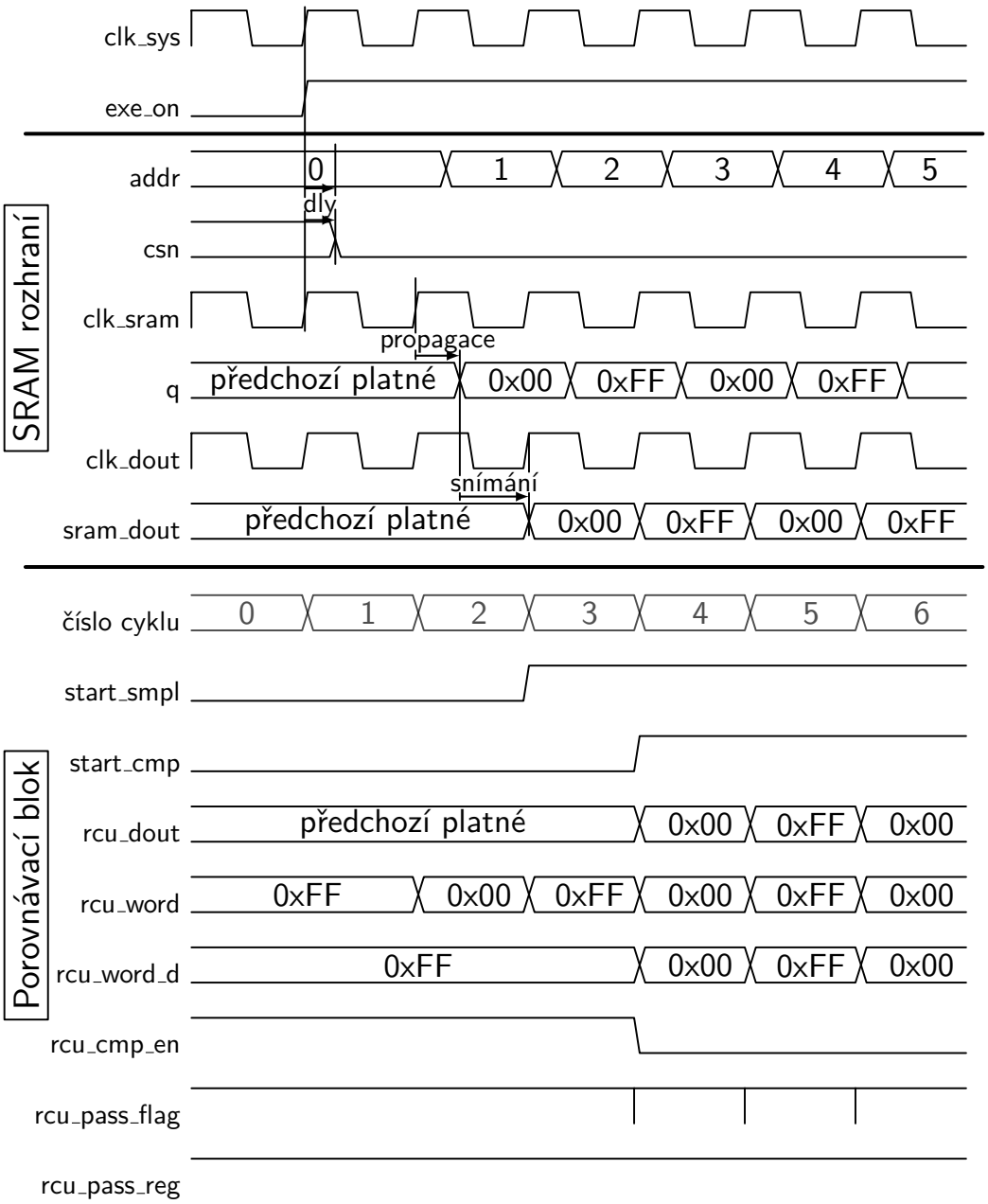
Porovnávání v testovacím čipu paměti SRAM generuje dva signály a to kombinační signál *rcu_pass_flag* a sekvenční signál *rcu_pass_flag_reg*. Porovnávací blok porovnává vždy načtená data z paměti SRAM s předpokládanými daty, které se mají vyskytovat na sudé nebo liché adrese. Tyto předpokládaná data jsou uložena v registrové mapě a uživatel je může měnit skrz SPI rozhraní.

Na obrázku 2.2 je znázorněna propagace dat z paměti a také průběh signálů při aktivním porovnávání čtených dat z paměti SRAM. Před tímto čtením byl



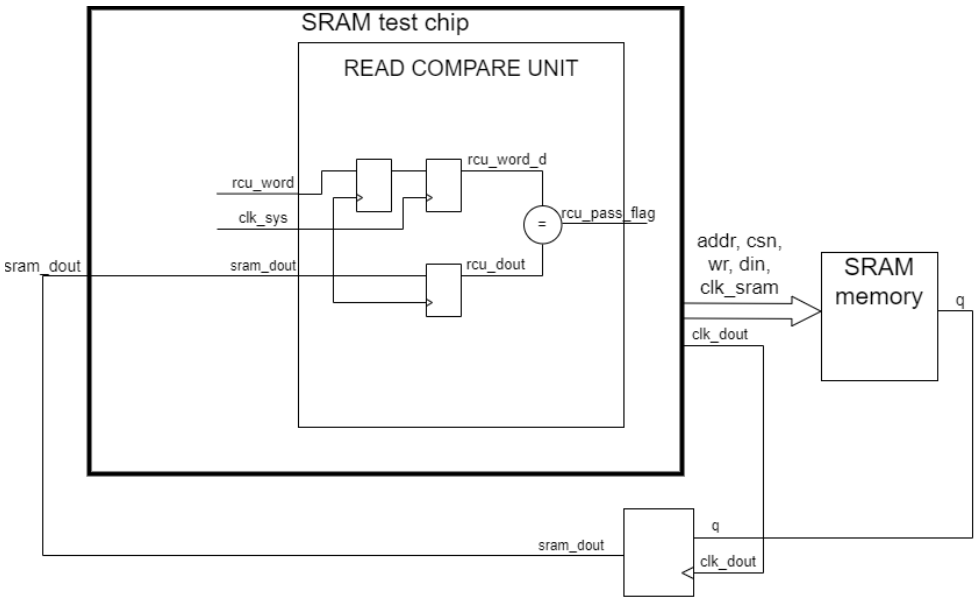
Obrázek 2.1: Detailní blokové schéma zpožďovacího bloku.

zapsáno do paměti šachovnicový tvar dat, tudíž je na sudých adresách hodnota výstupu „0x00“ a na lichých adresách je výstup paměti „0xFF“.



Obrázek 2.2: Diagram chování porovnávacího bloku při správném čtení.

Obrázek 2.3 slouží jako pomocný diagram, který ukazuje zjednodušené zapojení jednotlivých signálů, které se objevují v diagramu 2.2.



Obrázek 2.3: Zjednodušený diagram pro vysvětlení signálů z obrázku 2.2.

V horní části obrázku 2.2 vidíme rozhraní paměti SRAM při této operaci. Je zde znázorněno zpoždění hodinového signálu *clk_sram*, který je zpožděn pro splnění všech časových parametrů paměti. Ostatní ukázané signály (*addr* a *csn*) jsou oproti systémovým hodinám (zobrazeny v prvním řádku jako *clk_sys*) nezpožděny.

Při čtení produkuje pamět SRAM svůj výstup až po uplynutí přístupového času t_{qv}, který je delší než jeden čtecí cyklus této paměti. Lze tedy vidět, že na datový výstup *q* se hodnota z nulté adresy zpropagovala až při čtení další adresy.

Datový signál *q* je snímaný signálem *clk_dout*. Protože je tento signál ovládaný z testovacího čipu a je možné ho různě zpozdit, dokáže testovací čip specifikovat přesný přístupový čas, který je u každé vyrobené paměti lehce odlišný. Bohužel ale snímáním datového výstupu signálem *clk_dout* se hodnota signálu *sram_dout* zpožďuje o další hodinový cyklus.

Příprava dat k porovnání

Porovnávací blok byl navržen tak, aby počítal s dvou-hodinovým zpožděním, které je naznačeno v horní části obrázku 2.2. Vzhledem ke zpoždění je nutné upravit ovládací signály, aby byl porovnávací blok spuštěn ve správný čas. K aktivaci porovávacího bloku slouží signál *exe_on*, který aktivuje zároveň jak adresaci, tak i tvorbu signálů *csn* a *wr*. Tento signál se generuje v adresním bloku, popsaném kapitolou 2.1.3. Přesná implementace aktivace po-

```
rcu_read_mode <= '1' WHEN sram_read_mode = "01" OR
                          sram_read_mode = "11" ELSE '0';
p_start_compare : PROCESS (clk_sys, reset, rcu_reset,flag_reset)
BEGIN
  IF reset ='1' OR rcu_reset = '1' OR flag_reset='1' THEN
    start_compare_1 <= '0';
    start_compare_2 <= '0';
    start_compare_3 <= '0';
    start_compare_4 <= '0';
  ELSIF clk_sys'EVENT AND clk_sys='1' THEN
    IF (exe_on = '1')  THEN
      start_compare_1   <= rcu_read_mode;
      start_compare_2   <= start_compare_1;
      start_compare_3   <= start_compare_2;
      start_compare_4   <= start_compare_3;
    ELSE
      start_compare_1 <= '0';
      start_compare_2 <= '0';
      start_compare_3 <= '0';
      start_compare_4 <= '0';
    END IF;
  END IF;
END PROCESS;
```

Zdrojový kód 2.10: Aktivace porovnávacího bloku v RTL návrhu.

rovnávacího bloku je znázorněna ve zdrojovém kódu 2.10.

Signál *rcu_read_mode* je aktivní, kdykoli je testovací čip přepnut do režimu čtení, nebo do střídavého režimu čtení a zápisu. Poté se čeká na signál *exe_on*, který spustí aktivaci porovnávání. Na obrázku 2.2 můžeme vidět, že mezi náběžnou hranou signálu *exe_on* a prvními daty na signálu *sram_dout* jsou tři hodinové periody. O tolik cyklů se musí zpozdit start porovnávání. V případě střídavého režimu čtení a zápisu je třeba zpozdit start porovnávání dokonce o čtyři hodinové cykly, proto je také vytvořen signál *start_compare_4*.

Na obrázku 2.2 jsou znázorněny dva startovací signály – *start_smpl* a *start_cmp*. Signál *start_smpl* je ekvivalentem signálu *start_compare_2*, ukázaném ve zdrojovém kódu 2.10. Tento signál povoluje snímání dat ze signálu *sram_dout* do signálu *rcu_dout*, který je už porovnáván s předpokládanými hodnotami.

V závislosti na délce propagace dat z paměti je nutné zpozdit i očekávané hodnoty, vycházející z registrů *rcu_word_0* a *rcu_word_1*. Hodnoty *rcu_word* se nejdříve hradlem „AND“ upraví podle signálu *sram_msk_reg*, který značí aktivní bity výstupu z paměti. Poté se hodnota *rcu_word* určuje podle nejnižšího bitu adresy – pokud je tento bit lichý, předpokládaná hodnota výstupu bude ze sudého registru a naopak. Tímto krokem je řešen problém dlouhé přístupové doby paměti, která uvolní data z liché adresy až v dalším čtecím cyklu, kdy je adresa už sudá.

```
1 rcu_pass_flag_en <= '1' WHEN (start_compare_3 = '0' OR
2             (start_compare_4 = '0' AND sram_read_mode = "11") OR
3                         ( sram_read_mode = "10" )            OR
4                 ( toggle_bit = '1' AND sram_read_mode = "11") OR
5                 ( toggle_bit = '0' AND sram_csn_mode  = '1' ) OR
6                 ( bw       /= X"0" AND bw_mode        = '1' )
7                 )  ELSE '0';
```

Zdrojový kód 2.11: Podmínky pro povolení propagace výsledku porovnání.

Tento signál můžeme vidět na obrázku 2.2 pod názvem *rcu_word*. Pro správnou funkci porovnávání je ale tento signál potřeba zpozdit o dva hodinové cykly, stejně jako jsou data z paměti zpožděna svou cestou o dva hodinové takty. Takto zpožděný signál je ve zmiňovaném obrázku pod názvem *rcu_word_d*.

Povolení porovnání pomocí signálu *rcu_pass_flag_en*.

Druhý startovací signál *start_cmp* na obrázku 2.2 je ekvivalentní k signálu *start_compare_3*, který byl ukázán ve zdrojovém kódu 2.10. Tento signál v normálním režimu aktivuje signál *rcu_pass_flag_en*. Signál *rcu_pass_flag_en* slouží jako povolení zápisu do signálu *rcu_pass_flag*, který ukazuje aktuální výsledek porovnávání.

Protože testovací čip obsahuje různé režimy operací, je třeba aby signál *rcu_pass_flag_en* byl připraven na různé režimy. V ukázce zdrojového kódu 2.11 můžeme vidět jednotlivé podmínky, při kterých není povolena propagace výsledku porovnání.

Na prvním řádku ve zdrojovém kódu 2.11 je vidět základní podmínka normálního čtecího režimu – dokud je signál *start_compare_3* nulový, nebude se reagovat na výsledky porovnání. Splnění této podmínky můžeme vidět v diagramu na obrázku 2.2, kde se v okamžik aktivace signálu *start_cmp* aktivoval i signál *rcu_cmp_en*.

Na řádku číslo dva vidíme podmínku pro střídavý režim čtení a zápisu. Protože tento mód vždy začíná zápisem a čtení následuje až v druhém cyklu, je potřeba posunout i aktivaci porovnávání o cyklus později. Na třetím řádku je podmínka pro zapisovací režim, aby porovnávání při této operaci nikdy nebylo aktivní.

Další řádky aktivují porovnávání jen v ten hodinový cyklus, kdy jsou doopravdy data přijata a je třeba je porovnat. U režimu střídajícího zápis a čtení (čtvrtý řádek) je to vždy ten druhý cyklus (čtecí), který propaguje data do porovnávacího bloku. Proto je ve čtvrtém řádku signál *toggle_bit* v jedničce (kvůli zpoždění dat je tento signál v podmínce v jedničce a ne v nule). Narozdíl od předchozího režimu, na řádku číslo pět režim střídající signál *csn* aktivuje pamět v prvním cyklu, proto musí být pro aktivní porovnání signál *toggle_bit* nulový, ze stejného důvodu jako v předchozí podmínce. Poslední podmínka na

sedmém řádku ukazuje režim střídající hodnotu „ByteWrite“, která musí být nenulová, aby byl výstup propagován a mohl být porovnán.

Propagace signálů s výsledky

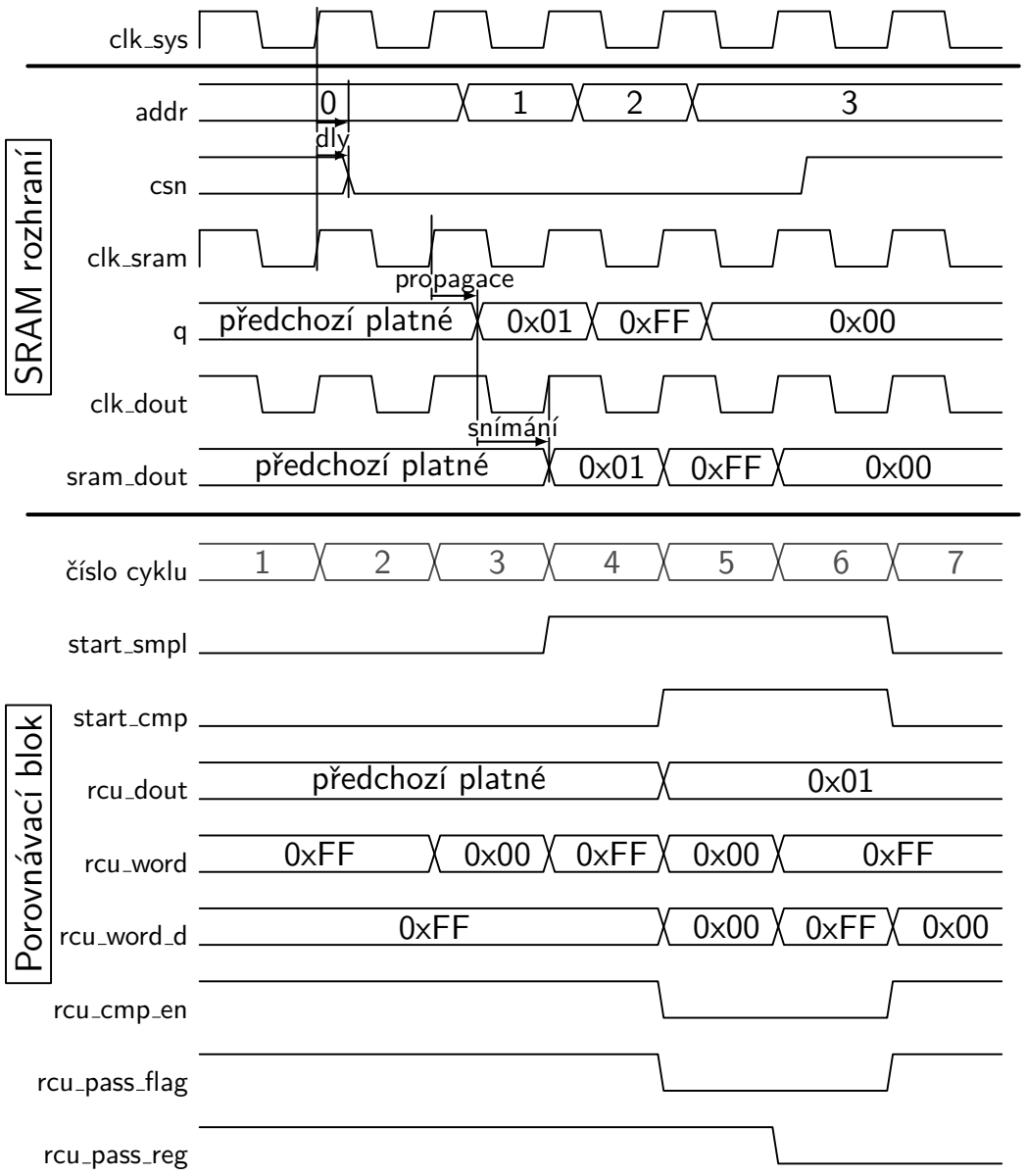
Po splnění podmínek aktivace signálu *rcu_pass_flag_en* se může začít propagovat výsledek porovnávání. Toto porovnávání se ukládá do kombinačního signálu *rcu_pass_flag*. Pokud je porovnání zakázáno, nebo je porovnání aktivováno a je v pořádku, signál je aktivní v hodnotě „1“. Pokud ale je porovnávání chybné, tak se signál *rcu_pass_flag* deaktivuje do hodnoty „0“.

Tato deaktivace může na krátkou dobu nastat i při správném porovnání a to při změně příchozích dat. Tento jev je možné vidět například na obrázku 2.2 při změně hodnot signálu *rcu_dout* z hodnot „0xFF“ na hodnoty „0x00“. V takovém případě se jedná o „glitch“ (krátká chybná změna dat, která se po určité době ustálí do původní polohy), který ale nijak neovlivní výsledek porovnávání.

Pro kontrolu celého porovnávání je vytvořen sekvenční signál *rcu_pass_flag_reg*. Tento signál má aktivní hodnotu v „1“ a do této hodnoty je i resetován na začátku operace. Pokud je porovnávání aktivní, tak se s náběžnou hranou hodin zkontroluje, jaká je hodnota na signálu *rcu_pass_flag*. Pokud je kontrolovaný signál aktivní, tak se nic nezmění. Ale pokud při náběžné hraně je tento signál neaktivní (to znamená, že porovnávání není správné), tak se signál *rcu_pass_flag_reg* deaktivuje do hodnoty „0“ a v té zůstane až do opětovného resetování.

Signál *rcu_pass_flag_reg* tedy ukazuje celkový výsledek porovnávání. Pokud nebyla nalezena ani jedna chyba, tento signál je po celou dobu stále aktivní. Pokud ale chyba byla nalezena, signál se deaktivuje a zastaví předčasně práci celého čipu. Taková situace je znázorněna v diagramu na obrázku 2.4. Tam můžeme vidět, že začátek operace probíhá úplně stejně jako při správném porovnávání. Ale data vycházející z nulté adresy neodpovídají šachovnicovému tvaru. Místo hodnot „0x00“ bylo z paměti přečteno „0x01“.

Když se data dostanou až do porovnávacího bloku, tak se při aktivaci porovnávání signálem *start_cmp* deaktivuje signál *rcu_pass_flag*. Při další náběžné hraně hodin se signál *rcu_pass_flag_reg* deaktivuje a zůstane v hodnotě „0“. Tento stav se dostane i do adresního bloku a tento stav zastaví další inkrementaci adresy. V porovnávacím bloku také dojde k zastavení přijímání dat a díky tomuto zastavení může uživatel poté přečíst, na jaké adrese se přečetly špatná data a jak ty data vypadaly. Kvůli zpoždění dat z paměti je vždy adresa posunuta o tři hodnoty dál, ale uživatel toto chování očekává.



Obrázek 2.4: Diagram chování porovnávacího bloku při chybném čtení.

2.2 Fyzická implementace

2.2.1 Syntéza

Po úspěšném RTL návrhu a jeho verifikaci popsané v kapitole 3.2 bylo potřeba vysyntetizovat vytvořený design. K syntéze se v zadavatelské firmě používá nástroj „Design Compiler“ od společnosti Synopsys[5]. Tento nástroj dokáže

projít zdrojové VHDL kódy a převést je do hradlové podoby.

Jako jeden z parametrů se tomuto nástroji dá knihovna, ze které má buňky vybírat. V této knihovně jsou obsaženy všechny vlastnosti logických buněk, takže tento program dokáže spočítat veškerá možná zpoždění a také celkovou velikost, kterou bude tento čip zabírat. Také se předloží tomuto nástroji minimální a maximální „pracovní roh“. Pracovní roh je souhrn podmínek, které určují jak rychle bude čip a jednotlivé buňky pracovat. Jednotlivými podmínkami rohu jsou napětí, technologický roh, teplota a pro všechny buňky pak zátěž a také rychlost náběžné a sestupné hrany. Maximální roh je tedy soustava podmínek, kdy čip bude pracovat nejpomaleji, v minimálním rohu bude naopak čip pracovat nejrychleji. V našem případě jsme měli tyto specifické rohy:

- maximální roh – pomalý proces, teplota je 85 °C a napětí je 1,08 V,
- minimální roh – rychlý proces, teplota je −40 °C a napětí je 1,32 V.

Pomocí těchto zadaných parametrů si „Design Compiler“ dokáže všechny zdrojové VHDL kódy převést do hradlové podoby vybrané knihovny. Knihovna byla vybrána od zadavatelské firmy EM Microelectronic, která tuto knihovnu bude používat při výrobě tohoto čipu. „Desing Compiler“ také optimalizuje všechny cesty, zda neobsahují nějaká hradla navíc, která nejsou potřebná, nebo zda nějaká celá část není nadbytečná. Pokud něco takového nastane, „Design Compiler“ tyto hradla nebo cestu smaže a tím vylepší výsledný design.

Syntézní omezení

Syntézní program potřebuje pro svou správnou funkčnost seznam omezení – „Constraints“. Těmito omezení uživatel specifikuje hodinové signály, jejich periodu a také specifikaci vnitřních signálů či výstupních kapacit. Všechny tyto omezení jsou sepsané v jednom souboru formátu „.sdc“ („Synopsys Design Constraints“), pro lepší editaci a čitelnost. Tento soubor se nachází u syntézního skriptu.

Pro správné syntetizovaní je nutné nastavit maximální hodinovou frekvenci všech hodinových vstupů. Toto omezení se vytváří příkazem „create_clock“, kde se specifikuje jaký vstupní signál bude hodinový a také jakou periodu budou dané hodiny mít. Tento čip má specifikované čtyři hodinové signály. Tři z nich vedou z vnějšku testovacího čipu – *pad_spi_clk*, *pad_clk_xtal* a *pad_clk_atb*. Čtvrtý hodinový signál se nazývá *spi_wr_data* a je vytvořen uvnitř čipu z hodinového signálu *pad_spi_clk*. Tento signál slouží k zápisu dat do registrové mapy a protože je odvozen od hodinového signálu *pad_spi_clk* je nutné ho vygenerovat příkazem „create_generated_clock“.

Dalším omezujícím pravidlem je příkaz „set_false_path“. Tento příkaz slouží k zneplatnění falešných cest, které se v designu můžou objevit. Syntézní program vždy hlídá jakékoli porušení předstihu a přesahu u všech klopných obvodů. Někdy je ale možné, že objeví takové porušení, které z hlediska funkčnosti

čipu nikdy nemůže nastat. K tomuto případu slouží příkaz „set_false_path“, který danou cestu označí za falešnou a syntézní program si toho nebude všímat.

Takové falešné cesty mohou nastat při vyšetřování časování ve zpožďovacím bloku. Protože je možnost nastavit libovolné zpoždění signálu, mohlo by se stát, že uživatel nastaví tak dlouhé zpoždění, že se data budou snímat ve zpožďovacím bloku chvíli před tím, než se v systémové doméně změní na nová. Tím pádem by zde vzniklo porušení přesahu a syntézní program toto porušení hlásí. V našem případě je to ale falešná cesta, protože uživatel vždy musí nastavit zpoždění takové, aby porušení přesahu nevytvořil. Proto je nutné pro všechny klopné obvody ve zpožďovacím bloku sepsat příkaz „set_false_path“ pro porušení přesahu.

Průběh syntézy

U tohoto čipu probíhala syntéza na dvě fáze. V první fázi bylo nutné vysyntetizovat samostatný zpožďovací blok. Tento blok byl navrhnut tak, aby byl symetrický a zpoždění v jeho multiplexorovém stromu bylo vyvážené. Syntéza by to ale nechápala a chtěla by tento strom prořezat a tím by už nebyl vyvážený. Proto byla v tomto bloku využita funkce „set_dont_touch“, která zakáže syntéznímu programu optimalizovat(mazat) označené buňky. Touto funkcí se označí všechny buňky uvnitř zpožďovacího bloku. Díky tomu je jisté, že strom pro zpoždění bude vždy úplný a symetrický. Tento blok se uloží a už se nebude syntetizovat znovu.

V druhém kroku je syntetizován zbytek celého designu. Zdrojový kód bloku, který byl syntetizován v předchozím kroku, je v tomto kroku vynechán a místo něho je dosazený již vysyntetizovaný blok. Tím se zaručí, že všechny zpožďovací bloky budou mít stejnou stavbu buněk. Tímto způsobem poté probíhá i procedura „Place and Route“ (umisťování vytvořených buněk na přesné místo na ploše („place“) a jejich propojování dráty („route“)). Tam se umístí na plochu první zpožďovací blok a další bloky se poté pouze zkopírují. Tím se zaručí i totožný čas šíření signálu skrz tyto bloky.

Kontrola výsledků

Výsledný design je nutný zkontrolovat, zda neztratil žádnou ze svých funkčností a syntéza proběhla v pořádku. První letmou kontrolou může být příkaz „link“ po běhu „Design Compiler“ programu. Tento příkaz vypíše všechny uzavřené bloky. Pokud proběhla syntéza v pořádku, tak je výstupem příkazu „link“ jeden blok, který obsahuje celý design. V případě chyby bychom na výstupu našli vícero bloků, které se nepovedlo syntetizovat nebo byly od designu odpojeny.

Pro kontrolu správnosti je nutné projít veškeré zprávy od syntézního programu, zda nenalezl nějaké chyby. Ať už chyby v časování, porušení předstihů či přesahů a nebo také chyby z důvodů nemožnosti syntetizovat určitou entitu.

Je důležité zkontrolovat vytvořené hodinové domény, zda syntézní program pochopil uživatelem zadané omezení a frekvence hodin odpovídá předpokládané frekvenci.

Dalším nástrojem pro kontrolu výsledné syntézy je formální verifikace, která se provede programem „Formality“, který je součástí balíčku „Design Compiler“ od společnosti Synopsys[5]. Tento program provede kontrolu, zda syntetizovaný netlist odpovídá RTL designu formálně i funkčně. Program „Formality“ proběhne v pořádku, pokud se žádné nesrovnalosti neukáží.

2.2.2 Place and Route

U tohoto návrhu dopadly všechny testy v pořádku a proto se syntetizovaný design předal „Place and Route“ inženýrovi, aby ho fyzicky implementoval. Pro „Place and Route“ je nezbytné předat vytvořený netlist (v jazyce Verilog) a také soubor s vytvořenými omezeními ve formátu „.sdc“.

Poté, co se čip fyzicky naimplementuje, tak je vrácen nový netlist (opět v jazyce Verilog) a k němu připojený soubor „SPEF“ – formát souboru určený pro přenos parazitních kapacit („Standard Parasitic Exchange Format“[6]). V něm jsou obsaženy veškeré parazitní kapacity všech použitých buněk, které slouží k sestavení možného zpoždění mezi buňkami[6].

Tyto dva soubory jsou použity pro tvorbu dalších souborů, které jsou důležité pro „post-layout“ verifikaci. Pomocí syntézních omezení, „SPEF“ souboru a také pomocí netlistu jsou v „PrimeTime“ (podprogram balíčku „DC Ultra“ od společnosti Synopsys[5]) programu vytvořeny soubory „SDF“ („Standart Delay Format[6]“). Tyto soubory slouží jako popis časových vlastností, které popisují celý implementovaný design[6]. Při verifikaci slouží tyto soubory jako náhrada fyzického čipu. Pro každý pracovní roh je vytvořen specifický „SDF“ soubor, který přísluší daným podmínkám.

2.2.3 Statická časová analýza

Statická časová analýza (STA) je metoda kontroly navrhnutého designu, zda splňuje veškeré časové požadavky a nedochází v něm k časovým hazardům (předstih, přesah a další problémy). STA se vykonává na syntetizovaném designu, při „Place and Route“ nebo i po provedení „Place and Route“. V zadavatelské firmě se tato analýza provádí programem „PrimeTime“ od společnosti Synopsys[5] a využívá se jako finální kontrola všech provedených prací až po dokončení „Place and Route“.

K správnému provedení STA je potřeba mít netlist s vytvořeným designem a k němu je potřeba soubor „SPEF“, který byl vytvořen při „Place and Route“ a který má v sobě uložené kapacity všech použitých buněk. Stejně jako u syntézního programu, i zde je potřebné mít seznam všech omezení („Constraints“) a také knihovnu s všemi použitelnými buňkami.

Pokud má STA vše potřebné pro její vykonání, tak začne zkoušet všechny možné kombinace šíření signálů v zadaných podmínkách (v pracovních rozích). Vždy měří délku šíření hodinového signálu a poté měří délku propagace datového signálu od první náběžné hrany hodin ke druhé. STA hlídá, zda nedojde k překročení předstihu nebo přesahu, tedy zda datový signál dorazí od změny při první náběžné hrany včas před druhou náběžnou hranou a také zda datový signál zůstane po první náběžné hraně dostatečně dlouho stabilní.

Výpis výsledků

Pro zjednodušený výpis potřebných informací se nachází v programu „PrimeTime“ příkaz „report_timing“. Tímto příkazem se můžou vypsat nejdůležitější cesty, v čele s tou kritickou – nejdelší cestou, která může způsobit nejpravděpodobněji problém. V každém výpisu cesty je vypsán počáteční a koncový bod (například výstup a vstup klopných obvodů) a poté všechny buňky, skrz které se signál propagoval. U každé buňky je vypsán čas, který byl třeba pro propagaci signálu. Pokud je čas nedostatečný a dochází k porušení předstihu nebo přesahu, STA vypíše čas, který chybí ke splnění tohoto požadavku a také klíčové slovo „Violated“ – porušený požadavek. Pokud jsou všechny požadavky splněny, STA vypíše čas, který ještě zbývá k hraně porušení a vypíše klíčové slovo „Met“ – splněné všechny požadavky.

V STA tohoto testovacího čipu nebyl nalezen žádný nesplněný požadavek, tudíž i analýza STA potvrdila, že se v čipu nenachází žádné chyby.

Kapitola 3

Verifikace

Verifikace se váže ruku v ruce s návrhem a je proto nezbytnou součástí této práce. Verifikace slouží k odhalení nesouladu funkčnosti čipu oproti specifikovaným vlastnostem nebo třeba pro zjištění maximální frekvence vytvořeného čipu.

Pro spuštění verifikace slouží dva skripty, které byly převzaty ze zadavatelské firmy, kde se už několik let využívají. První příkaz „recomp_all“ zkompiluje celý vytvořený design, verifikační soubory, pomocné knihovny a další potřebné soubory. Druhý příkaz „run_test“ spustí uživatelem zvolený test s příslušným nastavením.

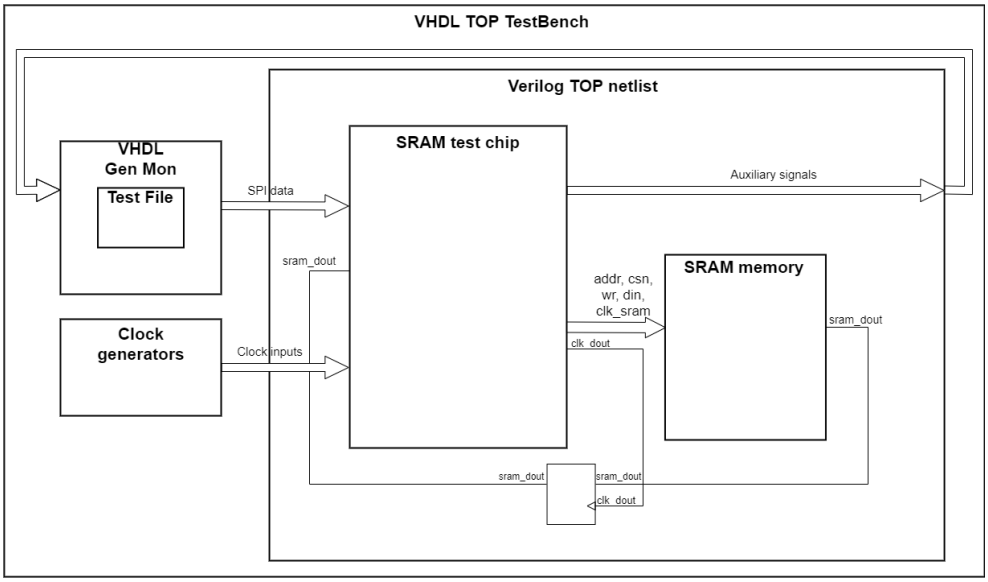
Mezi nastavení obou příkazů patří vybrání designu. Buď je možnost kompilovat a spouštet RTL design nebo například design po syntéze nebo po „Place and Route“. Pokud je zvoleno spuštění testu designu po provedení „Place and Route“, tak může uživatel vybrat například i specifický roh, který chce verifikovat. Pro snadné ladění nedokonalostí je možnost spuštění grafické vizualizace testu.

3.1 Testovací prostředí

Testovací prostředí se neskládá pouze z rozhraní vytvářeného čipu a paměti, ale obsahuje daleko více souborů a možností. Na obrázku 3.1 lze vidět zjednodušené zapojení testovacího čipu ve verifikačním prostředí, které bylo převzato ze zadavatelské firmy. Veškeré testovací soubory jsou již ve firmě několik let využívány, při verifikaci tohoto čipu byly vytvořeny pouze nové testy a v nich využívané pomocné procedury.

Každý test má svůj adresář, kde jsou uložené důležité věci pro jeho běh. Každý test musí obsahovat soubor „test.vhd“ (znázorněný na obrázku 3.1 jako „Test File“). Do tohoto souboru se zapisují veškeré pokyny, které jsou nezbytné k vykonání testu.

Obecná nastavení a všeobecně použitelné procedury se nachází v souboru „gen_mon.vhd“ (na obrázku 3.1 nazvaný jako „VHDL Gen Mon“), který ge-



Obrázek 3.1: Schéma zapojení testovacího čipu s pamětí při verifikaci.

neruje všechny ovládací signály dodávané do testovacího čipu. Jednotlivé hodinové generátory, které produkují hodinový signál do testovacího čipu jsou v oddělených entitách (na obrázku 3.1 se souhrně nazývají „Clock Generators“), ale dají se nastavovat a měnit jednotlivými procedurami, které se nachází v entitě „Gen Mon“.

Generující entita, generátory hodinového signálu a testovací čip s pamětí jsou propojené souborem a entitou „tb_top.vhd“ (na obrázku 3.1 označen jako „VHDL TOP TestBench“). Ten zastřešuje jednotlivé entity a zaručuje jejich správné zapojení, ať už se verifikuje RTL design nebo post-layout design. Vybrání designu se dá změnit parametrem skriptu „run_test“. Díky změně parametru dojde k načtení jiného konfiguračního souboru a v entitě „tb_top“ se zvolí jiná architektura s příslušným designem.

Testovací čip a paměť jsou připojené v entitě „Verilog Top netlist“. Tato entita byla dodána zadavatelem z „top-level“ analogového schématu, který přesně určuje, jaké má mít čip výstupní a vstupní signály a jak bude propojený s pamětí. Toto propojení je sice možné si navrhnout i svépomocí, ale protože tato entita odpovídá přesnému zapojení i na plošném spoji, tak je jisté, že testovací čip bude fungovat přesně tak, jak byl vytvořen a nedojde k žádné neočekávané chybě.

Model paměti SRAM je dodán zadavatelem stejně jako „top-level“ analogové schéma. Model obsahuje i všechny důležité parametry a časové kontroly, které jsou potřeba kontrolovat. Proto pokud dojde k porušení předstihu nebo přesahu vstupních signálů, model ihned oznámí chybu a test skončí s chybou.

3.2 RTL verifikace

RTL verifikace slouží v první řadě pro odhalení odlišností mezi navrhnutým designem a specifikovanými funkcemi. Proto jsem vytvořil verifikační plán, ve kterém jsem sepsal všechny specifikované funkce čipu pro které jsem následně vytvořil testy. Testy byly zaměřené takto:

- Základní test čtení a zápisu do paměti,
- Test registrové mapy,
- Testy výstupních signálů,
- Test mezních hranic při chybném čtení,
- Test režimu střídajícího čtení a zápis,
- Test režimu „Byte Write“,
- Testy signálu „Chip Select“ při čtení i zápisu a
- Test maximální frekvence.

3.2.1 Základní prvky testů

Všechny testy mají společné základní prvky, které jsou převzaté od zadavatelské firmy. Na začátku testu vždy probíhá procedura „gen_reset“ – procedura „generující reset“. Na začátku této procedury se zakáže činnost všech hodinových oscilátorů a nastaví se vnější signály do neutrální polohy – hodnoty „Z“ neboli vysoké impedance. Poté se aktivuje vnější asynchronní resetovací signál. Tím dojde k resetu celého čipu a po deaktivaci resetovacího signálu je testovací čip ve výchozí poloze.

Další převzatá procedura, která je společná pro všechny testy je procedura „spi_connect“. Tato procedura připraví rozhraní testovacího SPI modulu tak, aby bylo schopné vysílat signály do čipu a zároveň i přijímat signály od testovacího čipu. Po této proceduře se nastaví hodinový signál *clk_atb* do neaktivní polohy a poté už můžou přijít na řadu specifické procedury a funkce přímo pro dané testy.

Pro tento testovací čip byla vytvořena univerzální procedura *sram_start_operation*, která slouží k aktivaci testovacího čipu. Obsahuje veškeré nastavení, které uživatel může nastavit skrz její parametry a uživatel si nemusí pamatovat přesné názvy zapisovaných nebo čtených registrů. Její hlavičku můžete vidět ve zdrojovém kódu 3.1 Uživatel do parametrů této procedury uvede chtěnou operaci (čtení, zápis, střídání zápisu a čtení nebo další režimy), zvolí si počet slov, nad kterými se má operace provést, hodnoty datového vstupu, jednotlivých zpoždění signálů a předpokládaný výsledek operace.

```
PROCEDURE sram_start_operation (
  CONSTANT operation : std_logic_vector(1 DOWNTO 0) := C_OP_NONE;
  CONSTANT csn_mode   : std_logic                    := '0';
  CONSTANT bw_mode    : std_logic                    := '0';
  CONSTANT start_addr : std_logic_vector(15 DOWNTO 0) := X"0000";
  CONSTANT num_cycles : std_logic_vector(15 DOWNTO 0) := X"0400";
  CONSTANT datain  : std_logic_vector(31 DOWNTO 0) := X"FFFFFFFF";
  CONSTANT bw         : std_logic_vector(7 DOWNTO 0)  := X"0F";
  CONSTANT dl_clk     : std_logic_vector(7 DOWNTO 0) := X"00";
  CONSTANT dl_addr    : std_logic_vector(7 DOWNTO 0) := X"00";
  CONSTANT dl_wr      : std_logic_vector(7 DOWNTO 0) := X"00";
  CONSTANT dl_d       : std_logic_vector(7 DOWNTO 0) := X"00";
  CONSTANT dl_dout    : std_logic_vector(7 DOWNTO 0) := X"00";
  CONSTANT dl_csn     : std_logic_vector(7 DOWNTO 0) := X"00";
  CONSTANT divider    : natural := 0;
  CONSTANT clk_sel_atb: boolean := TRUE;
  CONSTANT direct_access : std_logic := '0';
  CONSTANT exp_rcu_pass_flag : std_logic := '1';
  CONSTANT exp_end_addr : std_logic_vector(15 DOWNTO 0) := X"0002"
) IS
```

Zdrojový kód 3.1: Hlavička procedury „sram_start_operation“.

Tato procedura se poté o vše postará. Zapíše veškeré požadované hodnoty do testovacího čipu, ke vstupním datům připraví odpovídajícím způsobem porovnávací registry a nastaví zpoždění operačních signálů. Po všech požadovaných zápisech se zablokují hodinové generátory a zapíše se do resetovacího registru, popsaném v kapitole 2.1.2. Po zapsání do resetovacího registru se zapíše do registru *rw_cfg* hodnota operace, která bude prováděna. Provede se resynchronizace nutná k propagaci všech změn uvnitř testovacího čipu a na závěr se aktivuje v registru *rw_cfg* startovací signál. Po zapsání tohoto signálu se aktivují i hodinové generátory a operace se aktivuje.

Tento styl ovládání operací má výhodu v jednotné aktivaci testovacího čipu. Vždy je tento postup stejný a proto se nemůžou stát neočekávané chyby, které by se mohli stát například vynecháním zápisu do nějakého podstatného registru.

Testy jsou implementovány tak, že po dokončení celé operace není nutné procházet celé průběhy signálů, zda fungovali korektně. Pokud v průběhu testu dojde k chybě, test tuto chybu vždy ohlásí. Po doběhnutí testů také vždy proběhne vyčtení důležitých registrů a kontrola, zda operace skončila tak, jak uživatel předpokládal. Vždy se přečte registr *rw_cfg*, ze kterého je jasné, zda čtení dopadlo bez chyby (bit *rcu_pass_flag* bude aktivní) nebo s chybou (bit bude neaktivní). Také se z tohoto registru pozná, zda operace doopravdy skončila (bit *done_bit* bude aktivní). Poté se také kontroluje adresa, na které skončil čip při provádění čtecí operace. Pokud bude totiž odlišná od předpokládané, je jasné, že při čtení došlo k chybě a testovací čip skončil dříve.

3.2.2 Popis jednotlivých testů

Jako první test pro tento čip byl vytvořen základní test, ve kterém se te i zapisuje do paměti. Je to obyčejný test, na kterém byla zkoušena základní funkčnost testovacího čipu. Tento test sloužil pro pochopení práce paměti SRAM, jak reaguje a jak například posílá datový signál do testovacího čipu. Pomocí něho byly odlaďovány obecné funkcionality čipu především vizuálně v grafickém prostředí.

Dalším základním testem je test registrové mapy. Tento test provádí několik základních operací s dostupnými registry a kontroluje správnost zapsání dat. Jeo úkol spočívá v zapsání dat do registrů, uložení zapsaných dat stejně, jak by to mělo probíhat uvnitř testovacího čipu a poté test čte data z registrů a porovnává je se svými uloženými hodnotami. Takto se nejdříve zapisují samé nuly, po skončení samých nul se zapíší do registrů samé jedničky.

Hlavní sada testů kontroluje chování čipu při různých dodatečných režimech, jako například při přímém zápisu, při střídání „Chip Select“ signálu nebo při střídání „Byte Write“ signálu. Jedná se o různé kombinace průběhů s bezchybným průběhem, poté s vnešením chyby do paměti a zkoušky, zda to čtecí cyklus přečte a zastaví se, nebo zda chybu mine z důvodů neaktivního „Chip Select“ signálu nebo jiných předpokládaných příčin. Také se testuje, kdy je zpoždění signálů ještě v pořádku a čtou se data správně, a kdy je zpoždění signálů již za hranou, a čtou se data jiná a tím pádem chybná.

Jeden z testů verifikuje správnou konečnou adresu a data po chybném čtení. Tento test vždy zapíše chybné data na určitou adresu a poté provede čtení celé paměti. Čtení je parametrizováno, že očekává chybné skončení na vybrané adrese. Po ukončení činnosti čipu je ještě přečten datový registr, který ukládá přečtené data z paměti. Z něho by vždy měl test přečíst chybná data, která proces čtení zastavila.

Test maximální frekvence slouží k potvrzení, že testovací čip je schopný fungovat správně na maximální dovolené frekvenci. Tento test při této nejvyšší frekvenci zapíše do celé paměti a poté paměť přečte.

Testy výstupních signálů mají za úkol vyvést z testovacího čipu všechny signály, které jsou přivedené k jejich multiplexoru. Poté se zaktivuje přivedený signál a ten může být zkontrolován. Tyto testy jsou jediné, které nejsou schopné se sami kontrolovat. K správné funkci těchto testů je nutné, aby je uživatel spustil v grafickém režimu a vždy se ujistil, že je na výstup propagovaný ten daný signál, který byl zvolený. Automatická kontrola a tvorba očekávaného průběhu všech signálů by byla složitá, proto byl zvolen tento způsob kontroly.

3.2.3 Výsledky RTL verifikace

Všechny testy jsou spouštěny po každé designové změně, která je provedena. Vždy je nutný úspěch ze všech testů, aby bylo potvrzeno, že designová změna

nepřináší žádné neočekávané chování čipu. Pokud se objeví chyba až při post-layout verifikaci, je třeba ji opravit znovu v RTL designu. Tím pádem se ale musí spustit veškerá verifikace na RTL úrovni, aby bylo jisté, že se opravou předchozí chyby nevytvořila nová a neočekávaná chyba.

V našem případě RTL verifikace po několika iteracích z RTL verifikace do post-layout verifikace a zase zpátky proběhla úspěšně. Pouze dva testy ukazují jedno chybové hlášení, ale to jsou zmíněné testy výstupních signálů. V těchto testech jsou vytvořena umělá upozornění na to, že daný test není automaticky kontrolován a je tedy potřeba ho vizuálně zkontrolovat.

3.3 Post-layout verifikace

Jak již bylo vysvětleno v kapitole 2.2.2, procedura „Place and Route“ byla prováděna expertem ze švýcarské mateřské firmy. Proto byla vždy důležitá domluva, ohledně provedených změn jak na straně RTL designu, tak na straně fyzické implementace. Při „Place and Route“ se snaží co nejvíce vybalancovat hodinový signál, aby přišel nejideálněji v jeden okamžik. Vždy to ale není 100% a proto se čip v post-layout verifikaci může chovat odlišně oproti RTL verifikaci.

Proto byla při této části verifikace nejdůležitější komunikace. „Place and Route“ inženýr musel správně popsat jeho řešení, které v aktuální verzi udělal a autor této práce musel správně popsat problém, který viděl v simulacích. Tento proces probíhá vždy na několik iterací a u tohoto čipu to nebylo jinak. Vždy bylo třeba upravit RTL design, provést RTL verifikaci, syntetizovat nový netlist, poté upozornit „Place and Route“ inženýra, který upraví fyzickou implementaci dle nového netlistu, provede potřebnou verifikaci a odešle zpátky post-layout netlist pro post-layout verifikaci.

3.3.1 Problematické části verifikace

Největší problém, který se objevil při verifikaci post-layout designu bylo balancování hodinového vstupu vedoucího do zpožďovacího bloku. Z digitálních simulací bylo vidět, že data do tohoto bloku přichází dříve než přichází do toho bloku hodinový signál. Proto bylo pro verifikaci nutné zpozdit příchozí hodinový signál do tohoto bloku. Ale „Place and Route“ inženýr neviděl tento problém a nedokázal pochopit, proč je to nutné. „Place and Route“ inženýrovi se tento problém v STA nikdy neobjevil, protože STA vždy počítala se zpožďovacím multiplexorem nastaveným na maximální hodnotu. Tímto způsobem bylo vybráno vždy nejdelší zpoždění a hodinový signál přicházel až po datech. Chyba se ale objevovala při post-layout verifikaci, kdy bylo zpoždění nastaveno na menší hodnoty a hodiny ve zpožďovacím bloku nedodržovaly správné časování signálů. Po vyřešení této nesrovnalosti stačilo najít správné zpoždění, kterým se musel zpozdit hodinový signál zpožďovacího bloku.

Dalším problémem bylo balancování výstupních multiplexorů. Bylo potřebné, aby se jednotlivé signály lišili v co nejmenším rozmezí. Pokud by byl jeden signál zpožděný ve výstupu například o 2 ns, ale druhý signál zpožděný pouze o 0,5 ns, uživatel by nerozpoznal, zda je první signál zpožděn zpožďovacím blokem nebo jinou příčinnou. Proto bylo nutné balancovat všechny výstupní signály tak, aby jejich propagace od vstupu do multiplexoru byla co nejmenší.

3.3.2 Výsledky post-layout verifikace

Výsledná verifikace post-layout designu probíhala primárně ve dvou pracovních rozích – v minimálním a maximálním rohu. V obou rozích proběhly po vyřešení zmíněných problémů všechny testy bez problémů a naprosto bez chyb. Chyby byly obsaženy pouze v testech sledujících výstupní signály. Tyto testy musely být vždy shlédnuté lidským okem, aby mohlo být prokázáno, že čip dokáže vyvést ven všechny zvolené signály.

3.4 Vektorové testy

Po úspěšném dokončení post-layout verifikace a prokázání bezchybné STA byl čip úspěšně přesunut do výroby. Avšak práce na tomto testovacím čipu neskončila. V normálním případě by bylo nutné vytvořit vektorové testy, které prokáží úspěšnou výrobu vytvořeného čipu. Ale protože tento čip funguje jako testovací čip paměti SRAM a není samostatný, vektorové testy tvořené pro tento čip slouží k otestování správné výroby právě připojených pamětí SRAM.

Při výrobě čipů ve velkých sériích, jsou čipy testovány v přístrojích, které testují celé wafery (pláty, na kterých se čipy vyrábí). Protože je ale tento čip vyráběn v malé sérii, testuje se jednotlivě v DIL (Dual In-Line) pouzdře v ATB přístroji. Tyto přístroje potřebují pro otestování čipu soubor („pattern“), který obsahuje na každém řádku jeden vektor – proto vektorové testy. Každý vektor obsahuje zvolené hodnoty vstupních signálů a také očekávané hodnoty výstupních signálů v daném čase. ATB přístroj poté vkládá do čipu v pravidelné periodě jednotlivé vektorové vstupní signály a kontroluje očekávané vektorové výstupy.

Pro vytvoření takových testů bylo potřeba přidat do testovacího prostředí pomocnou architekturu, která byla převzatá ze zadavatelské firmy. Tato architektura používá rozhraní, které v pravidelné periodě zaznamenává hodnoty vstupů a výstupů a ty ukládá do jednotlivých vektorů. Pomocí toho se tedy dokáže vytvořit soubor, který obsahuje vektorový popis průběhu operace testovacího čipu.

Z digitální simulace vektorového testu se vygeneruje vektorový soubor formátu „.sim_o“ – je to pouze záznam průběhu simulace. Pomocí převzatého skriptu se z tohoto souboru vytvoří soubor formátu „.jed“ – tento formát je standardizovaný pro využití v ATB přístrojích[7].

```
;        +--------------------------------- CH01 -- clk_atb (In)
;        |+-------------------------------- CH02 -- pad_reset (In)
;        ||+------------------------------- CH03 -- sdi (In)
;        |||+------------------------------ CH04 -- sdo (Out)
;        ||||+----------------------------- CH05 -- sck (In)
;        |||||+---------------------------- CH06 -- csb (In)
;        ||||||+--------------------------- CH07 -- dig_out1 (Out)
;        |||||||+-------------------------- CH08 -- dig_out2 (Out)
;        ||||||||+------------------------- CH09 -- out3 (Out)
;        |||||||||+------------------------ CH10 -- out4 (Out)
;        ||||||||||+----------------------- CH11 -- clk_xtal (In)
;        |||||||||||+---------------------- CH12 -- dummy12 (Out)
;        ||||||||||||+--------------------- CH13 -- dummy13 (Out)
;        |||||||||||||+-------------------- CH14 -- dummy14 (Out)
;        ||||||||||||||+------------------- CH15 -- dummy15 (Out)
;        |||||||||||||||+------------------ CH16 -- dummy16 (Out)
;        ||||||||||||||||+----------------- CH17 -- dummy17 (Out)
;        |||||||||||||||||+---------------- CH18 -- dummy18 (Out)
;        ||||||||||||||||||+--------------- CH19 -- dummy19 (Out)
;        |||||||||||||||||||+-------------- CH20 -- dummy20 (Out)
;        ||||||||||||||||||||+------------- CH21 -- dummy21 (Out)
;        |||||||||||||||||||||+------------ CH22 -- dummy22 (Out)
;        ||||||||||||||||||||||+----------- CH23 -- dummy23 (Out)
;        |||||||||||||||||||||||+---------- CH24 -- dummy24 (Out)
;        ||||||||||||||||||||||||+--------- CH25 -- dummy25 (Out)
;        |||||||||||||||||||||||||+-------- CH26 -- dummy26 (Out)
;        ||||||||||||||||||||||||||+------- CH27 -- dummy27 (Out)
;        |||||||||||||||||||||||||||+------ CH28 -- dummy28 (Out)
;        ||||||||||||||||||||||||||||+----- CH29 -- dummy29 (Out)
;        |||||||||||||||||||||||||||||+---- CH30 -- dummy30 (Out)
;        ||||||||||||||||||||||||||||||+--- CH31 -- dummy31 (Out)
;        |||||||||||||||||||||||||||||||+-- CH32 -- dummy32 (Out)
*FD      00000000000000000000000000000000
*V00000 000X00XXXXZXXXXXXXXXXXXXXXXXXXX CF
*V00001 000X10XXXXZXXXXXXXXXXXXXXXXXXXX TS0
```

Zdrojový kód 3.2: Ukázka hlavičky souboru „jed“.

3.4.1 Ukázka formátu „.jed“

Každý „.jed“ soubor obsahuje před výčtem vektorů hlavičku, ve které jsou specifikovány, na jakých pozicích jsou umístěny jednotlivé signály. Ve zdrojovém kódu 3.2 lze vidět vytvořenou hlavičku pro tento testovací čip a první dva vektory z testu.

Po hlavičce následují v souboru už pouze vektory, které začínají svým označením „*V“ a jejich číslem, a nebo komentáře, které začínají středníkem. Ve zdrojovém kódu 3.3 je vidět příklad zápisu do registru *bw_reg*.

Mezi vektory 34 až 49 můžeme vidět zápis adresy registru skrz rozhraní SPI. V pátém sloupci je uložený signál *pad_spi_clk*, který se v každém vektoru mění a tím vytváří hodinové pulsy. Ve třetím sloupci je signál *pad_spi_din*,

```
*V00034 000X00XXXXZXXXXXXXXXXXXXXXXXX
*V00035 000X10XXXXZXXXXXXXXXXXXXXXXXX
*V00036 000X00XXXXZXXXXXXXXXXXXXXXXXX
*V00037 000X10XXXXZXXXXXXXXXXXXXXXXXX
*V00038 000X00XXXXZXXXXXXXXXXXXXXXXXX
*V00039 000X10XXXXZXXXXXXXXXXXXXXXXXX
*V00040 000X00XXXXZXXXXXXXXXXXXXXXXXX
*V00041 000X10XXXXZXXXXXXXXXXXXXXXXXX
*V00042 001X00XXXXZXXXXXXXXXXXXXXXXXX
*V00043 001X10XXXXZXXXXXXXXXXXXXXXXXX
*V00044 000X00XXXXZXXXXXXXXXXXXXXXXXX
*V00045 000X10XXXXZXXXXXXXXXXXXXXXXXX
*V00046 001X00XXXXZXXXXXXXXXXXXXXXXXX
*V00047 001X10XXXXZXXXXXXXXXXXXXXXXXX
*V00048 001X00XXXXZXXXXXXXXXXXXXXXXXX
*V00049 001X10XXXXZXXXXXXXXXXXXXXXXXX
;0002:LBL_vec_write_ones_BW_REG__1:
*V00050 000X00XXXXZXXXXXXXXXXXXXXXXXX
*V00051 000X10XXXXZXXXXXXXXXXXXXXXXXX
*V00052 000X00XXXXZXXXXXXXXXXXXXXXXXX
*V00053 000X10XXXXZXXXXXXXXXXXXXXXXXX
*V00054 000X00XXXXZXXXXXXXXXXXXXXXXXX
*V00055 000X10XXXXZXXXXXXXXXXXXXXXXXX
*V00056 000X00XXXXZXXXXXXXXXXXXXXXXXX
*V00057 000X10XXXXZXXXXXXXXXXXXXXXXXX
*V00058 001X00XXXXZXXXXXXXXXXXXXXXXXX
*V00059 001X10XXXXZXXXXXXXXXXXXXXXXXX
*V00060 001X00XXXXZXXXXXXXXXXXXXXXXXX
*V00061 001X10XXXXZXXXXXXXXXXXXXXXXXX
*V00062 001X00XXXXZXXXXXXXXXXXXXXXXXX
*V00063 001X10XXXXZXXXXXXXXXXXXXXXXXX
*V00064 001X00XXXXZXXXXXXXXXXXXXXXXXX
*V00065 001X10XXXXZXXXXXXXXXXXXXXXXXX
*V00066 001X11XXXXZXXXXXXXXXXXXXXXXXX
```

Zdrojový kód 3.3: Ukázka zápisu do registru *bw_reg* v „.jed“ formátu.

sériový datový vstup do testovacího čipu. Pokud člověk přečte hodnoty na signálu *pad_spi_din*, které jsou vždy s náběžnou hranou, tak zjistí, že bylo skrz SPI přečtená binární hodnota „00001011“, která odpovídá hexadecimální hodnotě „0x0B“. Tato adresa odpovídá v testovacím čipu registru *bw_reg*.

Po zapsání adresy následuje komentář s popisem registru a následující vektory už značí přenos hodnoty, která má být uložena do zmíněného registru. Pokud člověk znovu postupně přečte hodnoty, které přichází v signálu *pad_spi_din* tak zjistí, že přijatá binární data jsou „00001111“, tedy hexadecimálně „0x0F“.

Ukončení SPI transakce se vždy projevuje zneplatněním signálu *pad_spi_csb*. Tento signál je v šestém sloupci a ve zdrojovém kódu můžeme vidět, že po dokončení přenosu adresy a dat k zápisu se tento signál ve vektoru číslo 66

deaktivoval.

3.4.2 Popis jednotlivých testů

Testy určené pro generování vektorů jsou odlišné od klasických testů určených pro verifikaci. V těchto testech nejde o dokonalé pokrytí všech možných případů, kde by se mohla vyskytovat chyba, ale jde o jednoduché operace s pamětí.

Tyto testy jsou poskládané ze základních SPI příkazů, skrz které se zapíší použité registry (předpokládaná čtená slova, datový vstup paměti, počet zápisů, startovací adresa a nastavení zpoždění paměťových signálů) a poté se skrz registr *rw_cfg* odstartuje příslušná operace. V průběhu celého testu se v pravidelné periodě snímají vstupy a výstupy testovacího čipu a zapisují se do vektorů.

Vektorové testy neobsahují žádné digitální kontroly, ale snímáním vstupů a výstupů vytváří tyto testy předpokládané data, které vstupují a přichází z paměti. Pokud by se při testování v přístroji ATB objevily jiné hodnoty, bude na to testovací inženýr upozorněn chybovým hlášením.

V případě této práce byly vytvořeny základní vektorové testy – zápis a čtení nul, jedniček nebo šachovnicového tvaru. Tyto testy jsou převzaty testovacím inženýrem, který je může kombinovat, upravovat a vytvářet komplikovanější testy pro nově vytvořenou paměť.

Závěr

Cílém této práce bylo analyzovat typy pamětí SRAM, navrhnout testovací čip pro určený typ paměti tohoto druhu, implementovat daný návrh a také ho zverifikovat. Za úkol bylo také provést Statickou časovou analýzu a vytvořit vektorové testy, které slouží k následnému testování SRAM pamětí.

Všechny tyto cíle byly splněny. Testovací čip byl navrhnut tak, že si uživatel může zvolit libovolné zpoždění signálů vedoucích do paměti tak, aby otestoval všechny časové požadavky paměti. Pomocí tohoto čipu lze charakterizovat velikost předstihu či přesahu paměti SRAM, nebo lze charakterizovat přístupovou dobu paměti. Do paměti lze zapsat stejná data do všech slov, nebo lze zapisovat na liché a sudé adresy jiná data. Lze také testovat správnost dat v paměti čtením, pro které slouží v testovacím čipu porovnávací blok. Blok vždy porovná data z paměti s předpokládanými hodnotami a pokud narazí na chybu, testování ukončí.

Tento testovací čip byl zverifikován jak na RTL úrovni, tak i na post-layout úrovni. Bez chyb prošla také Statická časová analýza. Po vyrobení tohoto čipu byly diskutovány s testovacím inženýrem vektorové testy, které byly zprovozněny a bylo potvrzeno, že tento testovací čip funguje bez problémů i v reálném provozu. V zadavatelské firmě se v této době používá pro charakterizaci časových parametrů nově vyrobených pamětí SRAM a díky jeho parametrizaci bude využit i u dalších modelů pamětí SRAM.

Tato práce byla výjimečná tím, že jsem se jako student podílel na reálném projektu od začátku mé práce s návrhovou firmou. Také není obvyklé, že jsem najednou komunikoval se třemi pobočkami firmy najednou – v české pobočce byl tento čip mnou navrhnut, verifikován a syntetizován, se švýcarskou pobočkou probíhala konzultace o podobě zadání (SRAM paměť byla navrhnuta ve Švýcarsku a proto se podíleli na návrhu), také byl švýcarským inženýrem proveden „Place and Route“ na plošný spoj a poté byl v americké pobočce testovacím inženýrem testován po výrobě a doteď je tam využíván.

Práce na tomto projektu mi dala spoustu znalostí a zkušeností, ať už z návrhového pohledu nebo také z pohledu jazykového. Být přítomen na me-

zinárodních poradách od začátku práce ve firmě bylo obohacující a jsem za tuto zkušenost velmi vděčný.

Bibliografie

1. GINOSAR, R. Metastability and Synchronizers: A Tutorial. *IEEE Design Test of Computers*. 2011, roč. 28, č. 5, s. 23–35. Dostupné z DOI: 10.1109/MDT.2011.113.
2. CUMMINGS, Clifford E; MILLS, Don. Synchronous Resets? Asynchronous Resets? I am so confused! How will I ever know which to use? In: *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers*. 2002.
3. SHAHDAD, M. An Overview of VHDL Language and Technology. In: *23rd ACM/IEEE Design Automation Conference*. 1986, s. 320–326. Dostupné z DOI: 10.1109/DAC.1986.1586107.
4. ŠŤASTNÝ, Jakub. FPGA prakticky. *BEN-technická literatura, Praha*. 2010.
5. *Design Compiler* [online]. Synopsys Inc. [cit. 2021-01-04]. Dostupné z: https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html.
6. IEEE Standard for Integrated Circuit (IC) Delay and Power Calculation System. *IEEE Std 1481-1999*. 2000, s. 1–400. Dostupné z DOI: 10.1109/IEEESTD.2000.7395044.
7. STANDARD, JEDEC. Standard Data Transfer Format Between Data Preparation System and Programmable Logic Device Programmer. *JESD3-C (Revision of JESD3-B)*. 1994, s. 1–35.

Příloha **A**

Seznam použitých zkratek

ADDR Address

ATB Automated Test Bench

BW Byte Write

CSB Chip Select Bit

CSN Chip Select Negated

DIL Dual In-Line

DRAM Dynamic Random Access Memory

NVM Non-Volatile Memory

PLL Phase-Locked Loop

RTL Register-transfer Level

SCK Serial Clock

SDC Synopsys Design Constraints

SDF Standart Delay Format

SDI Serial Data In

SDO Serial Data Out

SPEF Standard Parasitic Exchange Format

SPI Serial Peripheral Interface

SRAM Static Random Access Memory

STA Static Timing Analysis

VHDL Very High Speed Integrated Circuit Hardware Description Language

WR Write

Příloha **B**

Obsah přiloženého USB

- `readme.txt` ..... stručný popis obsahu USB
- `src` ..... adresář se zdrojovým souborem práce
  - `media` ..... grafické přílohy práce
  - `thesis.tex` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... adresář s textem práce
  - `thesis.pdf` ..... text práce ve formátu PDF