ՀՀ ԿՐԹՈՒԹՅԱՆ ԵՎ ԳԻՏՈՒԹՅԱՆ ՆԱԽԱՐԱՐՈՒԹՅՈՒՆ
ԵՐԵՎԱՆԻ ՊԵՏԱԿԱՆ ՀԱՄԱԼՍԱՐԱՆ

ՀԱՐՈՒԹՅՈՒՆ ՌՈՒԲԵՆԻ ԿՐԻԿՅԱՆ

ՎԵՐԱԿԱՌՈՒՑԱՎՈՐՎՈՂ ՀԱՇՎՈՂԱԿԱՆ ՀԱՄԱԿԱՐԳԵՐԻ ԿԱՌՈՒՑՄԱՆ ՄԻՋՈՑՆԵՐԻ ՄՇԱԿՈՒՄ ԵՎ ՀԵՏԱԶՈՏՈՒՄ

ԱՏԵՆԱԽՈՍՈՒԹՅՈՒՆ

Ե.27.01 «Էլեկտրոնիկա, միկրո և նանոէլեկտրոնիկա» մասնագիտությամբ տեխնիկական գիտությունների թեկնածուի գիտական աստիճանի համար

Գիտական ղեկավար՝ ՀՀ ԳԱԱ թղթակից անդամ,
ՀՀ գիտության վաստակավոր գործիչ,
տ.գ.դ., պրոֆ. Վ. Շ. Մելիքյան

ԵՐԵՎԱՆ 2016

ԲՈՎԱՆԴԱԿՈՒԹՅՈՒՆ

ՆԵՐԱԾՈՒԹՅՈՒՆ

<u>Թեմայի արդիականությունը:</u> Թվային ինտեգրալ սխեմաների (ԻՍ) զարգացումը հանգեցրել է ավելի մեծ չափերով, բազմաթիվ մտավոր սեփականության (ՄՍ) հանգույցների օգտագործմամբ տարբեր սինքրոազդանշանի տիրույթներով և պրոցեսորներով նախագծերի տարածմանը: Մեծ տարածում են գտել նաև կիսահաղորդչային բազմապրոցեսորային համակարգի (ԿԲՊՀ) տիպի նախագծերը: Այս ամենը հանգեցրել է թեստավորման տևողության զգալի աճի (2014 թվականին նախագծերի քանակը, որոնցում թեստավորումը կազմում է նախագծման տևողության 80% -ից ավելին, զգալիորեն աճել է): Թեստավորման տևողության և թեստավորման ճարտարագետների քանակի ղեկավարումը շատ կարևոր խնդիր է ժամանակակից թվային նախագծերում: Թեստավորման արտադրողականության աճը զգալիորեն զիջում է տարրերի քանակի աճին: Բացի դրանից, ժամանակակից թվային նախագծերում ծրագրային մասի նախագծումը գրեթե նույնքան ժամանակատար է, որքան ապարատային մասինը: Սովորական հաջորդական նախագծման դեպքում (երբ ծրագրային մասի նախագծումը սկսվում է առաջին ԻՍ-ի արտադրությունից հետո) ծրագրային մասի նախագծման մեծ տևողության հետևանքով հնարավոր է, որ նախատեսված ժամանակահատվածում նախագծի վաճառքի հետ կապված խնդիրներ առաջանան: Անհրաժեշտ է մշակել նոր միջոցներ՝ թեստավորման որակի, ինչպես նաև արդյունավետության բարձրացման համար:

Վերակառուցավորվող հաշվողական (ՎՀ) համակարգերի կիրառմամբ կառուցված նախատիպերի թեստավորման գործառույթների աճը կարող է զգալիորեն նվազեցնել թեստավորման տևողությունը և բարձրացնել որակը, ինչպես նաև ծրագրային մասի նախագծումը կարելի է սկսել առաջին ֆունկցիոնալ նախատիպի առկայության դեպքում: Ծրագրավորվող տրամաբանական ինտեգրալ սխեմաների (ԾՏԻՍ) զարգացումը տվել է ավելի մեծ ԻՍ-երի նախատիպերի կառուցման հնարավորություն (մեկ ԾՏԻՍ-ում հնարավոր է կառուցել 4 միլիոն տարրեր պարունակող ԻՍ-ի նախատիպ): Բացի դրանից, ժամանակակից ԾՏԻՍ-ների

մասնակի վերակառուցավորման հատկությունը կարող է նոր տիպի թեստավորման համակարգերի կառուցման հնարավորություն տալ:

ԾՏԻՍ նախատիպերով թեստավորման հիմնական խնդիրներից է սխալների հայտնաբերումը և տեղայնացումը: Այս գործընթացը, նախատիպի ներքին ազդանշանների տեսանելիության բացակայության պատճառով, իր բարդությամբ համադրելի է կիսահաղորդչում սխալների հայտնաբերմանը: Արդի ՎՀ համակարգերի կառուցման մեթոդներն ամբողջությամբ չեն ընդգրկում ԾՏԻՍ-ի զարգացման արդյունքում ստացված հնարավորությունները՝ նախատիպի ներքին ազդանշանների տեսանելիության բարձրացման տեսանկյունից: Այդ մեթոդները հնարավորություն են տալիս դիտարկելու ազդանշանները միայն շատ փոքր տևողությամբ: Թեստավորման տևողության նվազեցման համար անհրաժեշտ է բարելավել սխալների հայտնաբերման գործընթացը: Այդ նպատակով կարելի է կառուցել ՎՀ համակարգեր, որոնցով սինթեզվող հաստատումների միջոցով կկատարվի նախատիպի ներքին ազդանշանների դիտարկումը համակարգի աշխատանքի ընթացքում՝ առանց ընդհատման:

ԾՏԻՍ նախատիպի թեստավորման որակի գնահատման համար շատ կարևոր ցուցանիշի՝ ֆունկցիոնալ ծածկույթի հաշվարկի՝ բոլորի կողմից ընդունելի միջոց դեռևս չկա: Այդ պատճառով առաջարկվում է սինթեզվող հաստատումներն օգտագործել նաև այդ նպատակով:

Բացի վերը նշվածից, ՎՀ համակարգերը կարելի է նաև օգտագործել մի քանի սինքրոազդանշանով նախագծերի թեստավորման որակի բարձրացման նպատակով: Պարզապես դիտարկելով նախատիպի աշխատանքը՝ կարելի է ստուգել սինքրոազդանշանի տարբեր տիրույթներով անցնող ազդանշանների սինքրոնացման տրամաբանական հանգույցները: Սակայն սխալ վարքի դեպքում գրեթե հնարավոր չէ պարզել սխալի պատճառը: Այդ խնդրի լուծման նպատակով առաջարկվում է ներդնել հաստատումներ սինքրոնացման տրամաբանական հանգույցներում, որոնցով կդիտարկվեն համապատասխան ազդանշանները, և սխալների հայտնաբերումը կդառնա ավելի դյուրին:

Ատենախոսությունը նվիրված է վերը նկարագրված խնդիրների լուծման համար ՎՀ համակարգերի կառուցման միջոցների մշակմանը:

Հետազոտության առարկան: ՎՀ համակարգերի միջոցով կառուցված նախատիպերում սխալ վարքի հայտնաբերման, սխալների տեղայնացման, ֆունկցիոնալ ծածկույթի հաշվարկի եղանակները: Այդ միջոցների կիրառման ազդեցությունը հավելյալ ռեսուրսների օգտագործման և նախատիպի արագագործության վրա:

Աշխատանքի նպատակը: ՎՀ համակարգերի միջոցով կառուցված նախատիպերում սխալների հայտնաբերման և տեղայնացման, ինչպես նաև ֆունկցիոնալ ծածկույթի հաշվարկի միջոցների և թեստավորման ՎՀ համակարգերի կառուցման միջոցների մշակումը և մի քանի սինքրոազդանշանով նախագծերի ստուգման ՎՀ համակարգերի կառուցումը:

Հետազոտության մեթոդները: Ատենախոսության կատարման ընթացքում օգտագործվել են էլեկտրոնային սխեմաների մոդելավորման, ավտոմատացված համակարգերի կառուցման և տրամաբանական մակարդակների լեզուներով նկարագրման միջոցները:

Գիտական նորույթը.

- Մշակվել է համապիտանի հաջորդական դողի (ՀՀԴ) ղեկավարման թվային հանգույցի նախագծման նոր երթուղի՝ սինթեզվող հաստատումների կիրառմամբ, որը վերակառուցավորվող համակարգի ներքին ազդանշանների տեսանելիության բարձրացման հնարավորություն է տալիս:
- Առաջարկվել է նախագծման ընթացքում ներդրված հաստատումների կիրառումը նախատիպի թեստավորման որակի գնահատման ֆունկցիոնալ ծածկույթի հաշվարկի համար:
- Առաջարկվել է ՀՀԴ3.0 կապի մակարդակի թեստավորման վերակառուցավորվող հաշվողական համակարգի կառուցման միջոց՝ ֆունկցիոնալ թեստավորման հիմնական խնդիրների լուծման համար:

- Մշակվել է մի քանի սինքրոազդանշաններով թվային նախագծերի ամբողջական թեստավորման ՎՀ համակարգ, որում սինթեզվող հաստատումները ներդրվում են սինքրոազդանշանի տարբեր տիրույթների միջև անցում կատարող ազդանշանների սինքրոնացման տրամաբանական հանգույցներում՝ սխալների հայտնաբերման համար:
- Առաջարկվել է ծրագրավորվող ինտեգրալ սխեմայի մասնակի վերակառուցավորման հատկության օգտագործումը հավելյալ ռեսուրսների քանակի նվազեցման, ինչպես նաև արագագործության սահմանափակումների վերացման նպատակով:

<u>Պաշտպանության են ներկայացվում հետևյալ դրույթները.</u>

- Սինթեզվող հաստատումների կիրառմամբ վերակառուցավորվող համակարգի ներքին ազդանշանների տեսանելիության բարձրացման և ֆունկցիոնալ ծածկույթի հաշվարկի եղանակը.
- Համապիտանի հաջորդական դողի կապի մակարդակի թեստավորման ԾՏԻՍ նախատիպի կառուցման միջոցը.
- Մի քանի սինքրոազդանշանով թվային նախագծերի ամբողջական թեստավորման եղանակը.
- Նախատիպի կառուցման և հաստատումների ներդրման TAID ծրագրային և գործիքային միջավայրը:

<u>Աշխատանքի գործնական արժեքը:</u> ՎՀ համակարգերի կառուցման միջոցներն իրագործվել են TAID ծրագրային գործիքային միջավայրում, որը կատարում է ԾՏԻՍ նախատիպի կառուցման հոսքուղում տրված բոլոր քայլերը՝ օգտագործողին հարմար ինտերֆեյսով: Առաջարկված ՎՀ համակարգերի կառուցման միջոցը զգալիորեն պարզեցնում է սխալների հայտնաբերումը և տեղայնացումը, ինչպես նաև տալիս է ֆունկցիոնալ ծածկույթի հաշվարկի հնարավորություն ընդամենը 14-16% հավելյալ ռեսուրսների օգտագործմամբ և արագագործության չնչին նվազմամբ: TAID Instrumetor-ը հնարավորություն է տալիս օգտագործել ԾՏԻՍ-ի վերակառուցման հատկությունը՝ հավելյալ ռեսուրսների սահմանափակման և նախատիպի

արագագործության վրա թեստավորման համակարգի ազդեցության նվազեցման նպատակով:

Գիտական դրույթների հավաստիությունը հաստատված է բերված գիտական արդյունքների մոդելավորման միջոցով ստուգմամբ և գործնական փորձարկումների արդյունքների հետ համադրմամբ:

Ներդրումը: TAID ծրագրային միջոցը ներդրված է «Սինոփսիս Արմենիա» ՓԲԸ-ում: TAID Instrumentor ծրագրային գործիքը օգտագործվում է ՀՀԴ3.0, ՀՀԴ2.0 և ՀՀԴ1.1 ղեկավարման հանգույցների ապարատային մասի ԾՏԻՍ նախատիպերի կառուցման և սինթեզվող հաստատումների ներդրման նպատակով: TAID Debugger ծրագրային միջոցն օգտագործվում է ՀՀԴ նախատիպի թեստավորման ընթացքում դիտարկվող ազդանշանների սխալ վարքի հայտնաբերման և ֆունկցիոնալ ծածկույթի հաշվարկի համար:

Հրապարակումներ: Ատենախոսության հիմնական դրույթները ներկայացված են 5 գիտական հոդվածներում:

Ատենախոսության կառուցվածքը: Ատենախոսությունը բաղկացած է ներածությունից, 3 գլխից, եզրահանգումից, 105 անուն գրականության ցանկից և 3 հավելվածներից: Հիմնական տեքստը 121 էջ է, որը պարունակում է 52 նկար և 9 աղյուսակ: Աշխատանքի ընդհանուր ծավալը հավելվածների հետ 142 էջ է:

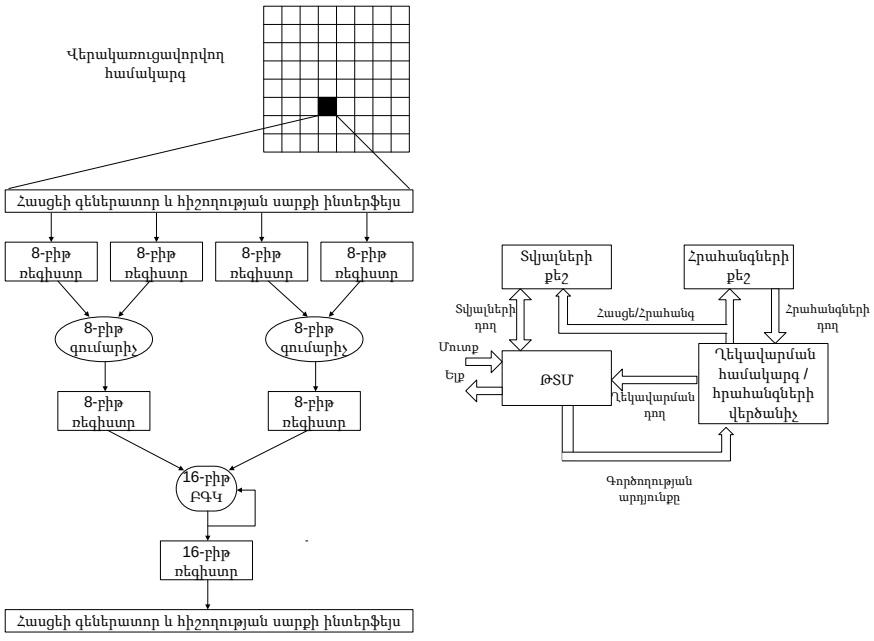
ԳԼՈՒԽ 1. ՎԵՐԱԿԱՌՈՒՑԱՎՈՐՎՈՂ ՀԱՇՎՈՂԱԿԱՆ ՀԱՄԱԿԱՐԳԵՐԻ ԿԱՌՈՒՑՄԱՆ ՄԻՋՈՑՆԵՐԻ ՄՇԱԿՄԱՆ և ՀԵՏԱԶՈՏՄԱՆ ԸՆԴՀԱՆՈՒՐ ՀԱՐՑԵՐ

1.1. Վերակառուցավորվող հաշվողական համակարգեր

Որոշակի ալգորիթմի իրականացման նպատակով գոյություն ունեն հաշվողական համակարգերի կառուցման երկու հիմնական եղանակներ՝ ապարատային և ծրագրային: Ապարատային եղանակով որոշակի գործողություն իրականացնելու նպատակով առաջինն օգտագործվում է մեկ խնդրի լուծման նպատակով կառուցված ԻՍ: Այդ ԻՍ-երն ունեն շատ բարձր արդյունավետություն և արագություն, քանի որ նախագծվում են միայն մեկ խնդրի լուծման նպատակով, և արտադրությունից հետո հնարավոր չէ ԻՍ-երի մեջ ֆունկցիոնալ փոփոխություններ կատարել: Պրոցեսորները հաշվարկը կատարելու ընթացքում կատարում են որոշակի հրահանգներ: Ծրագրային հրահանգները փոխելով՝ կարելի է հեշտությամբ փոխել համակարգի ֆունկցիոնալությունը: Այս եղանակն ավելի ճկուն է, բայց արագագործությունը ցածր է մեկ խնդրի համար նախագծված ԻՍ-երի համեմատ, քանի որ յուրաքանչյուր գործողություն կատարելու համար պրոցեսորը պետք է կարդա հրահանգը հիշողությունից, որոշի դրա նշանակությունը և կատարի համապատասխան գործողությունները, որի արդյունքում աճում է գործողություն կատարելու տևողությունը: ՎՀ համակարգերը միջանկյալ տեղ են գրավում ապարատային և ծրագրային միջոցների միջև՝ ունենալով ավելի բարձր արագագործություն ծրագրային և ավելի մեծ ճկունություն ապարատային միջոցների համեմատ:

Համակարգը, որում ծրագրավորվող տրամաբանությունն օգտագործվում է հաշվարկի կատարման նպատակով, կոչվում է ՎՀ համակարգ [1]: ՎՀ համակարգերը լայնորեն կիրառվում են ԻՍ-երի նախատիպի կառուցման և թեստավորման համար՝ որպես հարթակ: Այս համակարգերն առաջացել են 80-ական թվականներին՝ ԾՏԻՍ–ի վաճառքի համար հասանելիության ապահովման հետ մեկտեղ: ԾՏԻՍ-ի կառուցվածքը,

որը հնարավորություն է տալիս փոփոխել սարքի ֆունկցիոնալությունը անսահմանափակ քանակությամբ, խթանել է նոր բնագավառի առաջացումը, որում ապարատային բազմաթիվ ալգորիթմները կարելի է իրականցանել մեկ սարքի միջոցով, ինչպես ծրագրային ալգորիթմները՝ սովորական պրոցեսորով: Ալգորիթմի ապարատային իրականցման ավելի մեծ արագագործությունը համապատասխան ծրագրային ալգորիթմի համեմատ այս լուծումը դարձրել է գրավիչ հաշվողական մեծ հզորություն պահանջող խնդիրների լուծման, թվային ազդանշանների մշակման և այլ բնագավառներում [2-4]: Պատկերների մշակման 3 ալգորիթմի իրականացման պրոցեսորի (ծրագրային) և ԾՏԻՍ–ի արագագործությունների համեմատությունը ներկայացված է [5]–ում: Միևնույն ալգորիթմի ԾՏԻՍ–ով իրականացման արագությունը 7-12 անգամ մեծ է ծրագրային իրականացման համեմատ: ԾՏԻՍ-ները շատ ավելի արագագործ են որոշակի խնդիրների դեպքում, քանի որ դրանք կարելի է ձևափոխել յուրաքանչյուր ալգորիթմն իրականացնելիս: Վերակառուցավորվող և ֆիքսված հաշվողական համակարգերի հաշվարկի կատարման տարբերությունը ներկայացված է նկ.1.1–ում:



Նկ.1.1. Հաշվարկի կատարման ՎՀ համակարգի և հարվրդյան ճարտարապետությամբ պրոցեսորի համեմատությունը

ԾՏԻՍ-ը պարունակում է խնդրին համապատասխանող հաշվարկող միավորներ: Յուրաքանչյուր միավոր պարունակում է 8-բիթայնությամբ գումարիչներ և մեկ 16-բիթայնությամբ բազմապատկում, գումարում և կուտակում կատարող հանգույց, որոնք միմյանց հետ կապված են ռեգիստրներով: Ապարատային հասցեի գեներատորներն օգտագործվում են արտաքին հիշող սարքերից տվյալների ստացման նպատակով: Աջ մասում պատկերված է հարվրդյան ճարտարապետությամբ միկրոպրոցեսոր: Այն հաջորդաբար ստանում է հրահանգները դրանց համար նախատեսված քեշից: Տվյալները ստացվում են արտաքին հիշասարքից և տրվում թվաբանական տրամաբանական սարքին (ԹՏՍ)՝ լողացող կետով և ամբողջ թվերի համար նախատեսված ռեգիստրներով [6]: Վերակառուցավորվող համակարգերը հիմնականում բաղկացած են տվյալների հոսքուղուց և մինիմալ քանակությամբ ղեկավարող սխեմաներից այնպես, որ հաշվարկի համար նախատեսված միավորներն օգտագործվեն սինքրոազդանշանի յուրաքանչյուր պարբերությունում: Իսկ պրոցեսորների վրա կատարվող հաշվարկները հիմնված են հրահանգների հաջորդական հոսքի վրա և պարունակում են ճյուղավորումներ ու ցիկլեր: Վերակառուցավորվող համակարգերը կարող են ստանալ տվյալներ հիշողության բազմաթիվ հասցեներից տակտային ազդանշանի մեկ պարբերության ընթացքում, և տվյալների ստացման եղանակները կարելի է օպտիմալացնել խնդրին համապատասխան: Պրոցեսորները տվյալները ստանում են տվյալների քեշից, և դրա արդյունավետությունը կախված է հրահանգի կատարման ընթացքում տվյալի հասանելիությունից: Այսպիսով, ՎՀ-ի միջոցով հաշվարկը կատարելու համար անհրաժեշտ է այն բաժանել մասերի՝ զուգահեռ աշխատող, միմյանց հետ կապված երթուղիների միջոցով իրականացանելու նպատակով: Մինչդեռ սովորական պրոցեսորներն օգտագործում են հաջորդական հրահանգներ, իսկ հոսքուղու օգտագործումը և զուգահեռությունը կախված են պրոցեսորի կառուցվածքից:

ՎՀ համակարգի կառուցման գործընթացը հստակորեն ներկայացված է [1]–ում տրված օրինակում: Հետևյալ գործողությունն օգտագործվում է մատրիցների բազմապատկման, ինչպես նաև պատկերների մշակման մեջ փաթույթի գործողության իրականացման համար.

```
int   acc;
int coeff[n], data[n] ;
for(i=0; i<n; i++)
acc += coeff[i] * data[i];
```

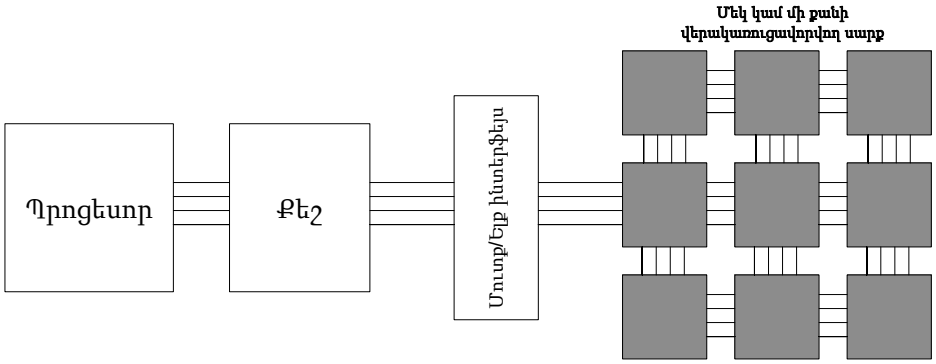
Այս գործողությունը ՎՀ-ում կատարելու համար օգտագործվում են ռեգիստրներ, բազմապատկում գումարում կուտակում (ԲԳԿ) կատարող տարրեր և հիշողության բլոկներ, որոնք, կախված խնդրից, կարելի է ծրագրավորել 8...64-բիթայնությամբ աշխատելու նպատակով: Տվյալներ կարելի է մատակարարել ինչպես ԻՍ-ի ներքին, այնպես էլ արտաքին հիշասարքերից: Թվաբանական գործողությունը կատարելու համար կարելի է օգտագործել ԾՏԻՍ-ում առկա ԲԳԿ միավորները, ինչպես նաև սարքերի նկարագրման լեզվով տրամադրվող, այսպես կոչված, 'սոֆթ' ՄՍ հանգույցներ: Ցիկլի գործողությունը պետք է իրականացվի տվյալների հոսքուղու և ղեկավարող սխեմաների միջոցով, որը կարելի է կատարել սինթեզի գործիքների միջոցով: Եվ վերջապես, համակարգի այլ կոմպոնենտների հետ աշխատելու համար անհրաժեշտ է ճշգրիտ իրականացնել ինտերֆեյսները: Արդյունքում կստանանք ներակայացված գործողությունը պարունակող խնդիրի լուծումը 10...100 անգամ ավելի մեծ արագագործությամբ՝ ծրագրային լուծման համեմատ, սակայն այս սխեման կառուցելու համար կպահանջվի կատարել շատ ավելի մեծ աշխատանք: Բազմաթիվ հետազոտություններ են կատարվում այդ աշխատանքն ավտոմատացնելու համար [7], և առաջարկվել են մեթոդներ որևէ ծրագրավորման լեզվից անմիջապես սարքի սինթեզի համար [8]:

1.1.1. Վերակառուցավորվող հաշվողական համակարգերի ճարտարապետությունները

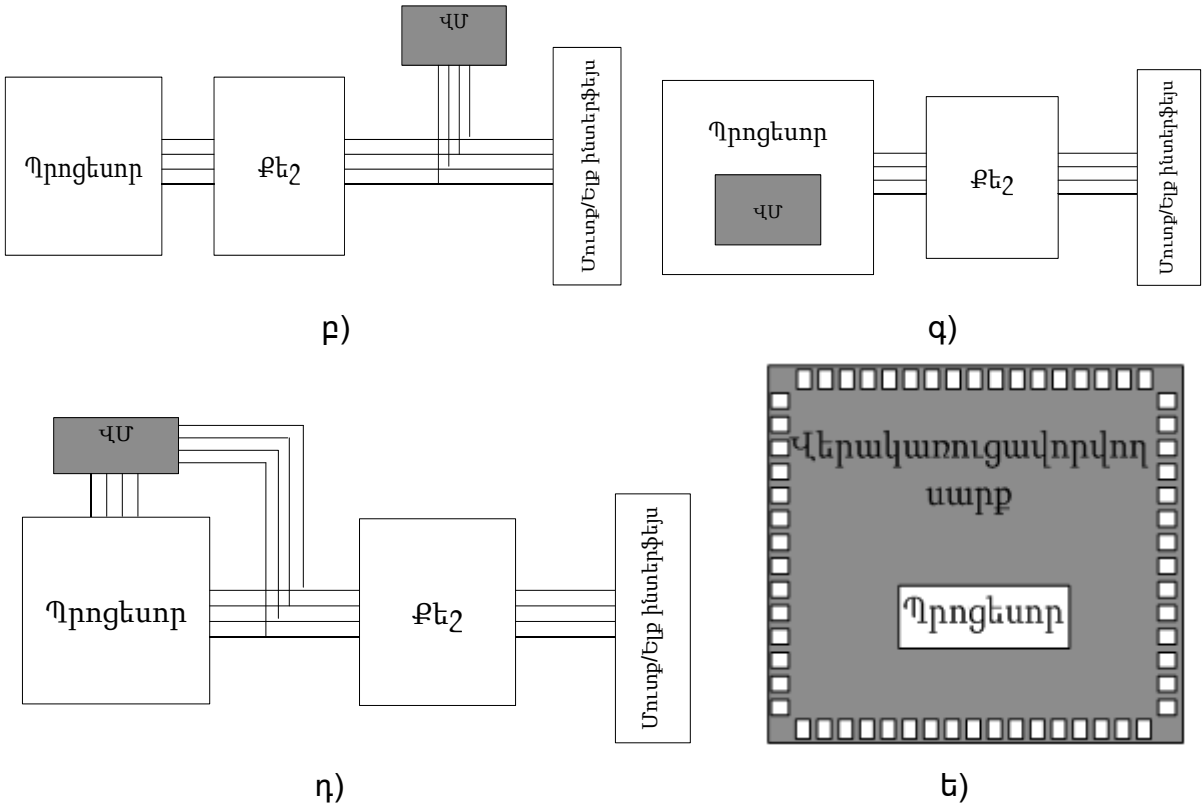
ՎՀ համակարգերը սովորաբար պարունակում են մեկ կամ մի քանի պրոցեսոր, վերակառուցավորվող մաս և հիշասարք: ՎՀ համակարգերը դասակարգվում են ըստ պրոցեսորի հետ կապի եղանակի: Բացի վերը նշվածից, այդ համակարգերը կարելի է դասակարգել ըստ որոշակի խնդրի լուծման համար սխեմայի կառուցման նպատակով օգտագործվող տարրերի չափի: Այդ տարրերը կարող են օգտագործվել ինչպես

որոշակի բիթի [9], այնպես էլ բիթերի խմբի [10] հետ գործողություններ կատարելու համար: Գոյություն ունեն պրոցեսորի հետ կապի 4 եղանակներ [11]:

Առաջին եղանակի դեպքում ՎՀ մասը օգտագործվում է որպես առանձին սարք և կապվում է հիշողության և պրոցեսորի հետ որևէ ստանդարտ ինտերֆեյսերով (նկ. 1.2ա): Այս դեպքում պրոցեսորի և ՎՀ մասի միջև տվյալների հոսքի արագությունը համեմատաբար ավելի փոքր է, և այդ պատճառով այն օգտագործվում է միայն, երբ հաշվարկի մեծ մասը պետք է կատարվի վերակառուցավորվող մասում [12, 13]: Երկրորդ եղանակի դեպքում կապը ավելի պարզ է և հետևաբար՝ ավելի էժան (նկ.1.2 բ և գ): ՎՀ մասը միացվում է պրոցեսորի համակարգային դողին: Այդպիսի կապի եղանակներ օգտագործվում են [14-19] հետազոտություններում: Երրորդի դեպքում կապի արագությունը շատ ավելի մեծ է, քանի որ ՎՀ-ն պրոցեսորի մեջ է (նկ. 1.2 դ): Այս եղանակը հնարավորություն է տալիս վերակառուցավորվող մասի միջոցով ստեղծելու նոր հրահանգներ յուրաքանչյուր խնդրի լուծման ընթացքում [20-24]: Չորրորդ եղանակի դեպքում պրոցեսորը ներդրված է ծրագրավորվող սարքի մեջ (նկ. 1.2 ե): Պրոցեսորը կարող է լինել ինչպես ՄՄ հանգույց, որը սովորաբար ներդրված է լինում ԾՏԻՍ-ի մեջ, այնպես էլ կառուցված լինել վերակառուցավորվող տարրերի միջոցով: Վերը նկարագրված կառուցվածքների պարամետրերը և օգտագործման բնագավառները ներկայացված են աղ. 1.1-ում:



ա)



Նկ. 1.2. ՎՀ համակարգերի դասակարգումը՝ ըստ պրոցեսորի հետ կապի եղանակի

Աղյուսակ 1.1

ՎՀ համակարգերի դասակարգումը և հիմնական պարամետրերը

ՎՀ անվանումը	**Պրոցեսորի և հիշասարքի միջև կապի արագությունը (մեգաբայթ/վ)**	**Հիշասարքի չափը (կիլոբայթ)**	**Նվազագույն ծրագրավորվող տարրի չափը**	**Կիրառությունը**
ա) RC2000[9]	528	152 հազար	1 բիթ	Պատկերների մշակում
բ) Pilchard[25] գ) Morphosys[26]	1064 800	20 2	1 բիթ 8 բիթ և ավել	Կոդավորում/ապակոդավորում
դ) Chess[21]	6400	12,288	8 բիթ և ավել	Պատկերների մշակում
ե)Xilinx UltraScale ԾՏԻՍ	>1600	24…94-հազար	1 բիթ և ավել	Բազմաթիվ բնագավառներ

ՎՀ համակարգերի կառուցման նպատակով հիմնականում օգտագործվում են ԾՏԻՍ-ներ, որոնք հնարավորություն են տալիս ծրագրավորելու սարքը բիթային

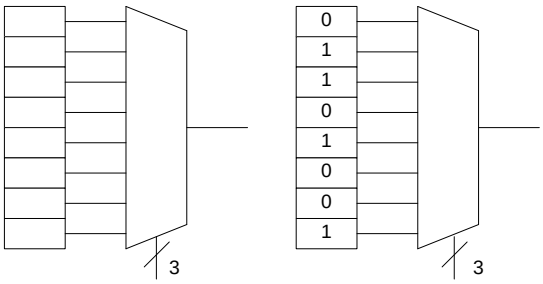
մակարդակում: Այս սարքերի տարածումը ՎՀ–ում պայմանավորված է վաճառքում հասանելի լինելով, ինչպես նաև ԾՏԻՍ–ի տարրերի քանակի՝ Մուրի օրենքով աճով [27]՝ ի տարբերություն այլ ՎՀ համակարգերի: ԾՏԻՍ–ի զարգացումը հնարավորություն է տվել կառուցելու միլիոնավոր ԻՍ տարրին համարժեք սխեմաներ մեկ սարքի մեջ: Xilinx UltraScale ԾՏԻՍ–ների նորագույն սերունդը տալիս է այդպիսի հնարավորություն [28]:

1.1.2. ԾՏԻՍ–ի կառուցվածքը և ծրագրավորումը

ԾՏԻՍ–ն ինտեգրալ սխեմա է, որը պարունակում է ծրագրավորվող տարրերի մատրից և ծրագրավորվող միջմիացումներ: Այն կարելի է ծրագրավորել որոշակի խնդրի լուծման համար արտադրությունից հետո, և խնդրի փոփոխման դեպքում վերածրագրավորել գործնականում անսահմանափակ քանակությամբ: ԾՏԻՍ–ի միջոցով խնդրի լուծման համար անհրաժեշտ սխեման կառուցելու համար օգտագործվող ռեսուրսները բաժանվում են հետևյալ մասերի.

- տրամաբանական բլոկներ,
- ծրագրավորվող միջմիացումներ:

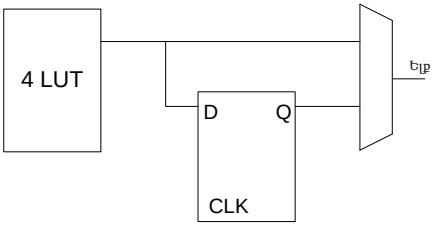
Ինչպես հայտնի է, կամայական գործողությունը կարելի է ներկայացնել Բուլյան հավասարման միջոցով [29]: Իսկ կամայական Բուլյան հավասարումն իրականացվում է իսկության աղյուսակի (ԻԱ) միջոցով: ԾՏԻՍ–ում ԻԱ-երն օգտագործվում են տրամաբանական գործողությունների կատարման նպատակով: Իսկության աղյուսակներին համապատասխան գործողությունը կատարող սարքը N-բիթ հիշողությամբ սարք է և N:1 մուլտիպլեքսեր (նկ.1.3): ԻԱ–ն կարող է կատարել կամայական գործողություն N մուտքերի միջև՝ պարզապես համապատասխան ձևով ծրագրավորելով հիշողության սարքը: ԾՏԻՍ-ն պարունակում է բազմաթիվ միևնույն մուտքերի քանակով ԻԱ–ներ: Օպտիմալ մուտքերի քանակի հաշվարկը լայնորեն ուսումնասիրվել է [30]:



Նկ.1.3. Իսկության աղյուսակների կառուցման նպատակով օգտագործվող սարքի կառուցվածքը և ծրագրավորման եղանակը

Մեծ մուտքերի քանակը՝ մի կողմից հնարավորություն է տալիս իրականացնելու ավելի բարդ գործողություններ մեկ տարրի միջոցով, որն իր հերթին նվազեցնում է միջմիացումներով պայմանավորված հապաղումը, քանի որ այլևս մի քանի տարրի միավորման անհրաժեշտություն չի լինում, սակայն, մյուս կողմից՝ մեծ մուտքերի քանակով ԻԱ-ները ավելի դանդաղագործ են, և ավելի փոքր մուտքերի քանակ պահանջող գործողություններ կատարելու համար ԻԱ–ն օգտագործվում է միայն մասնակի, որը հանգեցնում է ռեսուրսների կորստի: Չորս մուտքն օպտիմալ լուծում է, սակայն կախված խնդրից՝ մուտքերի քանակը կարող է տարբեր լինել [30]: Օրինակ՝ Xilinx–ի UltraScale սերնդում օգտագործվում են վեցմուտքանի ԻԱ–ներ [28]: Ծրագրավորվող բլոկում ԻԱ-ների օպտիմալ քանակը նույնպես լայնորեն ուսումնասիրված է [30]: Հաշվարկը ցույց է տվել, որ տրամաբանական գործողությունը մի քանի ԻԱ-ների միջև բաժանելու դեպքում կստացվեն ավելի փոքր մակերես և հապաղում: Այդ պատճառով արտադրվող ԾՏԻՍ–ների տրամաբանական բլոկները պարունակում են մի քանի ԻԱ:

Հաջորդական սխեմաների իրականցման համար ԾՏԻՍ–ի ծրագրավորվող բլոկները պարունակում են նաև տրիգերներ: Պարզագույն տրամաբանական հանգույցը ներկայացված է նկ.1.4–ում:



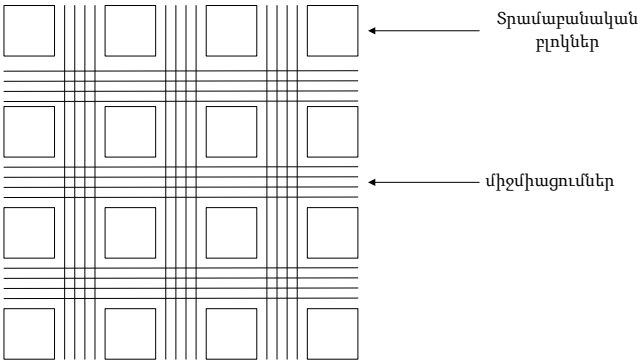
Նկ.1.4. Պարզագույն տրամաբանական հանգույցի կառուցվածքը

Այս հանգույցի ծրագրավորումը կատարվում է հետևյալ եղանակներով՝

- չորսմուտքանի ԻԱ-ի հիշողության բիթերը փոփոխելով,
- մուլտիպլեքսորի կառավարող ազդանշանը փոփոխելով,
- տրիգերի սկզբնական արժեքը համապատասխանաբար ընտրելով:

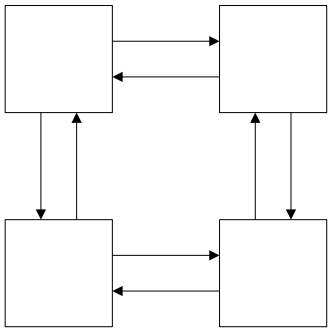
Արտադրության մեջ առկա ԾՏԻՍ–ների մեծ մասը օգտագործում է ստատիկ կամայական գրանցմամբ հիշասարքի (ՍԿԳՀ) բիթեր՝ վերը նշված կետերի արժեքները պահելու նպատակով: Փոփոխելով ՍԿԳՀ բիթերի պարունակությունը՝ կարելի է ծրագրավորել ԾՏԻՍ-ն:

ԻԱ-ն և D-տրիգերը միասին ԾՏԻՍ-ի տրամաբանական (ֆունկցիոնալ) հանգույցն են: ԾՏԻՍ-ը պարունակում է բազմաթիվ այդպիսի բլոկներ, որոնք անհրաժեշտ է միացնել մեծ խնդիրների լուծման համար նախատեսված սխեմաների կառուցման ընթացքում: Ժամանակակից ԾՏԻՍ-ներն ունեն այսպես կոչված կղզիների տիպի ճարտարապետություն: Տրամաբանական հանգույցները դասավորված են երկչափ զանգվածով շրջապատված միջմիացումներով (նկ.1.5): Բոլոր տրամաբանական գործողությունները կատարվում են այդ կղզյակներում, իսկ արդյունքը դուրս է բերվում արտաքին ելուստներ՝ ծրագրավորվող միջմիացումների միջոցով: Հաշվարկի բարդությունը հիմնականում սահմանափակվում է տրամաբանական բլոկների քանակով և միջմիացումների ցանցի բարդությամբ: Միջմիացումների կառուցվածքի ուսումնասիրությունն անհրաժեշտ է սկսել պարզագույն մոդելից, որն օգտագործվում է անմիջական հարևանությամբ գտնվող ֆունկցիոնալ բլոկների միջև կապ հաստատող միջմիացումների կառուցման համար (նկ.1.6): Բոլոր հարևան հանգույցների հետ կապի համար անհրաժեշտ են 4 մուտք և 4 ելք: Սա կապը հաստատելու պարզագույն եղանակն է, սակայն միջմիացումների մեծ քանակի դեպքում հապաղումների պատճառով առաջանում են խնդիրներ: 2x2–ի դեպքում հապաղումը զգալի չէ, սակայն իրավիճակը փոխվում է մեծ սխեմաների դեպքում: Օրինակ՝ երբ անհրաժեշտ է օգտագործել 1024x1024 ֆունկցիոնալ բլոկներ, հապաղումները զգալի են:



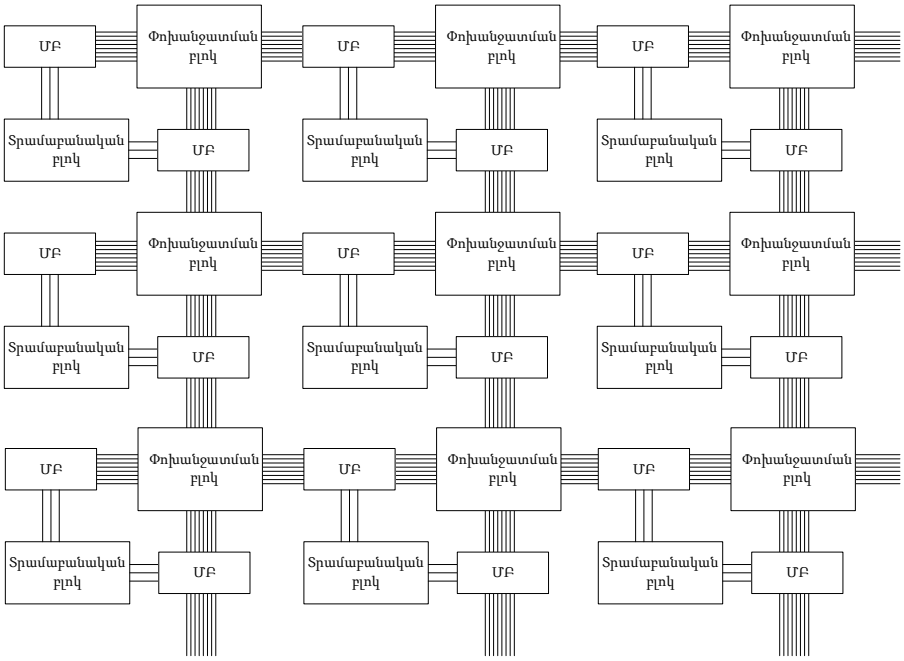
Նկ.1.5. Ժամանակակից ԾՏԻՍ–ների կառուցվածքը

Հապաղումը գծայնորեն աճում է 2 ֆունկցիոնալ բլոկների միջև հեռավորության աճի հետ, քանի որ միջմիացման ուղին դեպի նպատակակետ անցնում է բազմաթիվ ֆունկցիոնալ բլոկների միջով [1]: Վերը նկարագրված կապի միջոցը ժամանակակից ԾՏԻՍ-ներում օգտագործվում է միայն հարևան ֆունկցիոնալ բլոկների (ՖԲ) միջև կապեր հաստատելու նպատակով:



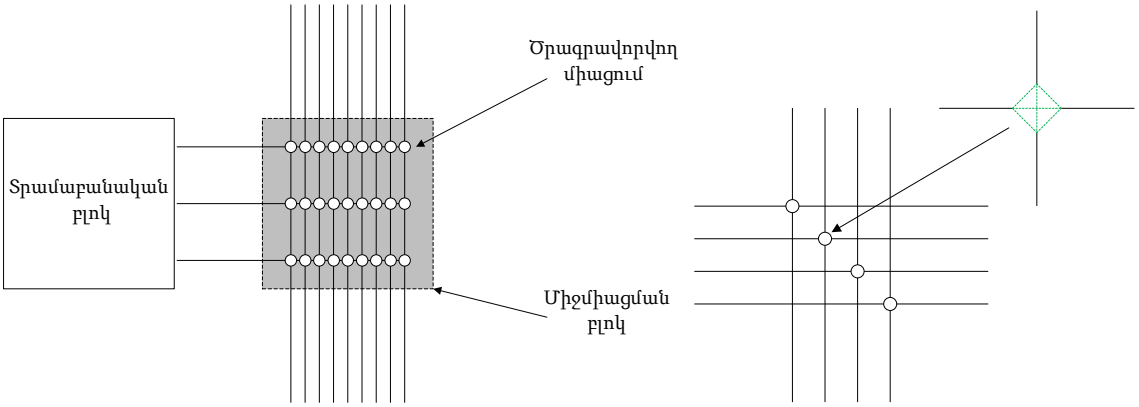
Նկ.1.6. ԾՏԻՍ–ում օգտագործվող պարզագույն միջմիացման եղանակը

Կառուցվող համակարգի բարդության աճի դեպքում միջմիացումների ցանցը կառուցելու համար անհրաժեշտ է ավելացնել լրացուցիչ հանգույցներ: Ժամանակակից ԾՏԻՍ-ների կառուցվածքը ավելի մոտ է նկ.1.7-ին: Այս կառուցվածքում միջմիացման բլոկը (ՄԲ) և փոխանջատման բլոկը (ՓԲ) օգտագործվում են միջմիացումները կառուցելիս: ՖԲ-ները միացվում են միջմիացման համար նախատեսված բլոկներին ՄԲ-ի միջոցով: ՄԲ-ն հնարավորություն է տալիս ՖԲ-ի մուտքերը/ ելքերը միացնել ուղղահայաց և հորիզոնական ուղիներին՝ մեծացնելով ծրագծման ճկունությունը: ՓԲ-ները տեղադրված են հորիզոնական և ուղղահայաց ուղիների հատման կետում: Հնարավոր միացման բոլոր եղանակները ներկայացված են կետագծերով (նկ 1.8.):



Նկ.1.7. ՄԲ-ների և ՓԲ-ների օգտագործմամբ միջմիացումների կառուցվածքը

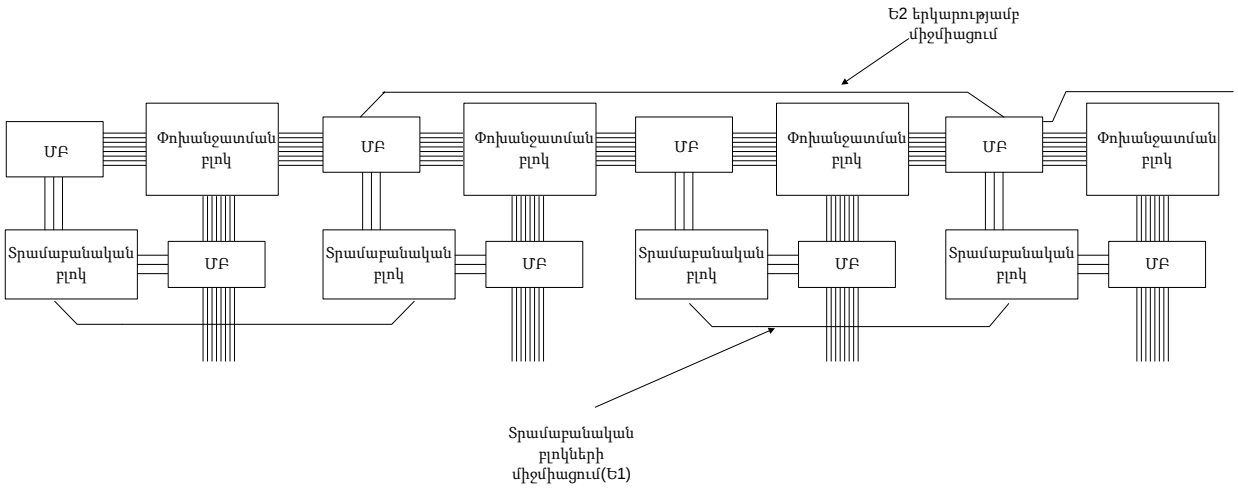
Միջմիացումների օպտիմալ կառուցվածք ստանալու համար կատարվել են բազմաթիվ ուսումնասիրություններ [31-34]: Այս կառուցվածքի դեպքում հեշտությամբ հնարավոր է իրականացնել հարևան երկու ՖԲ–ների միացումը, ինչպես նաև կապել ոչ հարևան ՖԲ-ները՝ առանց օգտագործելու այլ ՖԲ-ներ:



Նկ.1.8. ՓԲ-ի կառուցվածքը և կապի եղանակները հանգույցում

Առաջարկված մեթոդն ամբողջությամբ չի լուծում հապաղման խնդիրը: Այդ պատճառով ներմուծվել է միջմիացումների երկարության գաղափարը՝ կառուցվածքը բարելավելու համար: Օրինակ՝ Ե1 երկարությամբ միջմիացումը շրջանցում է միայն մեկ ՖԲ: Հնարավոր է ունենալ Ե4 և Ե8 երկարություններով միջմիացումներ, որոնք շրջանցում

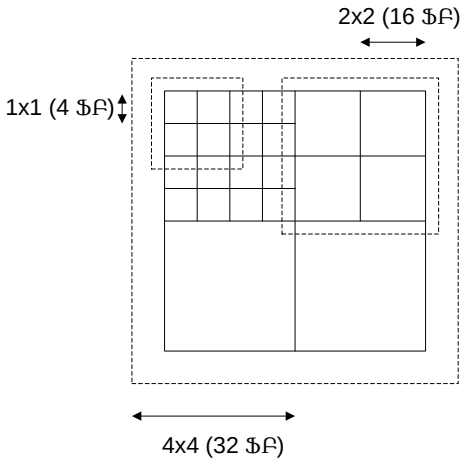
են համապատասխանաբար 4 և 8 ՖԲ: ՓԲ-ն կարող է օգտագործվել կարճ լարերից երկարներին անցնելու համար: Այս մոտեցումը թույլ է տալիս նվազեցնել հապաղումը՝ N հեռավորության վրա նվազեցնելով միջանկյալ ՓԲ-ների քանակը: Ե1 և Ե2 երկարություններով միջմիացումները ներկայացված են նկ.1.9-ում: Ե1–ն օգտագործվում է հարևան երկու բլոկների միացման համար, իսկ Ե2-ն ավելի մեծ հեռավորությամբ ՖԲ-ների համար՝ գրեթե միևնույն հապաղումով (Ե1):



Նկ.1.9. Ե1 և Ե2 տիպի միջմիացումներ

Հապաղման նվազեցման մեկ այլ մոտեցումը՝ երբ օգտագործվում է ծրագծման հիերարխիկ եղանակ, ներկայացված է նկ.1.10–ում: 1x1 չափի զանգվածը հիերարխիայի նվազագույն տարրն է, որում հնարավոր են միայն Նկ.1.6-ի տիպի միջմիացումներ: 2x2 –ը հիերարխիայի հաջորդ աստիճանն է, որը պարունակում է 16 ՖԲ, և նրա մեջ հնարավոր է օգտագործել ավելի երկար միջմիացումներ: Ճիշտ նախագծման դեպքում կարճ միջմիացումները գերակշռում են, իսկ երկար միջմիացումներն օգտագործվում են սահմանափակ քանակությամբ: Վերը նկարագրված միջմիացումների հանգույցները կարելի է կառավարել մեկ ՍԿԳՀ բիթով:

Ժամանակակից Xilinx-ի Virtex7 ԾՏԻՍ-ի կառուցվածքը ներկայացված է [35]-ում: Այն օգտագործում է միջմիացումների հիերարխիկ կառուցվածքը, որի փոքրագույն տարրը SLICE-ն է: SLICE-ը պարունակում է չորս վեցմուտքանի ԻԱ, 8 տրիգեր, մուլտիպլեքսոր և այլ տարրեր:



Նկ.1.10. Հիերարխիկ կառուցվածք ունեցող միջմիացումները

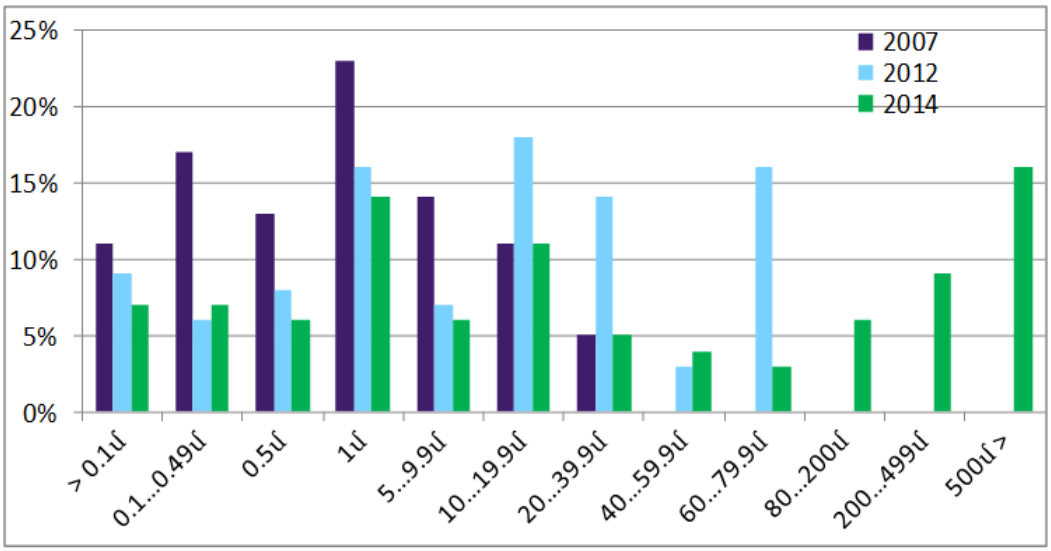
Բացի վերը նկարագրված ծրագրավորվող տրամաբանական մոդուլներից, ԾՏԻՍ-ը պարունակում է նաև ծրագծման ցանցին միացված թվային նախագծերում հաճախ օգտագործվող ԻՍ-ի հանգույցներ: Այս բլոկներն օգտագործվում են սպառվող, ծրագրավորվող ռեսուրսների նվազեցման, ինչպես նաև այն մասերի իրականացման համար, որոնք հնարավոր չէ կառուցել ԾՏԻՍ-ների միջոցով: Virtex7-ը պարունակում է հետևյալ բլոկները [35].

- Կամայական գրանցմամբ հիշասարքի բլոկներ,
- թվային ազդանշանների մշակման բլոկներ,
- տակտային ազդանշանի գեներատորներ և տակտային ազդանշանի բաշխման համար նախատեսված բլոկներ,
- արագագործ հաջորդական մուտքային/ելքային հանգույցներ,
- ապարատային ՄՍ հանգույցներ:

1.2. Թվային սխեմաների զարգացման միտումները

Թվային սխեմաների զարգացումը ցուցադրելու համար այս բաժնում ներկայացված են թվային ԻՍ-երի զարգացման միտումները տարրերի քանակի աճի, օգտագործվող ՄՍ հանգույցների քանակի, տարբեր սինքրոազդանշաններով տիրույթների քանակի աճի և այլ տեսանկյուններից: 2007, 2012 և 2014 թվականներին որոշակի տարրերի քանակով նախագծերի մասը բոլոր նախագծերի մեջ ներկայացված

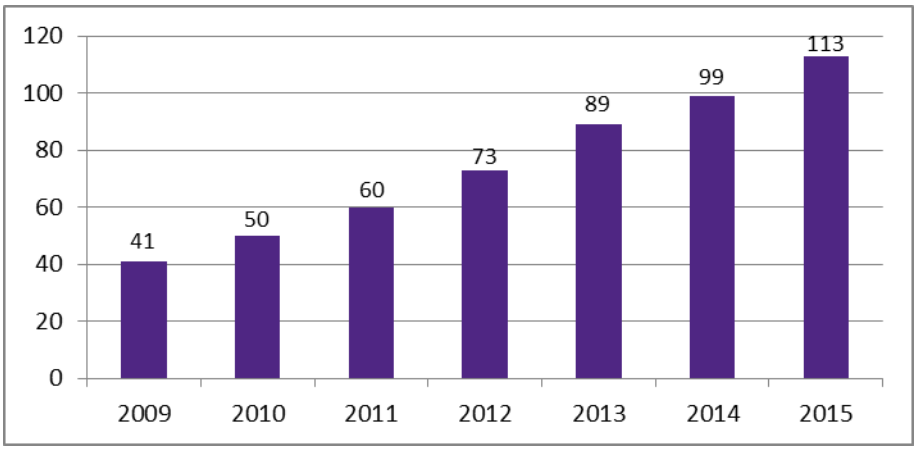
է նկ.1.11-ում (հիշասարքերը ուսումնասիրության ընթացքում հաշվի չեն առնվել) [36]: 2012 թվականին տարրերի քանակի վերին սահմանը 60 միլիոնն էր, իսկ 2014 թվականին այն անցել է 500 միլիոնը:



Նկ.1.11. Որոշակի տարրերի քանակով թվային նախագծերի տոկոսն ամբողջ նախագծերի մեջ 2007, 2012 և 2014 թվականներին

Հիմնական եզրակացությունն այն է, որ էլեկտրոնիկայի արդյունաբերությունն անցնում է ավելի մեծ չափերով նախագծերի: Ժամանակակից նախագծերի 31%-ի տարրերի քանակը գերազանցում է 80 միլիոնը, միևնույն ժամանակ նախագծերի 17%-ը 500 միլիոնից ավելի է: Պրոցեսորների նախագծերում տրանզիստորների քանակն աճել է 29000 տրանզիստորից X86-ում [37], որն արտադրվել 1979–ին, մինչև 1,72 միլիարդի 2005 թվականին արտադրված Dual Core Intanium-ում [38]: Փաստորեն, տարրերի քանակը 26 տարվա ընթացքում աճել է 60000 անգամ: Այս միտումը շարունակվելու է և ավելի փոքր տեխնոլոգիական գործընթացներ են կիրառվելու թվային նախագծերում: Բացի տարրերի քանակի աճից, կարևոր փոփոխություն է նաև այն, որ ԿԲՊՀ տիպի նախագծերի քանակը զգալիորեն աճել է [39]: Ուսումնասիրությունը ցույց է տալիս, որ նախագծերի քանակը, որոնք պարունակում են ներդրված պրոցեսոր, անցել է բոլոր նախագծերի 71%-ից, 41%-ը պարունակում է 2, իսկ 12%-ը՝ 8 և ավելի ներդրված պրոցեսորներ [36]: ԿԲՊՀ տիպի նախագծերն ունեն թեստավորման բարդության նոր մակարդակ ոչ ԿԲՊՀ տիպի նախագծերի համեմատ, քանի որ պետք է ստուգել նաև ծրագրի և ապարատի փոխհամաձայնեցումը, ինչպես

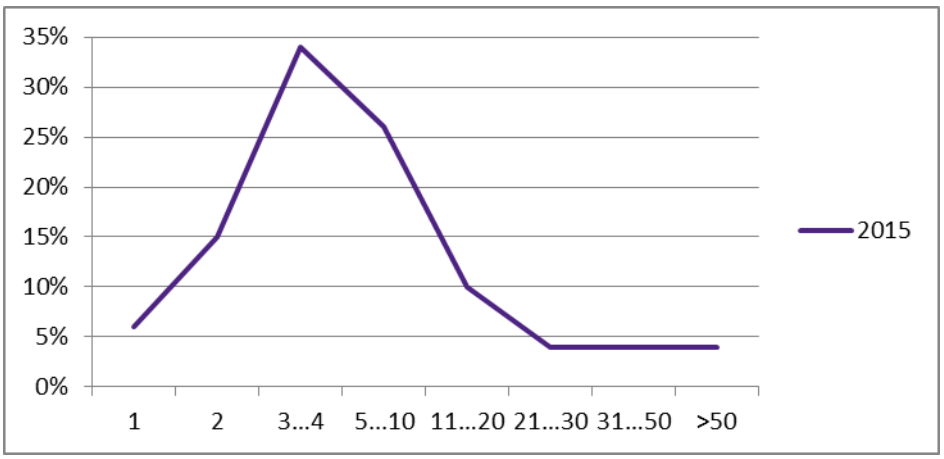
նաև ի հայտ է եկել նոր բարդ համակարգերի տեսակ՝ ցանց ԻՍ-ի վրա՝ ԿԲՊՀ-ում միջմիացումների կառուցման նպատակով [40]: Բացի ներդրված պրոցեսորների քանակի աճից, ժամանակակից ԿԲՊՀ–ներում հազվադեպ չէ 120-ից ավելի ՄՍ հանգույցների առկայությունը [39] (նկ. 1.12):



Նկ.1.12. ՄՍ հանգույցների քանակը ԻՍ-երում

ՄՍ հանգույցների կրկնակի օգտագործումը մի քանի ԿԲՊՀ–ներում նպաստում է նախագծման արտադրողականության աճին և որոշ դեպքերում նախատեսված ժամանակում նախագիծն ավարտելու միակ միջոցն է: ՄՍ հանգույցների 73%-ը տարբեր նախագծերում կրկնակի է օգտագործվում [36]: ՄՍ հանգույցների մեծ մասի տակտային ազդանշանների նկատմամբ պահանջները տարբեր են, և, հետևաբար, անհրաժեշտ են մեծ քանակությամբ տակտային ազդանշանների գեներատորներ, ինչն առաջացնում է թեստավորման նոր խնդիրներ՝ կապված ազդանշանների հետ, որոնք անցնում են ոչ սինքրոն տակտային ազդանշանների տիրույթներով: Նախագծերի 91%-ն ունի մեկից ավելի ոչ սինքրոն տակտային ազդանշան [36] (նկ.1.13):

Մի քանի ասինքրոն տակտային ազդանշան պարունակող նախագծերի թեստավորման խնդրի ուսումնասիրման ընթացքում պետք է նկատի ունենալ, որ մետաստաբիլության երևույթի հետ կապված համակարգային խափանումները չի կարելի ուսումնասիրել միջռեգիստրային փոխանցումների մակարդակի (ՄՌՓՄ) մոդելավորման ընթացքում, քանի որ մոդելն իդեալական է: Այս խնդիրներն ուսումնասիրելու համար անհրաժեշտ է ճշգրիտ ժամանակային պարամետրերով տարրերի մակարդակի նկարագրության մոդելավորում, սակայն դա հասանելի է դառնում միայն նախագծման վերջնական փուլում:



Նկ. 1.13. Սինքրոազդանշանների փիրույթների՝ որոշակի քանակով նախագծերի փոկոսը թվային նախագծերի մեջ

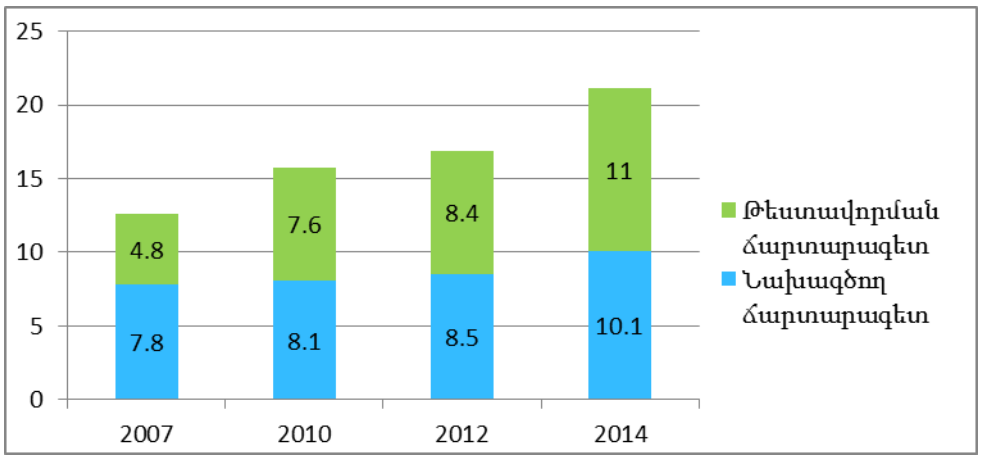
Բացի դրանից, հիմնականում միևնույն ՄՌՓՄ նկարագրությունն օգտագործվում է բազմաթիվ տեխնոլոգիական գործընթացներով ՄՍ հանգույցների կառուցման նպատակով, և ճարտարագետները տվյալներ չունեն հաճախորդի օգտագործած տեխնոլոգիական գործընթացի մասին, և, հետևաբար, նկարագրությունը պետք է թեստավորվի բազմաթիվ տարրերի գրադարաններով: Կան ծրագրային գործիքներ, որոնք կարելի է օգտագործել նախագծման սկզբնական փուլերում ՄՌՓՄ մոդելի կառուցվածքային վերլուծության միջոցով թեստավորում կատարելու նպատակով: Սակայն այդ ծրագրային գործիքների օգտագործումն ունի հետևյալ թերությունները [41].

- մեծ նախագծերի կառուցվածքային վերլուծության ժամանակատարություն,
- մեծ քանակությամբ կեղծ սխալների առաջացում,
- աշխատանքային ռեժիմների մեծ քանակություն,
- ոչ սինքրոն տակտային ազդանշանների մեծ քանակության պատճառով թեստավորման միջավայրի կառուցման բարդություն:

Ժամանակակից ԻՍ-երի մեծ մասը պարունակում է ծրագրավորելիության որոշակի հնարավորություն: Թվային նախագծերի ուսումնասիրությունը ցույց է տալիս, որ ԾՏԻՍ-ով կամ ծրագրավորվող սարքավորումներ օգտագործող նախագծերի քանակն աճել է: Բացի դրանից, աճել է նաև ԾՏԻՍ-ով նախագծերի քանակը, որոնք օգտագործում են ներդրված պրոցեսորներ՝ ծրագրային ճկունությունը ապարատայինին ավելացնելու նպատակով: Մեծ քանակությամբ մեկ խնդրի լուծման համար

նախագծված ԻՍ-երն ունեն ներդրված պրոցեսորներ: ԿԲՊՀ-ների մեծ մասը նույնպես ունի վերածրագրավորվող մասեր [42-44]: Վերը նշվածի արդյունքում աճել են ծրագրերի կարևորությունը և գործառույթները: Նույնիսկ որոշ դեպքերում ծրագրի նախագծումը ավելի բարդ է և ժամանակատար, քան ապարատային մասինը: Սովորական հաջորդական նախագծման գործընթացի դեպքում ծրագրերի նախագծումը կարելի է միայն սկսել, երբ ԻՍ-ի առաջին նախագիծը հասանելի է, որը կարող է խոչընդոտել չիպի՝ նախատեսված ժամանակում վաճառքում հայտնվելուն:

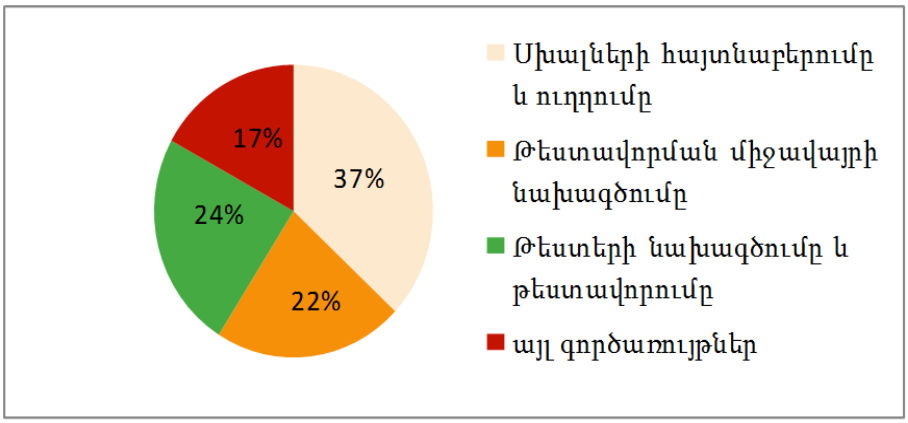
Թեստավորման վրա ծախսվող ժամանակը շատ կարևոր խնդիր է ժամանակակից թվային նախագծերի համար: Թեստավորման արտադրողականության աճը և ճարտարագետների քանակի աճի կառավարումը նույնպես կարևոր խնդիրներ են: Տարբեր նախագծերում, կախված բազմաթիվ պահանջներից, թեստավորման ճարտարագետների քանակը կարող է տարբեր լինել: Թեստավորող և թվային սխեմաներ նախագծող ճարտարագետների միջին քանակների համեմատությունը տրված է նկ. 1.14 –ում: 2007 - 2014 թվականներին միջին քանակի փոփոխությունը զգալի է [36]:



Նկ. 1.14. Թվային նախագծերում ներգրավված ճարտարագետների քանակը

Ժամանակակից թվային նախագծերում թեստավորման ինժեներների քանակը գերազանցում է թվային սխեմաներ նախագծողների քանակը: Թվային սխեմաներ նախագծող ինժեներները նույնպես ժամանակի զգալի մասը ծախսում են թեստավորման վրա: 2014 թվականի տվյալներով ճարտարագետների ժամանակի միջին հաշվով 47%-ը ծախսվում է թեստավորման վրա [36]: Ուսումնասիրությունները

ցույց են տալիս, որ ճարտարագետներն ավելի շատ ժամանակ են ծախսում սխալների հայտնաբերման և ուղղման, քան այլ խնդիրների լուծման վրա:



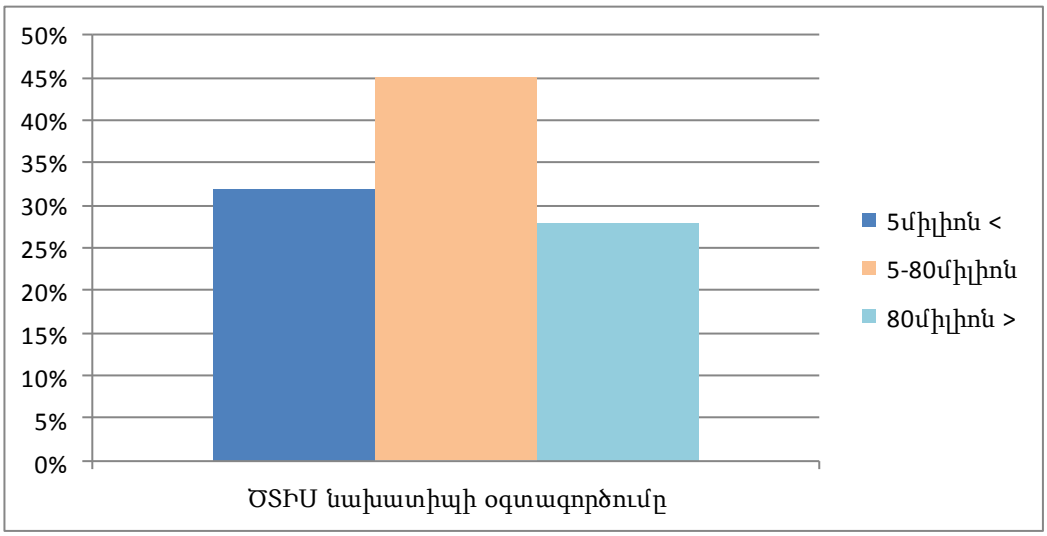
Նկ. 1.15. Թեստավորման ճարտարագետների գործառույթների մասնաբաժինները

Թեստավորման համար օգտագործվում են դինամիկ և ստատիկ մեթոդներ: Ստատիկ թեստավորման համար հիմնականում օգտագործում են ֆորմալ մեթոդներ: Դինամիկ թեստավորման համար օգտագործում են մոդելավորում: Մոդելավորման վրա հիմնված եղանակներն են.

- ՄՌՓՄ նկարագրության տողերի, վերջավոր վիճակների մեքենաների (ՎՎՄ) և անցումների ծածկույթը,
- SVA [45] լեզվով նկարագրված հաստատումների միջոցով հաշվարկվող ֆունկցիոնալ ծածկույթը,
- թեստավորումը՝ պատահական թեստ վեկտորների միջոցով:

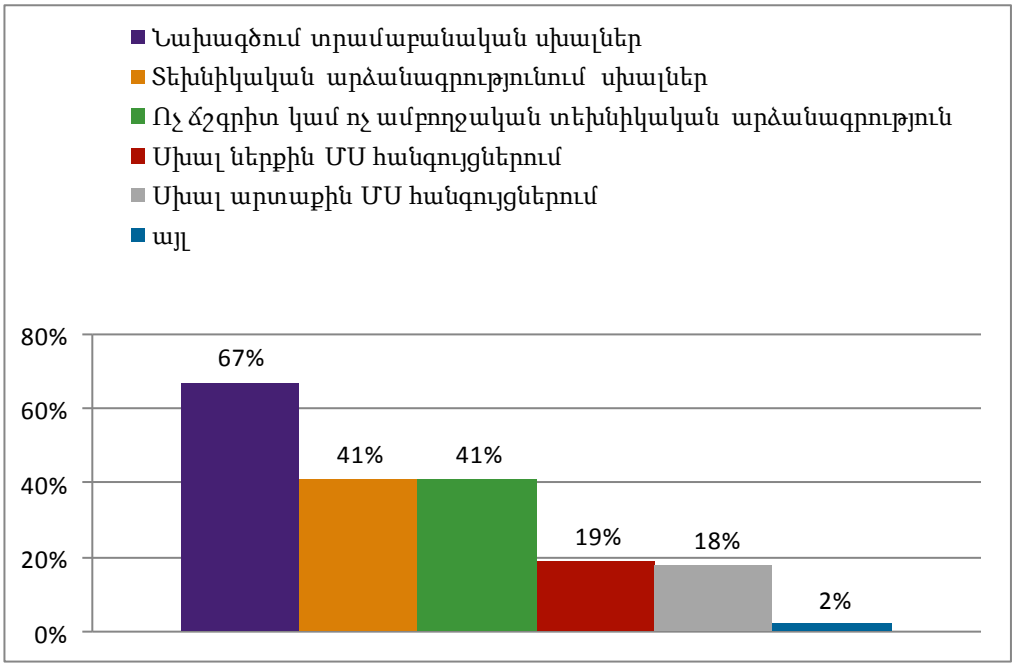
Վերը նշված եղանակների արդյունավետությունը բարձր է ՄՍ հանգույցների, ենթահամակարգերի դեպքում, սակայն սրանք կիրառելի չեն ԿԲՊՀ տիպի նախագծերի դեպքում: Նախագծերի բարդացման պատճառով մոդելավորման վրա հիմնված թեստավորման եղանակների արդյունավետությունը նվազել է, կամ աճը չի համապատասխանում նախագծերի բարդացմանը: Այս պնդումը ճիշտ է հատկապես այն նախագծերի դեպքում, որոնք պարունակում են պրոցեսորի մոդելներ: Հետևաբար, անհրաժեշտ են նոր մեթոդներ՝ թեստավորման համակարգերի արագագործության և արդյունավետության բարձրացման համար:

ԾՏԻՍ նախատիպի օգտագործումը կարող է օգտակար լինել մեծ նախագծերի դեպքում: ԾՏԻՍ նախատիպերը կարող են օգտագործվել, օրինակ, ԿԲՊՀ-ներում՝ նոր ծրագրային կամ ապարատային մասերի ավելացման դեպքում թեստավորումն իրականացնելու համար, ինչպես նաև դրանք կարող են հարթակ հանդիսանալ ծրագրերի նախագծման համար: Թվային նախածերի 33%-ն օգտագործում է ԾՏԻՍ նախատիպերը [36]: Թվային նախագծերում մեթոդի օգտագործման չափը՝ կախված նախագծերի տարրերի քանակից, տրված է նկ.1.16–ում: Նախագծերի քանակը, որոնք օգտագործում են ԾՏԻՍ նախատիպ, զգալիորեն նվազում է 40 միլիոնից ավելի տարրերի դեպքում: Նվազման հիմնական պատճառներն են սահմանափակումները՝ կախված ԾՏԻՍ-ներում առավելագույն տարրերի քանակից, ինչպես նաև մեծ նախագծի մի քանի ԾՏԻՍ-ում իրականացնելու նպատակով բաժանման խնդիրները:



Նկ.1.16. Նախատիպեր օգտագործող թվային նախագծերի քանակը

Ինտեգրալ սխեմաների նախագծման բնագավառի հիմնական խնդիրներից մեկը շարունակում է մնալ նախագիծը ժամանակին ավարտելը: Բացի դրանից, նախագծերի միայն 30%-ի դեպքում է հնարավոր լինում ստանալ ամբողջովին ֆունկցիոնալ սարք՝ առաջին անգամ արտադրված ԻՍ-ում [46-48]: Երկրորդ արտադրության հիմնական պատճառները ներկայացված են նկ.1.17-ում: Նախագծի ֆունկցիոնալ սխալները մնում են կրկնակի արտադրության հիմնական պատճառը:



Նկ.1.17. ԻՍ-երի կրկնակի արտադրությանը հանգեցնող սխալները

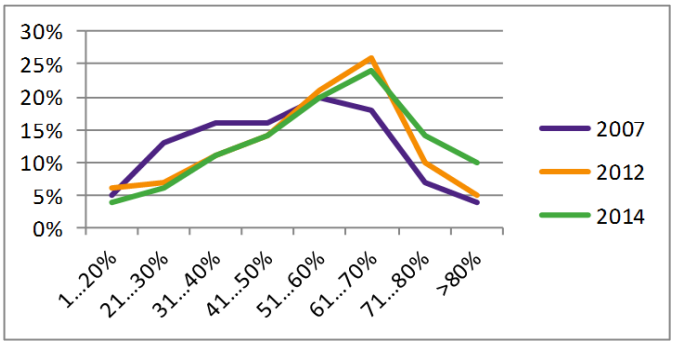
ԾՏԻՍ նախատիպի թեստավորման գործառույթների ընդլայնումը կարող է բարելավել ընդհանուր վիճակը, քանի որ դրա վրա հիմնված թեստավորման համակարգերը զգալիորեն արագագործ են և կարող են նվազեցնել թեստավորման ժամանակը: Պետք է նշել նաև, որ նախատիպի օգտագործումը չի կարող փոխարինել մոդելավորումը, սակայն որոշակի գործառույթների իրականացումը ԾՏԻՍ–ում հնարավոր է: Ժամանակակից ԾՏԻՍ–ները հնարավորություն են տալիս կառուցելու թեստավորման համար նախատեսված վերակառուցավորվող համակարգեր, որոնք պարունակում են պատահական թեստավորման հաջորդականությունների գեներացման համար նախատեսված հանգույցներ, ֆունկցիոնալ սխալների հայտնաբերման համար սինթեզվող դիտարկման միջոցներ, ինչպես նաև ֆունկցիոնալ թեստավորման որակը որոշելու մեխանիզմներ՝ ֆունկցիոնալ ծածկույթի գնահատման միջոցով:

1.3. Ֆունկցիոնալ թեստավորում

Ֆունկցիոնալ թեստավորումը ժամանակակից բարդ թվային սխեմաների նախագծման անհրաժեշտ բաղադրիչն է: Ինտեգրալ սխեմաների տարրերի քանակի աճը շարունակվում է Մուրի օրենքով [49], սակայն թեստավորման բարդությունն աճում

է ավելի արագ [50]: Ըստ 2014 թվականի տվյալների՝ թեստավորման միջին տևողությունը 57% է [36]: Նախագծերը, որոնցում թեստավորման վրա ծախսվող ժամանակը մեծ է 80%-ից զգալիորեն աճել են (նկ. 1.18): Բարձր մակարդակի ծրագրավորման լեզուների օգտագործումը հնարավորություն է տվել նախագծելու ավելի բարդ ֆունկցիոնալությամբ ինտեգրալ սխեմաներ: Ֆունկցիոնալությունը բարդացել է նաև հետերոգեն նախագծերի տարածման պատճառով, որոնցում համադրվում են ծրագրային և ապարատային, ինչպես նաև անալոգային և թվային բաղադրիչները: Թեստավորման արտադրողականության աճը զգալիորեն զիջում է տարրերի քանակի աճին: Անհրաժեշտ է մշակել նոր մեթոդներ՝ թեստավորման որակի բարձրացման, ինչպես նաև արդյունավետության աճի նպատակով: Գոյություն ունեն չորս հիմնական խնդիրներ [50], որոնք բարդացնում են թեստավորումը՝

1. վիճակների մեծ քանակությունը, որոնցում կարող է գտնվել համակարգը,
2. ոչ ճշգրիտ ֆունկցիոնալության հայտնաբերումը,
3. գոյություն չունի սարք, որի ֆունկցիոնալությունը կարելի է համարել բացարձակ ճշգրիտ, և որի հետ հնարավոր է համեմատել նոր սկսվող նախագիծը,
4. ֆունկցիոնալ ծածկույթի հաշվարկի մեթոդի բացակայությունը:



Նկ.1.18. Թեստավորման որոշակի ժամանակահատվածով նախագծերի տոկոսը թվային նախագծերի մեջ 2007, 2012 և 2014 թվականներին

Ամբողջական թեստավորման համար անհրաժեշտ է ստուգել համակարգը բոլոր հնարավոր վիճակներում, ինչպես նաև վիճակների միջև բոլոր հնարավոր անցումները: Այսինքն՝ անհրաժեշտ է ստուգել որոշակի վիճակում մուտքերի տարբեր կոմբինացիայի դեպքում արդյո՞ք համակարգն անցնում է նոր՝ ճշգրիտ վիճակի, թե՞ ոչ: Սակայն ժամանակակից թվային սխեմաների հնարավոր վիճակների քանակը

զգալիորեն աճում է, որը բարդացնում է թեստավորումը: Օրինակ՝ սպառվող հզորության նվազեցման համար բազմաթիվ նոր վիճակներ են ավելացվում որոշ ստանդարտ ինտերֆեյսների համար [51]:

Երկրորդ խնդիրն առաջանում է այն պատճառով, որ անհրաժեշտ է ստուգել՝ արդյոք համակարգի անցումը մի վիճակից մյուսին որոշակի մուտքերի դեպքում համապատասխանում է տեխնիկական արձանագրությանը: Հետազոտությունները ցույց են տալիս, որ սխալների քանակը գծայնորեն է կախված ՄՌՓՄ նկարագրության տողերի քանակի աճից [52]: Գոյություն ունեն թեստավորման բազմաթիվ մոդելներ: Այս մոդելների և տեխնիկական արձանագրության համապատասխանության ստուգումը բարդ խնդիր է, քանի որ չկա բացարձակ ճշգրիտ մոդել, որի հետ կարելի է համեմատել: Բացի դրանից, գոյություն չունի ֆունկցիոնալ ծածկույթի չափման միավոր, որը կարելի է օգտագործել՝ որոշելու համար, թե համակարգի ֆունկցիոնալության որ մասն է ստուգված: Չկա նաև ֆունկցիոնալ թեստավորման որակի գնահատման միջոց:

1.3.1. Ֆունկցիոնալ ծածկույթի հաշվարկը և խնդիրները

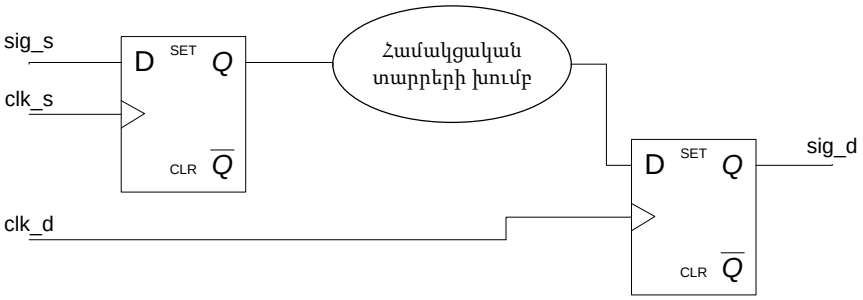
Թեստավորման որակի բնութագրման համար ներմուծված է ֆունկցիոնալ ծածկույթի գաղափարը [53]: Դրա հաշվարկը կատարվում է որոշակի լեզվով տրված վարքի նկարագրության միջոցով, որը ստուգում է կարևոր արժեքների, արժեքների խմբի կամ հաջորդականության, ինտերֆեյսի պահանջների, նախագծի հատկությունների կամ սահմանային պայմանների կատարումը: Ֆունկցիոնալ ծածկույթը շատ կարևոր է կամայական թեստավորման մեթոդի դեպքում, քանի որ ըստ դրա որոշվում է, թե երբ է թեստավորումն ավարտվել: 100% ֆունկցիոնալ ծածկույթը նշանակում է, որ թեստավորման նախագծում տրված բոլոր գործընթացներն ավարտվել են: Որոշակի ինտերֆեսյով տեղափոխված փաթեթների տվյալների չափը կարող է լինել ֆունկցիոնալ ծածկույթի մաս, և որոշակի ինտերֆեյսի ամբողջական թեստավորման համար անհրաժեշտ է ստուգել՝ արդյոք բոլոր հնարավոր չափերով փաթեթներն են փոխանցվել:

Նախագծին նոր ֆունկցիաներ ավելացնելու դեպքում թեստերը, որոնք ամբողջովին ստուգում են նախագծի նախկին տարբերակը, կարող են բավարար չլինել փոփոխությունից հետո: Օրինակ, երբ տվյալների տեղափոխման հերթի չափն աճում է, հին թեստը կարող է ամբողջովին չլցնել հերթը և չառաջացնել համապատասխան (հերթն ամբողջովին լցված է) պայմանը: Եթե նոր հատկություններ են ավելացվել, կարող է թեստի փոփոխության անհրաժեշտություն առաջանալ: Առանց ֆունկցիոնալ ծածկույթի՝ ենթադրվում է, որ օգտագործված թեստավորման եղանակները ստուգել են թեստավորման պլանով նախատեսված բոլոր պահանջները, որը կարող է չհամապատասխանել իրականությանը:

Ֆունկցիոնալ ծածկույթի չափման համար մոդելավորման ընթացքում օգտագործվում է SVA կամ PSL [54]: Սակայն այս լեզուների օգտագործումը ԾՏԻՍ նախատիպի մեջ հնարավոր չէ, քանի որ այս լեզուներով նկարագրված ծածկույթի միավորները սինթեզվող չեն: Նախատիպի թեստավորման ընթացքում ֆունկցիոնալ ծածկույթի՝ բոլորի կողմից ընդունելի չափման մեթոդը բացակայում է: Ֆունկցիոնալ թեստավորման քանակական բնութագրումը ԾՏԻՍ նախատիպի համար շատ բարդ խնդիր է, քանի որ տեսանելիությունը և կառավարումը սահմանափակ են: Բարդ է նաև թեստավորման համակարգի վարքն ուսումնասիրելու նպատակով սխալների ներդրումը: Փաստորեն, ֆունկցիոնալ թեստավորման որակի գնահատման եղանակներ չկան, և հնարավոր չէ գնահատել՝ թեստավորման պլանով նախատեսված ստուգումների որ մասն է իրագործվել:

1.3.2. Մի քանի տակտային ազդանշաններով թվային նախագծերի խնդիրները

Ինչպես նշվեց վերևում, ժամանակակից թվային նախագծերի մեծ մասը պարունակում է երկու և ավելի ասինքրոն տակտային ազդանշանների տիրույթներ: Այդ տիրույթների միջև տվյալների փոխանցումը ճշգրիտ կատարելը կարող է խնդիր լինել (նկ.1.19): Տարբեր սինքրոազդանշանների տիրույթներով անցնող ազդանշաններն անհրաժեշտ է սինքրոնացնել:



Նկ.1.19. Ազդանշանի անցումը տակտային տարբեր ազդանշանների տիրույթներով

Երբ տրիգերի ելքը միացվում է մեկ այլ տրիգերի մուտքին, և այդ տրիգերներն ունեն տարբեր ասինքրոն տակտային ազդանշաններ, կարող են առաջանալ հետևյալ խնդիրները՝

- մետաստաբիլություն,
- տվյալների կորուստ,
- տվյալների դողի վրա սխալ ազդանշանի առաջացում:

Այն դեպքերում, երբ ազդանշանը փոխվում է սինքրոազդանշանի ակտիվ ճակատին շատ մոտ, կա հավանականություն, որ տրիգերը կանցնի այնպիսի վիճակի, երբ ելքն ընդունում է անորոշ արժեք անհայտ տևողությամբ: Այս երևույթը տեղի է ունենում այն դեպքում, երբ ազդանշանը փոխվում է հաստատման կամ տեղակայման ժամանակահատվածներից որևէ մեկի ընթացքում: Այն սարքերի դեպքում, որոնք ունեն երկու կայուն վիճակ, ինչպիսին է տրիգերը, այս վիճակին անցումն անխուսափելի է [55]: Տրիգերների մետաստաբիլ վիճակին անցնելուց խուսափելու համար անհրաժեշտ է հաստատուն պահել ազդանշանը ամեն պարբերության ընթացքում տրիգերի հաստատման և տեղակայման ժամանակահատվածում: Այն դեպքերում, երբ ներգրավված են մեկից ավելի տակտային ազդանշաններ, այս պահանջի բավարարումը շատ բարդ է, իսկ երբեմն՝ անհնար:

Մետաստաբիլությունը մի քանի տակտային ազդանշաններով սխեմաների խնդիրների մեծ մասի պատճառն է: Այս երևույթի սխալ հասցեագրման դեպքում սխալը կհայտնաբերվի միայն կիսահաղորդչում: Եթե մետաստաբիլ վիճակին անցած տրիգերի ելքային ազդանշանը չափվի այն ժամանակ, երբ մետաստաբիլությունը հանդարտվել է, ապա համակարգի աշխատանքը չի խաթարվի: Սխալի առաջացման հավանականությունը հաշվելու համար անհրաժեշտ է հաշվի առնել երկու երևույթ՝

անցումը մետաստաբիլ վիճակին և ելքի չափումը, երբ տրիգերը գտնվում է մետաստաբիլ վիճակում:

Կամայական տրիգերի դեպքում գոյություն ունի ժամանակի որոշակի տիրույթ, որում ազդանշանի փոփոխման դեպքում մետաստաբիլ վիճակին անցումը երաշխավորված է [56]: Այդ տիրութը գտնվում է հաստատման և տեղակայման ժամանակահատվածում և կախված է տրիգերի պարամետրերից: Այդ տիրույթում ազդանշանի փոփոխման հավանականությունն որոշվում է հետևյալ բանաձևով.

$$T_{պ}/T_{տ} = T_{պ} \; x \; F_{տ}, \tag{1.1}$$

որտեղ $T_{պ}$ –ն այդ տիրույթի տևողությունն է, $T_{փ}$ –ն տակտային ազդանշանի պարբերությունն է, իսկ $F_{փ}$ –ն՝ հաճախությունը: Օգտվելով (1.1) -ից՝ կարելի է հաշվել սխալի առաջացման հաճախությունը (1.2).

$$F_{սխ} = F_{տվյալ} \; x \; F_{տ} \; x \; T_{պ} \, , \tag{1.2}$$

Կամայական տրիգերի ճշգրիտ աշխատանքի համար անհրաժեշտ է, որ տակտային ազդանշանի հաճախությունը մեծ լինի ($T_{տ} + T_{հ}$) –ից և $T_{պ}$ –ն շատ փոքր է ($T_{տ} + T_{հ}$) –ից: Կստանանք հետևյալ արտահայտությունը՝

$$T_{պ} \, x \, F_{տ} < 1 : \tag{1.3}$$

Ինչես ցույց է տրված [57] –ում, մետաստաբիլ վիճակում գտնվող տրիգերի ելքի չափման հավանականությունից մինչև ելքի ճշգրիտ արժեքի վերականգնումը որոշվում է հետևյալ բանաձևով՝

$$P = e^{-(t-d)/\tau} \, , \tag{1.4}$$

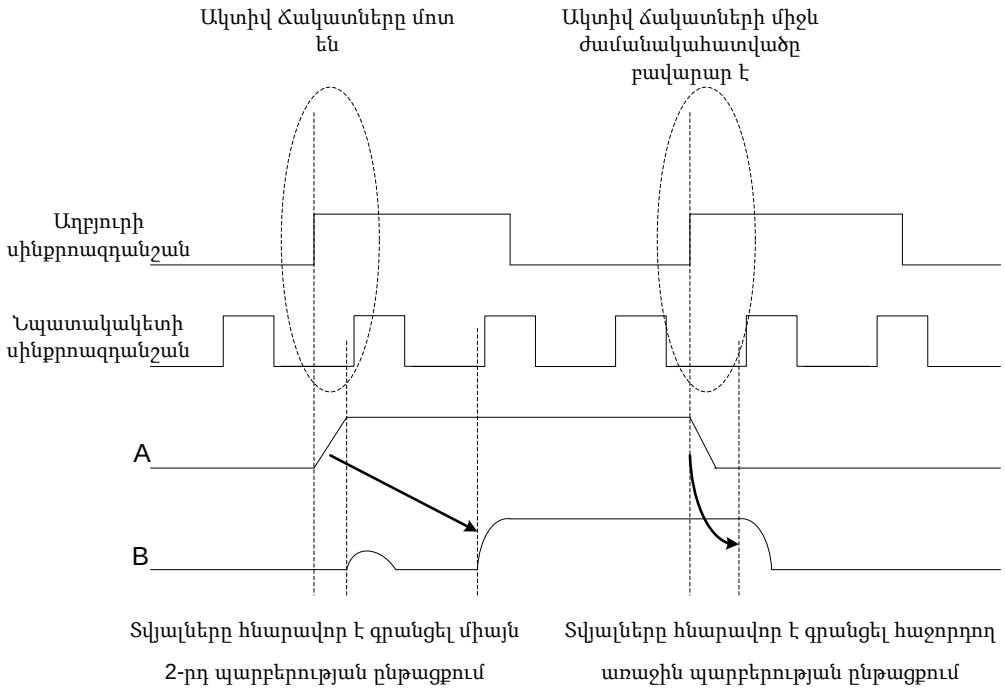
որտեղ t -ն ելքի չափման ժամանակն է, d -ն՝ տրիգերի հապաղումը, և τ –ն՝ ելքային ազդանշանի հաստատման ժամանակային հաստատունը:

Անցումը մետաստաբիլ վիճակին և ելքի չափումն անկախ պատահարներ են, քանի որ ազդանշանի աղբյուրի փոփոխությունը կախված չէ նշանակետի տակտային ազդանշանի ակտիվ ճակատից, և սխալի հավանականությունը հաշվելու համար

անհրաժեշտ է բազմապատկել այդ երկու երևույթների հավանականությունները: Կարելի է հաշվել սխալների միջև միջին ժամանակահատվածը (ՄԺՍՄ).

$$F = F_{տվյալ} \ x \ F_{ա} \ x \ T_{պ} \ x \ e^{-(t-d)/\tau} = {}^{1}/_{ՄԺՍՄ}: \qquad (1.5)$$

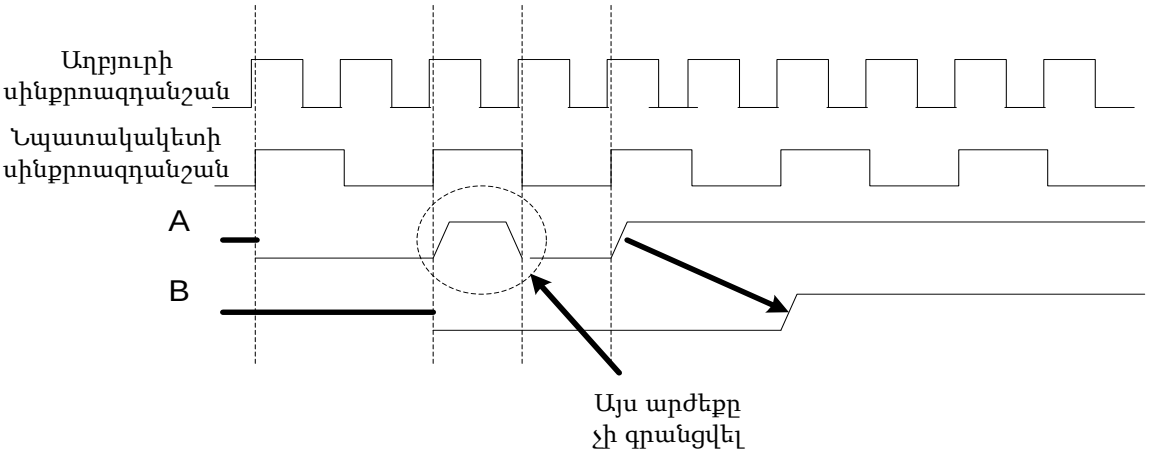
Ազդանշանի աղբյուրի գեներացրած տվյալները կարող են չգրանցվել նշանակակետի տակտային ազդանշանի առաջին պարբերության ընթացքում՝ տրիգերի մետաստաբիլ վիճակին անցման պատճառով: Եթե նշանակետի տակտային ազդանշանի ակտիվ ճակատը և մուտքային ազդանշանի փոփոխությունը մոտ լինեն, հավանական է՝ այդ փոփոխությունը հնարավոր չլինի հայտնաբերել նշանակետի առաջին պարբերության ընթացքում: Փոփոխությունը կհայտնաբերվի միայն հաջորդ պարբերությունում (նկ.1.20):



Նկ.1.20. Տրիգերի մետաստաբիլ վիճակում հայտնվելու արդյունքում տվյալների ուշացմամբ գրանցումը

Երբ աղբյուրի հաճախությունը երկու անգամ մեծ է նշանակետի հաճախությունից, և մուտքի ազդանշանի հաջորդականությունը “00101111” է, ապա ելքային ազդանշանի հաջորդականությունը կլինի “0011”: Ազդանշանի արժեքը երրորդ պարբերության ընթացքում ՝ մեկը, չի գրանցվի (նկ.1.21): Տվյալների կորուստ չի լինի, եթե աղբյուրի ազդանշանի ամեն փոփոխություն հայտնաբերվի նշանակետում, որի համար անհրաժեշտ է ազդանշանը կայուն պահել որոշակի նվազագույն

ժամանակահատվածում, որպեսզի հաստատման և տեղակայման պայմանները չխախտվեն [58]:



Նկ.1.21. Տվյալների կորուստը սխալ սինքրոնացման հետևանքով

Երրորդ կետում տրված խնդիրը կարող է առաջանալ այն դեպքում, երբ մի քանի ազդանշաններ, որոնք անցնում են տարբեր տիրույթներով, սինքրոնացված են առանձին-առանձին: Ինչպես վերը նշվեց, կարող է պահանջվել մի քանի նշանակետի պարբերություն, որպեսզի ազդանշանի փոփոխությունը հայտնաբերվի (տրիգերի մետաստաբիլ վիճակի պատճառով): Երբ բոլոր ազդանշանները փոխվում են միաժամանակ, որոշ ազդանշանների արժեքի փոփոխությունը կհայտնաբերվի նշանակետի առաջին պարբերության, իսկ մնացածի փոփոխությունը՝ հաջորդ պարբերության ընթացքում: Արդյունքում՝ կստացվի ազդանշանների ոչ ճշգրիտ կոմբինացիա, որը կարող է ֆունկցիոնալ խնդիրների պատճառ դառնալ: Եթե տրամաբանությունն այնպիսին է, որ մեկ պարբերության ընթացքում փոխվում է միայն մեկ ազդանշան, ապա այդ դեպքում հնարավոր է խուսափել խնդիրներից, քանի որ հապաղման մեկ պարբերություն մտցնելու դեպքում դողը կպահպանի նախկին ճշգրիտ արժեքը: Այսինքն՝ դողը կպահպանի արժեքը կամ կընդունի նոր՝ ճիշտ արժեք: Այս դեպքում օգտագործում են Գրեյի կոդավորումը, որում մեկ պարբերության ընթացքում հնարավոր է փոխել միայն մեկ բիթ:

1.4. ԾՏԻՍ նախատիպի կառուցման սկզբունքները

Նախատիպերի կառուցման նպատակով ԾՏԻՍ–ի ընտրությունը բնական էր, քանի որ այն պարունակում է մեծ քանակությամբ ծրագրավորվող տրամաբանություն: Առաջին սերնդի ԾՏԻՍ-ների դեպքում անհրաժեշտ էր օգտագործել բազմաթիվ ԻՍ-ներ՝ մեկ նախատիպի կառուցման նպատակով: Ժամանակակից ԾՏԻՍ-ները հնարավորություն են տալիս կառուցելու մեծ քանակությամբ տարրեր պարունակող թվային սխեմայի նախատիպը մեկ կամ մի քանի սարքի միջոցով, քանի որ պարունակում են ԻՍ-ի մի քանի միլիոն տարրերին համարժեք ծրագրավորվող հանգույցներ: ԾՏԻՍ-ի օգտակարությունը անվիճարկելի է, քանի որ այն հնարավորություն է տալիս սկսելու ծրագրերի նախագծումը մինչև սարքի արտադրությունը: Ծրագրավորողները հնարավորություն են ունենում նախագծման վաղ փուլերում օգտագործելու ֆունկցիոնալ սարքը: Բացի վերը նշվածից, ԾՏԻՍ նախատիպի օգտագործումը ունի հետևյալ առավելությունները.

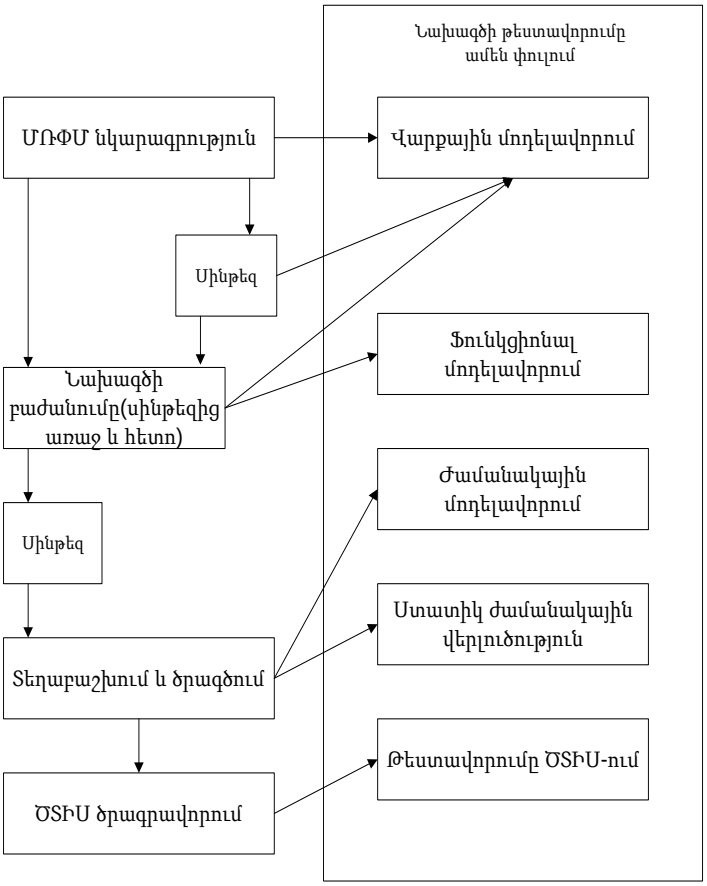
1. Հաղթահարում է ծրագրային մոդելավորման սահմանները: Որոշակի տարրերի քանակից սկսած մոդելավորումը հասնում է իր հնարավորությունների սահմանին՝ արագագործություն չունենալով, որպեսզի արդյունավետ օգտագործվի թվային սխեմաների համար: Եթե չկան ճշգրիտ ծրագրային մոդելներ, ապա ԾՏԻՍ նախատիպի օգտագործումը միակ լուծումն է:
2. Տրամադրում է ամբողջովին ֆունկցիոնալ սարք արտադրությունից (առաջին կիսահաղորդչից) առաջ:
3. Կարող է հանդիսանալ իդեալական թեստավորման հարթակ՝ ՄՍ հանգույցների, ինչպես նաև արդեն արտադրված այլ ԻՍ-երի հետ փոխհամաձայնեցված աշխատանքը ստուգելու նպատակով:
4. ԾՏԻՍ նախատիպերը կարող են օգտագործվել նաև հաճախորդներին սարքի ճշգրիտ աշխատանքը ցուցադրելու նպատակով:

Նախատիպի կառուցման երթուղին ներկայացված է նկ.1.22-ում: Սինթեզը կարող է կատարվել նախագծի մասերի բաժանումից ինչպես հետո, այնպես էլ առաջ: Սինթեզի արդյունքում ստացվում է որոշակի ԾՏԻՍ-ի համապատասխանող տարրերի մակարդակի նկարագրություն: Որոշ ծրագրային գործիքներ հնարավորություն են տալիս գնահատելու նաև արագագործությունը և նախագծում փոփոխություններ

կատարելով՝ ստանալու պլանավորված արդյունքները: ԾՏԻՍ-ում իրականացնելու համար նախագծի համապատասխանեցման ընթացքում ԿԲՊՀ ՄՌՓՄ-ն ձևափոխվում է՝ ԾՏԻՍ-ի տեխնոլոգիայով ավելի արդյունավետ կերպով իրականացվելու և որոշակի նախատիպի կառուցման հարթակին համապատասխանեցվելու համար: Այդ փոփոխությունները կարող են ենթադրել այն բաղադրիչների հեռացումը, որոնք նախատեսված չեն նախատիպում թեստավորման համար: Ինչպես նաև ԻՍ-ին հատուկ կառուցվածքների ձևափոխում՝ ԾՏԻՍ-ի համապատասխան կառուցվածքներով (օրինակ՝ տակտային ազդանշանի գեներատորներ և այլ ՄՍ հանգույցներ), հիշասարքերի չափերի փոքրացում՝ ԾՏԻՍ-ում ավելի արդյունավետ կերպով իրականացնելու համար:

Նախագիծը, մի քանի ԾՏԻՍ-ով կառուցելու համար, անհրաժեշտ է բաժանել որոշակի մասերի: Ինտեգրալ սխեմայի ՄՌՓՄ-ն բաժանվում է մոդուլների, որոնք իրականացվում են տարբեր ԾՏԻՍ-ներում: Այս քայլը անհրաժեշտ է այն դեպքում, երբ մեկ ԾՏԻՍ-ի ռեսուրսները չեն բավարարում: Հաջորդ քայլում կատարվում է ժամանակային սահմանափակումների, ինչպես նաև մուտքերի/ելքերի դիրքերի ընտրությունը ԾՏԻՍ-ում: Այս քայլը կարելի է կատարել սինթեզից հետո, սակայն ցանկալի է կատարել սինթեզից առաջ՝ ժամանակային սահմանափակումներին համապատասխանող տարրերի մակարդակի նկարագրությունը ստանալու համար: Տեղաբաշխումից և ծրագծումից հետո ԾՏԻՍ տարրերի մակարդակի նկարագրությունը ձևափոխվում է բիթերի մակարդակի նկարագրության, որն օգտագործվելու է ԾՏԻՍ-ի ծրագրավորման ընթացքում:

Ինչպես վերևում նշվեց, ԻՍ նախագիծը պետք է ձևափոխվի՝ ԾՏԻՍ-ում կառուցվելու համար: Որոշ մասեր պետք է ձևափոխվեն կամ փոխարինվեն՝ ԻՍ-ի և ԾՏԻՍ-ի ճարտարապետությունների տարբերությունների և ԾՏԻՍ-ի ռեսուրսների սահմանափակության պատճառով:



Նկ.1.22. Նախատիպի կառուցման երթուղին

Անհրաժեշտ է կատարել հետևյալ փոփոխությունները.

- Տակտային ազդանշանի ստացման եղանակները, որոնք օգտագործվում են ԻՍ–ում, չեն կարող կիրառվել ԾՏԻՍ–ում, քանի որ տրամաբանական տարրերի օգտագործումը տակտային ազդանշանի հաճախության փոփոխման համար արգելվում է: Ինչպես նաև փուլի թակարդմամբ հաճախային համակարգերը (ՓԹՀՀ) պետք է փոխարինվեն ԾՏԻՍ-ի համապատասխան տարրերով:
- ԻՍ-ի համար նախագծված հիշասարքերը չեն կարող օգտագործվել ԾՏԻՍ-ի մեջ: Պարզ հիշասարքերը պետք է փոխարինվեն ԾՏԻՍ-ի համապատասխան հիշասարքերով, իսկ բարդ կառուցվածք ունեցող հիշասարքերի դեպքում պետք է նախագծել/գրել փոխարինող մոդելներ ԾՏԻՍ-ի համար:
- Մակարդակով աշխատող տրիգերները և ասինքրոն հապաղման սարքերը պետք է ձևափոխվեն կամ փոխարինվեն:
- ԻՍ-ի անալոգային մոդուլները պետք է փոխարինվեն ԾՏԻՍ-ում առկա անալոգային ռեսուրսներով, կամ մեկ այլ անալոգային ԻՍ օգտագործելու

դեպքում հաղորդակցության և ղեկավարման լրացուցիչ մոդուլներ պետք է ավելացվեն:

ԾՏԻՍ նախատիպով թեստավորման հիմնական խնդիրներն են [59].

1. Մեծ նախագծերի բաժանումը մի քանի ԾՏԻՍ-ի մեջ: Սա կարևոր խնդիր է, քանի որ բաժանման ընթացքում սխալվելու հավանականությունը մեծ է: Բացի դրանից, մասերի բաժանելուց հետո անհրաժեշտ է մեծ քանակությամբ միջմիացումներ իրականացնել տարբեր ԾՏԻՍ-ների միջև, սակայն մուտքերի և ելքերի քանակը սահմանափակ է: [39] –ում ներկայացված են սխեմաներ, որոնք կարելի է օգտագործել խնդրի լուծման համար, սակայն թեստավորումը բարդանում է նախագծի բարդացման պատճառով:
2. Ամբողջովին ֆունկցիոնալ նախատիպի կառուցման համար երկար ժամանակ է անհրաժեշտ: Նախատիպը կարող է ունենալ ֆունկցիոնալ սխալներ հետևյալ պատճառներով՝

    ա) ԻՍ նախագծի ԾՏԻՍ-ում իրականացնելու համար ձևափոխում,

    բ) տպասալի նախագծային կամ ֆունկցիոնալ սխալներ,

    գ) թվային միջուկի նախագծի սխալներ:

    Ճշգրիտ սարքավորումների բացակայության դեպքում սխալների հայտնաբերման համար կարող են ամիսներ պահանջվել:
3. Սխալների պատճառի հայտնաբերումը խնդիր է, քանի որ ԾՏԻՍ ներքին ազդանշանների տեսանելիությունը բացակայում է: Ներքին ազդանշանների չափման/ստուգման համար անհրաժեշտ է դրանք դուրս բերել ԾՏԻՍ–ի ելքերին: Սակայն այդ ազդանշանների քանակը սահմանափակվում է ազատ ԾՏԻՍ-ի ելքերի քանակով, որոնք կարելի է օգտագործել թեստավորման նպատակներով: Կան ծրագրային միջոցներ ԾՏԻՍ-ի մեջ գործիքների ներդրման համար, որոնց միջոցով կարելի է որևէ ստանդարտ ինտերֆեյսով ԾՏԻՍ-ի ներքին ազդանշանները դուրս բերել չափման համար, սակայն առկա են հետևյալ սահմանափակումները.

    ա) Գործիքների ներդրումը հնարավոր է միայն մեկ ԾՏԻՍ-ում:

բ) Քանի որ օգտագործվում են ԾՏԻՍ-ի ներքին հիշողության ռեսուրսները, ուստի ազդանշանների քանակը և ամեն ազդանշանի դիտարկման տևողությունը (չափումների քանակը) սահմանափակ են:

գ) Քանի որ գործիքների ներդրումը կատարվում է ՄՌՓՄ-ում նոր ազդանշանի ավելացման համար, անհրաժեշտ է անցնել ԾՏԻՍ-ի ծրագրավորման համար նախատեսված բիթային նկարագրության ստացման բոլոր փուլերով:

4. Նախատիպի արագագործության սահմանափակումները, որոնք առաջանում են ԾՏԻՍ–ի՝ մեծ քանակությամբ ռեսուրսների օգտագործման պատճառով ժամանակային սահմանափակումներից: Ըստ [39]–ի՝ ռեսուրսների 70%-ի օգտագործումը զգալիորեն նվազեցնում է ԾՏԻՍ-ի արագագործությունը:
5. Նախատիպի կրկնակի օգտագործման հնարավորության բացակայությունը: Սա հիմնականում վերաբերում է այն դեպքերին, երբ նախատիպի կառուցման հարթակը (ԾՏԻՍ + տպասալ և այլն) կառուցվում է որոշակի ՄՍ հանգույցի համար:

1.5. Վերակառուցավորվող հաշվողական համակարգերի կառուցման՝ ֆունկցիոնալ թեստավորմանը նպատակաուղղված արդի մեթոդները

Ինչպես նախորդ ենթագլխում նշվեց, ԾՏԻՍ նախատիպերը սովորաբար կառուցումից անմիջապես հետո չեն աշխատում նախատեսված եղանակով և ունենում են բազմաթիվ սխալներ: Նույնիսկ ավելի ավարտուն վիճակում գտնվող նախատիպերը կարող են դրսևորել չնախատեսված վարք որոշակի սահմանային պայմաններում օգտագործման դեպքում: Տեսանելիությունը, որն ապահովվում է ԾՏԻՍ–ի մուտքերի և ելքերի միջոցով, բավարար չէ այդ սխալների տեղայնացման համար: Այս խնդիրների լուծման համար անհրաժեշտ է ԾՏԻՍ–ի ներքին ազդանշանների ավելի մեծ տեսանելիություն, որին հնարավոր չէ հասնել միայն մուտքերի և ելքերի դիտարկմամբ: Մի քանի ծրագրային և ապարատային գործիքներ գոյություն ունեն ԾՏԻՍ նախատիպի ներքին ազդանշանների չափման, որոշակի պատահարների և այլ ֆունկցիոնալ

սխալների հայտնաբերման համար: Որոշ ծրագրային գործիքներ տրամաբանական մոդուլներ են ավելացնում նախատիպի մեջ, որոնք կատարում են չափում՝ նախապես ընտրված ազդանշանների որոշակի պատահարից սկսած, և պահում են չափումները չօգտագործվող հիշողության հանգույցներում: Հետագայում այդ չափումները դուրս են բերվում ԾՏԻՍ-ից և տրվում օգտագործողին բազմաթիվ եղանակներով: Որոշ գործիքներ ավտոմատ կերպով ավելացնում են այդ մոդուլները՝ առանց օգտագործողի կողմից ՄՌՓՄ փոփոխության անհրաժեշտության, իսկ այլ գործիքների դեպքում անհրաժեշտ է ավելացնել սխալների հայտնաբերման մոդուլները՝ փոփոխելով ՄՌՓՄ-ն: Բացի դրանից, որոշ գործիքների դեպքում հնարավոր է կատարել ոչ մեծ փոփոխություններ՝ առանց տեղաբաշխման և ծրագծման գործընթացը կրկնելու: Ընդհանուր առմամբ, առաջարկվող մեթոդները բաժանվում են երկու մասի՝ անհապաղ և նախկին արժեքների դիտարկման ռեժիմներ:

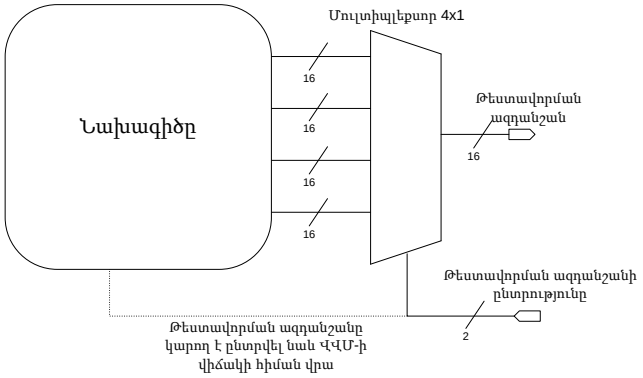
1.5.1. Ազդանշանների անհապաղ դիտարկման ռեժիմը

Իրական ժամանակում ազդանշանների չափումը հնարավորություն է տալիս չափելու ոչ միայն ազդանշանների արժեքները, այլ նաև հայտնաբերելու մրցավազքի պայմանները և ազդանշանների անսպասելի արժեքները: Այս դեպքում ազդանշանները դիտարկվում են օսցիլոսկոպի կամ տրամաբանության վերլուծիչի միջոցով: Որոշակի հանգույցների՝ ԾՏԻՍ-ի թեստավորման համար նախատեսված ելքերին միացնելով դիտարկումն, անհապաղ ռեժիմում ազդանշանների դիտարկման պարզագույն եղանակն է: Այս մեթոդի կիրառման դեպքում ԾՏԻՍ-ի տրամաբանական ռեսուրսները չեն օգտագործվում: Սովորաբար դիտարկումը կատարելու համար նախագծողը միացնում է իրեն հետաքրքրող հանգույցները ԾՏԻՍ-ի ելքերին՝ առանց տրամաբանական սխեմաներ օգտագործելու: Այս մեթոդի օգտագործման դեպքում ՄՌՓՄ փոփոխություններն անխուսափելի են՝ ավելի ցածր հիերարխիայի մակարդակից ազդանշանը բարձրագույն մակարդակին հասցնելու համար: ՄՌՓՄ փոփոխություններ պահանջող մեթոդներում նոր ազդանշանների/հանգույցների ավելացման դեպքում անհրաժեշտ է անցնել նախագծման բոլոր փուլերով, որը կարող է լինել ժամանակատար: Կարելի է բոլոր անհրաժեշտ հանգույցները դուրս բերել՝

լրացուցիչ տեղաբաշխման և ծրագծման գործընթացից խուսափելու համար: Սակայն ԾՏԻՍ մուտքերը և ելքերը սահմանափակ և շատ թանկարժեք ռեսուրսներ են:

Դիտարկվող հանգույցների քանակը մեծացնելու պարզ լուծում կարող է լինել մուլտիպլեքսորների օգտագործումը: Մուլտիպլեքսորի կառավարող ազդանշանը կարող է ինչպես միացվել ԾՏԻՍ–ի մուտքերին, այնպես էլ գեներացվել նախագծի վիճակից կախված որոշակի տրամաբանական հանգույցի ՎՎՄ-ի միոցով (նկ.1.23–ում): Այդ մեթոդի դեպքում կարելի է դիտարկել միանգամից 64 հանգույցի արժեքները՝ օգտագործելով ընդամենը 16 ելք և 2 մուտք (0 մուտք ներքին ղեկավարման եղանակի դեպքում): Կառավարող ազդանշանը երկրորդ տարբերակով գեներացնելու դեպքում կարելի է նվազեցնել օգտագործվող ելուստների քանակը:

«Synopsys» ընկերությունը տրամադրում է նախատիպի կառուցման հարթակներ, որոնցում ԾՏԻՍ-ի մի քանի ելքեր հատուկ առանձնացված են հանգույցների արժեքների դուրսբերման համար, այդ ելքերը միացված են տպասալի կանգնակներին և օգտագործվում են տրամաբանական վերլուծիչին կամ օսցիլոսկոպին միացման համար: Հանգույցներն ընտրելուց հետո «Synopsys»-ի կողմից տրամադրված ծրագրային միջոցը կատարում է անհրաժեշտ փոփոխությունները նախագծի տարրերի մակարդակի նկարագրության մեջ՝ Xilinx-ի FPGA Editor [60] ծրագրային գործիքի միջոցով:



Նկ.1.23. Մուլտիպլեքսորի միջոցով դիտարկվող ազդանշանի ընտրությունը

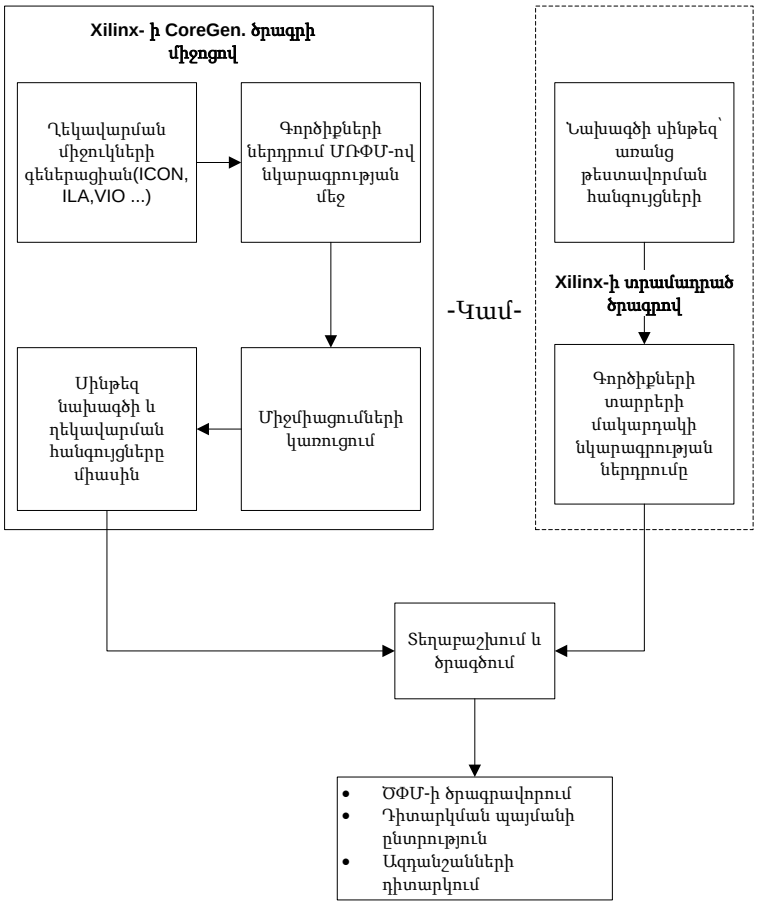
1.5.2. Ազդանշանների նախկին արժեքների դիտարկման ռեժիմը

Վերը նկարագրված մեթոդի հիմնական թերությունը ԾՏԻՍ-ի ելքերի/մուտքերի սահմանափակումն է: Նախկին արժեքների դիտարկման մեթոդներում օգտագործվում

է այլ եղանակ. դրանք չունեն այս սահմանափակումը: Այդ մեթոդների իրականացման նպատակով կառուցվող միջավայրերը բաղկացած են ապարատային և ծրագրային մասերից և օգտագործվում են ԾՏԻՍ–ի ծրագրավորվող ռեսուրսների ազդանշանների դիտարկման և չափման մոդուլների կառուցման նպատակով: Ծրագրային միջոցով կարելի է ընտրել պայմանը, որի կատարումից հետո պետք է սկսել ազդանշանի չափումը: Օգտվելով ԾՏԻՍ-ի՝ ծրագրավորման բիթերի կարդալու հատկությունից՝ կարելի է կարդալ կամայական տրիգերի արժեքը և ներդրված հիշողության պարունակությունը JTAG [61] ինտերֆեյսով: Չափված արժեքները հետագայում տեղափոխվում են համակարգիչ և որոշակի հարմար եղանակով տրվում են օգտագործողին: Այս մեթոդը շատ արդյունավետ է սխալների հայտնաբերման համար մեկ ԾՏԻՍ-ի օգտագործման դեպքում: Բացի դրանից, այն փաստը, որ դիտարկումը սկսվում է որոշակի պատահարից հետո, հնարավորություն է տալիս այս մեթոդը համատեղ օգտագործելու ծրագրային և այլ ապարատային միջոցների հետ, ինչպիսիք են, օրինակ, ազդանշանների գեներատորները: Xilinx–ի ChipScope [62] և Synopsys–ի Identify [63] ծրագրային գործիքներն օգտագործում են այս մեթոդը:

Xilinx–ի ChipScope ծրագրային գործիքը լրացուցիչ տրամաբանական հանգույցներ է ավելացնում կապի, որոշակի պայմանների հայտնաբերման և ազդանշանների չափման համար: Ավելացված մոդուլները և չափվող ազդանշաններն ունեն միևնույն սինքրոազդանշանը, որով խուսափում են մի քանի սինքրոազդանշաններ օգտագործելու խնդիրներից: Ապարատային գործիքների ավելացման 2 եղանակները տրված են նկ.1.24-ում: ՄՌՓՄ մակարդակում գործիքների ավելացման քայլերի հաջորդականությունը հետևյալն է.

1. օգտվելով Xilinx-ի CoreGenerator [64] ծրագրային գործիքից՝ գեներացվում են կապի համար նախատեսված հանգույցը և տրամաբանական վերլուծիչի բլոկը, որոնք պետք է ներդրվեն մակետի մեջ.



Նկ.1.24. Թեստավորման ղեկավարող հանգույցների ներդրումը

2. գեներացված մոդուլները ներդրվում են ՄՌՓՄ-ի մեջ կամ սինթեզված նախագծի մեջ՝ առանց ՄՌՓՄ-ն փոխելու.
3. հետագայում նախագիծն անցնում է տեղաբաշխման և ծրագծման փուլով՝ բիթերի մակարդակի նկարագրությունը ստանալու նպատակով:

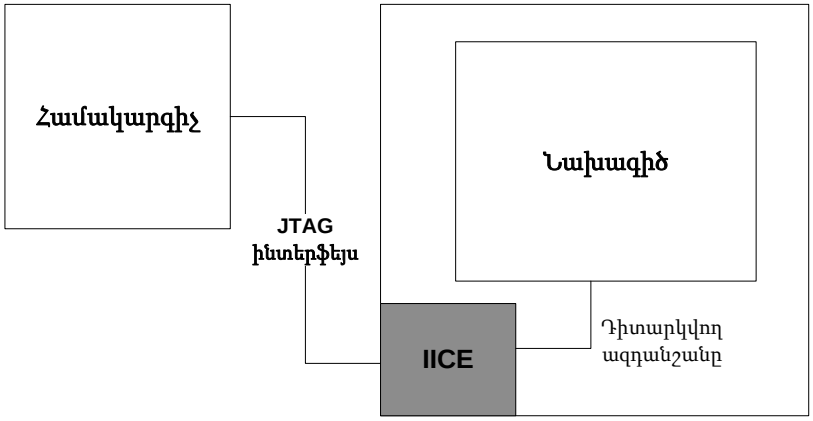
Երկրորդ եղանակը գործիքների ներդրումն է տարրերի մակարդակի նկարագրության մեջ:

1. ChipScope–ի միջուկների ներդրման գործիքի միջոցով տարրերի մակարդակի նկարագրության մեջ ընտրվում են տակտային ազդանշանը և այն հանգույցները, որոնց արժեքները պետք է չափել:
2. Միջուկների ներդրման գործիքն ավելացնում է բոլոր անհրաժեշտ մոդուլները, և արդյունքը փոխանցվում է տեղաբաշխման և ծրագծման փուլ:

Երկու դեպքերում էլ նախագիծը, որը պարունակում է բոլոր անհրաժեշտ մոդուլները, ծրագրավորվում է ԾՏԻՍ-ի մեջ, և օգտագործման եղանակը սխալների հայտնաբերման համար նույնն է: Օգտագործողը կապ է հաստատում ներդրված

գործիքների հետ JTAG ինտերֆեյսով և ընտրում է չափումը սկսելու պայմանը ու չափման եղանակը: Երբ ընտրված պայմանը կատարվում է, ներդրված գործիքները չափումները տեղափոխում են համակարգիչ:

Identify–ը նույն կերպ է աշխատում, ինչ ChipScope-ը, բացառությամբ նրա, որ գործիքների ներդրումը կատարվում է ՄՌՓՄ մակարդակում: ChipScope-ի նման այն նախագծին ավելացնում է ազդանշանների չափման և դիտարկման, չափումների պահպանման և համակարգչի հետ կապի համար նախատեսված մոդուլները: Բացի դրանից, այն տրամադրում է ազդանշանների ընտրման և դիտարկման մեխանիզմներ, որոնք նպաստում են թեստավորման այլ միջոցների հետ կապի միջոցով սխալների հայտնաբերմանը: Identify–ը բաղկացած է երկու մասից՝ Instrumentor և Debugger: Ներդրված գործիքի հիմնական մասը IICE-ն [63] է, որը ներդրվում է նախագծի մեջ (նկ.1.25): IICE–ն պարունակում է տրամաբանական սխեմաներ վերոնշյալ բոլոր գործողությունների համար:



Նկ.1.25. Identify ծրագրային գործիքով կառուցված դիտարկման համակարգը

Ծրագրային գործիքը ներդնում է մեկ կամ մի քանի IICE, որոնք չափում են ազդանշանները և կուտակում տվյալները ԾՏԻՍ-ի ԿԳՀ հանգույցներում: Identify-ի դեպքում նույնպես համակարգչի հետ կապը հաստատվում է JTAG ինտերֆեյսի միջոցով: Ազդանշանների չափումը սկսելու պայմանը ծրագրավորվում է IICE-ի մեջ: Identify-ի օգտագործման քայլերի հաջորդականությունը հետևյալն է.

1. Idetify Instrumentor–ի միջոցով օգտագործողն ընտրում է ազդանշանները, որոնք պետք է դիտարկվեն անմիջապես ՄՌՓՄ նկարագրության մեջ:
2. Idetify Instrumentor-ը ավտոմատ կերպով գեներացնում է IICE-ն.

3. Սինթեզից, տեղաբաշխման և ծրագծման գործընթացից հետո ԾՏԻՍ-ը ծրագրավորվում է, և IICE-ները կառավարվում են Idetify debugger–ով:
4. չափումը սկսելու պայմանի կատարվելուց հետո ծրագիրը ներբեռնում է ԾՏԻՍ-ի հիշողության պարունակությունը JTAG-ի միջոցով:
5. չափված տվյալները արտապատկերվում են որոշակի օգտագործման համար հարմար եղանակով:
6. ազդանշանների և չափումը սկսելու պայմանների փոքր փոփոխությունները կարելի է կատարել առանց սինթեզի և տեղաբաշխման ու ծրագծման գործընթացի:

Եզրակացություններ

1. Ներկայացված են ՎՀ համակարգերը, այդ համակարգերի և մեկ խնդրի լուծման նպատակով կառուցված ԻՍ-ների տարբերությունը: Տրված է ԾՏԻՍ-ների կառուցվածքը, որը հիմնականում օգտագործվում է ՎՀ համակարգերի կառուցման ընթացքում:

2. Ներկայացված են թվային սխեմաների զարգացման միտումները և զարգացման ընթացքում առաջացած ֆունկցիոնալ թեստավորման խնդիրները: Վերլուծված են ավանդական միջոցներով թեստավորման սահմանափակումները: Ներկայացված է նաև թվային նախագծերում թեստավորման ճարտարագետների քանակի և թեստավորման տևողության կառավարման խնդիրը:

3. Ներկայացված են մի քանի սինքրոազդանշանով նախագծերում առկա սխալների տեսակները և թեստավորման խնդիրները: Առաջարկվում է ՎՀ համակարգերի կիրառումը՝ այդպիսի նախագծերի թեստավորման համար:

4. Վերլուծված են արդի ՎՀ համակարգերով թեստավորմանը նպաստող մեթոդները և ԾՏԻՍ նախատիպերի հիմնական խնդիրները:

5. Առաջարկվում է ՎՀ համակարգերով կառուցվող նախատիպերի թեստավորման գործառույթների կիրառության սահմանների ընդլայնում և ֆունկցիոնալ ծածկույթի հաշվարկի միջոցների ներդրում նախատիպով թեստավորման ընթացքում:

ԳԼՈՒԽ 2. ՎԵՐԱԿԱՌՈՒՑԱՎՈՐՎՈՂ ՀԱՇՎՈՂԱԿԱՆ ՀԱՄԱԿԱՐԳԵՐԻ ԿԱՌՈՒՑՄԱՆ ԱՌԱՋԱՐԿՎՈՂ ՄԻՋՈՑՆԵՐԸ

2.1. Վերակառուցավորվող հաշվողական համակարգերի կառուցման առաջարկվող միջոցների մշակման խնդրի դրվածքը

Ինչպես առաջին գլխում նշվեց, ժամանակակից թվային նախագծերի տարրերի քանակն աճում է, օգտագործվում են մեծ քանակությամբ ՄՍ հանգույցներ, աճում են սինքրոազդանշանների և, ընդհանուր առմամբ, ԿԲՊՀ տիպի նախագծերի քանակները: Վերը նշվածի հետևանքով մոդելավորման վրա հիմնված եղանակները բավարար չեն՝ տրված ժամանակահատվածում թվային նախագծերն ամբողջական թեստավորելու համար: Մոդելավորման սահմանափակումների պատճառով առաջացած խնդիրների լուծման նպատակով մեծ տարածում է գտել ՎՀ համակարգերի օգտագործումը՝ թեստավորմանը նպաստելու համար: Առաջին գլխում նշված են ՎՀ համակարգերով կառուցված նախատիպերում հիմնական խնդիրները և այդ խնդիրների լուծման նպատակով օգտագործվող ՎՀ համակարգերի կառուցման մեթոդները:

Աշխատանքի նպատակը թվային նախագծերի թեստավորումն իրականացնելու համար ՎՀ համակարգերի կառուցման մեթոդների մշակումն է՝ սխալների հայտնաբերումը և տեղայնացումը ավելի դյուրին դարձնելու նպատակով, ինչպես նաև ԾՏԻՍ նախատիպերի թեստավորման որակի գնահատման ֆունկցիոնալ ծածկույթի հաշվարկի մեթոդի մշակումը: Այդ խնդրի լուծման համար առաջարկվում է նախագծման ընթացքում ներմուծել սինթեզվող հաստատումներ: Արդյունավետության գնահատման նպատակով կատարվել է համեմատություն՝ արդի այլ մեթոդների հետ հավելյալ ԾՏԻՍ ռեսուրսների, արագագործության սահմանափակումների, սխալների հայտնաբերման և այլ տեսանկյուններից: Որպես մեթոդի կիրառման օրինակ

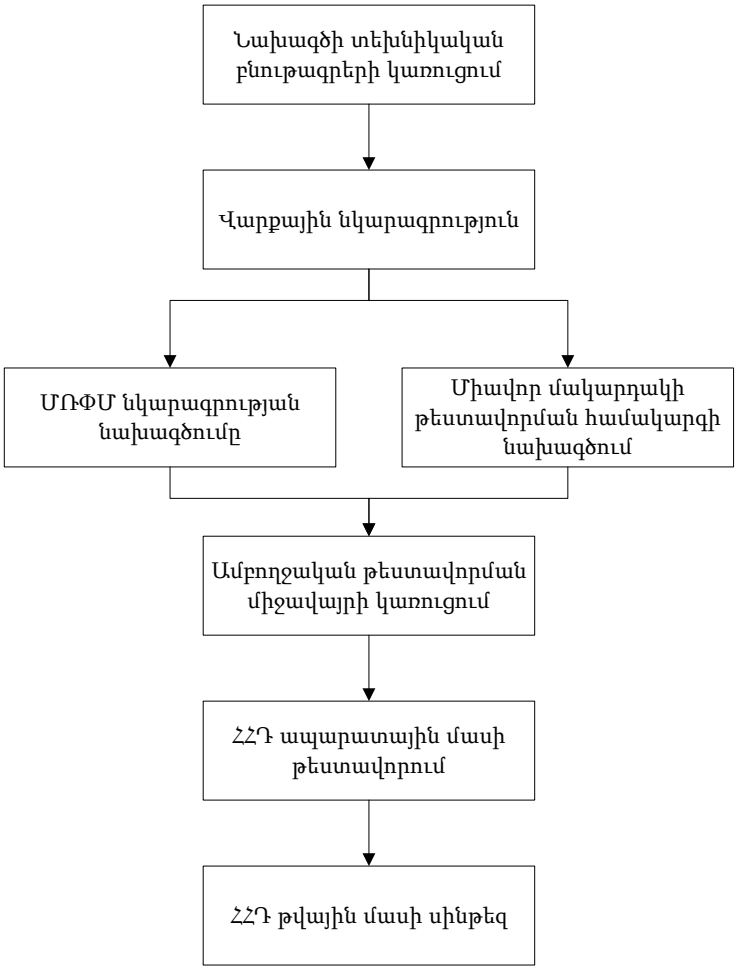
օգտագործվել է համապիտանի հաջորդական դողի (ՀՀԴ) [65] ինտերֆեյսի ղեկավարման թվային միջուկի նախատիպի կառուցումը:

2.2. Վերակառուցավորվող հաշվողական համակարգի օգտագործմամբ ՀՀԴ նախագծման և թեստավորման առաջարկվող միջոցը

Սովորաբար ՀՀԴ-ն օգտագործվում է ԿԲՊՀ-ներում՝ այլ սարքերի հետ միացման նպատակով: ՀՀԴ թեստավորումը պահանջում է ժամեր կամ նույնիսկ օրեր՝ մոդելավորման միջոցով ստուգման համար: Նույնիսկ եթե այն ամբողջովին թեստավորվել է, խնդիրներ կարող են առաջանալ երբ այն ինտեգրում են ԿԲՊՀ-ի մեջ: Որպեսզի համոզվենք, որ ՀՀԴ-ն կարող է օգտագործվել այլ ՄՍ հանգույցների հետ, պետք է ստուգել ինտերֆեյսները, որոնք օգտագործվում են այլ ՄՍ հանգույցներին միացնելու համար՝ նախագծված տեխնիկական արձանագրությանը համաձայն: ՀՀԴ–ն կառուցվում է ծրագրային և ապարատային մասերի համադրությամբ, իսկ ապարատային մասը բաղկացած է անալոգային և թվային մասերից: Այսինքն՝ նախորդ գլխում նկարագրված բոլոր խնդիրները պետք է դիտարկել նաև ՀՀԴ ղեկավարման միջուկի կառուցման ընթացքում, որն ամբողջովին թվային նախագիծ է: ՀՀԴ նախագծման ավանդական երթուղին ներկայացված է նկ.2.1–ում:

Նախագծի տեխնիկական նկարագրի կառուցումը կատարվում է ՀՀԴ նախագծողների ֆորումի [65] կողմից տրամադրված տեխնիկական արձանագրության հիման վրա: Նախագիծը բաժանվում է մասերի (ֆիզիկական, կապի, ծրագրային մասի հետ հաղորդակցության և այլ մակարդակներ), յուրաքանչյուր մաս նկարագրվում է առանձին: Նկարագրվում են նաև ինտերֆեյսները, որոնք պետք է օգտագործվեն՝ առանձին-առանձին մասերը միացնելու համար: Սխալների հայտնաբերմանն օժանդակելու նպատակով մշակվում են հաղորդակցության որոշակի եղանակներ, որոնք հնարավորություն են տալիս օգտագործելու հատուկ մոնիտորներ՝ սխալների հայտնաբերման և տեղայնացման համար:

Հաջորդ քայլով կատարվում է վարքային մոդելի նախագծումը: Նախագծի վարքային մոդելն անհրաժեշտ է նախատեսված ֆունկցիոնալության և տեխնիկական արձանագրության համապատասխանությունը ստուգելու համար: Վարքային նկարագրությունը հետագայում օգտագործվում է ՄՌՓՄ նախագծման/կառուցման ընթացքում որպես հիմք:



Նկ.2.1. ՀՀԴ նախագծման ավանդական երթուղին

Վարքային մոդելի թեստավորումից հետո կառուցվում է ՀՀԴ ղեկավարող միջուկի ՄՌՓՄ նկարագրությունը:

Նախագծի առանձին մասերի թեստավորումը կատարվում է միավորման մակարդակի թեստավորման միջավայրի կառուցման փուլում: Այն հնարավոր է սկսել ՄՌՓՄ նախագծման հետ միասին, քանի որ որպես հիմք է ընդունվում տեխնիկական նկարագրությունը, և հետևաբար՝ բոլոր անհրաժեշտ տվյալները և պահանջները

հայտնի են: Կատարվում է միավոր թեստի միջավայրի կառուցումը՝ որոշակի թեստավորման նկարագրման լեզվով՝ օգտագործելով կառուցման որոշակի մեթոդ: Կառուցվում են նաև հաստատումները, որոնք օգտագործվում են մոդելավորման ընթացքում ֆունկցիոնալ սխալների հայտնաբերման, ինչպես նաև ֆունկցիոնալ ծածկույթի հաշվարկի համար: Անհրաժեշտ հաստատումների ցանկը տրվում է ՀՀԴ տեխնիկական արձանագրության մեջ կամ այլ ՄՍ հանգույցների հետ կապի համար օգտագործվող միջմիացումների ստանդարտների արձանագրություններում, որին կարող են ավելացվել հաստատումները, որոնք հատուկ են որոշակի նախագծի: Այս փուլում նախագծվում են նաև սխալների ներմուծման եղանակներ, և ստուգվում է համակարգի վարքը սխալների դեպքում: Լիարժեք ֆունկցիոնալ թեստավորումը, այսինքն՝ սխալների ներմուծումը և ֆունկցիոնալ ծածկույթի բավարար տոկոսի ստացումը խելամիտ ժամկետներում, հնարավոր է միայն այս փուլում, քանի որ առանձին մասերի թեստավորումը հնարավոր է իրականացնել զուգահեռ, և յուրաքանչյուր մասի թեստավորումը չի պահանջում երկար ժամանակ: Բացի դրանից, մեծ է նաև համակարգի ղեկավարման հնարավորությունը, քանի որ նախագծողը հնարավորություն ունի անմիջապես առանձին մասերի ինտերֆեյսների միջոցով կատարելու սխալների կամ տարբեր աշխատանքային վիճակների ուսումնասիրության համար համապատասխան մուտքային տվյալների ներմուծում:

Միավոր մակարդակում թեստավորման կատարման որոշակի աստիճանից սկսած, այսինքն՝ ֆունկցիոնալ, տողերի, ՎՎՄ-ների և ազդանշանների փոփոխությունների ծածկույթի որոշակի տոկոսի ստացումից և թեստերի որոշակի մասի հաջողությամբ ավարտից հետո նախագծի առանձին մասերը միացվում են, և նախագիծն ինտեգրվում է ամբողջական թեստավորման համար կառուցված միջավայրին: Այս փուլում նախագծվում և օգտագործվում են հաստատումներ, որոնք նախատեսված են ՀՀԴ ղեկավարող միջուկի՝ այլ ՄՍ հանգույցների հետ կապի համար: Մոդելավորման տևողությունը զգալիորեն աճում է, և ֆունկցիոնալ ծածկույթի նախատեսված տոկոսն ապահովելու համար անհրաժեշտ են օրեր և նույնիսկ շաբաթներ: Կարելի է ասել, որ այս փուլում կատարվում է ԿԲՊՀ մակարդակի մոդելավորում, սակայն համակարգի այլ տարրերի փոխարեն օգտագործվում են

համապատասխան մոդելները, որոնք սովորաբար նախագծվում են SystemVerilog [45] սարքի նկարագրման և թեստավորման միջավայրերի կառուցման լեզվով:

Բացի վերը նշվածից, կառուցվում է նաև համակարգ, որը պետք է իրականացվի ԾՏԻՍ-ի նախատիպի միջոցով, ինչպես և համապատասխան թեստավորման միջավայր, որը պարունակում է ՀՀԴ և PCIe–ի [51] ղեկավարող միջուկները, որոշ դեպքերում՝ նույնիսկ պրոցեսորներ: Այս համակարգում միացումների ստուգման համար կատարվում է մոդելավորում, սակայն ԾՏԻՍ-ի նախատիպի լիարժեք թեստավորումը հնարավոր չէ:

Ապարատային թեստավորումը կատարվում է ԾՏԻՍ մակետի միջոցով: Այս փուլը սկսելու համար անհրաժեշտ է, որ ՄՌՓՄ–ն ավարտուն լինի, այսինքն՝ այն պետք է սկսել թեստավորման որոշակի նվազագույն մակարդակի հասնելուց հետո: Կատարվում է նախորդ գլխում բերված ԻՍ-ի ՄՌՓՄ նկարագրության ձևափոխությունը ԾՏԻՍ–ում իրականացնելու համար: ԾՏԻՍ-ը կառուցելու անհրաժեշտ փոփոխությունները կատարելուց հետո նախագծվում են սահմանափակումները, և գեներացվում են ԾՏԻՍ-ը ծրագրավորելու ֆայլերը: Թեստավորումը կատարվում է ՀՀԴ նախագծողների ֆորումի կողմից տրամադրված ծրագրերի միջոցով և պահանջներին համապատասխան:

Մինչև նախագծի վերջնական թեստավորումը և փաթեթավորումը՝ կատարվում է սինթեզ տարբեր տեխնոլգիական ստանդարտ տարրերի գրադարաններով, և անհրաժեշտ է, որ ժամանակային սահմանափակումները բավարարվեն բոլոր դեպքերում, քանի որ հաճախորդներին տրամադրվում է ՀՀԴ միջուկի ՄՌՓՄ նկարագրությունը, որը պետք է ժամանակային սահմանափակումների խախտումներ չունենա հաճախորդի կողմից օգտագործվող տարրերի գրադարանի դեպքում: Սինթեզի արդյունքում ստացված տարրերի մակարդակի նկարագրությունը թեստավորվում է ամբողջական թեստավորման համար կառուցված միջավայրում՝ ֆունկցիոնալ սխալների հայտնաբերման նպատակով: Ֆունկցիոնալ խնդիրներն այս փուլում կարող են առաջանալ սխալ սահամանափակումների օգտագործման, ՄՌՓՄ նկարագրության մեջ սխալ (չսինթեզվող) կառուցվածքների օգտագործման դեպքում, որոնք ձևափոխվում են սինթեզի գործիքի կողմից: Կատարվում է նաև տակտային

ազդանշանների տարբեր տիրույթներով անցումը սինքրոնացնող տրամաբանության կառուցվածքային թեստավորում, որում հաշվի է առնվում մուտքային/ելքային ազդանշանների սինքրոնացնող տակտային ազդանշանը, քանի որ հիմնականում ՀՀԴ-ն պետք է օգտագործվի ԿԲՊՀ-ում այլ ՄՍ հանգույցների և հաճախորդի կողմից նախագծված բաղադրիչների հետ: Նախագծի ամբողջական թեստավորումը կատարելուց հետո այն անհրաժեշտ է որոշակի եղանակով տրամադրել հաճախորդներին: Բացի ՄՌՓՄ-ից, հաճախորդին է տրամադրվում նաև թեստավորման որոշակի պարզ համակարգ, որը հետագայում կարող է օգտագործվել հաճախորդի ընտրած պարամետրերով՝ ՀՀԴ ղեկավարող միջուկի թեստավորան նպատակով:

2.2.1. Սինթեզվող հաստատումների օգտագործումը ԾՏԻՍ նախատիպում

Ավանդական թեստավորման եղանակները հիմնվում են պատահական թեստավորման խթանների ներմուծման և ելքային արժեքի ստուգման վրա: Սակայն նախագծերի բարդության աճի պատճառով բարդացել են ծածկույթի հաշվարկը, սխալների հայտնաբերումը և տեղայնացումը: Այդ խնդրի լուծման նպատակով ներմուծվել է նոր գործիք, որը կոչվում է հաստատում: SVA լեզվով նկարագրվում են հաստատումներ, որոնք պարունակում են նախագծի նախատեսված վարքի նկարագրությունը: Այն ապահովում է համապատասխանությունը նախագծի և իրականացման միջև: Հաստատումները.

- հնարավորություն են տալիս ավելի ճշգրիտ կերպով նկարագրելու վարքը, որը կարող է տարածվել մի քանի տակտային ազդանշանի պարբերություն,
- տրամադրում են միջոցներ՝ նախագծի և նախագծման ընթաքում համակարգի հետագա աշատանքի հետ կապված ենթադրությունների ավելի ճշգրիտ նկարագրման համար,
- բարելավում են ներքին միջմիացումների տեսանելիությունը,
- նպաստում են սխալների առաջացման պատճառների հայտնաբերմանը,

➢ֆունկցիոնալ ծածկույթի հաշվարկի միջոց են:

Այս գործիքներն օգտագործվում են թեստավորան միջավայրերում և սինթեզվող ՄՌՓՄ-ի մեջ: Գոյություն ունեն երկու տեսակի հաստատումներ՝ անհապաղ և զուգահեռ: Անհապաղ հաստատումներն օգտագործվում են always կամ initial [66] հանգույցներում և ունեն ընթացակարգային բնույթ: Բուլյան գործողությունները կատարվում են անհապաղ, առանց տակտային ազդանշանի ակտիվ ճակատին սպասելու: Օրինակ՝ այս արտահայտությունն ստուգում է՝ արդյոք տվյալներն ընդունողն ունի բավարար հիշողություն՝ ռեսուրսների տվյալների պահպանման համար, և բավարար հիշողության բացակայության դեպքում կարող է արդյոք ընդհատել մոդելավորումը: Հետևյալ հաստատումը պետք է օգտագործվի տվյալներն առաքողի համապատասխան ՎՎՄ- ում.

assert (numer_of_remote_credits != 0) $display ("Packet is accepted");

else $error("Packet accepted when there are no credits"):

Զուգահեռ հաստատումները հնարավորություն են տալիս ժամանակի ընթացքում բնութագրելու համակարգի ֆունկցիոնալությունը: Ստորև ներկայացվող պարզագույն հաջորդականությունն օգտագործվում է զուգահեռ հաստատման մեջ: Այն ստուգում է՝ արդյոք բոլոր անհրաժեշտ փաթեթներն են ուղարկվել ՀՀԴ աշխատանքային ռեժիմին անցումից առաջ.

```
sequence entry_to_u0_initialization1; @(posedge clk),

!entry_to_u0 ##1 entry_to_u0,

##1 tx_advert_rqst,

endsequence,

svc_u3link_tctrl_entry_to_u0_initialization1: cover property (entry_to_u0_initialization1):
```

Օրինակներում ներկայացված հաստատումները նախագծվել են [67] – ի հիման վրա և հատուկ են ՀՀԴ ղեկավարող միջուկին: Հետևյալ հաստատումն օգտագործվում է ՀՀԴ և այլ ՄՍ հանգույցների ինտերֆեյսների ստուգման նպատակով:

```
sequence s_axi_aw32_access;
```

```
@(posedge gm_clk)
xm_awvalid ##0 (xm_awaddr < 64'h0000000100000000) ##0 xm_awready;
endsequence
sva_axi_aw32_access: cover property (first_match(s_axi_aw32_access));
```

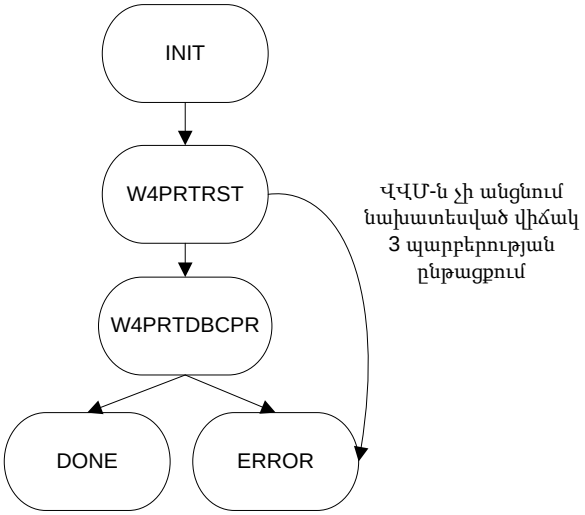
Այն ստուգում է՝ արդյոք կատարվել է 32-բիթ չափով գրանցման գործողություն, և օգտագործվում է ֆունկցիոնալ ծածկույթի հաշվարկի նպատակով: Ինչպես նախորդ գլխում նշվեց, այս արտահայտություններում ներկայացված հաստատումները հնարավոր չէ օգտագործել ԾՏԻՍ նախատիպի մեջ, քանի որ դրանք սինթեզվող չեն, և դրանց օգտագործումը մոդելավորումից հետո հնարավոր չէ: Սակայն հնարավոր է այս արտահայտություններն իրականացնել ՄՌՓՄ մակարդակի Verilog [66] լեզվով և կիրառել դրանք հետագայում նախատիպի մեջ՝ սխալների հայտնաբերումն ավելի դյուրին դարձնելու, ինչպես նաև ֆունկցիոնալ ծածկույթի հաշվարկի միջոց տրամադրելով՝ ՀՀԴ-ի ապարատային թեստավորման որակը գնահատելու նպատակով: Այս հաստատումներն օգտագործվում են այնպես ինչպես նախագծի այլ մոդուլները, որոնց մուտքերը միացվում են այն ազդանշաններին, որոնք անհրաժեշտ է դիտարկել: Դիտարկումը սկսվում է համակարգչի և նախատիպի միացումից անմիջապես հետո և կատարվում է մինչև աշխատանքի ավարտը կամ որևէ սխալի հայտնաբերումը: Հետևյալ պարզ հաստատումը օգտագործվում է ՀՀԴ հիմնական կանգնակների հանգույցում.

```
property port_rst,
@(posedge clk),
(reset_request) ##[1:3] ($past(fsm_prtsm) != PRT_RESET) && (fsm_prtsm == PRT_RESET) ##1 prt_dbc_pr,
endproperty,
svc_u31prt_dbc_port_reset: assert property (port_rst):
```

Այն ստուգում է՝ արդյոք կանգնակի ՎՎՄ-ն անցնում է PRT_RESET վիճակին համապատասխան ազդանշանի դեպքում, և արդյոք տակտային ազդանշանի մեկ պարբերության ընթացքում prt_dbc_pr ազդանշանը մեկ է: Իրականացված ՎՎՄ–ն ներկայացված է նկ. 2.2–ում: Այս մոդուլն ունի reset_request, fsm_prtsm և prt_dbc_pr մուտքերը և status, complete ելքերը: Աշխատանքի սկզբնական փուլում ՎՎՄ-ն գտնվում է INIT վիճակում: Առաջին պայմանի կատարումից հետո այն անցնում է W4PRTRST, որտեղ դիտարկում է ՀՀԴ հիմնական կանգնակների հանգույցի ՎՎՄ-ի վիճակը, և եթե 1-3 պարբերության ընթացքում այն չի անցնում PRT_RESET վիճակին, հաստատումն ավարտվում է անհաջող կարգավիճակով, և status ազդանշանն ընդունում է համապատասխան արժեքը: Հակառակ դեպքում, երբ 1-3 պարբերության ընթացքում հիմնական կանգնակների հանգույցի ՎՎՄ–ն անցնում է PRT_RESET վիճակին, պնդման ՎՎՀ-ն անցնում է W4PRTRST վիճակին, որտեղ դիտարկում է prt_dbc_pr ազդանշանը 1 պարբերության ընթացքում, և ազդանշանի՝ համապատասխան արժեքը չընդունելու դեպքում հաստատումն ավարտվում է անհաջողությամբ, հակառակ դեպքում՝ այն անցնում է DONE վիճակի, այսինքն՝ ավարտվում է հաջողությամբ:

ԾՏԻՍ նախատիպի օգտագործումը հնարավորություն է տալիս թեստավորել ՀՀԴ միջուկը մեծ արագությունների դեպքում՝ գրեթե կիսահաղորդչում իրականացված ԻՍ–ին հավասար [67]: Բացի դրանից, այն կարելի է թեստավորել որպես համակարգի մաս՝ հնարավորություն տալով հայտնաբերելու համակարգի այլ մասերի հետ փոխհամաձայնեցման խնդիրները, որոնք կարող են առաջանալ մոդելավորման ընթացքում ոչ ճշգրիտ մոդելների օգտագործման դեպքում:

Այն խնդիրը, ըստ որի գոյություն չունի մի մոդել, որն ամբողջովին համապատասխանում է տեխնիկական արձանագրությանը [50], ԾՏԻՍ նախատիպի դեպքում բացակայում է: Այս երկու հանգամանքները թույլ են տալիս պնդել, որ մոդելավորման որոշ գործառույթների իրականացումը ԾՏԻՍ նախատիպում կարող է նվազեցնել թեստավորման տևողությունը և արտադրված թվային ԻՍ-ի՝ ոչ ճշգրիտ աշխատելու հավանականությունը:



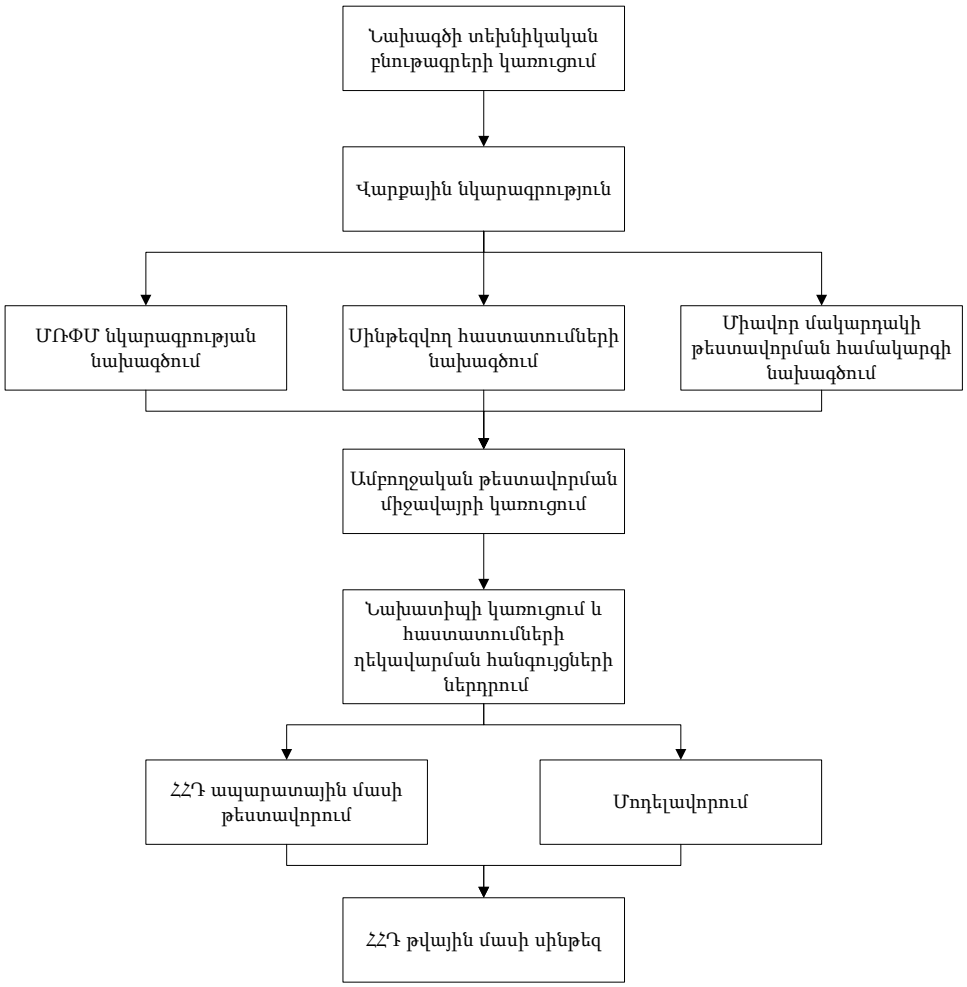
Նկ.2.2. Սինթեզվող հաստատման ՎՎՄ-ն

Ինչպես նշվեց նախորդ գլխում, մեծ նախագծերի դեպքում մոդելավորման արդյունավետ օգտագործումը հնարավոր չէ, և թեստավորման միակ միջոցը ԾՏԻՍ նախատիպն է: Սակայն ներքին ազդանշանների տեսանելիության և ղեկավարելիության բացակայությունը հնարավորություն չի տալիս այն արդյունավետ օգտագործել՝ թեստավորումն արագացնելու համար: Նախորդ գլխում ներկայացված մեթոդներով հնարավոր է ստանալ որոշակի տեսանելիություն, սակայն դա բավարար չէ, քանի որ նույնիսկ մեծ քանակությամբ գործիքների ներդրման դեպքում հնարավոր է ապահովել միայն մոդելավորման տեսանելիության շատ փոքր մասը:

Նկ. 2.3-ում ներկայացված է ՀՀԴ նախագծման առաջարկվող երթուղին [68]: Լրացուցիչ քայլեր են առաջարկվում, որոնց նպատակն է սինթեզվող հաստատումների նախագծումը, թեստավորումը և ներդրումը նախատիպի մեջ: Այս քայլերը պահանջում են լրացուցիչ ջանքերի գործադրում, սակայն մեկ անգամ այս հաստատումների նախագծումից և թեստավորումից հետո դրանք կարելի է կիրառել ՀՀԴ ղեկավարող միջուկների տարբեր տեսակների դեպքում, ինչպես նաև՝ այլ ՄՍ հանգույցներում: Առաջին և երկրորդ քայլերն ամբողջովին համապատասխանում են ավանդական նախագծման գործընթացին:

Սինթեզվող հաստատումների նախագծումը սկսվում է ՄՌՓՄ և միավորների թեստավորման միջավայրի նախագծման հետ միասին: Ելնելով առաջին քայլում նախատեսված ֆունկցիոնալության, ինտերֆեյսների արձանագրությունների և ՀՀԴ

նախագծողների ֆորումի պահանջներից, նախագծվում են հաստատումների սինթեզվող ՄՌՓՄ նկարագրություններ, որոնք թեստավորվում են միավոր թեստավորման միջավայրում: Սինթեզվող հաստատումների թեստավորման մի քանի եղանակ ներկայացված է [69]-ում: Բացի դրանից, նախագծվում են նաև հաստատումների ղեկավարման հանգույցներ, որոնք օգտագործվում են արդյունքների դուրսբերման, ինչպես նաև հաստատումների ակտիվացման համար:



Նկ.2.3. Առաջարկվող ՀՀԴ նախագծման երթուղին՝ սինթեզվող հաստրատումների կիրառմամբ

Ինչպես նշվեց նախորդ գլխում, ԾՏԻՍ նախատիպի օգտագործումը չի կարող փոխարինել մոդելավորմանը. այդ պատճառով պետք է կառուցվի ամբողջական թեստավորման միջավայր, որում պետք է ներդրվեն նախորդ քայլում նախատեսված հաստատումները և ղեկավարման համար նախատեսված տրամանաբանական բլոկները: Ի տարբերություն նախագծման ավանդական եղանակների, այս կետում

կատարվում է միայն հիմնական ֆունկցիոնալության և միջմիացումների ստուգումը, որը զգալորեն նվազեցնում է մոդելավորման տևողությունը:

Հաջորդ փուլում կառուցվում է նախատիպի նկարագրությունը, և հաստատումները պարունակող մոդուլները ղեկավարող հանգույցների հետ միասին ներդրվում են նախատիպի մեջ: Մոդելավորման կետն ավելացված է ՄՌՓՄ նկարագրության մեջ փոփոխությունների դեպքում հիմնական թեստավորումը կատարելու նպատակով: Նույն թեստավորումը կատարվում է նաև այս փուլում, սակայն այս դեպքում սխալների հայտնաբերումը և տեղայնացումը կատարվում են ավելի արագ սինթեզվող հաստատումերի միջոցով [68]:

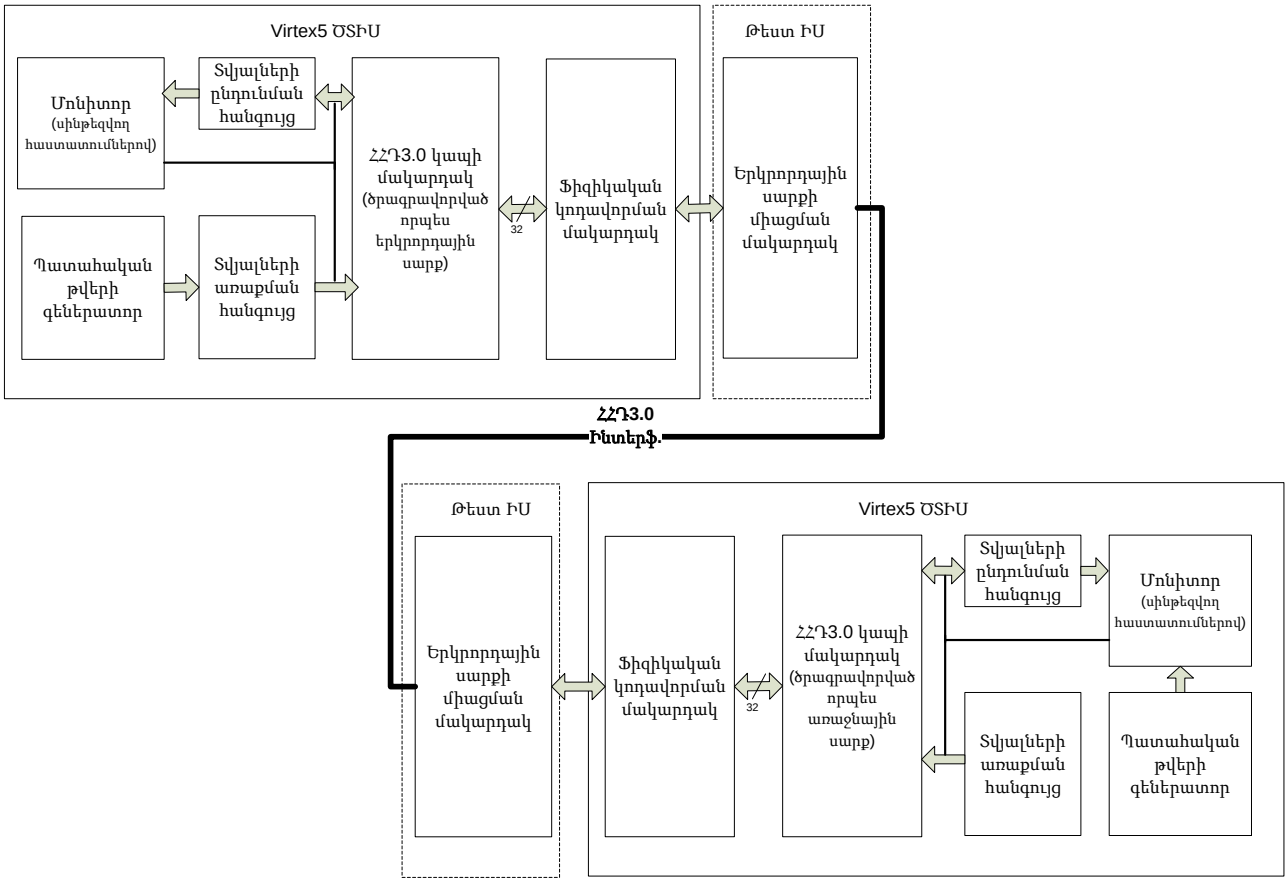
Սխալի հայտնաբերման դեպքում սինթեզվող հաստատումների աշխատանքի արդյունքներն ուսումնասիրելով հնարավոր է պարզել սխալի աղբյուրը պարունակող ՀՀԴ մասը, որից հետո միավոր թեստավորման միջավայրում հետագա ստուգումները կատարելով՝ կարելի է հեշտությամբ հայտնաբերել և ուղղել սխալները:

2.3. Առաջարկվող ՀՀԴ կապի մակարդակի թեստավորման վերակառուցավորվող հաշվողական համակարգը

Մեծ քանակով վիճակների թեստավորման խնդիրը բացահայտ կերպով երևում է ՀՀԴ կապի մակարդակի թեստավորման ընթացքում: Կապի և ֆիզիկական մակարդակները ՀՀԴ կարևորագույն օղակներ են, քանի որ անմիջապես կապ են հաստատում սարքերի միջև, այսինքն՝ իրական աշխարհի հետ: Բացի դրանից, հիմնականում ՀՀԴ արագագործությունը սահմանափակվում է կապի և ֆիզիկական մակարդակների հնարավորություններով [70]: Այդ պատճառով ՀՀԴ այդ մակարդակների թեստավորումը որակյալ ՄՍ հանգույցի նախագծման հիմնական խնդիրներից մեկն է: ԾՏԻՍ նախատիպի կառուցման ընթացքում օգտագործվում է նախորդ ենթագլխում ներկայացված մեթոդը:

Նախորդ գլխում ներկայացված ֆունկցիոնալ թեստավորման չորս հիմնական խնդիրների լուծման համար առաջարկվում է միավորների մակարդակի թեստավորման համակարգ, որն իրականացված է ՎՀ սարքի միջոցով: Առաջարկվող ՎՀ համակարգը

վիճակների մեծ քանակության թեստավորման խնդիրը լուծում է ՀՀԴ բաժանման միջոցով, իսկ երկրորդ և չորրորդ խնդիրները՝ սինթեզվող հաստատումների ներմուծմամբ: Երրորդ խնդիրը մասնակիորեն լուծվում է պատահական թվերի գեներատորի և մոնիտորների միջոցով, որոնք կատարում են ՀՀԴ ավելի բարձր մակարդակների գործառույթները [67]: Թեստավորման միջավայրը ներկայացված է նկ. 2.4–ում:



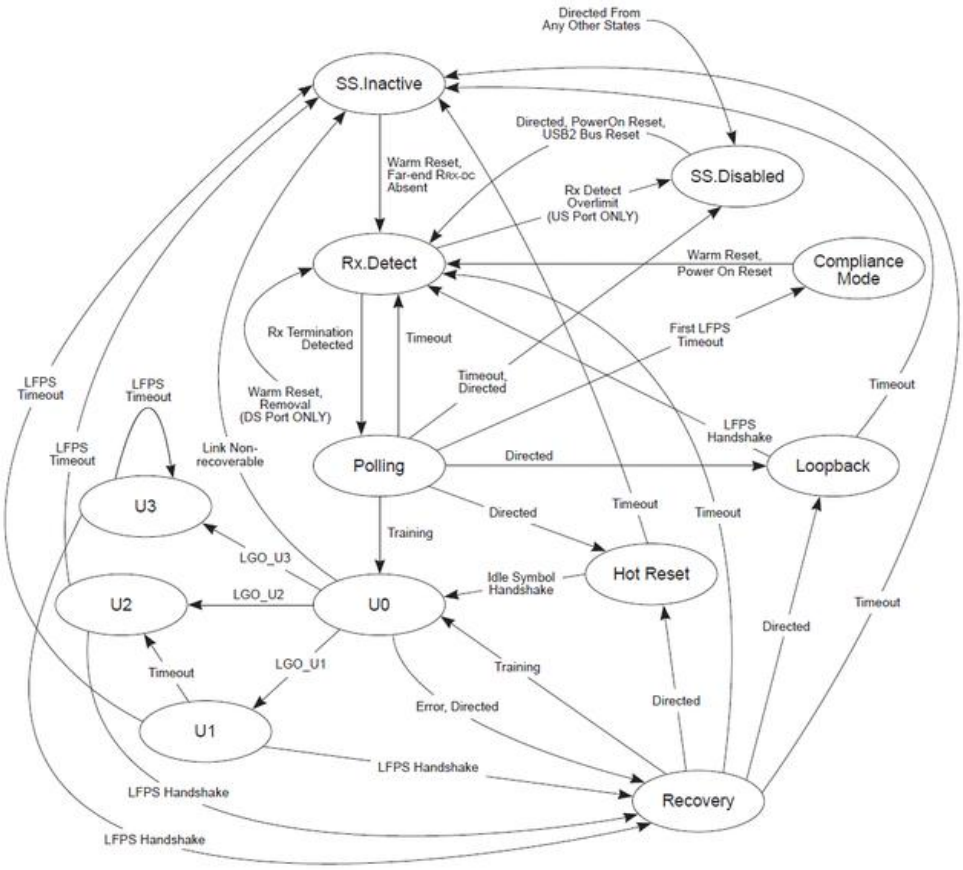
Նկ.2.4. Առաջարկվող ՀՀԴ կապի մակարդակի թեստավորման նախատիպի կառուցվածքը

Կապի մասնակիցների թվային մասերն իրականացվել են «Synopsys»-ի HAPS51T [71] նախատիպի կառուցման հարթակների միջոցով: Համակարգը պարունակում է երկու կապի մասնակից, որոնք միացվում են ՀՀԴ ինտերֆեյսի միջոցով: Թեստավորման որոշակի փուլում կարող է օգտագործվել նաև միայն մեկ մասը, որը միացվում է տրամաբանական վերլուծիչին (օրինակ՝ LeCroy [72] կամ Ellysis [73]): Նախագծված մակետավորման միջավայրը հնարավորություն է տալիս կատարելու կապի մակարդակի համար նախատեսված թեստավորման մեծ մասը՝

առանց ունենալու ՀՀԴ ղեկավարող միջուկի այլ մակարդակների ՄՌՓՄ նկարագրությունը: Այսինքն՝ նախագծման սկզբնական փուլում հնարավորություն է ընձեռվում գնահատելու համակարգի արագագործությունը և ստուգելու ՀՀԴ ֆիզիկական մակարդակի փոխհամաձայնեցումը անալոգային մասերի միջև: Ինչպես երևում է նկ.2.8-ից, նախատիպում չի օգտագործվում համակարգիչ կամ ծրագրային այլ լուծում: Հնարավոր է ստանալ արտադրված ԻՍ-ի արագագործությունը՝ առանց համակարգային սահմանափակումների, որոնք առաջանում են ծրագրային մասերի օգտագործման հետևանքով: Այն հնարավորություն է տալիս կատարելու տեսական սահմաններին մոտ արագություններով թեստավորում [67]:

Գոյություն ունի ՀՀԴ նախագծերի երեք տեսակ՝ առաջնային, երկրորդային և հանգուցային: Հիմնական սարքը՝ երկրորդային սարքերը համակարգչին կապելու միջոց է: Երկրորդային սարքը պարզ կերպով կարելի է նկարագրել որպես որոշակի սարքի ոչ ստանդարտ ինտերֆեյսի ձևափոխիչի ՀՀԴ ստանդարտ, օրինակ՝ ձայնագրման սարքերի համար I2S-ից [74] կամ SPDIF–ից [75] ՀՀԴ: Հանգուցային սարքի միջոցով հնարավոր է մի քանի երկրորդային սարքեր միացնել մեկ առաջնայինին: Այս բոլոր սարքերը պարունակում են միևնույն կապի մակարդակի բլոկները: Կապի մակարդակը նախատեսված է սկզբնական կապի հաստատման և արագագործ հաջորդական տվյալների հաղորդման դողի կարգաբերման համար [70]: Այն պարունակում է նաև տրամաբանական բլոկներ, որոնք, տվյալների փոխանցման բացակայության դեպքում, համակարգը տեղափոխում են էներգախնայող ռեժիմներից որևէ մեկը: ՀՀԴ կապի մակարդակի հիմնական գործառույթներից մեկը տվյալների փաթեթների կազմավորումն է: Կապի մակարդակի ՎՎՄ-ն պարունակում է 12 վիճակ, որոնցից ամեն մեկը նախատեսված է որոշակի գործողության համար: ՀՀԴ հիմնական ՎՎՄ-ն պատկերված է նկ.2.5–ում [70]:

Երկրորդային սարքի հիմնական ՎՎՄ-ն գտնվում է SS.Inactive վիճակում, երբ այն միացված չէ առաջնային սարքին, և միացման դեպքում անցնում է RX.Detect, իսկ առաջնային սարքն անցնում է RX.Detect վիճակին համակարգի սկզբնարժեքավորման ազդանշանի ոչ ակտիվ մակարդակի դեպքում:



Նկ.2.5. Կապի մակարդակի հիմնական ՎՎՄ-ն

Polling վիճակում կապի մասնակիցները համապատասխան տվյալների փոխանցումով սինքրոնացնում են տակտային ազդանշանները և կարգաբերում արագագործ տվյալների փոխանցման դողը: Սինքրոնացումն անհրաժեշտ է, քանի որ առաջնային և երկրորդային, ինչպես նաև հանգուցային սարքերը չունեն մեկ ընդհանուր տակտային ազդանշան: Հետևաբար, տվյալների առաքման և ստացման բլոկների տակտային ազդանշանները ասինքրոն են: Ոչ ճշգրիտ սինքրոնացման հետևանքով հնարավոր է տվյալների կորուստ: Սինքրոնացումը կատարվում է մեծ քանակությամբ 0–1 անցումներ ապահովող ազդանշանի գեներացմամբ: Այդ ազդանշանն օգտագործվում է փուլի թակարդմամբ հաճախության համակարգի (ՓԹՀՀ) կողմից տակտային ազդանշան գեներացնելու նպատակով: Այս գործընթացն ավելի մանրամասն ներկայացված է [70]-ում:

Եթե վերևում նկարագրված գործընթացը հաջողությամբ է ավարտվում, համակարգն անցնում է U0 վիճակին, որը և հիմնական աշխատանքային վիճակն է: U0 վիճակում կատարվում է տվյալների փոխանցումը: Կապի մակարդակն ապահովում է

փաթեթների, այսպես կոչված, վերնագրերի անխափան տեղափոխումը: Վերնագիրը պարունակում է հիմնական տվյալների չափը, երկրորդային սարքի հասցեն, ուղարկվող փաթեթի տեսակը և փաթեթը նշանակետին հասցնելու համար անհրաժեշտ այլ տվյալներ [70]: Հիմնական տվյալների ստուգումը կատարվում է ավելի բարձր մակարդակներում: Սխալների հայտնաբերման միջոցները նկարագրված են [70]-ում: U0 վիճակում փաթեթների վերնագրերում սխալի հայտնաբերման դեպքում առաջնային սարքի կապի մակարդակն անցնում է Recovery վիճակի, որում կրկնվում է տակտային ազդանշանի սինքրոնացումը և տվյալների փոխանցման դողի կարգաբերումը: Տեխնիկական արձանագրությամբ տրված ժամանակահատվածի ընթացքում տվյալների տեղափոխման բացակայության դեպքում համակարգն առաջնային սարքի հրահանգով անցնում է U1 կամ U2 էներգախնայող ռեժիմներից որևէ մեկին: Այդ երկու վիճակների միջև ընտրությունը պայմանավորված է ավելի բարձր մակարդակներից եկող հրահանգներով: U2 վիճակում, ըստ տեխնիկական արձանագրության, համակարգը պետք է սպառի ավելի փոքր էներգիա, քան U1-ում, սակայն այս վիճակից աշխատանքայինին անցումը կատարվում է ավելի մեծ ժամանակահատվածում: Այդ պատճառով ծրագիրը, որը կառավարում է ՀՀԴ-ն, կախված առաքող տվյալների չափից, որոշում է, թե որ էներգախնայող ռեժիմի անցման հրահանգը պետք է ուղարկվի: U3-ն ամենացածր էներգասպառմամբ վիճակն է, և հետևաբար՝ շատ ավելի երկար ժամանակ է անհրաժեշտ այս վիճակից աշխատանքայինին անցման համար: Այս վիճակին անցումը կատարվում է համակարգչի՝ էներգախնայող ռեժիմին անցման դեպքում: Տեխնիկական արձանագրության մեջ նկարագրված է նաև U3 վիճակում կապի մասնակիցների հաղորդակցության մեխանիզմը՝ ցածր հաճախային պարբերական ազդանշանը (ՑՀՊԱ): Այն անհրաժեշտ է, քանի որ U3 վիճակում հիմնական եղանակով հաղորդակցությունը առաջնային և երկրորդային սարքերի միջև բացակայում է, և էներգիայի խնայման նպատակով հիմնական տակտային ազդանշաններն անջատվում են: ՑՀՊԱ-ն օգտագործվում է առաջնային կամ երկրորդային սարքերի կողմից հրահանգների առաքման նպատակով: Այդ հրահանգների տեսակները նկարագրված են տեխնիկական արձանագրության մեջ: U1, U2 և U3 վիճակներից անցումը

աշխատանքային վիճակի կատարվում է Recovery վիճակում համակարգի կարգաբերման միջոցով: Կապի մակարդակի նախագիծը պարունակում է տվյալների առաքման, ստացման ուղիները և հիմնական ՎՎՄ-ն, որը, ինչպես վերևում նկարագրվեց, ղեկավարում է տվյալների հոսքը, ինչպես նաև սխալների դեպքում կրկնում է կապի հաստատման գործընթացը:

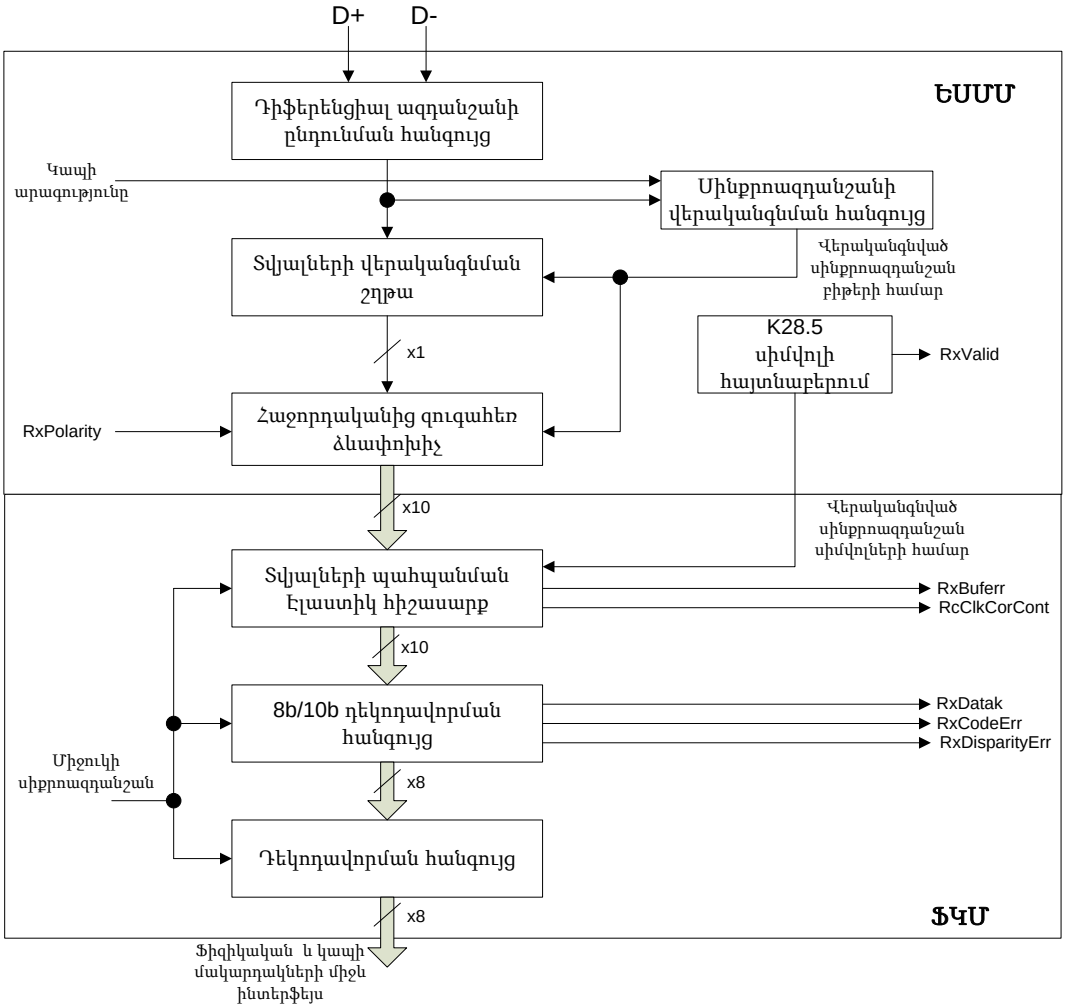
Ֆիզիկական մասն ապահովում է ՀՀԴ3.0 ստանդարտում նկարագրված ազդանշանների ավելի ցածր մակարդակ փոխանցումը: Այն բաղկացած է երկու մասից՝ կոդավորման ենթամակարդակ և երկրորդային սարքի միացման ենթամակարդակ [76]: Ֆիզիկական մակարդակի ընդունիչի կառուցվածքը ներկայացված է նկ. 2.6–ում: Կոդավորման ենթամակարդակը ամբողջովին թվային նախագիծ է: Այն պարունակում է 8b/10b կոդավորման-ապակոդավորման, ճկուն բուֆեր և կոդավորիչ-ապակոդավորիչ: Վերջինս նախատեսված է հաստատուն հոսանքի բալանսի պահպանման համար փոխանջատումների անհրաժեշտ քանակությունն ապահովելու համար: Կոդավորման ենթամակարդակն իրականացված է ԾՏԻՍ-ի մեջ և միացվում է կապի մակարդակին որոշակի ստանդարտ ինտերֆեյսով, որի դիտարկման նպատակով նախագծվել են սինթեզվող հաստատումներ:

Երկրորդային սարքի միացման ենթամակարդակը անալոգային նախագիծ է և, հետևաբար, հնարավոր չէ իրականացնել՝ օգտագործելով ԾՏԻՍ–ի ծրագրավորվող բլոկների միջոցով: Այդ պատճառով օգտագործվում է «Synopsys»-ում նախագծված թեստ ԻՍ-ն: Հնարավոր է օգտագործել նաև ԾՏԻՍ-ում առկա գերարագ տվյալների փոխանցման մուտք/ելք տարրերը [35]: Այս մակարդակում կատարվում են հետևյալ գործողությունները.

- անալոգային դիֆերենցիալ ազդանշանի ձևափոխում թվայինի,
- սինքրոազդանշանի և տվյալների վերականգնում,
- հաջորդական տվյալների ձևափոխում զուգահեռի:

Առաջարկվող ՎՀ համակարգի կառուցման մեթոդի դեպքում կապի մասնակիցները կարող են առաքել և ստանալ տվյալները, և, հետևաբար, անհրաժեշտ է աշխատանքի ընթացքում ձևափոխել համակարգը համապատասխան կերպով՝

կապի մասնակիցներից յուրաքանչյուրի տվյալների առաքման և ընդունման հանգույցների թեստավորման համար [68]:



Նկ.2.6. ՀՀԴ 3.0 ֆիզիկական մակարդակի կառուցվածքը

Այսինքն՝ մեկ մասի թեստավորման դեպքում հնարավոր է կապի մասնակիցներից մեկը կատարի առաջնային սարքի դեր և առաքի ղեկավարման փաթեթներ, իսկ մեկ այլ մասի դեպքում կատարի երկրորդային սարքի դեր և ընդունի տվյալները: Հետևյալ մոդուլները նախագծվել են թեստավորմանը նպաստելու և ՀՀԴ-ի ավելի բարձր մակարդակները փոխարինելու նպատակով.

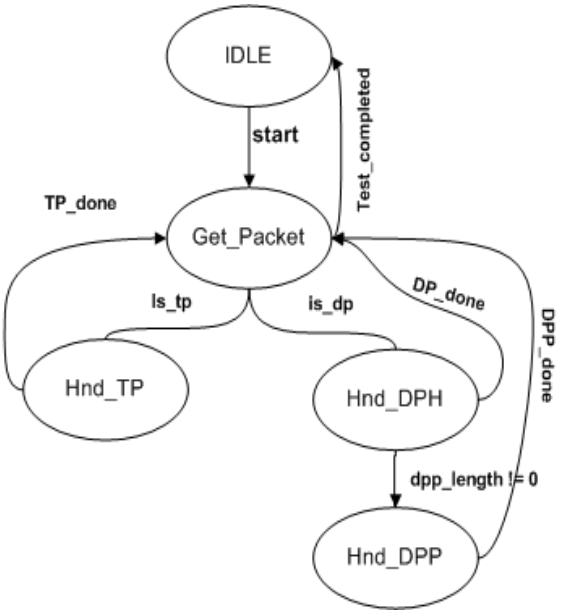
- տվյալների ընդունման և առաքման հանգույցներ,
- տվյալների առաքման և ինտերֆեյսի ղեկավարման հանգույց,
- մոնիտոր,
- պատահական թվերի գեներատոր:

Տվյալների ընդունման հանգույցը միացված է կապի տվյալների ընդունման ինտերֆեյսին: Այս հանգույցն ընդունում է փաթեթները, մշակում՝ օգտակար տվյալների և կապի ղեկավարման հատուկ փաթեթների առանձնացման նպատակով, և փոխանցում մոնիտորին: Այս մոդուլը պարունակում է երկու տարբեր տակտային ազդանշանով հերթ (կարդալու դեպքում ելքում առաջինը հայտնվում է ավելի շուտ գրանցված արժեքը)՝ տվյալների պահպանման և փոխանցման նպատակով: Հերթն ունի գրելու և կարդալու առանձին տակտային ազդանշաններ և, բացի վերը նշվածից, կատարում է նաև տարբեր տակտային ազդանշանների տիրույթներով անցնող տվյալների հոսքի սինքրոնացում: Այն կիրառվում է, քանի որ թեստավորման համակարգը և ՀՀԴ-ի կապի մակարդակը օգտագործում են տարբեր տակտային ազդանշաններ:

Նախատիպում արագագործության բարձրացման նպատակով բացակայում են ՀՀԴ ավելի բարձր մակարդակները և ղեկավարման ծրագրեր չեն օգտագործվում, հետևաբար՝ փաթեթների ամբողջական վերլուծում չի կատարվում: Սակայն կապի մակարդակի տվյալների առաքման և ընդունման ուղիների լիարժեք թեստավորման համար տվյալների ընդունման հանգույցը պետք է տեղեկություններ ունենա ուղարկվող բոլոր փաթեթների և դրանց պարունակության մասին: Այդ պատճառով օգտագործվում է, այսպես կոչված, ՙապարատային թեստ՚ (ԱԹ) տվյալների կառուցվածքը: Այն պարունակում է առաքվող փաթեթների տեսակները և առաքման հաջորդականությունը: Թեստավորման գործընթացը կատարելուց առաջ կատարվում է համապատասխան ԱԹ ընտրությունը, որը միևնույն պարունակությամբ օգտագործվում է կապի մասնակիցների կողմից: ԱԹ-ի պահպանման նպատակով օգտագործվում են ԾՏԻՍ-ում ներդրված ԿԳՀ հիշողության հանգույցները՝ ծրագրավորվող տրամաբանական տարրերի քանակը նվազեցնելու համար:

Ինտերֆեյսը ղեկավարող հանգույցները և մոնիտորները կարող են ծրագրավորվել՝ տվյալների առաքման և ընդունման ռեժիմներում աշխատելու նպատակով: Ինտերֆեյսի՝ առաքման ռեժիմում ծրագրավորվելու դեպքում այն կարդում է ԱԹ առաջին տարրի պարունակությունը և նկարագրված փաթեթն առաքում՝ ինտերֆեյսի ազդանշաններին համապատասխան արժեքները տալով, և

այդպես շարունակ՝ մինչև ԱԹ վերջին տարրը: Առաքման ռեժիմում ծրագրավորված ինտերֆեյսի ղեկավարման բլոկի հիմնական ՎՎՄ–ն ներկայացված է նկ. 2.7-ում:

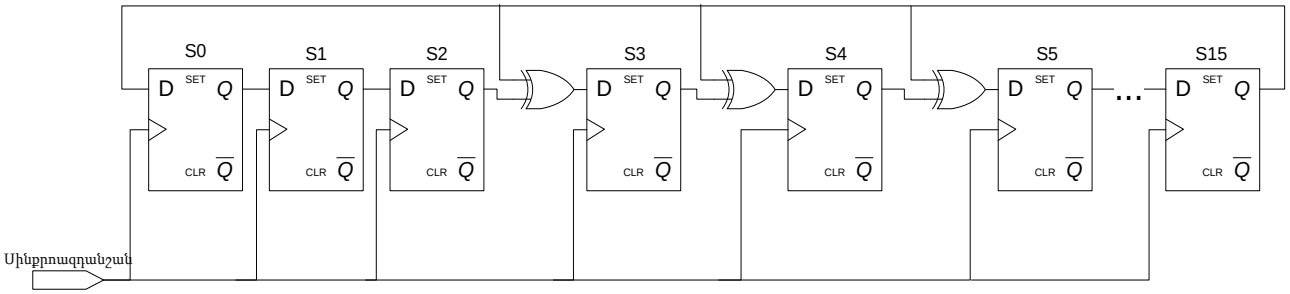


Նկ.2.7. Տվյալների առաքման ՎՎՄ-ն

Սկզբնարժեքավորման ազդանշանի ոչ ակտիվ վիճակին անցումից անմիջապես հետո ՎՎՄ-ն գտնվում է IDLE վիճակում: Start ազդանշանը ստանալով՝ ՎՎՄ-ն անցնում է Get_Packet վիճակին, որում հերթականությամբ կարդում է ԱԹ–ում պարունակվող փաթեթների նկարագրությունները: Կապի ղեկավարման փաթեթի դեպքում ՎՎՄ-ն անցնում է Hnp_TP վիճակին, որին առաքվում է ղեկավարման փաթեթը՝ կապի մակարդակի հրամանների համար նախատեսված մուտքերը համապատասխան կերպով ծրագրավորելով: Tp_done ազդանշանի մեկ արժեքը նշանակում է, որ փաթեթը հաջողությամբ առաքվել է կապի մակարդակի կողմից: ՎՎՄ-ն, Tp_done–ի մեկ արժեքը ստանալով, անցնում է Get_Packet վիճակին, որում կարդում է ԱԹ հաջորդ փաթեթի նկարագրությունը: Տվյալների փաթեթի դեպքում ՎՎՄ-ն անցնում է Hnd_DPH վիճակին, որում վերլուծվում է փաթեթի նկարագրությունը: Եթե առաքվող տվյալների չափը 0 է ([70]–ում նկարագրված են 0 չափով փաթեթների օգտագործման եղանակներ), ապա այն դիտարկվում է որպես ղեկավարման փաթեթ, և կատարվում է վերը նկարագրված քայլերի հաջորդականությունը, հակառակ դեպքում՝ ՎՎՀ–ն անցնում է Hnp_DPP վիճակին, որում ակտիվացվում է պատահական թվերի գեներատորը, և կապի մակարդակի ինտերֆեյսով տվյալները փոխանցվում են ՀՀԴ-ին, որն իր հերթին տվյալները փոխանցում է կապի մյուս մասնակցին:

ՀՀԴ-ով առաքվող տվյալները չեն ենթարկվում որոշակի օրինաչափության, հետևաբար՝ թեստավորումը ՀՀԴ ստանդարտի՝ իրական կյանքում օգտագործմանն ավելի մոտ դարձնելու համար ցանկալի է, որ տվյալների գեներացումը նույնպես չունենա որոշակի օրինաչափություն, սակայն, միևնույն ժամանակ, տվյալների փոխանցման ընթացքում անհրաժեշտ է հայտնաբերել սխալները: Այդ նպատակով օգտագործվում են 16 բիթանի երկու գծային հետադարձ կապով տեղաշարժվող ռեգիստր (ԳՀԿՏՌ), որոնցից ամեն մեկը գեներացնում է երկուական բայթ մեկ պարբերության ընթացքում: Այս մոտեցումը արագագործության սահմանափակումներ չի առաջացնում, քանի որ ՀՀԴ կապի ինտերֆեյսի տվյալների դողի լայնությունը 32 բիթ է, և օգտագործվում է 2 ԳՀԿՏՌ, որոնք ապահովում են անհրաժեշտ 4 բայթ տվյալները ամեն պարբերության ընթացքում: Միևնույն ԳՀԿՏՌ–ն օգտագործում է նաև մոնիտորը, և սկզբնարժեքավորման ընթացքում ստացվում է միևնույն արժեքը: ԳՀԿՏՌ կառուցման բազմանդամը նույնն է, ինչ ՀՀԴ-ում օգտագործվող կոդավորիչինը [70] (նկ. 2.8):

$$G(X) = X^{16} + X^5 + X^4 + X^3 + 1 \tag{2.1}$$



Նկ.2.8. Պատահական թվերի գեներացման ԳՀԿՏՌ-ն

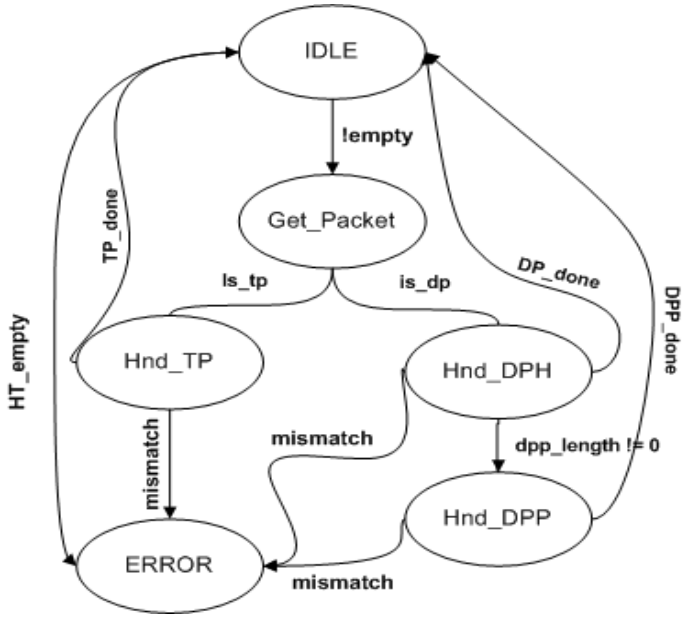
Տվյալների առաքման ռեժիմում մոնիտորը միացված է ընդունիչին և կատարում է միայն կապի ղեկավարման փաթեթների ստուգում: Այն միացվում է պատահական թվերի գեներատորին և ընդունիչին, երբ ծրագրավորված է ընդունման ռեժիմում: Տվյալների ստացման ռեժիմում մոնիտորը ստանում է տվյալները ընդունիչի հերթից և համապատասխան ազդանշանի միջոցով ակտիվացնում է պատահական թվերի գեներատորը: Քանի որ տվյալների առաքման և ստացման բլոկների ԳՀԿՏՌ-ներն ունեն միևնույն սկզբնական արժեքը և կառուցվածքը, ստացված և գեներացված

տվյալները պետք է համընկնեն: Չհամընկնելու դեպքում կարելի է ասել, որ տվյալների տեղափոխման ուղու մեջ սխալ կա:

Մոնիտորի մեջ ներդրված են կապի տվյալների առաքման և ընդունման քայլերի հաջորականության դիտարկումը կատարող ոչ մեծ քանակությամբ սինթեզվող հաստատումներ: Բոլոր անհրաժեշտ հաստատումների ներդրումը հնարավոր չէ: Մոնիտորի նախագիծը բավականին մեծ է: Նախագիծն ավելի շատ չբարդացնելու նպատակով փաթեթների վերլուծություն չի կատարվում նաև մոնիտորում: Ինչպես վերևում նշվեց, տեղափոխված տվյալների ճշգրտությունը ստուգելու նպատակով ներմուծվել է ԱԹ տվյալների կառուցվածքը, որն ընդհանուր է կապի մասնակիցների համար: Մոնիտորի հիմնական ՎՎՄ-ն ներկայացված է նկ.2.9-ում: Սկզբնական վիճակում, երբ ընդունիչից տվյալներ չեն ստացվել, այսինքն՝ ընդունիչի տվյալների հերթը դատարկ է, ՎՎՄ-ն գտնվում է IDLE վիճակում, որում հրահանգ է ուղարկվում պատահական թվերի գեներատորին՝ ԳՀԿՏՈ–ն սկզբնական վիճակին բերելու նպատակով: Երբ տվյալների ստացման հետևանքով ընդունման հերթի empty ազդանշանը ընդունում է 0 արժեքը, ՎՎՄ-ն անցնում է Get_Packet վիճակին, որում կարդում է հերթում պարունակվող փաթեթը, և եթե այն կապի ղեկավարման փաթեթ է, ապա անցնում է Hոp_TP վիճակին: Hոp_Tp վիճակում ստացված փաթեթը համեմատվում է ԱԹ-ում պարունակվող փաթեթի համապատասխան նկարագրության հետ և սխալի դեպքում անցնում է ERROR վիճակին, հակառակ դեպքում՝ IDLE վիճակին: Get_Packet վիճակում տվյալներ պարունակող փաթեթ ստանալու դեպքում ՎՎՄ-ն անցնում է Hոp_DPH վիճակին, որում համեմատում է փաթեթի վերնագիրը ԱԹ համապատասխան փաթեթի նկարագրության հետ:

Եթե տվյալները համընկնում են, և փաթեթում տվյալների պարունակությունը 0 չէ, ապա ՎՎՄ–ն անցնում է Hոp_DPP վիճակին: Hոp_DPP–ում համապատասխան ազդանշանով մոնիտորը ակտիվացնում է պատահական թվերի գեներատորը և համաեմատում է ստացված և գեներացված տվյալները: Եթե տվյալները չեն համընկնում, ՎՎՄ-ն անցնում է ERROR վիճակին, որը նշանակում է, որ փոխանցված տվյալում սխալ կա: Տվյալների համընկնման դեպքում ՎՎՄ-ն անցնում է IDLE

վիճակին՝ մինչև կապի մակարդակի տվյալների ստացման ինտերֆեյսից նոր փաթեթի ստացումը:

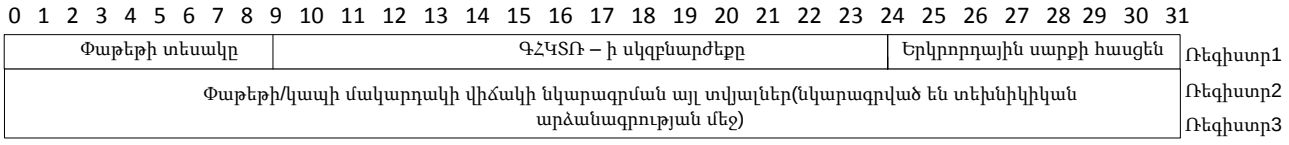


Նկ.2.9. Մոնիտորի հիմնական ՎՎՄ-ն

Տվյալների գեներացումն առաքող համակարգում և համեմատումն ընդունիչում ուսումնասիրելու համար անհրաժեշտ է դիտարկել տվյալների փաթեթի գոյության տևողությանը: Սկզբում փաթեթի նկարագրությունը վերցվում է ԱԹ-ից: Փաթեթի չափը 0 լինելու դեպքում այն դիտարկվում է այնպես, ինչպես կապի ղեկավարման փաթեթը: Առաքման բլոկը ՀՀԴ կապի համապատասխան ինտերֆեյսով փոխանցում է փաթեթը և գեներացված տվյալները ՀՀԴ կապի մակարդակին, որն իր հերթին այն փոխանցում է կապի մյուս մասնակցին: Ընդունելուվ փաթեթի վերնագիրը՝ այն գրանցվում է հերթի մեջ, որի դատարկ լինելու դեպքում համապատասխան ցուցչի արժեքը 0 է դառնում, և մոնիտորը, այդ ազդանշանը ստանալով, համեմատում է փաթեթը և ԱԹ համապատասխան նկարագրությունը: Տվյալները գրանցվում են մեկ այլ հերթի մեջ, որն իրականացված է ԾՏԻՍ-ում ներդրված ԿԳՀ հանգույցների միջոցով: Վերնագրի համընկնելու դեպքում համեմատվում են ստացված տվյալներն ընդունիչում գեներացվածի հետ, և համընկնման դեպքում փաթեթի մշակումն ավարտվում է: Սխալի դեպքում փաթեթի նկարագրությունը գրանցվում է նախատեսված հիշողության բլոկում, որի պարունակությունը թեստավորման ավարտից առաջ փոխանցվում է համակարգչին հաջորդական ինտերֆեյսով: Այսինքն՝ անհաջողության դեպքում

փաթեթի մշակումն ավարտվում է՝ այն գրանցելով հիշողության մեջ, իսկ այն փաթեթների մշակումը, որոնց փոխանցումը հաջողվում է առանց սխալների, ավարտվում է մոնիտորում: Այս եղանակը ընտրված է՝ ԾՏԻՍ-ի ռեսուրսների հնարավորինս քիչ օգտագործման նպատակով: Հիշողության դատարկ լինելու դեպքում կարելի է պնդել, որ ՄՌՓՄ նկարագրության մեջ տվյալների փոխանցման ուղիում սխալներ չկան, իսկ եթե այն դատարկ չէ, կարելի է հեշտությամբ պարզել, թե որ տեսակի փաթեթների դեպքում են սխալներ առաջանում, և փոխելով ՄՌՓՄ նկարագրությունը՝ կրկնել նույն թեստավորումը սխալներ առաջացնող փաթեթներով:

ԱԹ–ն հերթ է, որը պարունակում է տվյալ թեստում նախատեսված փաթեթների նկարագրությունները որոշակի հերթականությամբ: Միևնույն պարունակությամբ ԱԹ-ն կա ընդունող և առաքող մոդուլներում իրականացված ԾՏԻՍ-ի ԿԳՀ բլոկներում: ԱԹ միավորի պարունակությունը ներկայացված է նկ. 2.10-ում:



Նկ.2.10. Փաթեթի նկարագրությունը ԱԹ-ում

Չնայած նրան, որ ԾՏԻՍ նախատիպը համեմատվում է մինչ արտադրական թեստավորումը, սխալների հայտնաբերումը, տեղայնացումը և ֆունկցիոնալ ծածկույթի հաշվարկը գրեթե նույնքան բարդ են, որքան թեստավորումն արտադրությունից հետո [77]: Ինչպես նշվեց նախորդ գլխում, սխալների հայտնաբերման և տեղայնացման հիմնական խոչընդոտները պայմանավորված են ԾՏԻՍ ներքին ազդանշանների տեսանելիության և դրանց ղեկավարման հնարավորության բացակայությամբ: Այս խնդիրները կարելի է լուծել սինթեզվող հաստատումների ներդրմամբ, որոնք աշխատանքի ընթացքում դիտարկում են ՄՌՓՄ նկարագրության տրամաբանության որոշակի մասերը և հայտնաբերում ՀՀԴ տեխնիկական արձանագրության կամ իրականացմանը ներկայացվող պահանջները չբավարարող վարքի դեպքերը [77]:

Տվյալների առաքման հանգույցի թեստավորման համար նախագծվել են 34 ֆունկցիոնալ ծածկույթի հաշվարկի և 6 սխալ վարքի հայտնաբերման հաստատումներ: Ֆունկցիոնալ ծածկույթի համար նախագծված հաստատումը ստուգում է՝ կապի

մակարդակում [66]–ում նկարագրված մեխանիզմներով հայտնաբերվե՞լ է արդյոք սխալ կառուցված փաթեթ, թե՞ ոչ:

```
sequence Rx_lbad_tx_lrty_pkt;,
@(posedge link_clk) !rply_lbad_rqst ##1 rply_lbad_rqst,
##1 (tx_state == RPLY0)[->1] ##1 tx_last[TX_REQ_RPLY] [->1],
endsequence,
svc_u3link_tctrl_Rx_lbad_tx_lrty_pkt: cover property (Rx_lbad_tx_lrty_pkt):
```

Ստորև ներկայացվող հաստատումը նախագծված է սխալ վարքի հայտնաբերման համար: Այն դիտարկում է առաքվող տվյալների չափը և հայտնաբերում է 1024 բայթից ավելի տվյալների առաքումը: Այս սխալը կարող է առաջանալ, երբ բարձր մակարդակներից եկող փաթեթների սահմանները հայտնաբերված չեն, և հաջորդ փաթեթը հաշվարկվում է որպես մշակվող փաթեթի մաս: Այս սխալի առաջացման պատճառ կարող են լինել նաև բազմաթիվ այլ չնախատեսված վարքեր:

```
sva_link_tctrl_d_gt_1024: assert property( @(posedge link_clk),
!(data_length > 'd1028))else sva_link_error("The data payload is greater than 1024"):
```

Տվյալների ընդունման հանգույցի համար նախագծվել է 31 ֆունկցիոնալ ծածկույթի հաշվարկի և 5 սխալ վարքի հայտնաբերման հաստատում: Հետևյալ հաստատումը ստուգում է՝ արդյոք տվյալներ ընդունող բլոկը 8b/10b կոդավորման սխալի հայտնաբերման դեպքում պատասխանու՞մ է LBAD կապի ղեկավարման փաթեթով, և հիմնական ՎՎՄ-ն անցնու՞մ է RECOVERY վիճակին՝ կապի կարգաբերման նպատակով: Որպեսզի աշխատանքային վիճակին անցնելիս սխալ փաթեթի կրկնման դեպքում մշակումը կատարվի ճշգրիտ, համապատասխան ազդանշանն ընդունում է 1 արժեքը:

Ֆունկցիոնալ ծածկույթի հաշվարկի հաստատումը դիտարկում է փաթեթի սահմանները կազմող սիմվոլները: Այն հաջողությամբ է ավարտվում այն դեպքում, երբ այդ սիմվոլներից որևէ մեկը սխալ է:

```
sequence dpp_endf_err_1; @(posedge mac3_clk),
(dpp_length[1:0] != 2'h0) ##1 (dpp_length[1:0] == 2'h0) ##0 (pkt_state == DPP_XFR),
##0 (match4(pkt_window_9b, {EPF, END, END, END}) == 3),
```

```
##1 ({lnmcr_data_last, lnmcr_data_err} == 2'b10),
endsequence,
svc_u3link_rpkt_dpp_endf_err_1: cover property (dpp_endf_err_1):
```

Այս դեպքը նկարագրված է ՀՀԴ տեխնիկական արձանագրության մեջ: ՀՀԴ կապի մակարդակը պետք է ի վիճակի լինի հայտնաբերելու փաթեթի ավարտը նույնիսկ այն դեպքում, երբ սահմանը կազմող սիմվոլներից որևէ մեկը ճշգրիտ չէ: Եթե թեստավորումը հաջողությամբ է ավարտվել, և այս հաստատումը ծածկված է, ապա նախագծված ՀՀԴ կապի մակարդակը բավարարում է արձանագրության վերը նկարագրված պայմանը: Ֆունկցիոնալ ծածկույթի այս կետի բավարարումը կարելի է ապահովել՝ ֆիզիկական և կապի մակարդակների միջև թեստավորման լրացուցիչ հանգույց ավելացնելով, որը կներմուծի սխալ փաթեթը սահմանափակող սիմվոլը համապատասխան ժամանակահատվածում:

Կապի մակարդակի հիմնական ՎՎՄ–ի համար նախագծվել են 332 ֆունկցիոնալ ծածկույթի հաշվարկի և 12 սխալ վարքի հայտնաբերման հաստատումներ: Սխալ վարքի հայտնաբերման հաստատումը ստուգում է՝ արդյոք սխալների հաշվիչը ստանում է սկզբնական արժեքը համապատասխան ազդանշանի դեպքում: Այս վարքը նույնպես նկարագրված է ՀՀԴ արձանագրության մեջ: Պայմանը չբավարարելու դեպքում ՎՎՄ-ն, առանց որևէ պատճառի, աշխատանքային վիճակից կանցնի RECOVERY վիճակին, քանի որ սխալների քանակը զրոյացված չէ:

```
sequence reset_link_error_cnt_after_reset;  @(posedge link_clk),
ltmcs_link_err_ctr!=0,
##1  !link_reset_n,
##1 link_reset_n [->1],
##1 ltmcs_link_err_ctr==0,
endsequence:
```

Ֆունկցիոնալ ծածկույթի հաշվարկի հաստատումը ստուգում է՝ արդյոք թեստավորման ընթացքում կապի մակարդակն անցել է ցածր էներգասպառմամբ U2 վիճակին:

```
sequence successful_rem_u2_exit_recovery; @(posedge link_clk),
(link_state != U2) ##1 (link_state == U2),
##1 lfps_ux_exit [->1],
```

```
##1 (link_state != U2) [->1],
##1 (link_state == RECOV),
endsequence,
svc_u3ltssm_successful_rem_u2_exit_recovery: cover property,
(successful_rem_u2_exit_recovery):
```

Վերևում նկարագրված հաստատումների սինթեզը կատարվել է Synopsys's Synplify Premier [78] ծրագրային գործիքի միջոցով: Սինթեզի արդյունքները ներկայացված են աղ. 2.1–ում:

Աղյուսակ 2.1

ԾՏԻՍ–ի սպառվող ռեսուրսները միայն հաստատումների իրականացման դեպքում

Ռեսուրսները	**Տվյալներ առաքող հանգույց**	**Տվյալների ընդունման հանգույց**	**Կապի հիմանական ՎՎՄ**	**ՀՀԴ3.0 կապի մակարդակ + ՖԿՄ**
Slice տրիգեր	214	195	2087	2715
Slice ԻԱ	209	190	2036	7182

Սինթեզը կատարվել է Virtex5-ի [79] համար, քանի որ Synopsys-ի HAPS51T մակետի կառուցման հարթակում օգտագործվում է այդ ԾՏԻՍ-ը: Սինթեզը կատարվել է միայն սինթեզվող հաստատումների և արդյունքների գրանցող բլոկների ծախսած ռեսուրսների մասին ավելի ճշգրիտ պատկերացում կազմելու համար: Ինչպես երևում է աղյուսակից, թեստավորման համար նախատեսված բլոկները սպառում են ՀՀԴ կապի մակարդակի նախագծի 91% -ի չափ ծրագրավորվող տրամաբանական բլոկներ և 33%-ի չափ տրիգերներ: Այսքան ռեսուրսների օգտագործումը թույլատրելի է փոքր տարրերի քանակությամբ նախագծերի դեպքում, սակայն միևնույն մոտեցումը չի կարելի օգտագործել տարրերի մեծ քանակությամբ նախագծերի համար, քանի որ ԾՏԻՍ ռեսուրսները սահմանափակ են, և 70 % -ից ավելի օգտագործումը նվազեցնում է արագագործությունը: Սա առաջարկվող մեթոդի հիմնական թերությունն է:

ԾՏԻՍ–ի սարքերի մեծ մասը հնարավորություն է տալիս ձևափոխելու սարքի մի մասի կառուցվածքը՝ առանց ամբողջ սարքը վերածրագրավորելու, մասնակի ծրագրավորման ֆայլը ԾՏԻՍ-ի մեջ ներբեռնելու միջոցով [80]: Այս հատկությունը ՎՀ-

ներին տալիս է ավելի մեծ ճկունություն: Նախագիծը բաժանվում է երկու մասի՝ ստատիկ և վերակառուցավորվող: Մասնակի ծրագրավորման ընթացքում ստատիկ տրամաբանությունը չի փոփոխվում, և համակարգը շարունակում է անխափան աշխատանքը, իսկ վերակառուցավորվող մասերի կառուցվածքը փոփոխվում է մասնակի ծրագրավորման ֆայլին համապատասխան: Այս հատկության օգտագործման բազմաթիվ պատճառներ կան, որոնցից մեկն ավելի փոքր ԾՏԻՍ-ի օգտագործումն է միևնույն խնդրի լուծման նպատակով: Բացի դրանից, այս հատկությունը հնարավորություն է տալիս կառուցելու նոր տեսակի սարքեր, որոնք ավելի մոտ են ճկունության տեսանկյունից ծրագրային լուծումներին, սակայն պահպանում են ապարատային արագագործությունը: Մասնակի ծրագրավորման գործընթացը ներկայացված է [80]-ում: Մասնակի վերակառուցավորման հատկությունը կարելի է օգտագործել նախագծված թեստավորման համակարգում սպառվող ռեսուրսների նվազեցման նպատակով:

Տվյալների գրանցման և ՀՀԴ կապի մակարդակն իրականացվում են նախագծի ստատիկ մասում, իսկ հաստատումները՝ վերակառուցավորվող: Հաստատումները բաժանվում են խմբերի՝ ըստ ՀՀԴ կապի մակարդակի հիմնական ՎՎՄ վիճակի կամ ըստ թեստի նկարագրության: Առաջին դեպքում հաստատումների խմբեր կարող են լինել SS.Disable, Rx.Detect, Polling, Recovery, U0 և այլն: Մասնակի ծրագրավորման ֆայլեր պետք է գեներացվեն նշված խմբերից յուրաքանչյուրի համար: Աշխատանքի ընթացքում վերակառուցավորվող մասերի կառուցվածքը փոփոխվում է՝ կախված կապի մակարդակի առկա վիճակից: Օրինակ, U0 վիճակում վերակառուցավորվող համակարգերը ծրագրավորվում են U0 մասնակի ծրագրավորման ֆայլի միջոցով: Այս եղանակի դեպքում թեստավորումը կատարվում է առանց ընդհատման, քանի որ թեստավորման համակարգը վերակառուցավորվում է՝ կախված ՀՀԴ վիճակից, առանց արտաքին միջամտության: Սակայն այս եղանակը կիրառելու համար անհրաժեշտ է մասնակի ծրագրավորման մեծ արագություն, որը հնարավոր չէ ապահովել JTAG ինտերֆեյսի և զուգահեռ ծրագրավորման եղանակների միջոցով [81]: Պահանջվող արագագործությանը կարելի է հասնել ICAP–ի ղեկավարման բլոկի միջոցով [82]: Ըստ թեստի՝ բաժանման եղանակը չի պահանջում

վերակառուցավորման մեծ արագագործություն: Թեստը սկսվում է միայն մասնակի ծրագրավորման ավարտից հետո, երբ բոլոր հաստատումները հաջողությամբ ներդրված են: Այս մեթոդի օգտագործման արդյունքները ցածր էներգասպառման վիճակների ստուգման համար ներկայացված են աղ. 2.2–ում:

Աղյուսակ.2.2

Էներգախնայող վիճակների ստուգման նպատակով նախագծված հաստատումների ԾՏԻՍ–ի ռեսուրսները

Ռեսուրսները	**Տվյալներ առաքող հանգույց**	**Տվյալների ընդունման հանգույց**	**Կապի հիմանական ՎՎՄ**	**ՀՀԴ3.0 կապի մակարդակ + ՖԿՄ**
Slice տրիգեր	27	29	132	2715
Slice ԻԱ	23	16	202	7182

2.4. Մի քանի տակտային ազդանշանով նախագծերի թեստավորման եղանակը ՎՀ համակարգի կիրառմամբ

ՀՀԴ ստանդարտի զարգացման ընթացքում արագագործությունը զգալիորեն աճել է [70,83], ինչը հնարավորություն է տվել այն օգտագործելու բազմաթիվ բնագավառներում: Արդյունքում՝ համակարգի այլ մասերի հետ փոխհամաձայնեցված աշխատանքի համար ավելացել են տակտային ազդանշանների տիրույթները, որոնք ասինքրոն են մեկը մյուսի նկատմամբ, քանի որ գեներացվում են տարբեր տակտային ազդանշանի աղբյուրներից: Օրինակ՝ պրոցեսորի հետ աշխատելու համար կառուցված տրամաբանական հանգույցը պետք է սինքրոն լինի պրոցեսորի հիմնական դողի սինքրոազդանշանին, և տվյալները կամ հրահանգները պրոցեսորից պետք է փոխանցվեն ՀՀԴ ֆիզիկական մակարդակին տրամաբանական բլոկի միջոցով, որն իր հերթին պետք է սինքրոն լինի ֆիզիկական մակարդակին: Տվյալների փոխանցումը դեպի նպատակակետ կարող է տեղի ունենալ երկուսից ավելի տիրույթներով:

Ինչպես ՀՀԴ-ում, այնպես էլ գրեթե բոլոր ժամանակակից թվային նախագծերի դեպքում նախորդ գլխում նկարագրված մի քանի տակտային ազդանշաններով նախագծերում առկա խնդիրներից խուսափելու համար անհրաժեշտություն է

առաջանում սինքրոնացնելու ազդանշանի անցումը: Այդ նպատակով օգտագործվում են սինքրոնացնող տրամաբանական հանգույցներ [55]: Սինքրոնացնող տրամաբանական բլոկները անհրաժեշտ են այն դեպքում, երբ տրիգերի ելքը միացվում է տարբեր տակտային ազդանշանով մեկ այլ տրիգերի մուտքին: Հնարավոր չէ բավարարել ժամանակային բոլոր սահմանափակումները՝ առանց սինքրոնացնող տրամաբանության: Սինքրոնացնող տարրերը պետք է ապահովեն շատ մեծ հավանականություն, որ ազդանշանի փոփոխությունը նպատակակետում կբավարարի հաստատման և տեղակայման պահանջները: Այսինքն՝ սինքրոնացնող տարրերը ղեկավարում են մետաստաբիլության տարածումը և ապահովում են նպատակակետի տրիգերով ազդանշանի ճշգրիտ գրանցումը [58]:

ՀՀԴ-ն տվյալների ընդունման և առաքման ուղի և ղեկավարման բլոկ է: Թվային նախագծերի մեծ մասն ունի այդպիսի կառուցվածք: Այդ կառուցվածքներում սինքրոնացումը պետք է կատարվի տվյալների փոխանցման դողի և ղեկավարող ազդանշանների համար: Այդ նպատակով օգտագործվում են սինքրոնացման հետևյալ եղանակները.

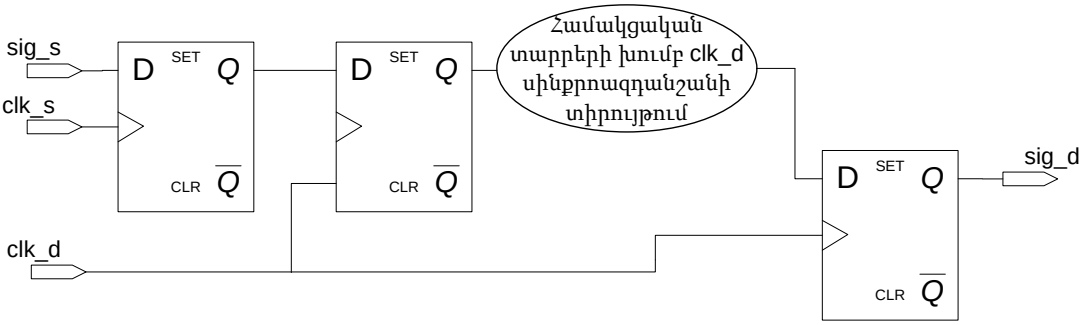
- մի քանի տրիգերով սինքրոնացում,
- իմպուլսների և պատահարների սինքրոնացում,
- տվյալների հոսքի սինքրոնացում:

Տվյալների փոխանցման սինքրոնացման տրամաբանության թեստավորումը թվային նախագծերի կառուցման կարևոր խնդիրներից է: Ինչպես ներկայացվեց նախորդ գլխում, թեստավորման ավանդական մեթոդները խնդիրը միայն մասնակիորեն են լուծում: Այդ պատճառով առաջարկվում է թեստավորումը կատարել ՎՀ-ով կառուցված մակետի միջոցով [83]: Թվային նախագծի ԾՏԻՍ նախատիպում կառուցելով թեստավորման առարկա և դիտարկելով նախատիպի աշխատանքը՝ կարելի է ապահովել ամբողջ համակարգը սինքրոնացնող տրամաբանության թեստավորումը: Սակայն թեստավորման ընթացքում սխալների դեպքում հնարավոր չէ պարզել սխալի պատճառը, այսինքն՝ չի կարելի պնդել, որ համակարգի խափանման պատճառը ոչ ճշգրիտ սինքրոնացումն է: Նույնիսկ այն դեպքում, երբ սխալի պատճառը սխալ սինքրոնացումն է, հնարավոր չէ գտնել ոչ ճշգրիտ սինքրոնացված ազդանշանը:

Այդ խնդրի լուծման համար առաջարկվում է ներդնել սինթեզվող հաստատումները բոլոր սինքրոնացնող տրամաբանական բլոկներում, որոնք աշխատանքի ընթացքում կկատարեն դիտարկում, և համակարգի սխալի դեպքում ներդրված հաստատումների վիճակների միջոցով հնարավոր կլինի պարզել սխալի պատճառը: Այս մասում ներկայացված են սինքրոնացման տրամաբանության տեսակները և յուրաքանչյուր տեսակի համար առաջարկվող հաստատումները:

Ազդանշանները, որոնք հազվադեպ են փոխվում և պահպանում են արժեքը, երկար ժամանակահատվածում, համարվում են ստատիկ: Այս ազդանշանների սինքրոնացման կարիք չկա նպատակակետում, քանի որ մետաստաբիլության հետևանքով առաջացած անորոշ արժեքների չափումը նպատակակետում ֆունկցիոնալ սխալի պատճառ չի կարող լինել: Նախագծողն է որոշում, թե որ ազդանշանները պետք է համարել ստատիկ որոշակի իրականացման դեպքում: Սակայն հնարավոր են դեպքեր, երբ որոշ ազդանշաններ փոփոխվում են նախատեսվածից հաճախ, ինչը կարող է խափանման պատճառ հանդիսանալ: Այդ դեպքում առաջարկվում է ներմուծել հաստատումներ, որոնք կդիտարկեն քվազի-ստատիկ ազդանշանները և որոշակի նվազագույն ժամանակից փոքր ժամանակահատվածում հաստատուն չլինելու դեպում կգրանցեն ազդանշանի չնախատեսված վարքը:

Ղեկավարման ազդանշանների փոխանցումը սինքրոազդանշանի այլ տիրույթ կարելի է կատարել մեկ տրիգերի միջոցով (նկ. 2.11): Այս մոտեցման դեպքում հնարավոր է տրիգերի մետաստաբիլ վիճակի անցումը համակցված տարրերին:

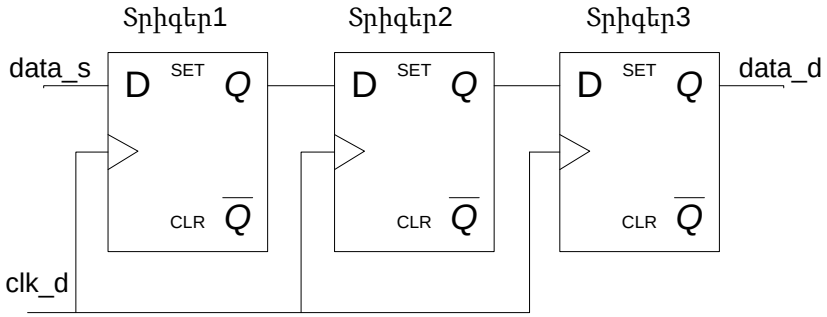


Նկ.2.11. Հանգուցային տրիգերի օգտագործմամբ սինքրոնացման սխեման

Մեկ տրիգերի դեպքում սխալի առաջացման հաճախությունը որոշվում է հետևյալ բանաձևով [57]՝

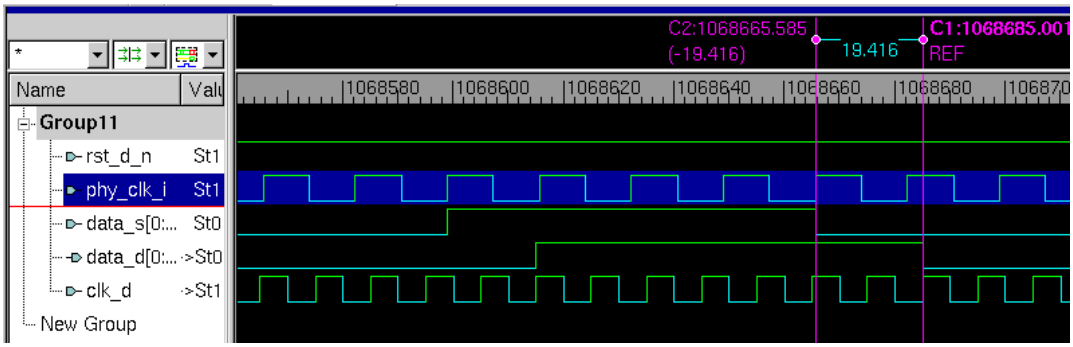
$$F = \frac{F_{սպյալ} \, x \, F_{սռ} \, x \, T_{պլ}}{e^{t_h/\tau}}, \qquad (2.2)$$

որտեղ t_h–ը մետաստաբիլության հանդարտեցման ժամանակն է, այսինքն՝ այն ժամանակը, որն անհրաժեշտ է տրիգերի ելքում կայուն արժեքի հաստատման համար: Փաստորեն t_h–ի համապատասխան արժեքի ընտրության դեպքում հնարավոր է խուսափել մետաստաբիլության՝ դեպի համակցական տարրեր տարածումից: Սակայն դա հնարավոր չէ կատարել, քանի որ սինթեզի ծրագրային միջոցները ձգտում են փոքրացնելու տարրերի չափերը՝ մակերեսի խնայման նպատակով: Դրա պատճառով օգտագործվում են մի քանի հաջորդաբար միացված տրիգերներ (նկ.2.12): Սինքրոնացման այս եղանակը ավելի մանրամասն ներկայացված է [55]-ում:



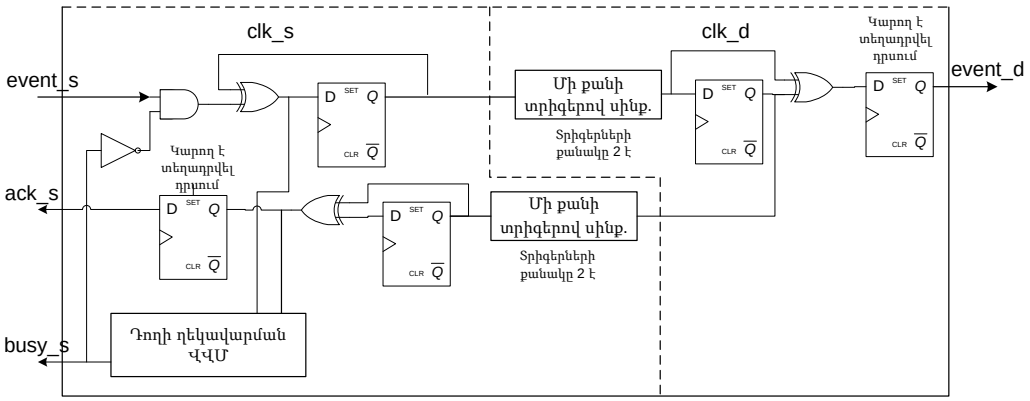
Նկ.2.12. Մի քանի տրիգերներով սինքրոնացումը

Այս եղանակի թեստավորման համար առաջարկվում է ներմուծել հաստատումներ, որոնք կստուգեն՝ արդյոք ազդանշանը հաստատուն է մնացել նպատակակետի սինքրոազդանշանի նախատեսված քանակությամբ պարբերությունների ընթացքում: Նկ.2.13-ում ներկայացված է երկու տրիգերի օգտագործմամբ սինքրոնացումը:



Նկ.2.13. Երկու տրիգերով սինքրոնացման օրինակ

Որոշ դեպքերում անհրաժեշտություն է առաջանում մեկ սինքրոազդանշանի տիրույթում տեղի ունեցող որևէ իրադարձության մասին տեղեկացնելու մեկ այլ տիրույթի տրամաբանական բլոկներին: Սինքրոնացնող տրամաբանական բլոկի կառուցվածքը ներկայացված է նկ.2.14–ում [55]:



Նկ.2.14. Պատահարների սինքրոնացման տրամաբանական հանգույցի կառուցվածքը

event_s մուտքային ազդանշանի միջոցով աղբյուրի տրամաբանական բլոկը տեղեկացնում է մեկ այլ սինքրոազդանշանին որոշակի իրադարձության դեպքում: Օրինակ, փաթեթների առաքման ընթացքում տեխնիկական արձանագրության մեջ տրված փոխանցման սխալների քանակի գերազանցման դեպքում ՀՀԴ կապի մակարդակը պետք է տեղեկացնի բարձր մակարդակներին: Հետևյալ պահանջների ստուգման համար օգտագործվել են սինթեզվող հաստատումները.

- հարցման պատասխանը եկել է նպատակակետից (ack_s ազդանշանով),
- առանց հարցման՝ պատասխան չի եկել,
- մուտքային ազդանշանը անփոփոխ է մնացել՝ մինչ պատասխան ստանալը,
- փոխծածկված հարցումներ չկան:

Վերջին կետի ստուգման համար նախատեսված հաստատումը հետևյալն է.

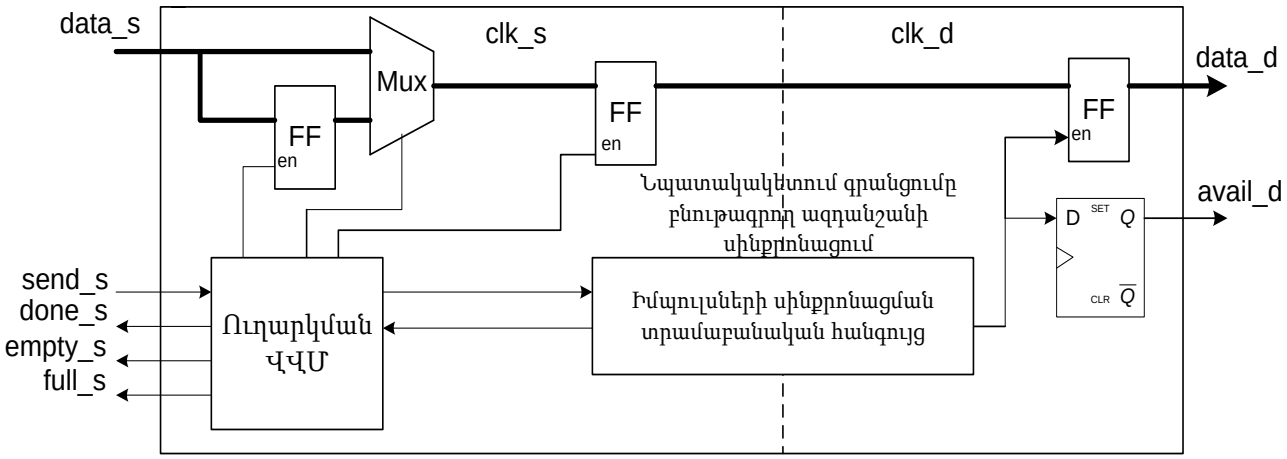
```
always @(posedge event_s_mod or negedge rst_s_n or negedge rst_d_n),
begin,
if (!rst_s_n || !rst_d_n),
flag <= 1'b0,
```

```
    else if (event_s_mod),
      flag <= 1'b1,
  end,
always @(posedge event_d or negedge rst_s_n or negedge rst_d_n begin,
    if (!rst_s_n || !rst_d_n),
      flag <= 1'b0,
    else if (event_d && !flag_reg),
      flag <= 1'b0,
 end,
…
property NoExtraEventOccurs,
@(posedge event_d) disable iff ( !rst_s_n || !rst_d_n ),
flag,
endproperty:
```

flag ազդանշանն ընդունում է մեկ արժեք այն դեպքում, երբ պատահարը տեղի է ունեցել, այսինքն՝ մուտքային ազդանշանի մեկ արժեքի դեպքում, և պահպանում է արժեքը, մինչ ack_s ազդանշանի միջոցով սինքրոնացնող տրամաբանությունը կտեղեկացնի նպատակակետում պատահարի գրանցման մասին: Նկարագրված հաստատումը մուտքային ազդանշանի դրական ճակատի դեպքում ստուգում է flag ազդանշանը և վերջինիս մեկ արժեքի դեպքում տեղեկացնում է սխալ վարքի հայտնաբերման մասին:

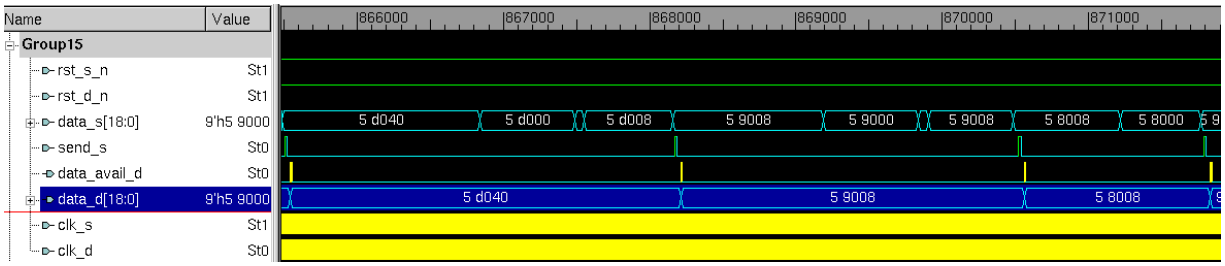
Ղեկավարման ազդանշանների՝ դողի սինքրոազդանշանների տարբեր տիրույթներով անցումը սինքրոնացնելու համար չի կարելի օգտագործել մի քանի տրիգերներով կառուցվածքը՝ նախորդ գլխում ներկայացված սխալ ազդանշանի առաջացումից խուսափելու համար: Նման դեպքերում օգտագործվում է նկ. 2.15–ում ներկայացված կառուցվածքը: send_s ազդանշանը ընդունում է մեկ արժեքը՝ մուտքային ազդանշանի հաստատուն լինելու դեպքում: Տվյալները պահվում են ներքին հիշողության բլոկում և նպատակակետում գրանցվում են avail_d ազդանշանի միջոցով:

Այս կառուցվածքի դեպքում միջանկյալ սխալ արժեքների փոխանցումից հնարավոր է խուսափել: Նկարագրված գործընթացը ներկայացված է նկ.2.16-ում [55]:



Նկ.2.15. Ղեկավարող ազդանշանների դողի սինքրոնացման հանգույցը

Սինքրոնացման այս եղանակի դեպքում գործում են միևնույն պահանջները, ինչ նախորդի դեպքում:

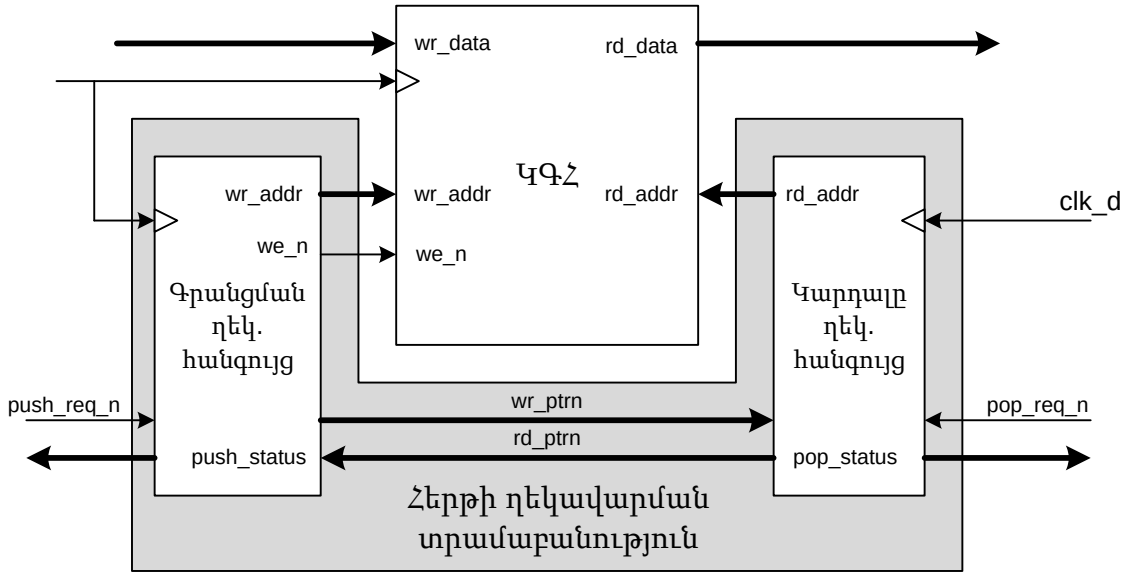


Նկ.2.16. Ղեկավարող ազդանշանների դողի սինքրոնացման օրինակ

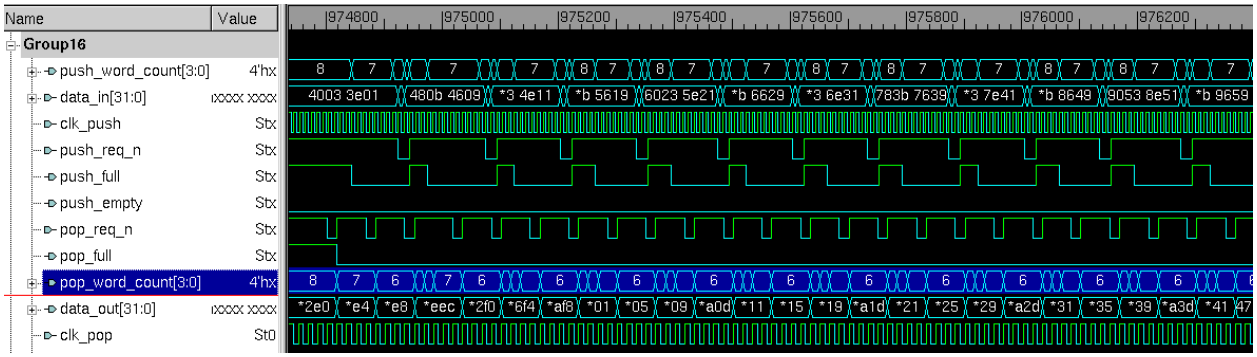
Տվյալների հոսքը համարվում է տվյալների՝ մեկը մյուսի հետևից տեղափոխումը սարքի մեջ առկա հիշողության բլոկների սահմաններում: Հոսքի սինքրոնացման համար օգտագործվում է նկ.2.17–ում ներկայացված երկու սինքրոազդանշանով հերթը: Տվյալները գրանցվում են աղբյուրի սինքրոազդանշանով և դուրս են բերվում նպատակակետի սինքրոազդանշանով:

Գրեյի կոդավորումն օգտագործվում է՝ կարդալու և գրանցելու ցուցիչները կառուցելու նպատակով: Ինչպես նախորդ գլխում ներկայացվեց, Գրեյի կոդավորումն անհրաժեշտ է միջանկյալ սխալ ազդանշաններից խուսափելու համար, քանի որ

միաժամանակ մի քանի բիթի փոփոխություն հնարավոր չէ: Հերթի աշխատանքը ներկայացված է նկ.2.18-ում [55]:



Նկ.2.17. Հերթի միջոցով տվյալների հոսքի սինքրոնացման հանգույցի կառուցվածքը



Նկ.2.18. Հերթի միջոցով տվյալների հոսքի սինքրոնացման օրինակ

Տվյալների հոսքի այս եղանակով սինքրոնացումը հաջողությամբ իրականացնելու համար անհրաժեշտ է, որ սինքրոնացման տրամաբանությունը բավարարի հետևյալ պահանջները [85].

- հերթի ցուցիչների արժեքների թռիչքներ չկան Գրեյի կոդավորման դեպքում,
- հերթի կարդալու ցուցիչը կայուն է տվյալների դուրսբերման հրահանգի դեպքում,
- գրանցման կամ կարդալու հրահանգներ չկան, երբ հերթի պարունակությունը ջնջվում է սկզբնարժեքավորման ազդանշանի հետևանքով,

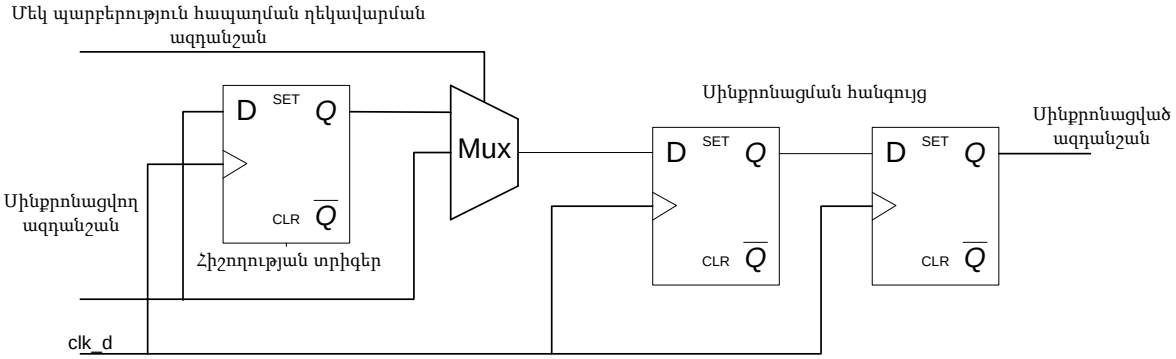
- գրանցման հրահանգ չկա, երբ հերթն ամբողջովին լցված է,
- դուրսբերման հրահանգ չկա, երբ հերթը դատարկ է:

Վերջին երկու պահանջների համար նախատեսված հաստատումները հետևյալն են.

```
// Assertion for POP error,
sequence pop_when_empty,
! (pop && empty && !push),
endsequence : pop_when_empty,
POP_error: assert property (@ (posedge clk) pop_when_empty),
// Assertion for PUSH error,
sequence push_when_full,
!(push && full && !pop),
 endsequence : push_when_full,
 PUSH_error : assert property (@(posedge clk) push_when_full):
```

Ղեկավարման դողերի դեպքում հաճախ նախագծողը, կախված տեխնիկական նկարագրությունից, որոշում է, որ բավարար է միայն մի քանի տրիգերով սինքրոնավորման միջոցի կիրառումը: Այդպիսի որոշման պատճառ կարող է լինել այն հանգամանքը, որ ղեկավարման ազդանշանները միաժամանակ չեն օգտագործվում կամ մեկի ազդեցության արդյունքում մյուսի դիտարկումը անիմաստ է: Ինչպես նախորդ գլխում նշվեց, տվյալների այդպիսի սինքրոնացման արդյունքում մետաստաբիլության հետևանքով ազդանշանի մեկ կամ ավելի պարբերությունների հապաղումը կարող է սխալի պատճառ դառնալ: Սակայն մետաստաբիլությունը պատահական երևույթ է, և հնարավոր է, որ թեստավորման ընթացքում սխալի հայտնաբերման համար անհրաժեշտ քանակությամբ տեղի չունենա: Նախագծողի այս ենթադրությունները թեստավորելու համար առաջարկվում է ներմուծել մի քանի տրիգերներով կառուցվածքի՝ նկ. 2.19-ում ներկայացված կառուցվածքը: Այս կառուցվածքը նախատեսված է մետաստաբիլության արդյունքը ստանալու համար: Ավելացվում է մեկ, այսպես կոչված, հիշողության տրիգեր, և մուլտիպլեքսորի միջոցով ընտրություն է կատարվում ներկա և նախկին արժեքների միջև: Այսինքն՝ հնարավոր է

մտցնել հապաղում նպատակակետի սինքրոազդանշանի մեկ պարբերության չափով: Կարելի է օգտագործել նաև հիշողության մի քանի տրիգեր՝ հապաղման պարբերությունների քանակը մեծացնելու համար:



Նկ.2.19. Նպատակակետի սինքրոազդանշանի՝ մեկ պարբերության չափով հապաղում մտցնող տրամաբանական հանգույցի կառուցվածքը

Թեստավորման համակարգը մուլտիպլեքսորի միջոցով կարող է հապաղումները մտցնել կամայական սինքրոնացուցչի մեջ և շարունակել թեստը: Այս եղանակով հնարավոր է գտնել սխալ նախագծման հետևանքով առաջացած խնդիրները և անհրաժեշտության դեպքում կիրառել Գրեյի կոդավորում ղեկավարման դողի ազդանշանների համար:

Եզրակացություններ

1. Մշակվել է ՀՀԴ ղեկավարման հանգույցի նախագծման նոր երթուղի, որը թույլ է տալիս ներդրված սինթեզվող հաստատումների միջոցով զգալիորեն պարզեցնել ԾՏԻՍ նախատիպում սխալների հայտնաբերման և տեղայնացման գործընթացները:

2. Մշակվել են ՀՀԴ3.0 կապի մակարդակի թեստավորման ՎՀ համակարգի կառուցման միջոցներ: Նախագծված համակարգը հնարավորություն է տալիս կատարելու ՀՀԴ3.0 կապի մակարդակի ամբողջական թեստավորում առանց համակարգի այլ մասերի և, հետևաբար, առանց ծրագրային մասի արագագործության

սահմանափակումների, ինչպես նաև տրամաբանական վերլուծիչներով թեստավորում՝ մինչև ՀՀԴ-ի ամբողջական նախագծի հասանելիությունը:

3. Մշակվել է ֆունկցիոնալ ծածկույթի հաշվարկի եղանակ՝ սինթեզվող հաստատումների կիրառմամբ:

4. Առաջարկվել է ՎՀ համակարգերի մասնակի վերակառուցավորման հատկության կիրառումը՝ ԾՏԻՍ ռեսուրսների քանակի նվազեցման և թեստավորման համակարգի արագագործության բարձրացման նպատակով:

5. Մշակվել են սինթեզվող հաստատումների միջոցով մի քանի տակտային ազդանշաններով թվային նախագծերում սինքրոնացման հանգույցների թեստավորման եղանակներ:

ԳԼՈՒԽ 3. ՎԵՐԱԿԱՌՈՒՑԱՎՈՐՎՈՂ ՀԱՇՎՈՂԱԿԱՆ ՀԱՄԱԿԱՐԳԻ ԿԱՌՈՒՑՄԱՆ ԾՐԱԳՐԱՅԻՆ ՄԻՋՈՑԻ ՆԿԱՐԱԳՐՈՒԹՅՈՒՆԸ ԵՎ ՕԳՏԱԳՈՐԾՈՒՄԸ

3.1. Վերակառուցավորվող հաշվողական համակարգի կառուցման ծրագրային միջոցի կառուցվածքը և աշխատանքի սկզբունքը

ՎՀ համակարգերի կիրառմամբ ՀՀԴ թեստավորման առաջարկվող մեթոդի կիրառումից առաջ անհրաժեշտ է ներդնել ապարատային մասը և անցնել նախատիպի կառուցման բոլոր փուլերով: Մեթոդների կիրառման դեպքում անհրաժեշտ են փոփոխություններ ՄՌՓՄ նկարագրության մեջ: Ճարտարագետի կողմից այս փոփոխությունների կատարման դեպքում սխալվելու հավանականությունը մեծ է, քանի որ փոփոխվում է ՄՌՓՄ նկարագրությունը նախագծի ցածր մակարդակներում, որի արդյունքում լրացուցիչ մուտքային/ելքային ազդանշաններ են ավելանում բարձր մակարդակներում: Ինչպես նախորդ գլխում ներկայացվեց, անհրաժեշտ է մեծ աշխատանք կատարել սինթեզվող հաստատումների ներդրման, ինչպես նաև, մասնակի վերակառուցում օգտագործելու դեպքում, մասնակի ծրագրավորման բիթային նկարագրությունների գեներացման համար: ՀՀԴ ղեկավարման միջուկում օգտագործվում են սինքրոնացման բազմաթիվ հանգույցներ, և նախատեսված բոլոր հաստատումների ներդրումը շատ բարդ գործընթաց է: Այդ պատճառով պետք է հնարավորինս ավտոմատացնել գործընթացը:

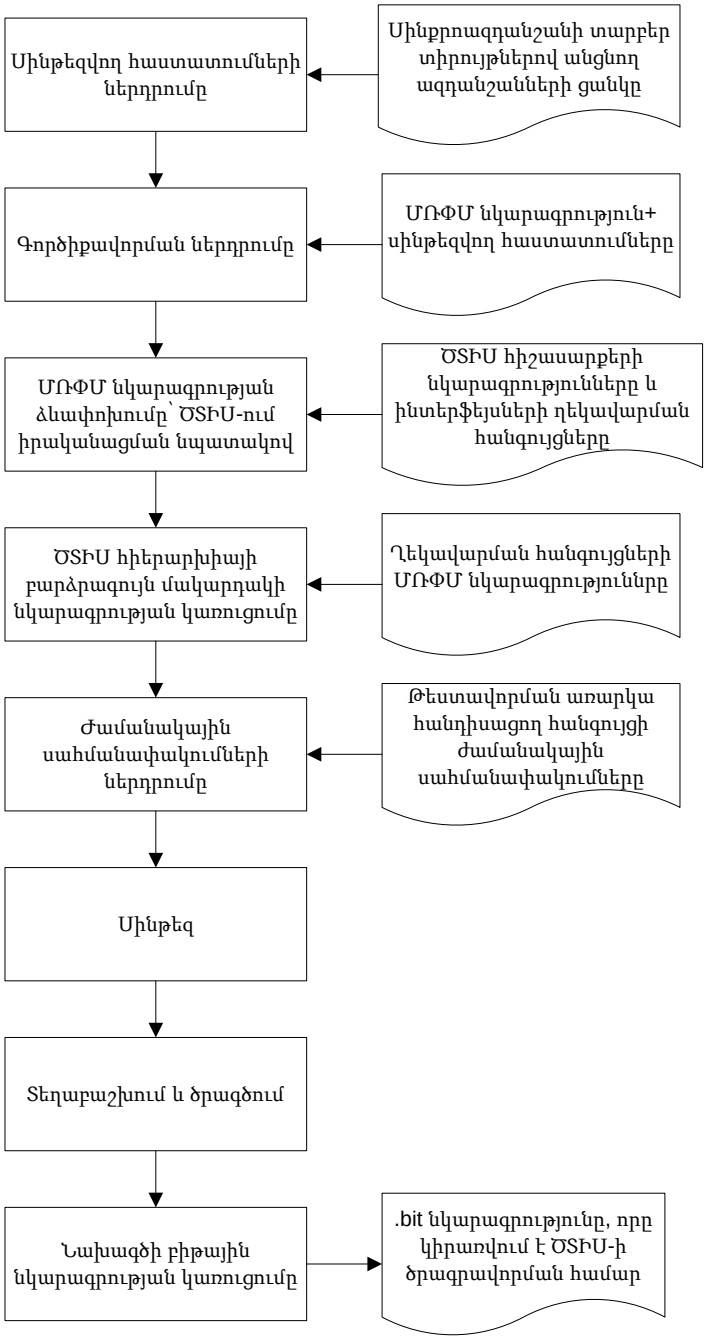
Այս գլխում ներկայացված են ծրագրային և ապարատային միջոցները, որոնք նախագծված են վերը նկարագրված խնդիրների լուծման համար: TAID (Test Assertion Instrumentor and Debugger) ծրագրային միջոցը նախատեսված է սինքրոնացման տրամաբանական բլոկներում սինթեզվող հաստատումների, ՀՀԴ-ն նախագծում ավելացված հաստատումների գործիքավորման և արդյունքների դուրսբերման համար անհրաժեշտ տրամաբանության ներդրման համար: Ծրագրային մասը նախագծված է

Linux Ubuntu [86, 87] օպերացիոն համակարգի համար, Perl [88] և TCL [89] սկրիպտավորման լեզուներով, իսկ ապարատային մասը՝ Verilog [90, 91] սարքերի նկարագրման լեզվով: Գրաֆիկական ինտերֆեյսը կառուցված է QT5.5-ի լեզվով [92]: TAID-ը պարունակում է նաև ՀՀԴ ծրագրային ապահովմանը լրացում՝ նախագծված C [93] ծրագրավորման լեզվով, և օգտագործվում է ավելացված թեստավորման ապարատային մասի ղեկավարման և արդյունքների դուրսբերման համար: TAID ծրագիրը բաղկացած է երկու մասից՝ գործիքավորման ներդրման մասից և արդյունքների մշակման համար նախատեսված ՀՀԴ ծրագրային ապահովման հավելվածից:

TAID Insrumentor-ի օգտագործման երթուղին ներկայացված է նկ.3.1–ում: TAID Instrumentor–ն օգտագործվում է մակետի կառուցման, իսկ TAID Debugger-ը՝ աշխատանքի ընթացքում դիտարկման համար: TAID Instrumentor ծրագրային միջոցի օգտագործման համար անհրաժեշտ է նախագծումը կատարել նախորդ գլխում ներկայացված մեթոդով: Այսինքն՝ անհրաժեշտ է սինթեզվող հաստատումների ներդրումը կատարել ՀՀԴ ՄՌՓՄ նկարագության նախագծման ընթացքում և կատարել մոդելավորում միավոր մակարդակի թեստավորման միջավայրում: ՄՌՓՄ նկարագրության մեջ ներդրվում են հաստատումները պարունակող մոդուլները: Դիտարկման առարկա ազդանշաններն իրականացվում են որպես մուտքային, իսկ արդյունքների դուրսբերումը կատարվում է հատուկ այդ նպատակով իրականացված ինտերֆեյսի միջոցով:

ՄՌՓՄ նկարագրության մեջ հաստատումները ներդրվում են հետևյալ կերպ.

```
`ifdef USB3_LINK_ASSERT_ON,
`include "usb3_link_u3ltssm_sva.v",
`include "usb3_link_txctl_sva.v",
`include "usb3_link_rxctl_sva.v",
….
`endif:
```



Նկ.3.1. TAID Insrumentor ծրագրի երթուղին

Իսկ usb3_link_u3ltssm_sva.v – ն պարունակում է հետևյալը՝

```
usb3_link_u3ltssm_sva U_LTSSM_SVA (
    .reset_n        (mac3_reset_n),
     .link_clk      (mac3_clk),
// inputs
    .ls_u0          (lstate_u0),
    .ls_u1          (lstate_u1),
```

```
        .ls_poll        (lstate_poll),
        .ls_rxdet       (lstate_rxdet),
    .ls_lpbk        (lstate_lpbk),
    .ls_cmply       (lstate_cmply),
          .....
    // Control/Status interface
     .csr_addr        ( csr_addr ),
     .csr_wr_data     ( csr_wr_data ),
     .csr_write       ( csr_write ),
     .csr_req         ( csr_req ),
     .csr_rdy         ( csr_rdy ),
     .csr_rdata       ( csr_rdata )
    ):
```

csr_* ազդանշաններն օգտագործվում են արդյունքների դուրսբերման, ինչպես նաև հաստատումների ակտիվացման համար: Ներդրված մոդուլների անվանումների ընտրությունը կատարվում է հետևյալ կերպ. [անվանումը]_sva, որտեղ փակագծերում նշվածի փոխարեն պետք է տրվի այն մոդուլի անվանումը, որում ներդրվում են թեստավորման հանգույցները: Անվանումների այսպիսի ընտրությունը կատարվում է, որպեսզի TAID ծրագիրը ի վիճակի լինի հայտնաբերելու թեստավորման մոդուլները գործիքավորման ներդրման ընթացքում: Հիմնական թեստավորման հաստատումների ներդրման ավտոմատացումը TAID ծրագրի միջոցով իրականացված չէ, քանի որ ըստ առաջարկված մեթոդի՝ այդ գործընթացը պետք է կատարվի ճարտարագետի կողմից ՄՌՓՄ նկարագրության նախագծման ընթացքում: Մի քանի սինքրոազդանշաններով նախագծերի դեպքում սինքրոնացման տրամաբանությունը դիտարկելու համար նախագծված հաստատումների ներդրումը կատարվում է ավտոմատ կերպով: Այս գործընթացի համար անհրաժեշտ է Spyglass [94] ծրագրի միջոցով գեներացնել սինքրոնացման տրամաբանական հանգույցների ցուցակը, որը հանդիսանում է մուտքային տվյալ՝ գործընթացի կատարման համար: Spyglass ծրագրի գեներացրած սիքրոնացման մոդուլների ցանկի մի մասը ներկայացված է աղ. 3.1–ում:

Աղյուսակ 3.1

Հատված Spyglass ծրագրային գործիքի միջոցով ստացված երկու ասինքրոն սինքրոազդանշանի տիրույթների միջև անցում կատարող ազդանշանների ցանկից

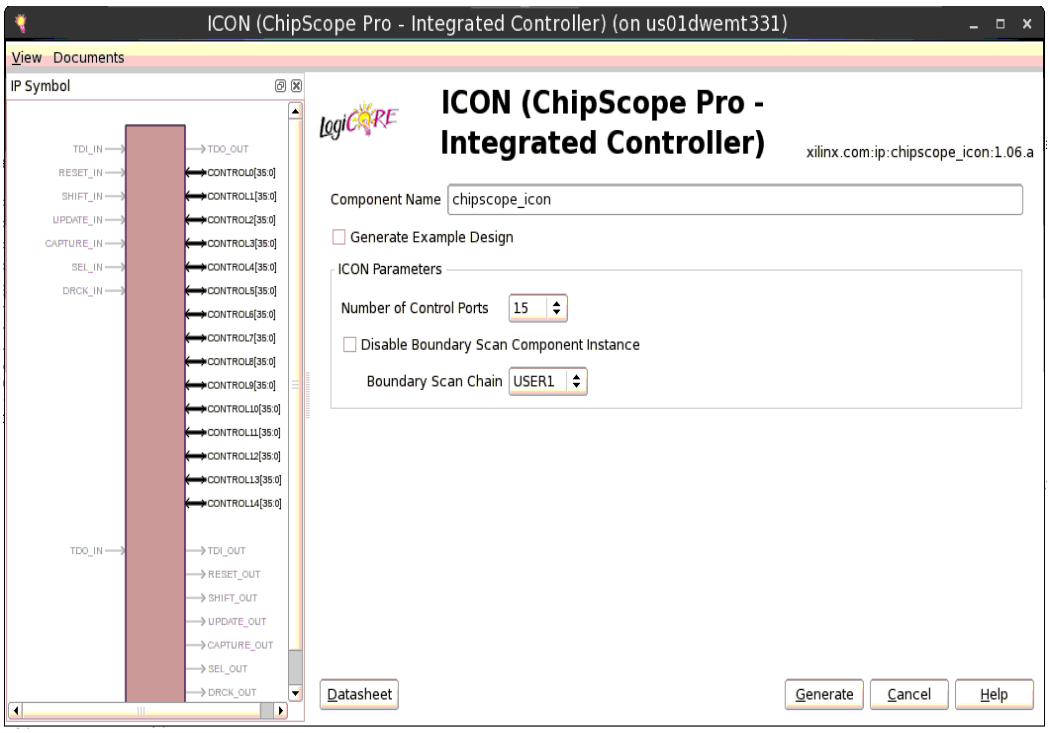
Աղբյուրի տրիգերը	Նպատակակետի տրիգերը	Սինքրոնացման եղանակը
ehci_ohci_top.u_ehci_ahb.u_uhc2_ehci. u_uhc2_lpsync.u_uhc2_bussync_h2p. \data_in_latch_reg[27]	ehci_ohci_top.*. u_uhc2_bussync_h2p. bus_sync_out_reg[27]	Enable Based Synchronizer
ehci_ohci_top.u_ehci_ahb.u_uhc2_ehci. u_uhc2_lpsync.u_uhc2_bussync_h2p. \data_in_latch_reg[26]	ehci_ohci_top.*. u_uhc2_bussync_h2p. bus_sync_out_reg[26]	Enable Based Synchronizer
ehci_ohci_top.ss_word_if_i	ehci_ohci_top.* .u_uhc2_rhprt_0. u_uhc2_rhlpm.vld_hshk_r_reg	does not require synchronization (long-delay/quasi-static)
ehci_ohci_top.ss_word_if_i	ehci_ohci_top.* .u_uhc2_rhprt_0 .u_uhc2_rhlpm. retry_cntr_r_reg[1-0]	does not require synchronization (long-delay/quasi-static)
ehci_ohci_top.u_ohci_0_ahb.*. U_FIFO_CTL.U_POP_FIFOFCTL. \this_addr_g_int_reg[2]	ehci_ohci_top.*. U_PUSH_FIFOFCTL.U_sync. GEN_FST2 .sample_meta_reg[2]	Conventional multi-flop for metastability
ehci_ohci_top.u_ohci_0_ahb.*. U_FIFO_CTL.U_POP_FIFOFCTL. \this_addr_g_int_reg[1]	ehci_ohci_top.*. U_PUSH_FIFOFCTL. U_sync.GEN_FST2. sample_meta_reg[1]	Conventional multi-flop for metastability

Spyglass CDC ծրագրային միջոցը տրամադրում է բոլոր այն ազդանշանների ցուցակը, որոնք անցնում են մի քանի սինքրոազդանշանի տիրույթներով: Աղյուսակում ներառված են աղբյուրի տրիգերի ամբողջական անվանումը, նպատակակետի տրիգերի անվանումը և սինքրոնացման մեթոդը: Քվազի-ստատիկ ազդանշանների ցանկը, որը հանդես է գալիս որպես մուտքային տվյալ Spyglass CDC ծրագրին, նույնպես արտացոլվում է աղյուսակում: Այսինքն՝ Spyglass CDC կառուցվածքային

վերլուծության միջոցով TAID ծրագրին տալիս է սինթեզվող հաստատումների ներդրման համար անհրաժեշտ բոլոր տվյալները:

Գործիքավորման ներդրման փուլում կատարվում է բոլոր անհրաժեշտ ապարատային միջոցների ներդրումը, որոնք ապահովում են TAID Debugger ծրագրի և ԾՏԻՍ նախատիպի միջև ինտերֆեյսը: Ներդրված ապարատային գործիքներն ունեն ղեկավարման և դիտարկման առանձին ռեգիստրներ, որոնք նախագծված են թեստավորման համակարգի կառավարման նպատակով: Կախված ԾՏԻՍ նախատիպի կառուցվածքից՝ ինտերֆեյսը ծրագրի և ապարատային մասերի միջև կարող է տարբեր լինել: Օրինակ՝ համակարգչի օգտագործման դեպքում կարելի է օգտագործել AHB[95] կամ APB [96] ինտերֆեյսներ և PCIE [51] ղեկավարման միջուկը կառուցել համապատասխան կերպով: PCIE–ի ղեկավարման միջուկը ծրագրավորվում է այնպես, որ ունենա մեկ լրացուցիչ ֆունկցիա, որը նախատեսված է թեստավորման ինտերֆեյսի ապահովման համար: Այսինքն՝ եթե ՀՀԴ միջուկի համար անհրաժեշտ է *N* ֆունկցիա, ապա գործիքավորումից հետո PCIE-ն ունենում է *N+1* ֆունկցիա: Թեստավորման համակարգի ռեգիստրների հասցեն հայտնի է դառնում համակարգչին միացնելուց և օպերացիոն համակարգում PCIE ղեկավարման միջուկի գրանցվելուց հետո, որը հետագայում օգտագործվում է թեստավորման համակարգի ղեկավարման համար TAID Debugger ծրագրի միջոցով:

Համակարգչի բացակայության դեպքում թեստավորման համակարգի ղեկավարման և արդյունքների դուրսբերման նպատակով օգտագործվում է JTAG ինտերֆեյսը Xilinx–ի ChipScope ծրագրի միջոցով: Այդ նպատակով ԾՏԻՍ-ի նախատիպին ավելացվում են ICON և VIO միջուկները: Այդ միջուկները գեներացվում են Xilinx–ի CoreGenerator միջոցով՝ TAID–ում համապատասխան պարամետրերի ընտրության դեպքում: ICON ղեկավարման միջուկի ծրագրավորման պատուհանը ներկայացված է նկ. 3.2–ում: CoreGenerator ծրագրի միջոցով գեներացվում է ICON-ի և VIO-ի տարրերի մակարդակի նկարագրությունը ընտրված սարքի համար, որը հետագայում օգտագործվում է տեղաբաշխման և ծրագծման ընթացքում այլ թեստավորման մոդուլների և ՀՀԴ ղեկավարման միջուկի հետ միասին:



Նկ.3.2. ICON դիտարկման և ղեկավարման ինտերֆեյսի միջուկի ծրագրավորման CoreGenerator-ի պատուհանը

Գեներացվում են նաև միջուկի ժամանակային սահմանափակումները, որոնք օգտագործվում են նախատիպի կառուցման հետագա փուլերում: Բացի դրանից, գեներացվում են միջուկների մոդելները՝ ԾՏԻՍ-ի նախատիպի նախագծի բարձրագույն մակարդակի նկարագրության ստուգման նպատակով: Համապատասխան պարամետրերի ընտրությունից հետո գեներացված միջուկները ներդրվում են հետևյալ եղանակով՝

```
chipscope_icon U_ltssm_icon
(
 // Inouts
 .CONTROL0       (CONTROL0[35:0]),
 .CONTROL1       (CONTROL1[35:0]),
 ...
);

chipscope_vio U_ltssm_ctrl_vio
(
```

```
 // Outputs
 .SYNC_OUT     (lnmcc_SYNC_OUT),
 // Inouts
.CONTROL       (CONTROL0[35:0]),
....
 // Inputs
.CLK           (link_clk) );
```

նախագծի ՄՌՓՄ նկարագրության բարձրագույն մակարդակում:

ՄՌՓՄ նկարագրությունն անհրաժեշտ է ձևափոխել՝ ԾՏԻՍ–ում իրականացնելու պահանջներին համապատասխան: Օգտագործվում է Synopsys–ի ProtoCompiler [97] ծրագրային գործիքը: Կատարվում են ՄՌՓՄ նկարագրության հետևյալ ձևափոխությունները.

- ասինքրոն հապաղումները կամ մակարդակով ակտիվացվող տրիգերները փոխարինվում են կամ ձևափոխվում,
- հիշողության բլոկները փոխարինվում են համապատասխան ԾՏԻՍ բլոկներով, և չափերը փոքրացվում են, քանի որ ԻՍ–երում օգտագործվող հիշասարքերը մեծ են և առանց փոփոխության ներդրման դեպքում սպառում են մեծ քանակությամբ ԾՏԻՍ ռեսուրսներ,
- սինքրոազդանշանի ուղու վրա տեղադրված տարրերը հեռացվում են,
- սինքրոազդանշանի ձևափոխությունները կատարվում են ԾՏԻՍ-ի համապատասխան հանգույցների օգտագործմամբ,
- ԻՍ-ի մուտքային և ելքային տարրերը հեռացվում են,
- անալոգային մասերի համար ներդրվում են հանգույցներ՝ արտաքին սարքերի ղեկավարման համար:

Բացի վերը նկարագրված փոփոխություններից, նախագծի ՄՌՓՄ նկարագրությունը պետք է կառուցված լինի այնպես, որ համապատասխան պարամետրի միջոցով կատարվի ընտրություն ԾՏԻՍ-ի իրականացման և արտադրության համար նախատեսված նկարագրությունների միջև, քանի որ հնարավոր չէ ծրագրային գործիքների միջոցով ՄՌՓՄ նկարագրությունը ամբողջովին համապատասխանեցնել ԾՏԻՍ-ի իրականացմանը:

ԾՏԻՍ նախատիպի հիերարխիայի բարձրագույն մակարդակի նկարագրության կառուցման փուլում գեներացվում են թեստավորման բոլոր մոդուլների ՄՌՓՄ նկարագրությունները, և համակարգչի օգտագործման դեպքում կառուցվում են ղեկավարման բոլոր անհրաժեշտ միջուկները: Թեստավորման մոդուլների կառուցումը կատարվում է TAID ծրագրում առկա Perl սկրիպտավորման լեզվով նախագծված ծրագրի միջոցով, իսկ ղեկավարման միջուկների կառուցման նպատակով օգտագործվում է ՄՍ հանգույցների ծրագրավորման գործիք:

Ժամանակային սահմանափակումների ավելացման փուլում պետք է որպես մուտքային տվյալներ տրվեն ՀՀԴ ղեկավարման միջուկի և օգտագործվող այլ միջուկների ժամանակային սահմափակումները, որոնք այս փուլում համապատասխանեցվում են ԾՏԻՍ նախատիպին: Բացի դրանից, ավելացվում են թեստավորման համակարգի ժամանակային սահմանափակումները, որոնք տրվում են հետևյալ կերպ.

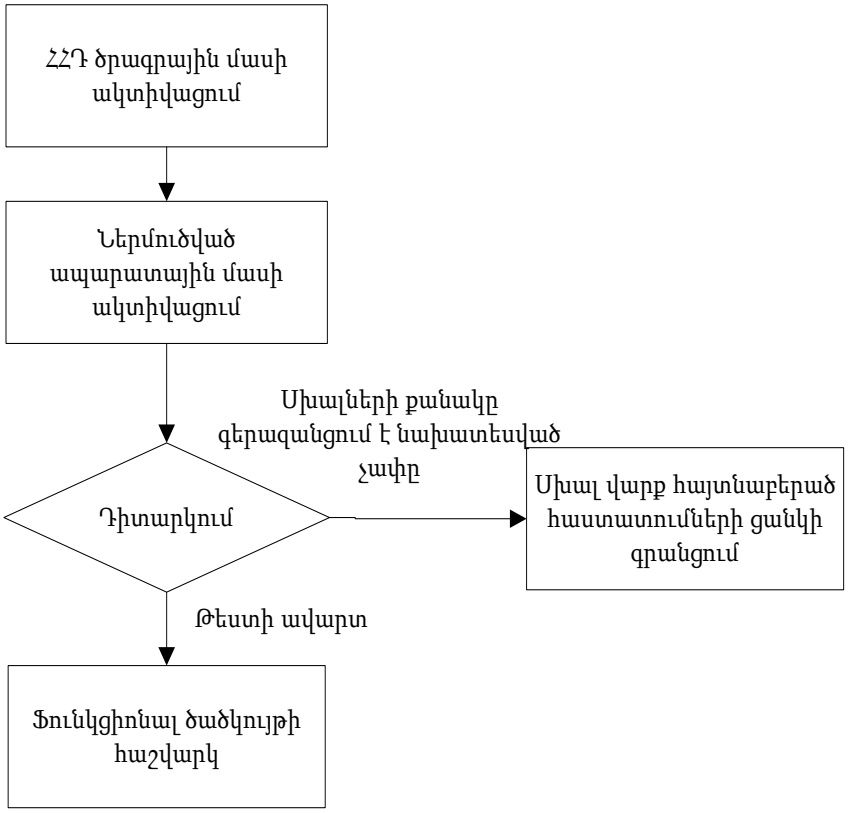
```
define_path_delay  -from {rclk_fx_i} -to {rmmi_tx_symbol_clk}        -max  10
define_path_delay  -from {rclk_fx_i} -to {rmmi_rx_symbol_clk}        -max  10
define_path_delay  -from {rclk_fx_i} -to {fpga_pcie_clk_125}         -max  8
define_path_delay  -from {rclk_fx_i} -to {utmi_clk}                  -max  10
```

Այս փուլում, որպես մուտքային տվյալ տրվում է նաև ԾՏԻՍ –ի և՛ մուտքերին, և՛ ելքերին միացվող նախատիպի ազդանշանների նկարագրությունը: Հետևյալ նկարարագրությունը կիրառվում է HAPS51T նախատիպի կառուցման հարթակի համար.

```
define_attribute {p:refclka_p} syn_loc {D16}
define_attribute {p:refclka_n} syn_loc {C16}
define_io_standard {p:utmi_rxvalid} syn_pad_type {LVCMOS25} syn_io_slew {FAST}
define_io_standard {p:utmi_rxerror} syn_pad_type {LVCMOS25} syn_io_slew {FAST}
```

Սինթեզի կատարման համար TAID-ը օգտագործում է Synopsys–ի ProtoCompiler ծրագիրը: Տեղաբաշխումը և ծրագծումը, ինչպես նաև ԾՏԻՍ–ի ծրագրավորման ֆայլի գեներացումը կատարվում են Xilnx-ի տրամադրած ծրագրերի միջոցով:

ԾՏԻՍ մակետի միջոցով թեստավորման ընթացքում դիտարկումը կատարվում է C լեզվով նկարագրված ՀՀԴ ծրագրային ապահովման հավելվածի միջոցով: Ծրագրային հավելվածն ակտիվացվում է <<insmod TAID_debugger.ko>> հրամանով: Հավելվածի ակտիվացման հետ միասին կատարվում է նաև ներդրված ապարատային մասի ակտիվացումը, և սկսվում է հաստատումների դիտարկումը: Համակարգիչ պարունակող համակարգի դեպքում համակարգիչը պետք է օգտագործի Linux օպերացիոն համակարգ, քանի որ հավելվածը նախագծված է միայն այդ միջավայրում կիրառելու նպատակով: Թեստավորման ռեգիստրների ցանկը ներկայացված է աղ. 3.2-ում: Դիտարկման ընթացքում սխալների որոշակի քանակի դեպքում ապարատային մասը համապատասխան ռեգիստրի միջոցով տեղեկացնում է ծրագրային մասին դիտարկման ավարտի և տվյալների դուրսբերման անհրաժեշտության մասին (նկ. 3.3): Սխալների առավելագույն քանակը, որի դեպքում դիտարկումը պետք է ավարտվի, ծրագրավորվում է ապարատային մասի համապատասխան ռեգիստրի մեջ:



Նկ.3.3. TAID Debugger հավելվածի երթուղին

Աղյուսակ 3.2

Թեստավորման ապարատային մասի ղեկավարման ռեգիստրները

Ռեգիստր	Սկզբնակետը (+գրանցման սկզբնակետ)	Նկարագրությունը
CMD	0x0	Օգտագործվում է թեստավորման ապարատային մասի ղեկավարման նպատակով:
ERRCNT	0x4	Առավելագույն սխալների քանակը որի դեպքում պետք է ավարտել դիտարկումը:
INTMASK	0x8	Օգտագործվում է ընդհատման կանչի ազդանշանի ղեկավարման նպատակով:
GENINT	0xC	Ընդհատման կանչի դեպքում ծրագրային մասը պատճառը գտնելու նպատակով կարդում է այս ռեգիստրը:
ASTAUS[i]	0xF + i*4	Սինթեզվող հաստատումների նույնականացման համարը և դիտարկման արդյունքները:

Դիտարկումը կարող է ավարտվել նաև թեստավորման ավարտի դեպքում՝ հավելվածի միջոցով: Թեստավորումը հաջողությամբ ավարտելու դեպքում ֆունկցիոնալ ծածկույթի տվյալները դուրս են բերվում ապարատային մասից համապատասխան ռեգիստրների միջոցով: ASTATUS ռեգիստրը 32 բիթ է և պարունակում է 16-ական բիթ տվյալներ երկու հաստատման դիտարկման արդյունքների մասին: Այդ ռեգիստրի մեկ հաստատման համար նախատեսված կառուցվածքն ունի հետևյալ տեսքը՝

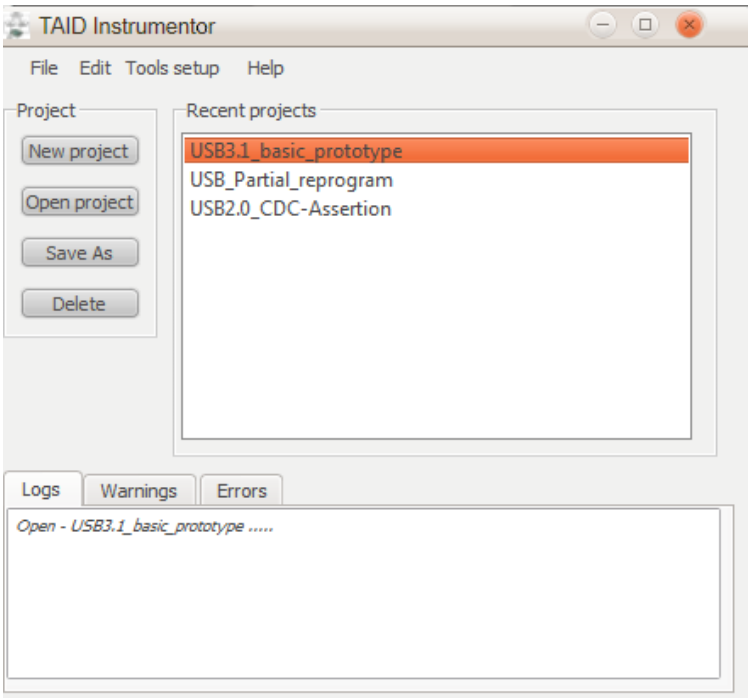
```
struct aststus16 {

uword uid; // unique ID for each assertion

uByte active; // 1 – active, 0 - diabled

uByte status; // 0 – no error, 1 -failed

} // end aststus16
```

Այն բաղկացած է նույնականացման համարից, հաստատման վիճակից և դիտարկման արդյունքը բնութագրող փոփոխականներից: Նույնականացման համարը արդյունքը բնութագրող փոփոխականի հետ միասին օգտագործվում է ֆունկցիոնալ ծածկույթի հաշվարկի, ինչպես նաև սխալների տեղայնացման նպատակով:

Հաստատման վիճակը բնութագրող փոփոխականով ակտիվացվում է դիտարկումը, որի միջոցով հայտնի սխալների դեպքում կարելի է ընդհատել դիտարկումը կեղծ արդյունքներից խուսափելու նպատակով:

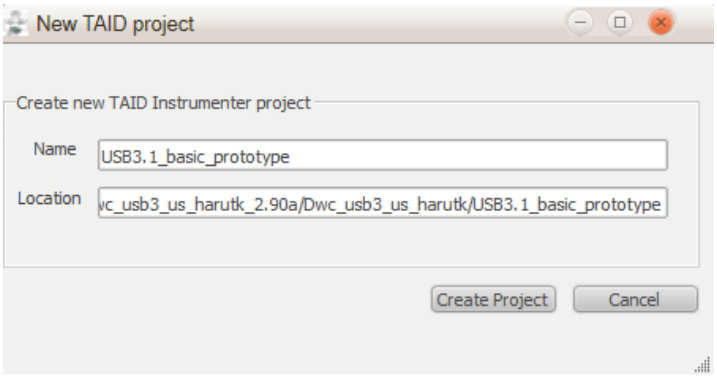
3.2. Վերակառուցավորվող հաշվողական համակարգի կառուցման ծրագրային միջոցի գրաֆիկական ինտերֆեյսը

TAID ծրագրի հիմնական պատուհանը ներկայացված է նկ.3.4-ում: Այն հնարավորություն է տալիս ստեղծելու նոր նախագիծ **«New Project»** դաշտի միջոցով, ինչպես նաև ընտրելու նկ.3.1-ում տրված հոսքուղու որոշակի փուլում գտնվող նախագծերը **«Recent projects»** կամ **«Open project»** դաշտերից որևէ մեկի միջոցով:



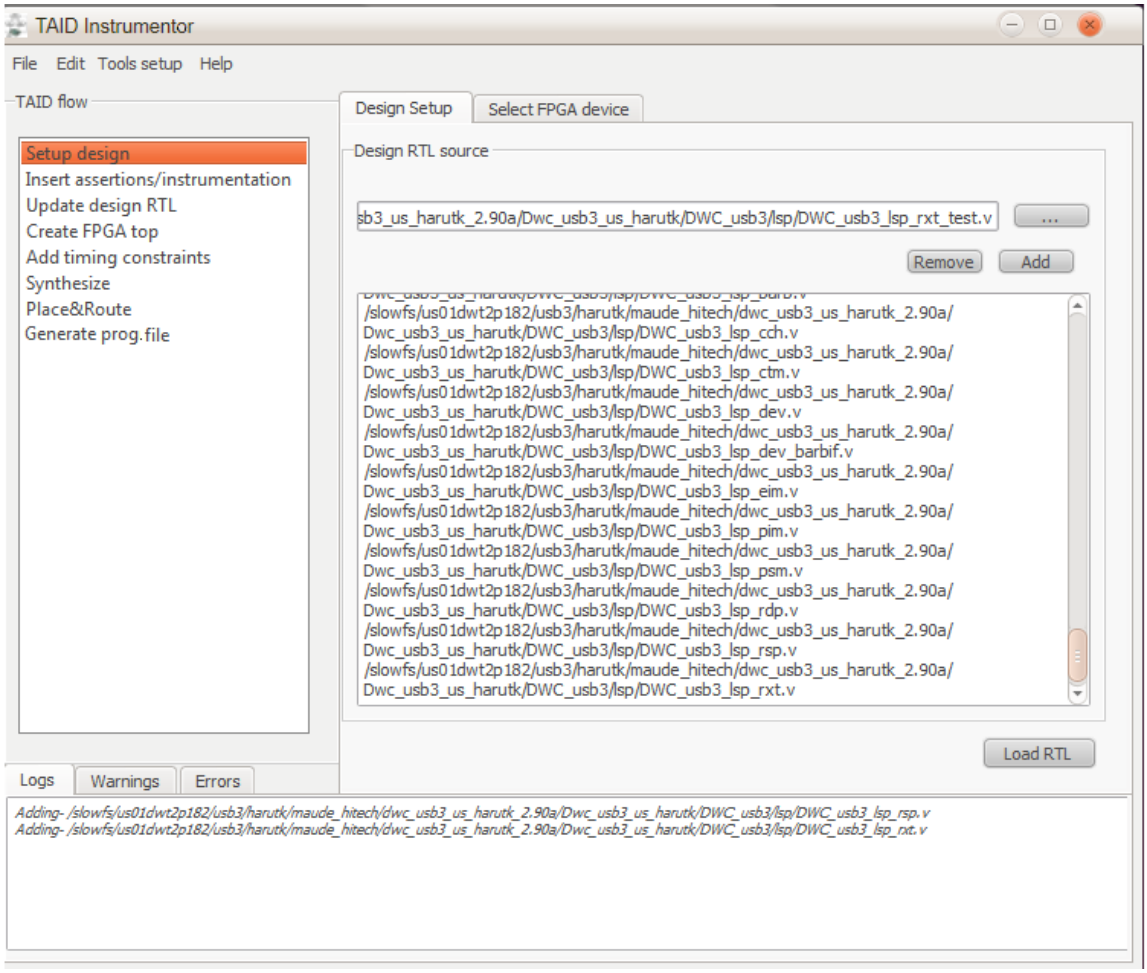
Նկ.3.4. TAID ծրագրի հիմնական պատուհանը

Նոր նախագիծ ընտրելու դեպքում **«New TAID project»** պատուհանում տրվում են նոր նախագծի անվանումը և կառուցման վայրը (նկ.3.5):



Նկ.3.5. Նոր նախագծերի կառուցման պատուհանը

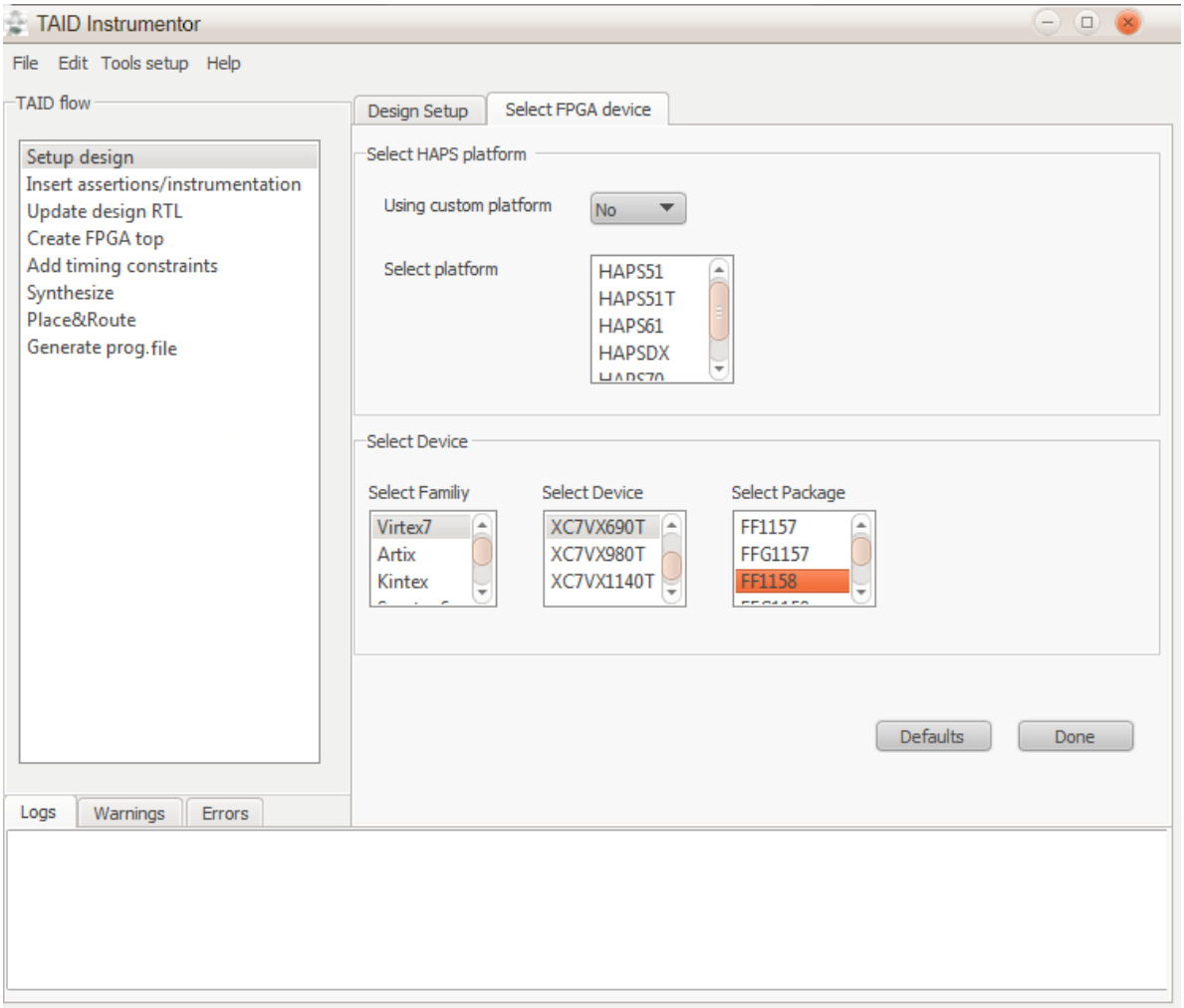
Նոր նախագիծ կառուցելուց հետո բացվում է հիմնական աշխատանքային պատուհանը: **«TAID flow»** դաշտում տրված են նկ.3.1 հոսքուղու քայլերը միևնույն հերթականությամբ և **«Setup Design»**–ը, որի միջոցով տրվում են նախագծի ՄՌՓՄ նկարագրությունը, և կատարվում է ԾՏԻՍ սարքի ընտրությունը (նկ. 3.6):



Նկ.3.6. ՄՌՓՄ նկարագրության ներմուծման պատուհանը

«Load RTL» դաշտի ընտրության դեպքում կատարվում է ՄՌՓՄ նկարագրության ստուգում VCS [98] ծրագրի միջոցով օգտագործված սխալ կառուցվածքների

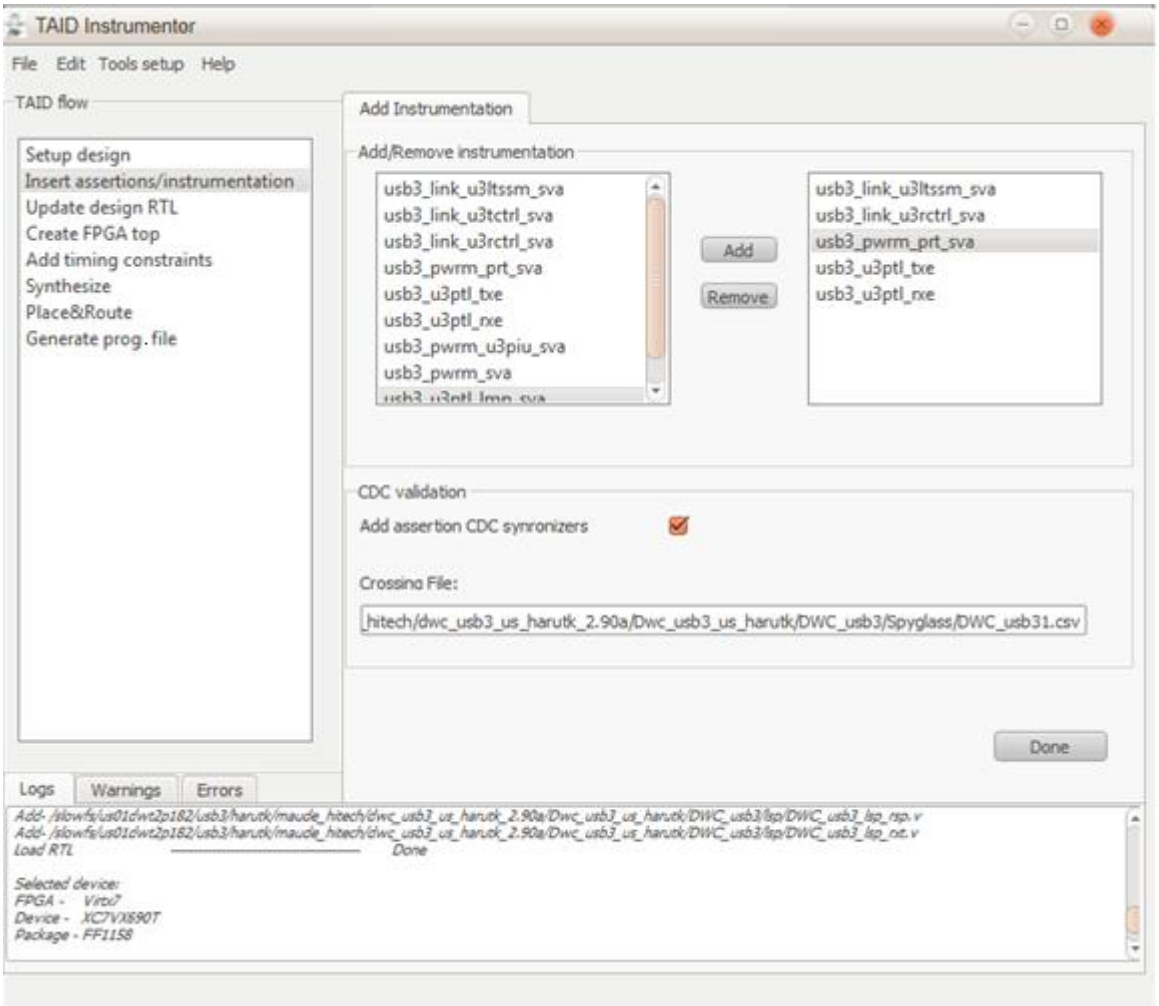
հայտնաբերման նպատակով: **«Select FPGA device»** պատուհանով կատարվում են ԾՏԻՍ նախատիպի կառուցման հարթակի ընտրությունը Synopsys-ի HAPS հարթակի օգտագործման դեպքում և ԾՏԻՍ սարքի ընտրությունը այլ տպասալերի օգտագործման դեպքում: Երբ **«Using custom platform»**–ի դաշտի արժեքը **«Yes»** է, կատարվում է ԾՏԻՍ-ի ընտրությունը, հակառակ դեպքում ընտրվում է HAPS հարթակներից որևէ մեկը (նկ. 3.7):



Նկ.3.7. ԾՏԻՍ-ի և նախատիպի կառուցման հարթակի ընտրության պատուհանը

Հաջորդ քայլում կատարվում է նկարագրության մշակումը, որի ընթացքում հայտնաբերվում են նախագծողի կողմից ներդրված սինթեզվող հաստատումների մոդուլները, որոնց անվանումները բավարարում են TAID պահանջները (նկ. 3.8): **«Add/Remove instrumentation»** դաշտի ձախ մասում տրվում են բոլոր հայտնաբերված հաստատումների հանգույցները, իսկ աջ մասում՝ բոլոր այն հանգույցները, որոնց համար պետք է ներդրվի գործիքավորում՝ թեստավորման

ընթացքում օգտագործման համար: **«CDC validation»** դաշտում **«Add assertion CDC syncronizers»**-ի միջոցով ակտիվացվում է սինքրոնացման համակարգերի համար նախատեսված հաստատումների և արդյունքների դուրսբերման համար նախատեսված գործիքավորման ներդրումը: **«Crossing file»** դաշտում տրվում է Spyglass CDC ծրագրի միջոցով գեներացված բոլոր անցումների և սինքրոնացման եղանակների նկարագրությունը, որը հետագայում օգտագործվում է հաստատումների ներդրման նպատակով:



Նկ.3.8. TAID ծրագրի հաստատումների ներդրման պատուհանը

«TAID flow» դաշտում տրված քայլերը կատարվում են հերթականությամբ, և արդյունքում ստացվում է նախագծի և թեստավորման համակարգի բիթային նկարագրությունը, որն օգտագործվում է ԾՏԻՍ-ի ծրագրավորման փուլում: TAID debugger ծրագրային միջոցի համար գրաֆիկական ինտերֆեյս կառուցված չէ: Այն օգտագործվում է ՀՀԴ ծրագրային ապահովման նմանությամբ:

3.3. Վերակառուցավորվող հաշվողական համակարգի կառուցման ծրագրային միջոցի արդյունավետության գնահատումը

Այս ենթագլխում ներկայացված է համակարգի արդյունավետության գնահատումը երեք տեսանկյունից՝ սպառվող ԾՏԻՍ ռեսուրսներն ու արագագործության սահմանափակումները, ֆունկցիոնալ ծածկույթի հաշվարկի հնարավորությունը և սխալի հայտնաբերման գործընթացի արդյունավետությունը:

Ինչպես բազմիցս նշվել է, ԾՏԻՍ-ի միջոցով կառուցված թեստավորման համակարգերի կառուցման հիմնական խնդիրը օգտագործված ռեսուրսն է: Թեստավորման համակարգերը, որոնք կառուցելու համար կիրառվում են մեծ քանակությամբ ԾՏԻՍ ռեսուրսներ, չի կարելի արդյունավետ կերպով կիրառել, քանի որ ամբողջական նախատիպը չի կարող բավարարել ժամանակային սահմանափակումները: Այդ պատճառով անհրաժեշտ է առաջին հերթին դիտարկել սպառվող ռեսուրսները: Բացի սպառվող ռեսուրսներից, անհրաժեշտ է դիտարկել արագագործության սահմանափակումները, որոնք առաջանում են թեստավորման հանգույցների ներդրումից: Այն ազդանշանների համար, որոնց դիտարկման նպատակով տեղադրված են մեծ քանակությամբ հաստատումներ, ժամանակային սահմանափակումներին համապատասխանությունը կարող է հնարավոր չլինել: Անհրաժեշտ է դիտարկել ՀՀԴ3.0–ում օգտագործվող բոլոր հաստատումների ցանկը (աղ. 3.3): Կա հաստատումների երկու հիմնական տեսակ՝ այլ ՄՍ հանգույցների հետ կապի ինտերֆեյսների և ՀՀԴ3.0-ի ներքին ազդանշանների ստուգման: ՀՀԴ3.0-ի միջուկն օգտագործում է PIPE [76] ինտերֆեյսը ֆիզիկական մակարդակի հետ կապի հաստատման, AHB Slave-ը [95] ծրագրային մասից հրամանների ստացման, իսկ Master-ը համակարգի հիշասարքի հետ կապի համար (PCIE-ի ղեկավարման հանգույցի միջոցով կամ անմիջապես միացված է պրոցեսորի տվյալների դողին): Ներքին ազդանշանների ստուգման հաստատումները բաժանվում են խմբերի՝ ըստ փոխանցման տեսակի, համակարգի վիճակի, ՀՀԴ3.0-ի մակարդակների մասերի և այլն: Սինթեզը, տեղաբաշխումը և ծրագծումը կատարվել են վատագույն դեպքի համար, երբ բոլոր սինթեզվող հաստատումները ներդրված են ֆունկցիոնալ ծածկույթի

հաշվարկի նպատակով: Սինքրոազդանշանի՝ տարբեր տիրույթների միջև ազդանշանի անցումը սինքրոնացնող տրամաբանության համար նախատեսված պնդումները հաշվի չեն առնվել:

Աղյուսակ 3.3

ՀՀԴ ղեկավարման միջուկի և այլ ՄՍ հանգույցների ստուգման համար նախատեսված հաստատումները

Հաստատումների խմբերը	**Հաստատումների ենթախմբերը**	**Բոլորը**	**Ծածկված**
PIPE ինտերֆեյս	Ազդանշանների արժեքները P0 վիճակում	70	64
	Ազդանշանների արժեքները P1 վիճակում	16	16
	Ազդանշանների արժեքները P2 վիճակում	22	18
	Ազդանշանների արժեքները P3 վիճակում	34	8
	Սինքրոազդանշանի ազդանշանի հաճախությունը	3	1
	Ազդանշանների արժեքները սկզբնարժեքավորման ազդանշանի դեպքում	37	33
	Ցածր էներգասպառմամբ վիճակների միջև անցումները	8	5
	Տվյալների ընդունիչի հայտնաբերումը P2, P3 վիճակում	4	4
	ՑՀՊԱ-ի հայտնաբերումը և առաքումը	16	13
AHB առաջնային	Կարդալ	46	25
	Գրանցել	57	30
	Փոխանցման սխալը	5	0
	Փոխանցումը կրկնելու հրամանը	15-կարդալ 19-գրանցել	5-կարդալ 5-գրանցել
	Split տեսակի փոխանցումները	15-կարդալ 19-գրանցել	5-կարդալ 5-գրանցել
	Փոխանցման հասցեի կառուցվածքը	31	29
AHB երկրորդային	Կարդալ	18	18

	Գրանցել	18	18
Փոխանցումների տեսակները	Conrtol	8-ղեկ. 80-տ. մուտք 80-տ. ելք	8-ղեկ. 76-տ. մուտք 68-տ. ելք
	Bulk	286-մուտք 390-ելք	286-մուտք 390-ելք
	ISOC	12-ելք 20-մուտք	12-ելք 20-մուտք
	Interrupt	173-մուտք 279-ելք	146-մուտք 240-ելք
Կապի մակարդակը	Կապի սկզբնարժեքավորումը և կարգաբերումը	28	28
	Կապի ղեկավարումը	88	30
	Սխալների մշակումը	123	50
	Հիմնական ՎՎՄ	41	41
ՀՀԴ2.0	LPM էներգիայի խնայողության նպատակով ներմուծված հավելվածը	89	63
	UTMI ինտերֆեյսը	24	24
ՀՀԴ2.0 փոխանցումների տեսակները	Conrtol	63-ղեկ. 180-տ. մուտք 135-տ. ելք	63-ղեկ. 159-տ. մուտք 132-տ. ելք
	Bulk	98-մուտք 101-ելք	94-մուտք 98-ելք
	ISOC	61-մուտք 48-ելք	59-մուտք 44-ելք
	Interrupt	225-մուտք 150-ելք	209-մուտք 139-ելք

Առաջարկվող մեթոդի կիրառման դեպքում սպառվող հավելյալ ԾՏԻՍ ռեսուրսների չափը ՀՀԴ3.0 առաջնային սարքի նախատիպում ներկայացված է աղ. 3.4-ում: Հաստատումների ներդրման դեպքում օգտագործվող ԻԱ-ների քանակն աճել է 14%-ով, իսկ ռեգիստրների քանակը՝ 16%-ով: Օգտագործվել են լրացուցիչ 6 RAM36 [99] հանգույց և 2 լրացուցիչ BUFGCTRL [100]՝ սինքրոազդանշանի ծառի կառուցման համար: Չնայած այն հանգամանքին, որ մեծ քանակությամբ լրացուցիչ ԾՏԻՍ

ռեսուրսներ են օգտագործվել, ինչպես երևում է աղյուսակից, ամբողջական թեստավորման համակարգը և ՀՀԴ3.0 նախատիպը կառուցելու նպատակով օգտագործված է XC7VX690T [101] ԾՏԻՍ-ի ռեսուրսների 50%-ից քիչ մասը: Ռեսուրսների այդպիսի օգտագործումը չի կարող խնդիրներ առաջացնել [39]:

Աղյուսակ 3.4

Առաջարկվող մեթոդի դեպքում սպառվող հավելյալ ռեսուրսները

ԾՏԻՍ ռեսուրսի տեսակը	**ՀՀԴ3.0 նախատիպը առանց գործիքավորման**	**Առաջարկվող մեթոդով կառուցված ՀՀԴ3.0 նախատիպը**	**Օգտագործված ԾՏԻՍ ռեսուրսների չափը (%)**
Slice ԻԱ	145174	165498	38,2
Slice ռեգիստր	65628	76334	8,8
մուլտիպլեքսոր	2331	2643	1,2
RAMB36	97	103	7,1
RAMB18E1*	6	6	0,2
BUFGCTRL	19	21	59,38
MMCME2_ADV**	4	4	20
BUFR**	2	2	2,5

*օգտագործվում է սինքրոազդանշանի ծառի կառուցման նպատակով [100]
**ներդրված հիշասարքի հանգույց [99]

Ինչպես վերևում նշվեց, հաստատումների ներդրման արդյունքում հնարավոր է արագագործության նվազում: Նվազման հիմնական պատճառը մեծ քանակությամբ հաստատումների օգտագործումն է մեկ ազդանշանի (տրիգերի ելքի) դիտարկման նպատակով, որի արդյունքում այդ ազդանշանի որոշ համակցական ուղիներով անցման հապաղումը չի բավարարում սահմանված պայմանները: Այդպիսի խնդիրներ կարող են առաջանալ, երբ մեկ ազդանշանը ՀՀԴ ղեկավարման հանգույցի տարբեր վիճակներում ունի տարբեր գործառույթներ, որոնց ստուգման նպատակով ներդրվում են բազմաթիվ հաստատումներ: ՀՀԴ նախատիպի սինքրոազդանշանների սինթեզի արդյունքում ստացված հաճախությունները, կառուցված առաջարկվող մեթոդով և առանց գործիքավորման, ներկայացված են աղ. 3.5-ում: Ինչպես երևում է աղյուսակից, առաջարկվող մեթոդի դեպքում bus_clk և pipe_pclk սինքրոազդանշանների սինթեզի արդյունքում ստացված հաճախությունները փոքր են պահանջվածից: Ստացված արդյունքները թույլատրելի են ԾՏԻՍ-ի օգտագործմամբ կառուցված նախատիպի դեպքում, և փորձարկումների ընթացքում խնդիրներ չեն առաջացել: Թեստավորման համակարգի ներդրման հետևանքով առաջացած արագագործության

սահմանափակումները կարելի է նվազեցնել մասնակի վերակառուցավորման միջոցով [77]: Ինչպես նախորդ գլխում ներկայացվեց, անհրաժեշտ է նախագիծը բաժանել մասերի՝ որպես չափանիշ ընդունելով ՀՀԴ որոշակի վիճակում կիրառումը կամ՝ ըստ թեստի: Մասնակի վերակառուցավորման օգտագործումը ավելի նպատակահարմար է ԿԲՊՀ-ների նախատիպերի դեպքում, երբ ռեսուրսները և ժամանակային սահմանափակումները ավելի խիստ են:

Աղյուսակ 3.5

ՀՀԴ3.0 նախատիպի ժամանակային պահանջները և ստացված արդյունքները

Սինքրոազդանշան	Պահանջվող հաճախություն (ՄՀց)	Ստացված հաճախությունը՝ առանց գործիքավորման (ՄՀց)	Ստացված հաճախությունը առաջարկված մեթոդով (ՄՀց)
bus_clk	125	126,8	117,9
pcie_pipe_clk	250	347,5	347,5
pipe3_pclk_in	125.0	169,5	160,3
pipe_pclk	125	144,3	123
ram_clk*	85	47,3	47,3
rclk	100	234	216
utmi_clk	60	61,4	60,9
suspend_clk_32k	1	814	814

*այս սինքրոազդանշանի հաճախությունը կարող է փոփոխվել տեղաբաշխումից և ծրագծումից հետո, և ժամանակային սահմանափակումները կիրառված են հաճախության առավելագույն արժեքի համար

Կառուցված նախատիպի թեստավորումը կատարվել է [102-105]-ում ներկայացված թեստավորման եղանակներով: Ֆունկցիոնալ ծածկույթը ներկայացված է աղ. 3.3-ում: Ինչպես երևում է աղյուսակից, PIPE ինտերֆեյսի հաստատումների մեծ մասը ծածկված է: Վատագույն արդյունքը ստացվել է P3 վիճակի համար նախատեսված հաստատումների դեպքում, և անհրաժեշտ են լրացուցիչ թեստեր այդ վիճակում համակարգի ուսումնասիրման նպատակով: AHB առաջնային ինտերֆեյսի ցածր ֆունկցիոնալ ծածկույթը պայմանավորված է ՀՀԴ3.0-ի կառուցվածքով: AHB առաջնային ինտերֆեյսը օգտագործվում է համակարգի հիշասարքի և ՀՀԴ-ի միջև կապի համար՝ շրջանցելով պրոցեսորը, որում ոչ բոլոր փոխանցման եղանակներն են անհրաժեշտ: Կապի մակարդակի սխալների մշակման և կապի ղեկավարման

գործառույթների ցածր ծածկույթը նույնպես պայմանավորված է առաջնային սարքի կառուցվածքով: AHB երկրորդային սարքերի բոլոր հաստատումները ծածկված են, քանի որ դրանք նախատեսված են միայն պարզագույն փոխանցման եղանակի համար, որն օգտագործվում է ՀՀԴ3.0 ղեկավարման ռեգիստրների արժեքները կարդալու և նոր արժեքներ գրանցելու համար: Փոխանցումների տարբեր տեսակների մեծ ֆունկցիոնալ ծածկույթը պայմանավորված է Synopsys-ի ծրագրային գործիքի կիրառմամբ, որի միջոցով հնարավոր է ղեկավարել փոխանցումների գրեթե բոլոր պարամետրերը:

Սխալի հայտնաբերման և տեղայնացման արդյունավետությունը ցուցադրելու նպատակով բերված է ՀՀԴ3.0 տեխնիկական արձանագրության մեջ ներկայացված կապի հաստատման քայլերի հաջորդականության ընթացքում սխալ վարքի հայտնաբերման օրինակ: Դիտարկվում է կապի մակարդակի այնպիսի սխալը, որի դեպքում կապի հաստատումը հնարավոր չէ: Այսպիսի սխալի օրինակ է երկրորդային սարքի ոչ բավարար քանակությամբ TS1 սիմվոլների առաքումը Polling.Active վիճակում: Այդպիսի սխալ կարող է առաջանալ կապի հաստատման ընթացքում սխալ տվյալների առաքման հետևանքով կամ կապի մասնակիցներից որևէ մեկի՝ տեխնիկական արձանագրությանը չհամապատասխանելու դեպքում: Այդպիսի սխալի հայտնաբերման նպատակով անհրաժեշտ է դիտարկել ՀՀԴ3.0 կապի մակարդակի հիմնական ՎՎՄ ֆունկցիոնալ ծածկույթի հաստատումները: Ըստ տեխնիկական արձանագրության՝ հետևյալ անցումներն են անհրաժեշտ՝ հիմնական աշխատանքային վիճակին անցման, այսինքն՝ կապի հաստատման համար.

1. Rx.Detect.Active->Polling.LFPS,
2. Polling.LFPS->Polling.RxEQ,
3. Polling.Rxeq->Polling.Active,
4. Polling.Active->Polling.Configuration,
5. Polling.Configuration->Polling.Idle,
6. Polling.Idle->Hot Reset,
7. Polling.Idle->Polling.LoopBack,
8. Polling.Idle->U0:

Այսինքն՝ անհրաժեշտ է առաջին հերթին դիտարկել ՀՀԴ3.0 կապի մակարդակի հիմնական ՎՎՄ-ի համար նախատեսված հաստատումների ցանկը: Կապի հաստատման ընթացքում առաջացած խնդիրների դեպքում կարելի է պնդել, որ վերը նշված անցումներից որևէ մեկը չի կատարվել: Վերը նկարագրված սխալի դեպքում 4-րդ կետում տրված անցումը չի կարող կատարվել և, հետևաբար, այդ հաստատումը ծածկված չէ: Այդ անցման համար անհրաժեշտ է, որ տվյալների ուղարկման հանգույցը առաքի, իսկ տվյալների ընդունման հանգույցը ստանա նախատեսված քանակությամբ TS1 սիմվոլները: Այսինքն՝ հաջորդ քայլում պետք է դիտարկել տվյալների առաքման և ընդունման հանգույցների հետևյալ հաստատումները.

- PollAct_TxTS1,
- PollAct_PollConfg_TS1:

Առաջինը ստուգում է՝ արդյոք նախատեսված սիմվոլներն ուղարկվել են, և նշված սխալի դեպքում այն պետք է ծածկված լինի: Երկրորդը ներդրված է տվյալների ուղարկման հանգույցում և ստուգում է՝ արդյոք նախատեսված քանակությամբ սիմվոլները ստացվել են կապի մյուս մասնակցից: Տրված սխալի դեպքում վեջինս ծածկված չի կարող լինել, հետևաբար սխալի հիմնական պատճառը հայտնաբերվեց: Հետագա՝ ավելի խորը ուսումնասիրություն կատարելու համար անհրաժեշտ է դիտարկել PIPE ինտերֆեյսի, այսինքն՝ ՀՀԴ3.0-ի ֆիզիկական մակարդակի աշխատանքային P0 վիճակի հաստատումները, որոնցում տրված սխալի դեպքում որևէ խնդիր պետք է որ չլինի: Այսինքն՝ սխալն առաջացել է ֆիզիկական մակարդակում կամ կապի մյուս մասնակցի պատճառով: Կապի մյուս մասնակցի համար դիտարկելով միևնույն հաստատումները՝ կարելի է հայտնաբերել սխալ տրամաբանությունը կամ, վատագույն դեպքում, հայտնաբերել սխալ պարունակող հանգույցը և շարունակել սխալի հայտնաբերման և ուղղման գործընթացը մոդելավորման միավոր մակարդակի միջավայրում:

Քննարկումն ամբողջացնելու համար անհրաժեշտ է դիտարկել առաջարկված մեթոդի համեմատությունը առաջին գլխում ներկայացված արդի այլ մեթոդների հետ (աղ. 3.6):

Աղյուսակ 3.6

Առաջարկվող և արդի մեթոդների համեմատությունը

	Synopsys Identify	Xilinx ChipScope	TAID
Արագագործության սահմանափակում	մեծ քանակությամբ ազդանշանների գործիքավորման դեպքում	մեծ քանակությամբ VIO և ICON օգտագործելու դեպքում	Միևնույն ազդանշանի դիտարկման բազմաթիվ հաստատումների օգտագործման դեպքում:
Հավելյալ ԾՓՀ ռեսուրսներ	փոքր քանակությամբ՝ միայն IICE միջուկի կառուցման նպատակով	փոքր քանակությամբ VIO և ICON հանգույցների համար	Մեծ քանակությամբ. • հաստատումներ, • ղեկավարման հանգույցներ:
Ֆունկցիոնալ ծածկույթի հաշվարկ	չկա	չկա	Տալիս է ԾՏԻՍ նախատիպում ֆունկցիոնալ ծածկույթի հաշվարկի հնարավորություն:
Սխալի հայտնաբերում և տեղայնացում	անհրաժեշտ են գիտելիքներ ղեկավարման միջուկի կառուցվածքի մասին	անհրաժեշտ են գիտելիքներ ղեկավարման միջուկի կառուցվածքի մասին	Ավելի պարզ է, քանի որ անհրաժեշտ է միայն տեխնիկական արձանագրության մեջ:
Գործիքավորման ներդրում	ավտոմատացված է	ավտոմատացված է	Անհրաժեշտ է ավելացնել հաստատումները նախագծման փուլում:

Եզրակացություններ

1. Սինթեզվող հաստատումների ներդրումը կատարվել է TAID ծրագրային և ապարատային միջավայրում: TAID Instrumentor-ի միջոցով կատարվում է անհրաժեշտ ապարատային բոլոր բաղադրիչների ներդրումը, ինչպես նաև նախագծի ՄՌՓՄ նկարագրության համապատասխանեցումը ԾՏԻՍ-ում իրականացմանը: TAID Debugger ՀՀԴ ծրագրային ապահովման հավելվածը կիրառվում է աշխատանքի ընթացքում հաստատումների ակտիվացման և արդյունքների դուրսբերման նպատակով: TAID-ի գրաֆիկական ինտերֆեյսը զգալիորեն պարզեցնում է նախատիպի կառուցման գործընթացը:

2. Առաջարկված մեթոդով կատարվել է ՀՀԴ առաջնային սարքի թեստավորման ֆունկցիոնալ ծածկույթի հաշվարկը: Ներկայացված է առաջարկված մեթոդով սխալի

հայտնաբերման և տեղայնացման գործընթացը, որը հնարավորություն է տալիս ՀՀԴ ճարտարագետներին (ծրագրային և ապարատային մասերի) հեշտությամբ հայտնաբերել և ուղղել սխալները՝ առանց թվային մասի իրականացման մասին ամբողջական պատկերացում ունենալու:

3. Համեմատություն է կատարվել արդի և առաջարկվող ՎՀ համակարգերի կառուցման միջոցների միջև: Առաջարկվող մեթոդը հնարավորություն է տալիս՝ հաշվելու ֆունկցիոնալ ծածկույթը և զգալիորեն պարզեցնելու սխալների հայտնաբերման գործընթացը: Սակայն առաջարկվող միջոցների դեպքում հավելյալ ԾՏԻՍ ռեսուրսների օգտագործումը և արագագործության սահմանափակումներն անխուսափելին են, իսկ ներկայացված այլ մեթոդների դեպքում այդ խնդիրներն առաջանում են միայն մեծ քանակությամբ ազդանշանների դեպքում:

ԵԶՐԱՀԱՆԳՈՒՄ

1. Մշակվել է համապիտանի հաջորդական դողի (ՀՀԴ) ղեկավարման թվային հանգույցի նախագծման նոր երթուղի՝ սինթեզվող հաստատումների կիրառմամբ, որը ՎՀ համակարգի ներքին ազդանշանների տեսանելիության բարձրացման հնարավորություն է տալիս: Ներդրված հաստատումները զգալիորեն պարզեցնում են սխալների պատճառների հայտնաբերման գործընթացը: Ի տարբերություն այլ մեթոդների, որոնց արդյունավետ կիրառման համար անհրաժեշտ են գիտելիքներ նախագծի իրականացման վերաբերյալ, առաջարկված մեթոդը չի պահանջում դրանք, և գիտելիքները միայն ՀՀԴ ինտերֆեյսի տեխնիկական արձանագրության վերաբերյալ բավարար են:

2. Առաջարկվել է նախագծման ընթացքում ներդրված հաստատումների կիրառումը՝ նախատիպի թեստավորման որակի գնահատման՝ ֆունկցիոնալ ծածկույթի հաշվարկի նպատակով:

3. Մշակվել է ՀՀԴ3.0 ինտերֆեյսի կապի մակարդակի թեստավորման ՎՀ համակարգի կառուցման միջոց՝ ֆունկցիոնալ թեստավորման հիմնական խնդիրների լուծման նպատակով: Առաջարկված համակարգը պարունակում է միայն ապարատային մաս, այդ իսկ պատճառով՝ ՀՀԴ ինտերֆեյսի տվյալների փոխանցման արագության ծրագրային սահմանափակումները բացակայում են: Թեստավորման ընթացքում դա թույլ է տալիս ստանալ տվյալների փոխանցման այնպիսի արագություն, որն առավելագույնս մոտ է տեխնիկական արձանագրությունում նշվածին:

4. Մշակվել է մի քանի սինքրոազդանշաններով թվային նախագծերի ամբողջական թեստավորման նախատիպ՝ հիմնված ՎՀ համակարգի վրա: Սինթեզվող հաստատումները ներդրվում են սինքրոազդանշանի տարբեր տիրույթների միջև անցում կատարող ազդանշանների սինքրոնացման տրամաբանական հանգույցներում՝ սխալների հայտնաբերման նպատակով: Ի տարբերություն ծրագրային միջոցների, որոնք կատարում են նախագծի կառուցվածքային վերլուծություն, առաջարկվող մեթոդը կատարում է ֆունկցիոնալ թեստավորում նախատիպում, հետևաբար՝ արդի մեթոդների թերություններն այս մեթոդում բացակայում են:

5. Նախատիպի կառուցման, թեստավորման ապարատային մասի և սինթեզվող հաստատումների ներդրման նպատակով նախագծվել է TAID Instrumentor ծրագրային գործիքային միջավայրը: TAID Debugger ՀՀԴ ծրագրային ապահովման հավելվածն օգտագործվում է նախատիպի թեստավորման ընթացքում ազդանշանների դիտարկման, արդյունքների դուրս բերման և հաստատումների ղեկավարման նպատակով: TAID Instrumentor ծրագրային միջոցով կառուցված ՀՀԴ3.0 նախատիպը վատագույն դեպքում, երբ ներդրված են ֆունկցիոնալ ծածկույթի հաստատումները, սպառում է 14%-ով ավելի իսկության աղյուսակների կառուցման նպատակով օգտագործվող տարրեր և 16%-ով ավելի տրիգերներ: Բացի այդ, նախատիպի արագագործությունն աննշան նվազում է: Սակայն, այդ արդյունքն ընդունելի է ԾՏԻՍ-ով կառուցված նախատիպի դեպքում: Արագագործության բարձրացման և հավելյալ ռեսուրսների նվազեցման նպատակով օգտագործվել է ԾՏԻՍ-ի մասնակի ծրագրավորման հատկությունը:

ՕԳՏԱԳՈՐԾՎԱԾ ԳՐԱԿԱՆՈՒԹՅՈՒՆ

1. Gokhale M.B., Graham P.S. Reconfigurable computing: Accelerating computation with field-programmable gate arrays.- Springer Science & Business Media, 2006.- 233 p.
2. Sotiriades E., Kozanitis C., Dollas A. FPGA based architecture for DNA sequence comparison and database search // IEEE International Parallel and Distributed Processing Symposium, IPDPS-2006.- Apr. 25, 2006.- 8 p.
3. FPGA Acceleration for Simultaneous Image Reconstruction and Segmentation based on the Mumford-Shah Regularization / W. Zhang, L. Shen, T. Page, et al // Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.- 2015.- P. 261-261.
4. Rencher M., Hutchings B. Automated target recognition on SPLASH 2 // IEEE Field-Programmable Custom Computing Machines: Proceedings, The 5th Annual IEEE Symposium.- Apr. 16, 1997.- P. 192-200.
5. Asano S., Maruyama T., Yamaguchi Y. Performance comparison of FPGA, GPU and CPU in image processing // IEEE Field Programmable Logic and Applications, FPL-2009.- Aug. 31, 2009.- P. 126-131.
6. Hennessy J., Patterson D.- Computer architecture: a quantitative approach.- Elsevier, 2011.- 476 p.
7. Design patterns for reconfigurable computing / A. DeHon, J. Adams, M. DeLorimier, el al // IEEE Field-Programmable Custom Computing Machines, FCCM 2004: 12th Annual IEEE Symposium on.- Apr. 2, 2004.- P. 13-23.
8. De Micheli G. Hardware synthesis from C/C++ models // Proceedings of the conference on Design, Automation and Test in Europe, ACM.- Jan. 1, 1999.- P. 80.
9. Celoxica, RC2000 Development and evaluation board data sheet // http://www.eetimes.com/document.asp?doc_id=1217588
10. PipeRench: A reconfigurable architecture and compiler / S.C. Goldstein, H. Schmit, M.

Budiu, et al // Computer.- 2000.- Vol. 33, issue 4.- P. 70-77.

11. Compton K., Hauck S. Reconfigurable computing: a survey of systems and software // ACM Computing Surveys.- Jun. 1,2002.- Vol. 34, issue 2.- P. 171-210.
12. Cadence Design Systems Inc, Palladium Datasheet // https://www.cadence.com/rl/Resources/datasheets/incisive_enterprise_palladium.pdf
13. Mentor Graphics, Vstation Pro: High Performance System Verification // http://www.businesswire.com/news/home/20031027005194/en/Mentor-Graphics-Announces-Scalable-Verification-Platform-Technologies
14. MorphoSys: case study of a reconfigurable computing system targeting multimedia applications / H. Singh, G. Lu, R. Maestre, et al // Proceedings of the 37th Annual Design Automation Conference, AMC.- Jun. 1, 2000.- P. 573-578.
15. Hauser J.R., Wawrzynek J. Garp: A MIPS processor with a reconfigurable coprocessor // IEEE Field-Programmable Custom Computing Machines, Proceedings, The 5th Annual Symposium.- Apr. 16, 1997.- P. 12-21.
16. MorphoSys: a reconfigurable processor targeted to high performance image application / G. Lu, M.H. Lee, H. Singh, et al // Parallel and Distributed Processing.- Springer Berlin Heidelberg.- Apr. 12, 1999.- P. 661-669.
17. Memory addressing organization for stream-based reconfigurable computing / M. Herz, R. Hartenstein, M. Miranda, et al // Proceedings of the 9th IEEE International Conference on Electronics, Circuits and Systems.- Sep. 15, 2002.- P. 1-5.
18. Programmable active memories: Reconfigurable systems come of age / J.E. Vuillemin, P. Bertin, D. Roncin, et al // Very Large Scale Integration (VLSI) Systems, IEEE Transactions on.- Mar. 4, 1996.- Vol. 4, issue 1.- P. 56-69.
19. Wittig R.D., Chow P. OneChip: An FPGA processor with reconfigurable logic // IEEE FPGAs for Custom Computing Machines.- Apr. 17, 1996.- P. 126-135.
20. Javaid H., Pittman R.N., Forin A. A Framework for Automated Acceleration of Application Binaries on eMIPS.- Feb., 2011.- 20 p.

21. A reconfigurable arithmetic array for multimedia applications / A. Marshall, T. Stansfield, I. Kostarnov, et al // Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays, ACM.- Feb. 1, 1999.- P. 135-143.

22. Mirsky E., DeHon A. MATRIX: a reconfigurable computing architecture with configurable instruction distribution and deployable resources // FPGAs for Custom Computing Machines: IEEE Symposium.- Apr. 17, 1996.- P. 157-166.

23. The Raw microprocessor: A computational fabric for software circuits and general-purpose programs / M.B. Taylor, J. Kim, J. Miller, et al // IEEE Micro.- Aug. 7, 2002.- Vol. 22, issue 2.- P. 25-35.

24. Razdan R., Smith M.D. A high-performance microarchitecture with hardware-programmable functional units // Proceedings of the 27th annual international symposium on Microarchitecture, ACM.- Nov. 30, 1994.- P. 172-180.

25. Pilchard-a reconfigurable computing platform with memory slot interface / P.H.W. Leong, M.P. Leong, O.Y. Cheung, et al // IEEE Field-Programmable Custom Computing Machines, FCCM'01.- Mar. 29, 2001.- P. 170-179.

26. The MorphoSys dynamically reconfigurable system-on-chip / G. Lu, H. Singh, M.H. Lee, et al // IEEE Evolvable Hardware, Proceedings of the First NASA/DoD Workshop.- Jul. 19-21, 1999.- P. 152-160.

27. Shannon L., Cojocaru V., Dao C.N., Leong F. Technology Scaling in FPGAs: Trends in Applications and Architectures // IEEE Field-Programmable Custom Computing Machines (FCCM), 23rd Annual International Symposium.- May 2, 2015.- P. 1-8.

28. UltraScale Architecture and Product Overview, v2.7.- Feb. 17, 2016.- http://www.xilinx.com/ support/documentation/data_sheets/ds890-ultrascale-overview.pdf

29. Grimaldi R.P. Discrete and Combinatorial Mathematics: An Applied Introduction.- 2nd ed.- Addison-Wesley Longman Publishing Co., Inc, 1989.- 800 p.

30. Brueck S.R. There are no fundamental limits to optical lithography: International Trends in

Applied Optics.- 2002.- P. 85-110.

31. Dehon A. Nanowire-based programmable architectures // ACM Journal on Emerging Technologies in Computing Systems (JETC).- Jul. 1, 2005.- Vol. 1, issue 2.- P. 109-162.

32. Dehon A. Deterministic addressing of nanoscale devices assembled at sublithographic pitches // Nanotechnology, IEEE Transactions.- Nov. 4, 2005.- Vol. 4, issue 6.- P. 681-687.

33. Dehon A., Naeimi H. Seven strategies for tolerating highly defective fabrication // Design & Test of Computers, IEEE.- Jul., 2005.- Vol. 22, issue 4.- P. 306-315.

34. Trimberger S. Effects of FPGA architecture on FPGA routing // Proceedings of the 32nd annual ACM/IEEE Design Automation Conference.- Jan. 1, 1995.- P. 574-578.

35. 7 Series FPGAs Configuration, User Guide, v1.10.- June 24, 2015.-http://www.xilinx.com/support/documentation/user_guides/ug470_7Series_Config.pdf

36. Foster H.D. Trends in functional verification: A 2014 industry study // Design Automation Conference (DAC), 52nd ACM/EDAC/IEEE.- Jun. 8, 2015.- P. 1-6.

37. Pryce D. 80486 32-bit CPU breaks new ground in chip density and operating performance. Intel Corp: product announcement, EDN (Press release).- May 11, 1989.

38. Dual-Core Intel® Itanium® Processor: Reference Manual Update.- http://www.intel.com/content/dam/doc/product-brief/high-performance-computing-itanium-9100-powering-mainframe-solutions-on-flexible-industry-standard-servers-brief.pdf

39. Doug A., Lesea L., Richter R. FPGA-based prototyping methodology manual.- Synopsys, San Jose, Happy About, 2011.- 470 p.

40. Network on chip: An architecture for billion transistor era / A. Hemani, A. Jantsch, S. Kumar, et al // Proceeding of the IEEE NorChip Conference.- Nov. 5, 2000.- Vol. 31.- 8 p.

41. Lee Y., Kim N., Kim J.B., Min B. Millions to thousands issues through knowledge based SoC CDC Verification // IEEE SoC Design Conference (ISOCC).- Nov. 4, 2012.- P. 391-394.

42. Rajsuman R. System-on-a-chip: Design and Test.- Artech House, Inc.- Jul. 1, 2000.- 294 p.

43. Bricaud P. Reuse methodology manual: for system-on-a-chip designs.- Springer Science &

Business Media, Dec. 6, 2012.- 281 p.

44. United States patent application US 13/966,028. Microcontroller programmable system on a chip with programmable interconnect / W.S. Snyder; Cypress Semiconductor Corporation.- Aug. 13, 2013.

45. SystemVerilog 3.1a Language Reference Manual.- http://www.ece.uah.edu/~gaede/cpe526/SystemVerilog_3.1a.pdf

46. Dorsch R., Wunderlich H.J. Tailoring ATPG for embedded testing // Test Conference Proceedings, International.- 2001 .- P. 530-537.

47. Aitken R., Marinissen E.J. Guest Editors' Introduction: Addressing the Challenges of Debug and Diagnosis // Design & Test of Computers, IEEE.- May 30, 2008.- Vol. 25, issue 3.- P. 206-207.

48. Synopsys Global Users Survey 2011 (N-646).

49. Mack C.A. Fifty Years of Moore's Law // IEEE Transactions on Semiconductor Manufacturing. – 2011. – P. 202-207.

50. Molina A., Cadenas O. Functional verification: approaches and challenges // Latin American applied research.- Jan., 2007.- 37.1.- P. 65-69.

51. PCI Express® Base Specification Revision 4.0.- https://doc.xdevs.com/doc/Standards/PCI/PCI_Express_Base_4.0_Rev0.3_February19-2014.pdf

52. Fine S., Ziv A. Coverage directed test generation for functional verification using bayesian networks // Design Automation Conference Proceedings, IEEE.- June 2, 2003.- P. 286-291.

53. Functional verification of large ASICs / A. Evans, A. Silburt, G. Vrckovnik, et al // Design Automation Conference Proceedings. IEEE.- Jun. 19, 1998.- P. 650-655.

54. Foster, H. Marschner E., Wolfsthal, Y. IEEE 1850 PSL: The next generation // Design and Verification Conference and exhibition, DVCon'05.- Feb., 2005.- 7 p.

55. IP Solutions for Synchronizing Signals that Cross Clock Domains. https://www.synopsys.com/dw/doc.php/wp/dw_crossclockdomains_wp.pdf?cmp=IPClockdomainWP-IP-whitepaper-wp

56. Ginosar R. Metastability and synchronizers: A tutorial // IEEE Design & Test of Computers.- Sep. 20, 2011.- P. 23-35.

57. Hand N. The need for an automated clock-domain crossing verification solution.- Mentor Graphics Corporation, 2006.- 20 p.

58. Verma S., Dabare A.S. Understanding clock domain crossing issues // EE Times.- Dec., 2007.

59. Five Challenges to FPGA-Based Prototyping.- http://www.eetimes.com

60. ISE In-Depth Tutorial, v14.1, Apr 24, 2012.-http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ise_tutorial_ug695.pdf

61. JTAG 101, IEEE 1149.x and Software Debug.- http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/jtag-101-ieee-1149x-paper.pdf

62. ChipScope Pro 11.4 Software and Cores, v11.4, December 2, 2009.- http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/chipscope_pro_sw_cores_ug029.pdf

63. Synopsys Identify Datasheet.- http://www.synopsys.com/tools/implementation/fpgaimplementation/capsulemodule/identify_ds.pdf

64. CORE Generator Guide.- http://www2.informatik.hu-berlin.de/~fwinkler/psvfpga/synthese/ISE-Dokumentaion/docs/cgn/cgn.pdf

65. USB-IF.- http://www.usb.org/

66. Thomas D., Moorby P. The Verilog® Hardware Description Language.- Springer Science & Business Media, Sep. 11, 2008.- 380 p.

67. Krrikyan H. USB IP development flow based on Synthesizable Assertions // 10th International Conference on Semiconductor Micro- and Nanoelectronics, September 11-13, 2015.- Yerevan, Armenia, 2015.- P.147-152.

68. Krrikyan H. Data Generator and Synthesizable Monitor for USB3.0 Link Layer Testing // 10th International Conference on Semiconductor Micro- and Nanoelectronics, September 11-13, 2015.- Yerevan, Armenia, 2015.- P.141-146.

69. Mostafa M., Safar M., El-Kharashi M.W., Dessouky M. SystemVerilog Assertion Debugging based on Visualization, Simulation Results, and Mutation // Microprocessor Test and Verification Workshop, 15th International.- Dec. 15-16, 2014.- P. 55 - 66.

70. Universal Serial Bus 3.1 Specification.- July 26, 2013. – 631 p.

71. http://www.synopsys.com/Systems/FPGABasedPrototyping/Pages/HAPS.aspx

72. http://teledynelecroy.com

73. http://www.ellisys.com/products/usbex350

74. I^2S bus specification.- https://www.sparkfun.com/datasheets/BreakoutBoards/I2SBUS.pdf

75. SPDIF interface.- http://www.epanorama.net/documents/audio/spdif.html

76. PHY Interface For the PCI Express, SATA, and USB 3.1 Architectures.-http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/phy-interface-pci-express-sata-usb31-architectures-ver43.pdf

77. Krrikyan H., Hovhannisyan T., Manukyan S. Prototyping system for USB3. 0 link layer using synthesizable assertions and partial reconfiguration // IEEE Computer Science and Information Technologies (CSIT).- Yerevan, September, 2015.- P. 238-241.

78. http://www.synopsys.com/tools/implementation/fpgaimplementation/pages/synplify-premier.aspx

79. Virtex-5 FPGA User Guide, v5.4.- March 2012.- http://www.xilinx.com/support/documentation/user_guides/ug190.pdf

80. Partial Reconfiguration User Guide (v14.5) April 26, 2013, www.xilinx.com

81. 7 Series FPGAs Configuration.- June 24, 2015.- http://www.xilinx.com/support/documentation/user_guides/ug470_7Series_Config.pdf

82. Liu S., Pittman R.N., Forin A. Minimizing partial reconfiguration overhead with fully streaming DMA engines and intelligent ICAP controller // FPGA.- Feb. 21, 2010.- P. 292.

83. http://www.usb.org/developers/docs/usb20_docs/

84. Melikyan V., Krrikyan H., Hovhannisyan T., Babayan D. Synthesizable Assertions for Clock Domain Crossing Verification for the USB Prototype // Proceedings of the Republic of

Armenia National Academy of Sciences and National Polytechnic University of Armenia. Series of Technical Sciences.- 2016.-V.69, N 2.- P. 138-150.

85. Krrikyan H. Functional verification method of synchronization schemes within USB2.0 prototype // Proceeding of Engineering Academy of Armenia (PEAA).- 2016.- V.13, N 1.- P. 143-146.

86. Beck M., Magnus R., Kunitz U. Linux Kernel Internals.- Addison-Wesley Longman Publishing Co., Inc.- Apr. 1, 2002.- 482 p.

87. https://wiki.ubuntu.com

88. Уолл Л., Кристиансен Т., Орвант Дж. Программирование на Perl.- М.: Символ-Плюс, 2006.- 1152 с.

89. Welch B.B., Hobbs J., Welch B. Practical Programming in Tcl and Tk.- Prentice Hall Ptr, 2003.- 960 p.

90. Ciletti M.D. Advanced Digital Design with the Verilog HDL.- 2 edition.- Prentice Hall, 2010. - 984 p.

91. Thomas D., Moorby P. The Verilog Hardware Description Language. 5th ed. – Springer, 2002. - 386 p.

92. Саммерфилд М. Qt. Профессиональное программирование. Разработка кроссплатформенных приложений на C++.- М.: Символ-Плюс, 2011.- 560 с.

93. Прата С. Язык программирования С: Лекции и упражнения.- М.: Вильямс, 2006.- 960 с.

94. https://www.synopsys.com/Tools/Verification/static-formal-verification/Pages/spyglass-cdc.aspx

95. AHB-Lite Protocol Specification, v1.0.- http://www.eecs.umich.edu/courses/eecs373/readings/ARM_IHI0033A_AMBA_AHB-Lite_SPEC.pdf

96. APB Protocol Specification , v1.0.- https://web.eecs.umich.edu/~prabal/teaching/eecs373-f11/readings/ARM_AMBA3_APB.pdf

97. http://www.synopsys.com/Prototyping/FPGABasedPrototyping/Pages/protocompiler.aspx

98. http://www.synopsys.com/Tools/Verification/FunctionalVerification/Pages/VCS.aspx

99. 7 Series FPGAs Memory Resources, v1.11.- November 12, 2014.- http://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf
100. 7 Series FPGAs Clocking Resources, v1.11.2.- June 12, 2015.- http://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf
101. 7 Series FPGAs Overview, v1.17.- May 27, 2015.- http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf
102. http://www.usb.org/developers/compliance/ssusb_testing/USB_3_0_Link_Layer_Test_Specification_2013_08_02.pdf
103. http://www.usb.org/developers/tools/
104. http://www.usb.org/developers/compliance
105. http://compliance.usb.org/cv

ՀԱՎԵԼՎԱԾ1

SYNOPSYS®

№ 406/16 05 2016

Հաստատում եմ
տնօրեն Հ. Մուսայելյան

Հարություն Ռուբենի Կրրիկյանի «Վերակառուցավորվող հաշվողական համակարգերի կառուցման միջոցների մշակում և հետազոտում» թեմայով թեկնածուական ատենախոսության արդյունքների

ՆԵՐԴՐՄԱՆ ԱԿՏ

ԵՊՀ «Ռադիոֆիզիկայի» ֆակուլտետի 3-րդ կուրսի ասպիրանտ Հ. Ռ. Կրրիկյանի կողմից մշակված TAID ծրագրավորվող տրամաբանական ինտեգրալ սխեմաներով(ԾՏԻՍ) նախատիպերի կառուցման ծրագրային միջոցը ներդրված է «ՍԻՆՈՓՍԻՍ ԱՐՄԵՆԻԱ» ՓԲԸ-ում: TAID Instrumnetor ծրագրային միջոցն օգտագործվում է համապիտանի հաջորդական դողի(ՀՀԴ) ղեկավարման հանգույցների ապարատային մասի ԾՏԻՍ նախատիպերի կառուցման, և սինթեզվող հաստատումների ներդրման համար: TAID Debugger ծրագրային միջոցն օգտագործվում է ՀՀԴ-ի նախատիպի թեստավորման ընթացքում դիտարկվող ազդանշանների սխալ վարքի հայտնաբերման, և թեստավորման ֆունկցիոնալ ծածկույթի հաշվարկի համար: ՀՀԴ3.0, ՀՀԴ2.0 և ՀՀԴ1.1 ղեկավարման հանգույցների նախատիպերը կառուցվում են և թեստավորումը կատարվում է TAID ծրագրային ապարատային միջավայրում:

Լուծումների խմբի հետազոտման և
Մշակման բաժնի II կարգի կառավարիչ՝ Մ.Հարությունյան

"ՍԻՆՈՓՍԻՍ ԱՐՄԵՆԻԱ" ՓԲԸ
0026, ՀՀ, ԵՐԵՎԱՆ, ԱՐՇԱԿՈՒՆՅԱՑ 41
ՀԵՌ.՝ (+374 10) 49 21 00, ՖԱՔՍ՝ (+374 10) 49 26 96
ՀՎՀՀ 02236362

"SYNOPSYS ARMENIA" CJSC
41 ARSHAKUNYATS AVE., YEREVAN, ARMENIA, 0026
TEL.: (+374 10) 49 21 00, FAX: (+374 10) 49 26 96
TAX PAYER'S ID 02236362

ՀԱՎԵԼՎԱԾՉ

- **Կապի մակարդակի հիմնական ՎՎՄ հաստատումների ներդրման հանգույցը**

```
module ltssm_top(  ux_exit_timer_run, lgen_ux_exit_active,
                    link_clk,  reset, timeout,
                    u1_state, u2_state, u3_state, link_state,
                    ux_exit_lgen_timer, lfps_ux_exit,
                    // ports used to get coverage data. Connected to wrapper before csr* interface
                   get_one_byte,
                   byte_data,
                   addr,
                    // outputs
                   done1,
                  status1,
                  read_valid
            );

input [3:0] link_state ;
input [1:0] u1_state,u2_state,u3_state ;
input ux_exit_timer_run, lgen_ux_exit_active, timeout,ux_exit_lgen_timer,lfps_ux_exit;
output [12:0] done1;
output [13*2-1:0] status1;
//
input  get_one_byte;
input [3:0] addr;
output [7:0] byte_data;
//outputs
output read_valid;
//clock and reset
input link_clk, reset;
localparam [3:0] U1 = 4'b1,
        U2 = 4'b10,
        U3 = 4'b11,
        RECOV = 4'b0111,
        INACT = 4'b0100;
localparam [1:0] U1_EXIT = 2'b11,
        U2_EXIT = 2'b11,
        U3_EXIT = 2'b11;
localparam [3:0] LTSSM_U1_EXIT   = 4'b0000,
        LTSSM_U2_EXIT   = 4'b0001,
        LTSSM_U3_RW     = 4'b0010,
        U1_LOC_EXIT     = 4'b0011,
        U2_LOC_EXIT     = 4'b0100,
        U3_LOC_EXIT     = 4'b0101,
        U1_LOC_EXIT_REC = 4'b0110,
        U2_LOC_EXIT_REC = 4'b0111,
        U3_LOC_EXIT_REC = 4'b1000,
        U1_LOC_EXIT_INACT = 4'b1001,
        U2_LOC_EXIT_INACT = 4'b1010,
        U1_REM_EXIT     = 4'b1011,
        U2_REM_EXIT     = 4'b1100;
wire [2*13-1 :0]      status;
wire [12:0]     done;
reg  [12:0]     done_d;
```

```
reg  [3:0]      count;
reg  clean;
reg  [13*8-1:0] gl_stat;

wire [12:0]     done_p ;
wire            read_valid;
reg  [7:0]      byte_data_r;
assign done1= done;
assign status1 = status;
// Read data
assign read_valid = (count == 13);
always @(done or reset)
begin
   if(reset) count = 0;
   else count = count+1;
end
assign byte_data = byte_data_r ;

always @(posedge link_clk)
begin
  if(reset) begin
       byte_data_r <= 8'b0;
       clean <= 0;
     end
     else if(get_one_byte)
     begin
         case(addr)
           4'd1:byte_data_r <= gl_stat[7:0];
           4'd2:byte_data_r <= gl_stat[15:8];
           4'd3:byte_data_r <= gl_stat[23:16];
           4'd4:byte_data_r <= gl_stat[31:24];
           4'd5:byte_data_r <= gl_stat[39:32];
           4'd6:byte_data_r <= gl_stat[47:40];
           4'd7:byte_data_r <= gl_stat[55:48];
           4'd8:byte_data_r <= gl_stat[63:56];
           4'd9:byte_data_r <= gl_stat[71:64];
           4'd10:byte_data_r <= gl_stat[79:72];
           4'd11:byte_data_r <= gl_stat[87:80];
           4'd12:byte_data_r <= gl_stat[95:88];
           4'd13:byte_data_r <= gl_stat[103:96];
         endcase
     end
end

//  Write data to reg
assign done_p[0]  = !done[0] & done_d[0];
assign done_p[1]  = !done[1] & done_d[1];
assign done_p[2]  = !done[2] & done_d[2];
assign done_p[3]  = !done[3] & done_d[3];
assign done_p[4]  = !done[4] & done_d[4];
assign done_p[5]  = !done[5] & done_d[5];
assign done_p[6]  = !done[6] & done_d[6];
assign done_p[7]  = !done[7] & done_d[7];
assign done_p[8]  = !done[8] & done_d[8];
assign done_p[9]  = !done[9] & done_d[9];
assign done_p[10]  = !done[10] & done_d[10];
assign done_p[11]  = !done[11] & done_d[11];
assign done_p[12]  = !done[12] & done_d[12];
```

```
always @(posedge link_clk)
begin
  if(reset) done_d <= 13'b0;
  else  done_d[12:0] <= done[12:0];
end

always @(posedge link_clk)
begin
  if(reset)
  begin
       gl_stat <= {104{1'b0}};
  end
 else
   begin
     case(done_p)
       13'b0000000000001: begin
           gl_stat[7:0] <={status[1:0],LTSSM_U1_EXIT,2'b00};
       end
       13'b0000000000010: begin
          gl_stat[15:8]<={status[3:2],LTSSM_U2_EXIT,2'b00};
       end
       13'b0000000000100: begin
           gl_stat[23:16] <={status[5:4],LTSSM_U3_RW,2'b00};
       end
       13'b0000000001000: begin
          gl_stat [39:32]<={status[7:6],U1_LOC_EXIT,2'b00};
       end
       13'b0000000010000: begin
          gl_stat[47:40]<={status[9:8],U2_LOC_EXIT,2'b00};
       end
       13'b0000000100000: begin
         gl_stat[55:48] <={status[11:10],U3_LOC_EXIT,2'b00};
       end
       13'b0000001000000: begin
         gl_stat[63:56] <={status[13:12],U1_LOC_EXIT_REC,2'b00};
       end
       13'b0000010000000: begin
         gl_stat[71:64] <={status[15:14],U2_LOC_EXIT_REC,2'b00};
       end
       13'b0000100000000: begin
         gl_stat[79:72] <={status[17:16],U3_LOC_EXIT_REC,2'b00};
       end
       13'b0001000000000: begin
         gl_stat[87:80] <={status[19:18],U1_LOC_EXIT_INACT,2'b00};
       end
       13'b0010000000000: begin
         gl_stat[95:88] <={status[21:20],U2_LOC_EXIT_INACT,2'b00};
       end
       13'b0100000000000: begin
         gl_stat[103:96] <={status[23:22],U1_REM_EXIT,2'b00};
       end
       13'b1000000000000: begin
         gl_stat[31:24] <={status[25:24],U2_REM_EXIT,2'b00};
       end
       default: gl_stat <= gl_stat;
     endcase
   end
end
```

```
ltssm_u1_exit u_ltssm_u1_exit( .link_state(link_state),
                               .u1_state(u1_state),
                               .ux_exit_timer_run(ux_exit_timer_run),
                               .lgen_ux_exit_active(lgen_ux_exit_active),
                               .link_clk(link_clk),
                               .reset(reset),
                               .done(done[0]),
                               .status(status[1:0])
                             );
ltssm_u2_exit u_ltssm_u2_exit( .link_state(link_state),
                               .u2_state(u2_state),
                               .ux_exit_timer_run(ux_exit_timer_run),
                               .lgen_ux_exit_active(lgen_ux_exit_active),
                               .link_clk(link_clk),
                               .reset(reset),
                               .done(done[1]),
                               .status(status[3:2])
                             );
ltssm_u3_wakeup u_ltssm_u3_wakeup(
                               .link_state(link_state),
                               .u3_state(u3_state),
                               .lgen_ux_exit_active(lgen_ux_exit_active),
                               .link_clk(link_clk),
                               .reset(reset),
                               .done(done[2]),
                               .status(status[5:4])
                             );

remote_uxexit #(U1) u1_loc_exit(
                               .link_state(link_state),
                               .ux_exit_lgen_timer(ux_exit_lgen_timer),
                               .timeout(timeout),
                               .link_clk(link_clk),
                               .reset(reset),
                               .done(done[3]),
                               .status(status[7:6])
                             );
remote_uxexit #(U2) u2_loc_exit (
                               .link_state(link_state),
                               .ux_exit_lgen_timer(ux_exit_lgen_timer),
                               .timeout(timeout),
                               .link_clk(link_clk),
                               .reset(reset),
                               .done(done[4]),
                               .status(status[9:8])
                             );
remote_uxexit #(U3) u3_loc_exit (
                               .link_state(link_state),
                               .ux_exit_lgen_timer(ux_exit_lgen_timer),
                               .timeout(timeout),
                               .link_clk(link_clk),
                               .reset(reset),
                               .done(done[5]),
                               .status(status[11:10])
                             );
successful_local_ux_exit #(U1, U1_EXIT, RECOV) u1_loc_exit_rec(
                                        .link_state(link_state),
```

```
                                    .ux_state(u1_state),
                                    .timeout(timeout),
                                    .link_clk(link_clk),
                                    .reset(reset),
                                    .done(done[6]),
                                    .status(status[13:12])
                                  );
successful_local_ux_exit #(U2, U2_EXIT, RECOV) u2_loc_exit_rec(
                                    .link_state(link_state),
                                    .ux_state(u2_state),
                                    .timeout(timeout),
                                    .link_clk(link_clk),
                                    .reset(reset),
                                    .done(done[7]),
                                    .status(status[15:14])
                                  );
successful_local_ux_exit #(U3, U3_EXIT, RECOV) u3_loc_exit_rec(
                                    .link_state(link_state),
                                    .ux_state(u3_state),
                                    .timeout(timeout),
                                    .link_clk(link_clk),
                                    .reset(reset),
                                    .done(done[8]),
                                    .status(status[17:16])
                                  );
successful_local_ux_exit #(U1, U1_EXIT, INACT) u1_exit_fail_ss_inact(
                                    .link_state(link_state),
                                    .ux_state(u1_state),
                                    .timeout(timeout),
                                    .link_clk(link_clk),
                                    .reset(reset),
                                    .done(done[9]),
                                    .status(status[19:18])
                                  );
successful_local_ux_exit #(U2, U2_EXIT, INACT) u2_exit_fail_ss_inact(
                                    .link_state(link_state),
                                    .ux_state(u2_state),
                                    .timeout(timeout),
                                    .link_clk(link_clk),
                                    .reset(reset),
                                    .done(done[10]),
                                    .status(status[21:20])
                                  );
successful_rem_ux_exit_recovery #(U1) u1_rem_exit(
                                    .link_state(link_state),
                                    .lfps_ux_exit(lfps_ux_exit),
                                    .timeout(timeout),
                                    .link_clk(link_clk),
                                    .reset(reset),
                                    .done(done[11]),
                                    .status(status[23:22])
                                   );
successful_rem_ux_exit_recovery #(U2) u2_rem_exit  (
                                    .link_state(link_state),
                                    .lfps_ux_exit(lfps_ux_exit),
                                    .timeout(timeout),
                                    .link_clk(link_clk),
                                    .reset(reset),
```

```
                                .done(done[12]),
                                .status(status[25:24])
                               );
endmodule
```

- **ՀՀԴ ցածր էներգասպառմամբ վիճակների ստուգման որոշ հաստատումների ՄՌՓՄ նկարագրություն**

```
module ltssm_u1_exit(
                    link_state,
                    u1_state,
                    ux_exit_timer_run,
                    lgen_ux_exit_active,
                    link_clk,
                    reset,
                    done,
                    status
                      );


input [3:0] link_state;
input [1:0] u1_state;
input ux_exit_timer_run;
input lgen_ux_exit_active;

// clock
input link_clk;

// reset
input reset;

// status
output done;
output [1:0] status;
reg [1:0] status_r;
reg done_r;

assign done = done_r;
assign status = status_r;

parameter [3:0] U1 = 4'b1;
parameter [1:0] U1_EXIT = 2'b11;



localparam [2:0] IDLE=3'b000,
          WAIT_U1EXIT = 3'b001,
          CHECK_TIMER_RUN = 3'b010,
          LGEN_U1EXIT = 3'b011,
          PASS = 3'b100,
          FAIL=3'b101;


reg [2:0] next_as_state;
reg [3:0] link_state_d;
```

```
wire entry_u1 = !(link_state_d == U1) && (link_state == U1) ;

// generate delayed version of link_state
always @(posedge link_clk) begin
  if(reset)
  begin
    link_state_d <= 0;
  end
  else
  begin
    link_state_d <= link_state;
  end
end

always @(posedge link_clk)
begin
  if(reset)
  begin
      status_r <= 2'b00;
      done_r   <= 1'b0;
      next_as_state <= IDLE;
  end
  else
    case(next_as_state)
      IDLE:begin
           if(entry_u1)
             next_as_state <= WAIT_U1EXIT;
          end
      WAIT_U1EXIT: begin
                    if(u1_state== U1_EXIT)
                      next_as_state <= CHECK_TIMER_RUN ;
                end
      CHECK_TIMER_RUN: begin
                     if(ux_exit_timer_run) next_as_state <= LGEN_U1EXIT;
                     else next_as_state <= FAIL;
                    end
      LGEN_U1EXIT: begin
                     if(!lgen_ux_exit_active) next_as_state <= FAIL;
                     else next_as_state <= PASS;
                 end
      PASS: begin
              status_r <= 2'b00;
              done_r   <= 1;
            end
      FAIL: begin
              status_r <= 2'b10;
              done_r   <= 1;
            end
      default: begin
              next_as_state <= IDLE;
              status_r      <= 2'b11;
              done_r        <= 0;
            end
    endcase
end

endmodule
```

```
module ltssm_u2_exit(
                    link_state,
                    u2_state,
                    ux_exit_timer_run,
                    lgen_ux_exit_active,
                    link_clk,
                    reset,
                    done,
                    status
                      );


input [3:0] link_state;
input [1:0] u2_state;
input ux_exit_timer_run;
input lgen_ux_exit_active;

// clock
input link_clk;

// reset
input reset;

// status
output done;
output [1:0] status;
reg [1:0] status_r;
reg done_r;

assign done = done_r;
assign status = status_r;


localparam [2:0] IDLE=3'b000,
          WAIT_U2EXIT = 3'b001,
          CHECK_TIMER_RUN = 3'b010,
          LGEN_U2EXIT = 3'b011,
          PASS = 3'b100,
          FAIL=3'b101;
parameter [3:0] U2 = 4'b10;
parameter [1:0] U2_EXIT = 2'b11;

reg [2:0] next_as_state;
reg [3:0] link_state_d;

wire entry_u2 = !(link_state_d == U2) && (link_state == U2) ;

// generate delayed version of link_state
always @(posedge link_clk) begin
  if(reset)
  begin
    link_state_d <= 0;
  end
  else
  begin
    link_state_d <= link_state;
  end
```

```
end

always @(posedge link_clk)
begin
  if(reset)
  begin
      status_r <= 2'b00;
      done_r   <= 1'b0;
      next_as_state <= 0;
  end
  else
    case(next_as_state)
      IDLE:begin
           if(entry_u2)
             next_as_state <= WAIT_U2EXIT;
          end
      WAIT_U2EXIT: begin
                    if(u2_state== U2_EXIT)
                      next_as_state <= CHECK_TIMER_RUN ;
                end
      CHECK_TIMER_RUN: begin
                     if(ux_exit_timer_run) next_as_state <= LGEN_U2EXIT;
                     else next_as_state <= FAIL;
                    end
      LGEN_U2EXIT: begin
                    if(lgen_ux_exit_active) next_as_state <= PASS;
                    else next_as_state <= FAIL;
                end
      PASS: begin
              status_r <= 2'b00;
              done_r   <= 1;
           end
      FAIL: begin
              status_r <= 2'b10;
              done_r   <= 1;
           end
     default: begin
              next_as_state <= IDLE;
              status_r      <= 2'b11;
              done_r        <= 0;
           end
       endcase
end

endmodule

module ltssm_u3_wakeup(
                  link_state,
                  u3_state,
                  lgen_ux_exit_active,
                  link_clk,
                  reset,
                  done,
                  status
                    );


input [3:0] link_state;
```

```
input [1:0] u3_state;
input lgen_ux_exit_active;

// clock
input link_clk;

// reset
input reset;

// status
output done;
output [1:0] status;
//reg [1:0] status_r;
//reg done_r;


localparam [3:0] IDLE=3'b000,
          WAIT_U3EXIT = 3'b001,
          LGEN_U3EXIT = 3'b011,
          PASS = 3'b100,
          FAIL=3'b101;
parameter  [3:0] U3 = 4'b10;
parameter  [1:0] U3_EXIT = 2'b11;

reg [2:0] next_as_state;
reg [3:0] link_state_d;



assign done =( next_as_state == PASS) || (next_as_state == FAIL);
assign status = ( next_as_state == FAIL)?2'b11:2'b00;


wire entry_u3 = !(link_state_d == U3) && (link_state == U3) ;

// generate delayed version of link_state
always @(posedge link_clk) begin
  if(reset)
  begin
    link_state_d <= 0;
  end
  else
  begin
    link_state_d <= link_state;
  end
end

always @(posedge link_clk)
begin
  if(reset)
  begin
//       status_r <= 2'b00;
//       done_r   <= 1'b0;
       next_as_state <= IDLE;
  end
  else
    case(next_as_state)
      IDLE:begin
```

```
            if(entry_u3)
             begin
               next_as_state <= WAIT_U3EXIT;
//               status_r <= 2'b01 ;
//               done_r <= 0 ;
             end
            end
        WAIT_U3EXIT: begin
                    if(u3_state== U3_EXIT)
                      next_as_state <= LGEN_U3EXIT ;
                 end
        LGEN_U3EXIT: begin
                    if(!lgen_ux_exit_active) next_as_state <= FAIL;
                    else next_as_state <= PASS;
                 end
        PASS: begin
          next_as_state <= next_as_state;
//               status_r <= 2'b00;
//               done_r   <= 1;
            end
        FAIL: begin
          next_as_state <= next_as_state;
//               status_r <= 2'b10;
//               done_r   <= 1;
            end

       default: begin
                next_as_state <= IDLE;
//              status_r      <= 2'b11;
//              done_r        <= 0;
            end
    endcase
end

endmodule


module remote_uxexit (
                  link_state,
                  ux_exit_lgen_timer,
                  timeout,
                  link_clk,
                  reset,
                  done,
                  status
                    );

input [3:0] link_state;
input ux_exit_lgen_timer;
input timeout;

// clock
input link_clk;
// reset
input reset;
// status
output done;
output [1:0] status;
```

```
reg [1:0] status_r;
reg done_r;

assign done = done_r;
assign status = status_r;

parameter [3:0] UX= 4'b0001;
parameter [2:0] UX_EXIT_RESP_TX_US=3'b111;
localparam [1:0] IDLE=2'b00,
          LFPS_UXEXIT=2'b01,
          PASS = 2'b10,
          FAIL=2'b11;

reg [1:0] next_as_state;
reg [3:0] link_state_d;
reg ux_exit_lgen_timer_d;

wire entry_ux = !(link_state_d == UX) && (link_state == UX) ;
wire timer_st = !(ux_exit_lgen_timer == UX_EXIT_RESP_TX_US) && (ux_exit_lgen_timer_d ==
UX_EXIT_RESP_TX_US) ;

// generate delayed version of link_state
always @(posedge link_clk) begin
  if(reset)
  begin
    link_state_d <= 0;
    ux_exit_lgen_timer_d <= 0;
  end
  else
  begin
    link_state_d <= link_state;
    ux_exit_lgen_timer_d <= ux_exit_lgen_timer;
  end
end

always @(posedge link_clk)
begin
  if(reset)
  begin
      status_r <= 2'b00;
      done_r   <= 1'b0;
      next_as_state <= IDLE;
  end
  else
    case(next_as_state)
      IDLE:begin
           if(entry_ux)
             next_as_state <= LFPS_UXEXIT;
          end
      LFPS_UXEXIT : begin
                      if( timer_st ) next_as_state <= PASS;
                      else if(timeout) next_as_state <= FAIL;
                    end
      PASS: begin
              status_r <= 2'b00;
              done_r   <= 1;
            end
      FAIL: begin
```

```
                status_r <= 2'b11;
                done_r   <= 1;
            end
        default: begin
                next_as_state <= IDLE;
                status_r      <= 2'b11;
                done_r        <= 0;
            end
  endcase
end
endmodule


module successful_local_ux_exit(
                                link_state,
                                 ux_state,
                                 timeout,
                                 link_clk,
                                 reset,
                                 done,
                                 status
                                );

input [3:0] link_state;
input [1:0] ux_state;
input timeout;

// clock
input link_clk;
// reset
input reset;
// status
output done;
output [1:0] status;
reg [1:0] status_r;
reg done_r;

parameter [3:0]  UX = 1,EXP_STATE=4'b0111;
parameter [1:0] UX_EXIT = 2'b11;


localparam         [2:0] IDLE=3'b000;
localparam          [2:0] WAIT_UXEXIT = 3'b001;
localparam          [2:0] LEAVE_UX = 3'b010;
localparam          [2:0] PASS = 3'b011;
localparam          [2:0] FAIL=3'b100;


reg [2:0] next_as_state;
reg [3:0] link_state_d;

wire entry_ux = !(link_state_d == UX) && (link_state == UX) ;

// generate delayed version of link_state

assign status = status_r;
assign done = done_r;
```

```
always @(posedge link_clk) begin
  if(reset)
  begin
    link_state_d <= 0;
  end
  else
  begin
    link_state_d <= link_state;
  end
end

always @(posedge link_clk)
begin
  if(reset)
  begin
      status_r <= 2'b00;
      done_r   <= 1'b0;
      next_as_state <= IDLE;
  end
  else
  begin
    case(next_as_state)
      IDLE:begin
           if(entry_ux)
             next_as_state <= WAIT_UXEXIT;
          end
      WAIT_UXEXIT: begin
                    if(ux_state== UX_EXIT)
                      next_as_state <= LEAVE_UX;
                end
      LEAVE_UX : begin
                    if(timeout) next_as_state <= FAIL;
                    else if(link_state == EXP_STATE) next_as_state <= PASS;
                end
      PASS: begin
              status_r <= 2'b00;
              done_r   <= 1;
            end
      FAIL: begin
              status_r <= 2'b11;
              done_r   <= 1;
            end
      default: begin
              next_as_state <= IDLE;
              status_r      <= 2'b11;
              done_r        <= 0;
            end
  endcase
 end
end

endmodule


module successful_rem_ux_exit_recovery(
                                link_state,
                                lfps_ux_exit,
                                timeout,
```

```
                                link_clk,
                                reset,
                                done,
                                status
                               );

input [3:0] link_state;
input lfps_ux_exit;
input timeout;

// clock
input link_clk;
// reset
input reset;
// status
output done;
output [1:0] status;
reg [1:0] status_r;
reg done_r;

parameter  [3:0] UX = 1;
parameter  [1:0] UX_EXIT = 2'b11;
parameter  [3:0] RECOV = 4'b111;

localparam [2:0] IDLE=3'b000,
          WAIT_LFPS_EXIT = 3'b001,
          LEAVE_UX = 3'b011,
          PASS = 3'b100,
          FAIL=3'b101;

reg [2:0] next_as_state;
reg [3:0] link_state_d;

wire entry_ux = !(link_state_d == UX) && (link_state == UX) ;

// generate delayed version of link_state
assign status = status_r;
assign done = done_r;
always @(posedge link_clk) begin
  if(reset)
  begin
    link_state_d <= 0;
  end
  else
  begin
    link_state_d <= link_state;
  end
end
always @( posedge link_clk)
begin
  if(reset)
  begin
      status_r <= 2'b00;
      done_r   <= 1'b0;
      next_as_state <= IDLE;
  end
  else
    case(next_as_state)
```

```
        IDLE:begin
             if(entry_ux)
               next_as_state <= WAIT_LFPS_EXIT;
            end
        WAIT_LFPS_EXIT: begin
                        if(lfps_ux_exit)
                          next_as_state <= LEAVE_UX;
                    end
        LEAVE_UX : begin
                        if(timeout) next_as_state <= FAIL;
                        else if(link_state == RECOV) next_as_state <= PASS;
                   end
        PASS: begin
                 status_r <= 2'b00;
                 done_r   <= 1;
              end
        FAIL: begin
                 status_r <= 2'b11;
                 done_r   <= 1;
              end
       default: begin
                 next_as_state <= IDLE;
                 status_r      <= 2'b11;
                 done_r        <= 0;
              end
     endcase
end
endmodule
```

ՀԱՎԵԼՎԱԾ 3

Նկարների ցանկ

Աղյուսակների ցանկ

Հապավումների ցանկ

ԻՍ – ինտեգրալ սխեմա
ԾՏԻՍ – ծրագրավորվող տրամաբանական ինտեգրալ սխեմա
ՄՍ – մտավոր սեփականություն
ԿԲՊՀ – կիսահաղորդչային բազմապրոցեսորային համակարգ
ՎՀ – վերակառուցավորվող հաշվողական
ՀՀԴ – համապիտանի հաջորդական դող
ԹՏՍ – թվաբանական տրամաբանական սարք
ԻԱ – իսկության աղյուսակ
ՍԿԳՀ – ստատիկ կամայական գրանցմամբ հիշասարք
ՖԲ – ֆունկցիոնալ բլոկ
ՄԲ – միջմիացման բլոկ
ՓԲ – փոխանջատման բլոկ
ՄՌՓՄ – միջռեգիստրային փոխանցումների մակարդակ
ՓԹՀՀ – փուլի թակարդմամբ հաճախային համակարգ
ՎՎՄ – վերջավոր վիճակների մեքենա
ԳՀԿՏՌ – գծային հետադարձ կապով տեղաշարժվող ռեգիստր
ՑՀՊԱ – ցածր հաճախային պարբերական ազդանշան
ԲԿԳ – բազմապատկում գումարում կուտակում
ԱԹ – ապարատային թեստ