Uporaba v mikrokrmilnik vgrajenih programirljivih logičnih vezij

Andrej Kenda[1]

[1]*Ime in naslov organizacije prvega avtorja*
E-pošta: ak8655@student.uni-lj.si

Povzetek

FPGA integrirana vezja so v praksi nemalokrat nepogrešljiva, vendar pa je njihova implementacija velikokrat zahtevna. V članku raziskujemo enega izmed odgovorov na to težavo, ki ga ponuja proizvajalec Texas Instruments v svojih mikrokrmilnikih z dodatnim t.i. CLB koprocesorjem. Na grobo je opisana sestava omenjenega koprocesorja, katere razumevanje je bistveno za koriščenje takšnega sistema. Kasneje pa je implementacija predstavljena še na praktičnem primeru simulacije inkrementalnega dajalnika, ki prikaže širšo sliko uporabnosti koprocesorja.

1 Uvod

V prejšnjem članku smo pregledali zgradbo in delovanje sistema CLB (angl. configurable logic block), ki je odgovor proizvajalca Texas Instruments na številne težave pri implementaciji FPGA (angl. field-programable gate array).

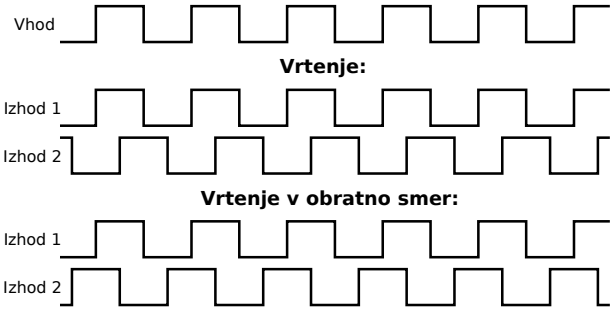
Kot je bilo opisano v prejšnjem članku, za programiranje omenjenega koprocesorja uporabljamo orodje "CLB Tool". To nam preko grafičnega vmesnika omogoča nastavljanje različnih podsklopov ter generira kodo za uporabo v programu. Poleg generirane kode pa nam omogoča vpogled v vse podsklope preko simulacijskih datotek ter dodatnega diagrama povezav.

Primer praktične uporabe CLB koprocesorja je simulacija inkrementalnega dajalnika. Kot je to dejstvo v veliko področjih, je tudi tu možnih več pristopov. Tudi implementacija, opisana v nadaljevanju ni bila očitna že od samega začetka, samo raziskovanje je vključevalo nemalo neuspelih poskusov.

2 Idejna zasnova

Zasnova simuliranega inkrementalnega dajalnika je prikazana na sliki 1, kjer lahko opazimo da je na vhod CLB koprocesorja priključen kvadratni signal s polovičnim delovnim ciklom.

V CLB koprocesorju se vhodni signal uporabi za referenco in "na" njem proizvede dva izhodna signala, ki sta drug drugemu zamaknjena za 90°. Poleg tega pa je potrebno dodati tudi sposobnost za spremembo smeri "vrtenja", kar pomeni da ob določenih pogojih začne "prehitevati" prej počasnejši signal.



Slika 1: Idejna zasnova simuliranega inkrementalnega dajalnika z uporabo CLB koprocesorja

Projekt je zasnovan na predlogi, ki temelji na knjižnicah "Bitfield". Te v grobem vsebujejo le preproste funkcije in poimenovanje in poimenovanje registrov. V času pisanja pa je za delo s CLB koprocesorjem na voljo le "Driverlib", ki vsebuje dokaj bolj kompleksne funkcije.

3 Nastavljanje CLB podkslopov

Zaradi postavitve projekta na knjižnicah "Bitfield" in temu pripadajočega izostanka orodja CLB Tool, je za konfiguracijo najlažja uporaba spletnega orodja "SysConfig", ki vsebuje povsem enak vmesnik, kot ga ima projekt s knjižnicami "Driverlib" [3].

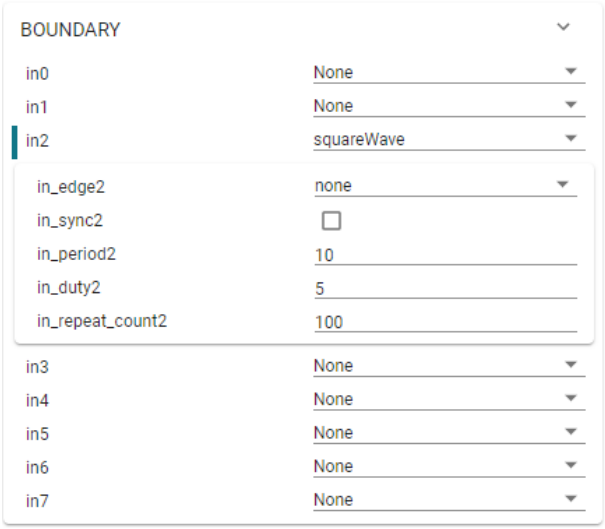
3.1 Splošne nastavitve

Ker CLB obravnavamo kot ločen sistam, je najlažje da se pred komunikacijo z matično napravo poslužujemo simulacije. V ta namen je v orodju "CLB Tool" za modul "BOUNDARY" na voljo možnost simuliranega kvadratnega signala. Za implementacijo inkrementalnega dajalnika je kot simuliran vhod definiran vhod in2, kar prikazuje tudi slika 2.

3.2 Nastavljanje parametrov

Emulacijo inkrementalnega dajalnika smo implementirali na naslednji način:

- Za vir ure/hitrosti smo uporabili eCAP modul v funkciji PWM generatorja s katerim smo generirali 50% PWM signal, ki mu lahko nastavljamo frekvenco. Ta modul smo uporabili saj ima omogoča



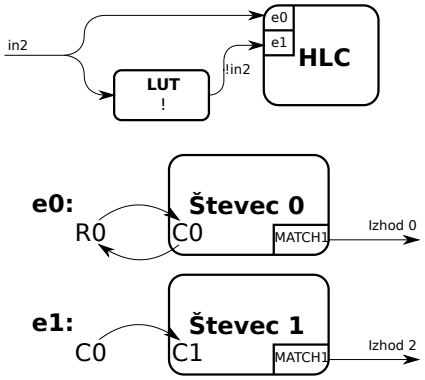
Slika 2: Konfiguracija modula “BOUNDARY” v orodju “CLB Tool”

32 bitno nastavitev periode (najnižja frekvenca = $200MHz/2^{32} \approx 0,046$).

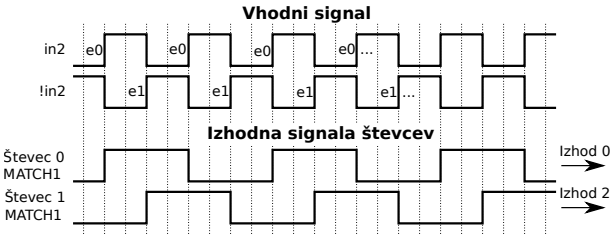
- Preko števca 0 in HLC-ja temu urinemu signalu prepolovimo frekvenco.
- Preko števca 1 in HLC-ja generiramo fazno zamaknjen signal s prepolovljeno frekvenco. Dodatno lahko ta signal invertiramo.

Zasnova inkrementalnega dajalnika je prikazana na sliki 3 in uporablja 1 LUT4 modul, 2 števca in HLC modul. Implemetirana je na naslednjih korakih, ki se navezujejo na poteke signalov s slike 3 in 4.

- Vhod “in2” je direktno in negirano preko LUT4 modula speljan v vhoda “e0” in “e1” HLC modula (vhoda za proženje dogodkov).
- V HLC modulu se dogodki prožijo ob pozitivnem robu prožilnega signala, torej se pripadajoč dogodek sproži na vsako periodo signala “in2” (s 180° zamikom).
- HLC modul operira z vrednostmi “R0”, “R1”, “C0” in “C1”, pri tem je začetna vrednost “R0” nastavljena na 1.
- Oba števca sta nastavljena, da ne štejeta, nimata vhodnih signalov, le vrednost “MATCH1” je nastavljena na 1 - uporabljena sta le kot komparatorja.
- Ob dogodku “e0” HLC modul zamenja vrednosti “C0” in “R0”, pri tem se izvršijo naslednji ukazi:
  - **MOV C0, R1** - premakni vrednost C0 v R1,
  - **MOV R0, C0** - premakni vrednost R0 v C0,
  - **MOV R1, R0** - premakni vrednost R1 v R0,
- Pri tem se ob vsakem pozitivnem robu vhodnega signala “e0” zamenja polariteta izhodnega signala “MATCH1”, števca 0 - perioda signala je dvakrat večja od vhodnega.
- Ob dogodku “e1”, ki se izvrši s 180° zamikom glede na “e0”, se v števec 1 vpiše vrednost števca 0 - “MATCH1” števca 1 je tako enak signalu “MATCH1” števca 0, le da je zakasnjen za četrtino njegove periode. Pri tem se izvrši ukaz:
  - **MOV C0, C1** - premakni vrednost C0 v R1,
- Signala “MATCH1” obeh števcev sta speljana na izhod.



Slika 3: Shema boljše zasnove inkrementalnega dajalnika s pomočjo CLB koprocesorja



Slika 4: Potek signalov boljše zasnove inkrementalnega dajalnika s slike 3

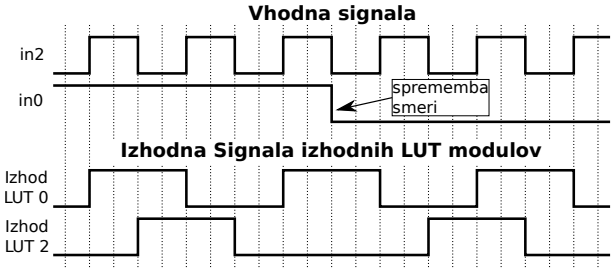
3.3 Obračanje smeri inkrementalnega dajalnika

V drugem primeru iz poglavja 3.2 smo parametre nastavili tako, da je signal “MATCH1” iz števca 1 vedno zakasnjen glede na signal “MATCH1” iz števca 0. Zato je potrebno dodati tudi logiko za obračanje smeri simuliranega inkrementalnega dajalnika, saj želimo simulacijo obeh smeri vrtenja. Za to uporabimo dodaten vhod “in0” v “BOUNDARY” modulu. Tega prav tako nastavimo na simuliran kvadraten signal, le da mu periodo občutno povečamo (pribl. 1O-kratnik periode signala “in2”).

Simulirani signal obravnavamo kot zunanjo tipko, ki v normalnem stanju zavzema logično 1, ob pritisku pa logično 0. To uporabimo pri izhodu iz CLB koprocesorja, kjer je signal potrebno speljati preko izhodnega LUT3 modula;

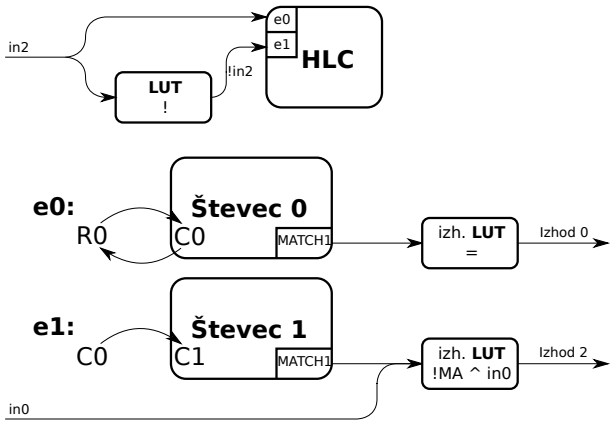
- Signal “MATCH1” števca 0 je speljan preko izhodnega LUT3 modula 0, pri tem se ne spremeni.
- Signal “MATCH1” števca 1 pa je speljan preko izhodnega LUT3 modula 2. Pri tem se njegova negirana vrednost z logično operacijo “XOR” primerja s signalom “in0”.

Kot prikazuje slika 5, je signal izhodnega LUT3 modula 0 vedno enak, signal izhodnega LUT3 modula 2 pa se spreminja glede na stanje signala “in0”. Pri tem prejšnji signal glede na to stanje prehiteva oz. za njim zaostaja.



Slika 5: Prikaz izhodnih signalov iz CLB koprocesorja pri simulaciji inkrementalnega dajalnika, glede na simulirane vhodne signale

Celotna shema končne implementacije je prikazana na sliki 6.



Slika 6: Celotna shema končne implementacije simuliranega inkrementalnega dajalnika z uporabo CLB koprocesorja

4 Vhodni signali v CLB

vhodni signali v poglavju 3 so bili za čas nastavljanja podsklopov zaradi enostavnejšega dela le simulirani. Za delovanje pa je potrebno signal speljati iz enega od modulov na integriranem vezju. Kot je razbrati iz poglavja 3.2, potrebujemo kvadratni vhodni signal z nastavljivo frekvenco ter signal za obračanje smeri, ki bo v naravnem stanju prevzel logično 1, ob pritisku pa logično 0.

4.1 Generiranje kvadratnega signala

Za vhodni signal v CLB je uporabljen izhod iz “eCAP” modula, ki je v osnovi števec. Ta odločitev je nastala zgolj zaradi 32-bitnega števca v omenjenem modulu, kar omogoča bolj natančno določitev periode. Modul preko pulzno-širinske modulacije generira kvadratni signal s polovičnim delovnim ciklom. Implementacija funkcije za usposobitev “eCAP” modula v načinu “apwm” je prikazana na seznamu 1.

```
void ECAP_apwm_init(float freq, float duty)
{
    /* Setup APWM mode on CAP1, */
    /* set period and compare registers */

    /* Enable APWM mode */
    ECap1Regs.ECCTL2.bit.CAP_APWM = 1;
    /* Set Period value */
    ECap1Regs.CAP1 = CPU_FREQ / freq;
    ECap1Regs.CAP3 = CPU_FREQ / freq;
    /* Set Compare value */
    ECap1Regs.CAP2 = ECap1Regs.CAP1 * duty;
    ECap1Regs.CAP4 = ECap1Regs.CAP3 * duty;

    /* Initialize GPIO pins for APWM1. */
    InitAPwm1Gpio();
}
```

Seznam 1: Implementacija funkcije za usposobitev “eCAP” modula v “apwm” načinu

Opazimo lahko, da je za usposobitev potrebno določiti le dolžino periode in primerjalne vrednosti. Slednja je namenjena preklopu iz visokega v nizko stanje pri generiranju “PWM” signala. Preko te funkcije tako generiramo signal z določitvijo frekvence in delovnega cikla (“freq” in “duty”) Za boljši vpogled v sistem je preko “GPIO” modula ta signal speljan tudi iz naprave, kjer ga lahko opazujemo z osciloskopom ali logičnim analizatorjem.

Ta signal je nato priključen na lokalni vhod 2 CLB koprocesorja, ter opravlja vlogo prej simuliranega vhoda “in2”.

4.2 Signal za obračanje smeri

Za obračanje smeri je bil izbran signal “GPIO24”. Ta je nastavljen kot vhod z “pull-up”, kar pomeni, da bo v odprtem položaju zavzel logično 1. Preko vhodnega “X-bar”-a in globalnega vodila je priključen na vhod 0 CLB koprocesorja in s tem prevzema mesto prej simuliranega signala “in0”.

5 Delovanje inkrementalnega dajalnika

Za lažjo uporabo simuliranega inkrementalnega dajalnika so vse nastavitve, ki niso del orodja “CLB Tool” zapakirane v funkciji “CLB1_init”. Za končnega uporabnika pa je vse skupaj razkrito v funkciji, prikazani na seznamu 2

```
void InitEncoder(float frequency)
{
    /* Initialize ECAP in APWM mode */
    /* with 1/2 duty cycle. */
    ECAP_apwm_init(frequency*2, 0.5);

    /* Initialize the CLB1 block. */
    CLB1_init(CLB1_BASE);
}
```

Seznam 2: Implementacija funkcije za nastavitev inkrementalnega dajalnika

Funkciji nastavimo frekvenco inkrementalnega dajalnika in preko tega se nastavi “eCAP” modul z dvokratnikom te frekvence. Ta ima seveda vlogo vhodnega signala v CLB koprocesor. S seznama 2 lahko vidimo tudi klic prej omenjene funkcije “CLB1_init”, ki skrbi za nastavitev vseh ostalih parametrov CLB koprocesorja.

6 Zaključek

Iz praktičnega primera lahko razberemo, da je CLB koprocesor uporaben dodatek za marsikatero aplikacijo, kjer bi z njim lahko nadomestili FPGA čip. V grobem lahko rečemo, da trditve proizvajalca (ob ignoriranju tržnih izmišljotink seveda) v veliki meri držijo. CLB lahko marsikje nadomesti FPGA, je lažji za programiranje, ter je seveda direktno na samem integriranem vezju, kar med drugim v veliko primerih pomeni tudi nižjo ceno.

Grafični vmesnik "CLB Tool" zaradi svoje narave zahteva veliko potrpežljivosti. Če je to težava pri tako majhni aplikaciji, si je lahko predstavljati težavnost pri obsežnih projektih.

CLB pa žari pri majhnih projektih, kjer je bistvenega pomena hitra implementacija ter se manipulira z manjšim številom signalov. Prav tako je vsakemu programerju omogočena enostavna uporaba, brez da bi se popolnoma posvečal takšnim vezjim.

7 Dodatno branje

Več informacij o sami implementaciji CLB je opisanih v [1]. Za migracijo obstoječih FPGA projektov na CLB pa je precej dober uvod vir [2].

Literatura

[1] Nima Eskandari. *Designing With the C2000™ Configurable Logic Block (CLB)*. SPRACL3. ZDA, avg. 2019. URL: https://www.ti.com/lit/an/spracl3/spracl3.pdf?ts=1608544448925&ref_url=https%253A%252F%252Ftraining.ti.com%252F.

[2] Peter Galicki. *How to Migrate Custom Logic From an FPGA/CPLD to C2000™ Microcontrollers*. SPRACO2A. Rev. 2020-7. Texas Instruments Inc. ZDA, sep. 2019. URL: https://www.ti.com/lit/an/spraco2a/spraco2a.pdf?ts=1618290495856&ref_url=https%253A%252F%252Ftraining.ti.com%252F.

[3] *SysConfig*. Texas Instruments Inc. 2021. URL: https://dev.ti.com/sysconfig/.