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

Pri oblikovanju vdelanih sistemov se zaradi preprostega programiranja in cenovne ugodnosti najpogosteje poslužujemo mikrokrmilnikov različnih proizvajalcev. Na začetku (včasih "celo" na sredini) oblikovalnega procesa, v včasih preveliki ponudbi, izberemo čip, ki najbolje ustreza našim zahtevam.

Obstajajo pa tudi aplikacije, ki jim konvencionalni mikrokrmilniki niso kos. Tu nastopi iskanje drugačnih rešitev. Če imamo srečo, morda odkrijemo kakšen ASIC (angl. application specific integrated circuit), ki zadošča našim potrebam, vendar je na določenih področjih teh dokaj malo. V določenih aplikacijah nam ne preostane drugega oz. za boljšo rešitev odkrijemo FPGA (angl. field-programable gate array). Ti nam omogočajo postavitev sistema na nizkem nivoju, ki ga lahko prikrojimo natanko našim zahtevam. FPGA v veliko aplikacijah namestimo poleg prej omenjenega mikrokrmilnika in ga uporabimo kot dodaten procesor.

Kljub temu, da so FPGA čipi precej zmogljivi, pa nastopi težava pri sami izvedbi. Kadra, ki ima globoko znanje takšnih sistemov je zaradi zahtevnosti precej malo. Poleg tega pa uporaba dveh čipov za pri številnih aplikacijah nemalokrat ni idealna, saj ti čipi dostikrat zahtevajo uporabo zunanjega RAM in Flash spomina, za komunikacijo s katerima smo odgovorni sami. Razne komunikacije med posameznimi modulil običajno terjajo največ razvojnega časa, hkrati pa je poleg FPGA ponavadi prisoten tudi mikrokrmilnik, ki je hkrati dodatna finančna obremenitev in še ena linija komunikacije za implementacijo.

Na naštete težave je proizvajalec Texas Instruments odgovoril s koprocesorjem CLB (angl. configurable logic block), ki je vdelan v nekaj njihovih mikrokrmilnikov. Ta naj bi po proizvajalčevih trditvah tako nadomestil dodaten FPGA (in s tem razvijalcu prihranil marsikatero uro), kot tudi omogočil programiranje s preprostim vmesnikom [1].

2 Predstavitev CLB

Slika 1 prikazuje shematski prikaz komponent v CLB koprocesorju, ki je del mikrokrmilnika F28379D, proizvajalca Texas Instruments. Na sliki je razvidno, da je omenjeni CLB sestavljen iz štirih med seboj enakih podsklopov (angl. Tile). Vsak podsklop je sestavljen iz procesorja (angl. CELL) ter vmesnika za "priklop" signalov iz matične naprave (angl. CPU I/F).

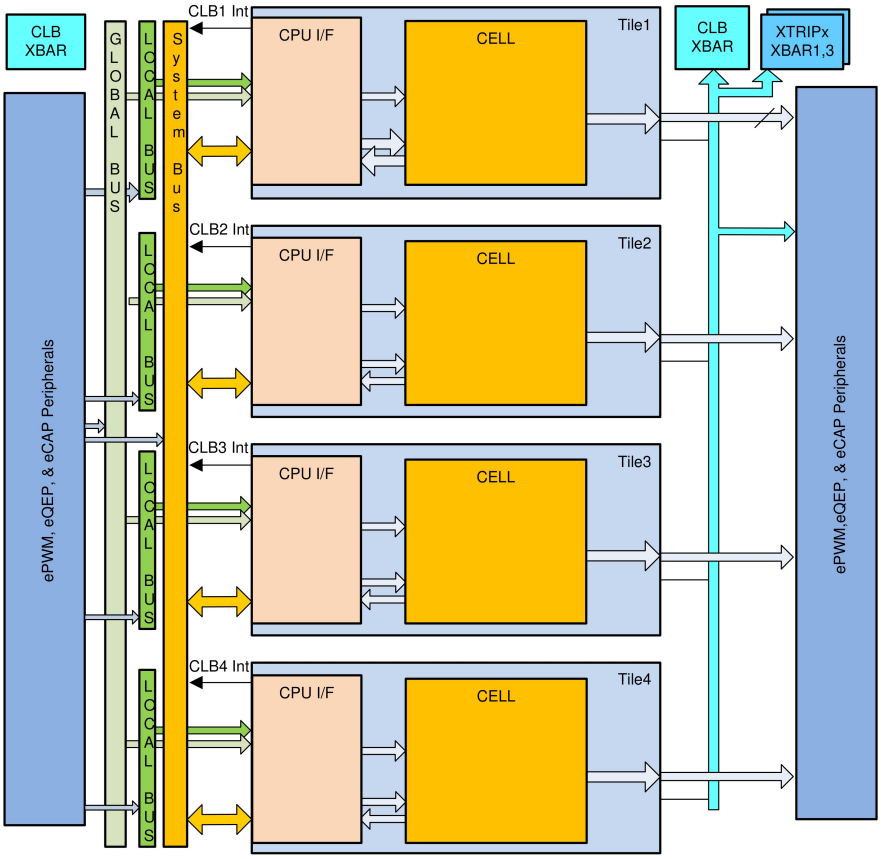
V CLB lahko vhodne signale pripeljemo preko treh vodil, lokalnega, globalnega in sistemskega vodila. Teh se za vhode v CLB poslužujemo glede na samo postavitev ostalih dodatkov, ki so poleg CLB-ja prisotni na čipu [6, Pogl. 26.3]. Posebnost omenjenega mikrokrmilnika je tudi prisotnost t.i. "crossbar"-jev, ki omogočajo dovod vhodnih signalov preko lokalnega in globalnega vodila, ter odvod signalov iz CLB-ja. Ti signali lahko izvirajo iz različnih dodatkov, kot so eCAP, ePWM, GPIO... iz CLB-ja pa lahko v prav te dodatke tudi "pripeljemo" izhodni signal iz CLB-ja, kot vhod.

Interakcija pa ni omejena le z dodatnimi napravami in matičnim procesorjem, temveč je mogoča tudi med različnimi podsklopi CLB-ja. Možno pa je tudi proženje prekinitev, na katere lahko procesor ustrezno reagira.

3 Zgradba CLB modulov

V poglavju 2 je omeneno, da je sam CLB sestavljen iz štirih podsklopov. Na sliki 2 je videti, da vsak podsklop vsebuje [6, Pogl. 26.4]:

- 3 štirivhodne LUT4 (angl. 4-input lookup table),
- 8 trivhodnih izhodnih LUT3 (angl. 3-input lookup table),
- 3 števce,
- 3 FSM (angl. finite state machine),
- 1 HLC (angl. high level controller) in
- 1 nastavljiv preklopni blok.



Slika 1: Shematski prikaz komponent CLB koprocesorja [6, Pogl. 26.2]

3.1 LUT moduli

LUT4 modul omogoča modulacijo signala preko logičnih operacij. Vhodne signale (slika 3 - "IN0"-"IN3") lahko preko enačbe z "AND", "OR", "NOT" in "XOR" operacijami odvedemo v izhodni signal [2, Pogl. 3.3].

Razlika med "LUT4" in izhodnim "LUT3" modulom je le, da ima slednji tri namesto štirih vhodov, ter je njegov izhod vezan direktno na izhod CLB-ja, kar je tudi ena izmed njegovih funkcij [6, Pogl. 26.4.4-26.4.5].

3.2 Števci

Števec je precej kompleksen modul v CLB-ju, ki ga lahko nastavimo za delovanje kot števec, seštevalnik, komparator ali operator zamika. S slike 4 je moč razbrati štiri funkcijske vhode; "RESET", "MODE 0", "MODE 1" in "EVENT". Prvi od teh vhodnih signalov ob visoki vrednosti števec postavi na vrednost 0, "MODE 0" ob visoki vrednosti omogoči štetje, "MODE 1" nastavi smer štetja (visoko - navzgor, nizko - navzdol), signal "EVENT" pa ob pozitivnem robu sproži dogodek, na katerega ta odreagira z nastavljeno računsko operacijo, katere parametre nastavimo z drugima signaloma (slika 4 - "Static controls" in "LOAD VALUE"). Do vsebine števca lahko dostopamo preko HLC modula (pogl. 3.4), kjer je ta označena z zaporedno številko števca (npr. števec 1 - C1).

Kot izhod iz števca lahko spremljamo signale "ZERO", "MATCH1" in "MATCH2". Prvi prevzame visoko vrednost kadar je vrednost števca 0, ostala dva pa se vklopita, kadar je vrednost števca enaka nastavljeni pripadajoči vrednosti. Ko števec doseže svojo maksimalno vrednost (2^{32}) ta "prelije" in začne šteti od 0. Temu obnašanju se ponavadi želimo izogniti [6, Pogl. 26.4.2].

3.3 FSM moduli

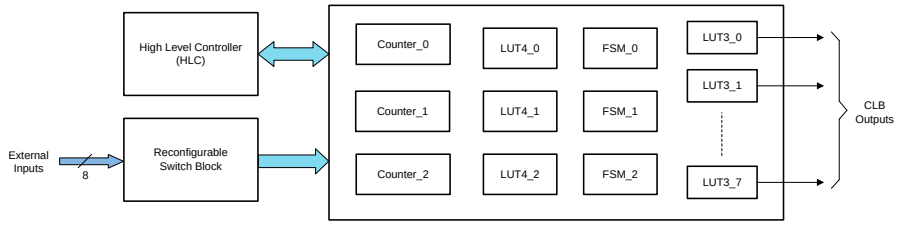
FSM modul omogoča implementacijo avtomata stanj. Le to se izvede preko dveh internih LUT blokov; "Output LUT", "S0 Next State LUT" in "S1 Next State LUT" (slika 5), te so po zgradbi povsem enake opisanim v poglavju 3.1. Za implementacijo avtomata stanj uporabljamo le slednji LUT tabeli, ki služita interni modulaciji signalov in operirata z zunanjima signaloma "EXT_IN0", "EXT_IN1" in vhodoma "S0" in "S1", ki prevzameta prejšnjo vrednost izhoda pripadajoče tabele.

Če nimamo potrebe po implemetaciji avtomata stanj, se lahko poslužimo tudi "Output LUT", in modul uporabimo kot običajno LUT tabelo (izhod "FSM_LUT_OUT").

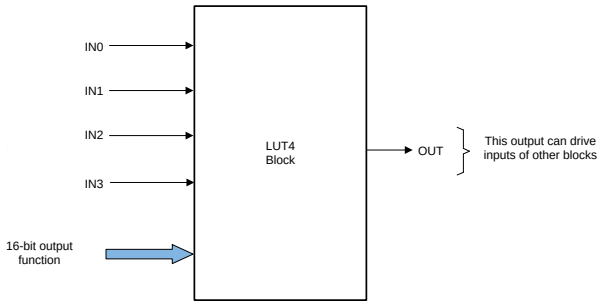
Če aplikacija ne potrebuje uporabe FSM modula, se lahko poslužimo še vhodov "EXTRA_EXT_IN0" in "EXTRA_EXT_IN1". V tem primeru celoten modul deluje kot štirivhodni LUT modul.

3.4 HLC modul

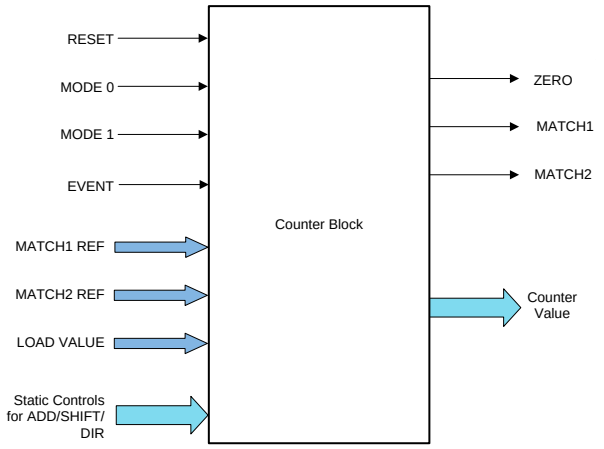
HLC modul je za razliko od ostale naprave veliko bolj kompleksen. S slike 6 je iz bloka za procesiranje dogod-



Slika 2: Shematski prikaz komponent CLB koprocesorja [6, Pogl. 26.2]



Slika 3: Shematski prikaz štirivhodnega LUT4 modula CLB koprocesorja [6, Pogl. 26.4.4]



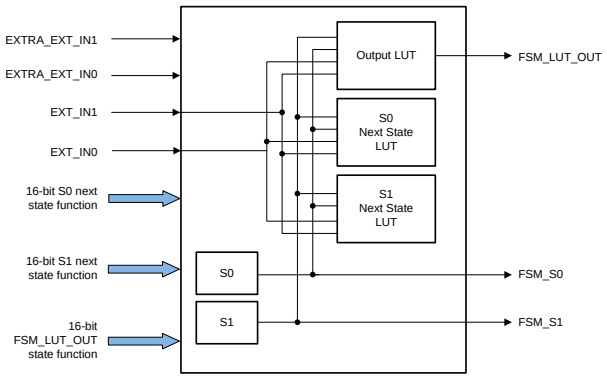
Slika 4: Shematski prikaz števca CLB koprocesorja [6, Pogl. 26.4.2.1]

kov in štirih registrov, ki služijo za shranjevanje spremenljivk “R0”-“R3”[6, Pogl. 26.4.6].

Modul lahko direktno upravljamo tudi z matičnim procesorjem, s katerim modul lahko komunicira tudi preko “push-pull” metode zapisovanja v skupni spomin.

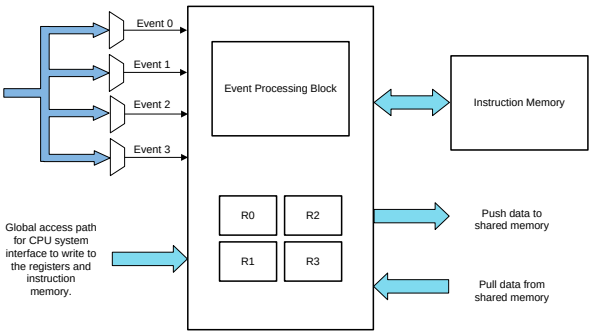
Operiramo lahko s prej omenjenimi spremenljivkami “R0”-“R3”, z vrednostmi števcev “C1”-“C2” in z obemi “MATCH” vrednostmi v števcih. Za proženje dogodkov ima modul štiri vhodne signale. Vsak od teh sproži sebi pripadajoč program, zapisan v instrukcijskem spominu. Ta lahko na omenjenih spremenljivkah izvede naslednje operacije [6, Pogl. 26.4.6.2]:

- ADD/SUB - seštevanje in odštevanje,
- MOV/MOV_T1/MOV_T2 - premikanje,



Slika 5: Shematski prikaz FSM modula CLB koprocesorja [6, Pogl. 26.4.3]

- PUSH/PULL - pisanje in branje v skupnem spominu,
- INTR - proženje prekinitev.



Slika 6: Shematski prikaz HLC modula CLB koprocesorja [6, Pogl. 26.4.6]

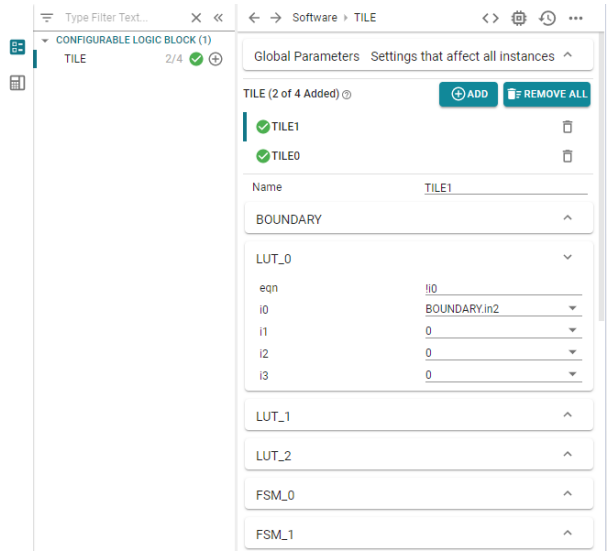
3.5 Preklopni modul

Ta modul je namenjen izbiri vhodnih signalov, ki lahko izvirajo iz zunanjih virov, drugih CLB podkslopov (angl. tile) ali pa signal generira modul sam. Zadnji vir se uporablja le za simulacijske namene. Modul je v orodju “CLB Tool” (Pogl. 4) zaradi postavitve predstavljen pod imenom “BOUNDARY” [2, Pogl. 3.3].

4 Uporaba CLB

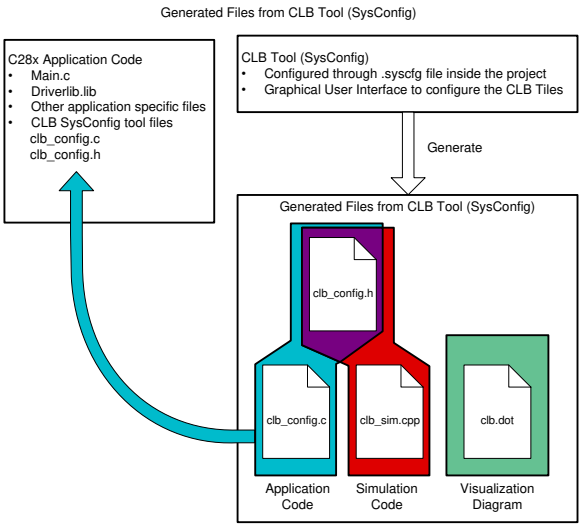
Za samo programiranje CLB koprocesorja v praksi je proizvajalec Texas Instruments postavil grafični vmesnik

"CLB Tool" (slika 7), ki nam preko izbire določenih parametrov zgenerira kodo za uporabo v našem projektu.



Slika 7: Grafično okolje "CLB Tool"

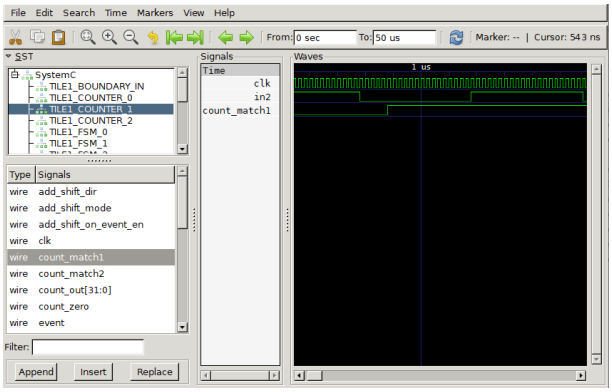
Grafični vmesnik za vsako od komponent, naštetih v poglavju 3, dovoljuje izbiro opisanih parametrov, vhodnih in izhodnih signalov ter splošno konfiguracijo modulov. Kot je prikazano na sliki 8, orodje generira 2 aplikacijski datoteki; "clb_config.h" ter "clb_config.c". Ti datoteki lahko seveda brez težav vključimo v naš "C" program.



Slika 8: Struktura projekta pri uporabi orodja "CLB Tool" [2, Pogl. 1]

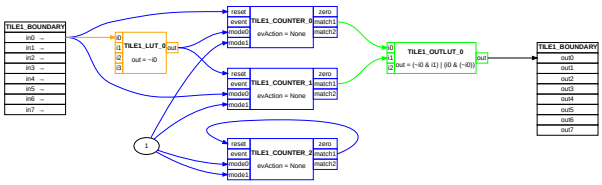
Poleg aplikacijske kode, pa nam orodje dovoljuje tudi generiranje simulacijskih datotek. Te se na koncu prevedejo v datoteko "CLB.vcd", ki omogoča vpogled v notranje signale CLB preko zunajega programa za prikazovanje grafov (slika 9).

Orodje avtomatsko generira tudi datoteko za grafični pregled povezav v koprocesorju. Ta se prevede v obliko ".html", ki jo lahko odpremo v vsakem brskalniku (slika



Slika 9: Pregled notranjih signalov CLB-ja s programom "GTK Wave"

10)[2, Pogl. 1].

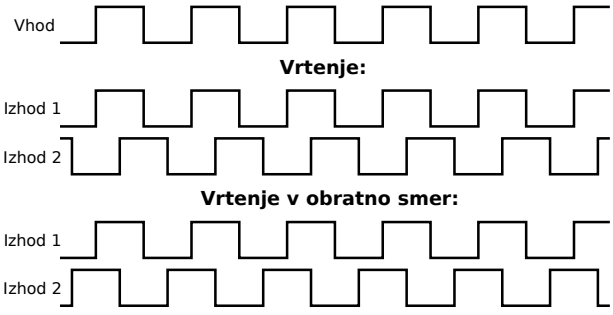


Slika 10: Primer generiranega ".html" diagrama

5 Praktičen primer

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. Naša idejna zasnova je prikazana na sliki 11, 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 11: Idejna zasnova simuliranega inkrementalnega dajalnika z uporabo CLB koprocesorja

Projekt je zasnovan na predlogi, ki temelji na

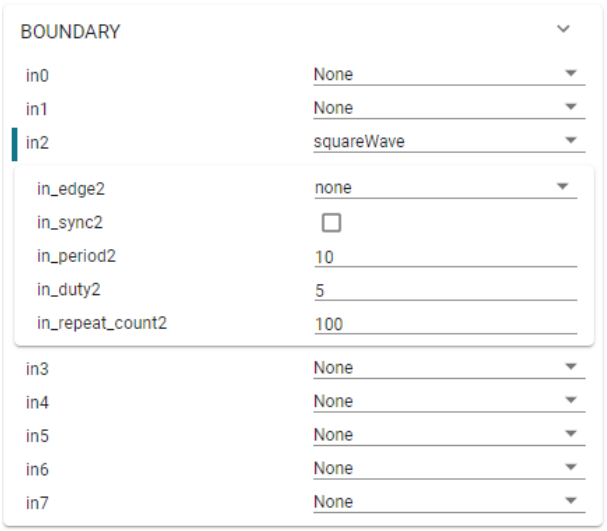
knjižnicah “Bitfield”. Te v grobem vsebujejo le preproste funkcije in poimenovanje in poimenovanje registrov. V času pisanja pa je za delo s CLB koprocesorjem na voljo le “Driverlib”, ki vsebuje dokaj bolj kompleksne funkcije.

5.1 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” [5].

5.1.1 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 (Pogl. 4). Za implementacijo inkrementalnega dajalnika je kot simuliran vhod definiran vhod in2, kar prikazuje tudi slika 12.



Slika 12: Konfiguracija modula “BOUNDARY” v orodju “CLB Tool”

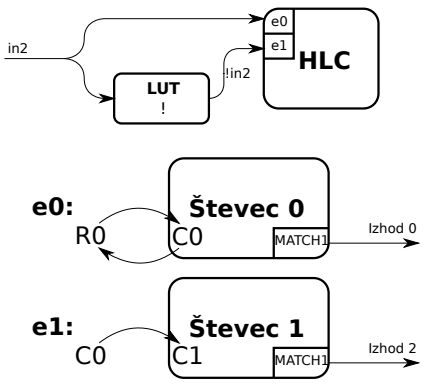
5.1.2 Nastavljanje parametrov

Emulacijo inkrementalnega dajalnika smo implementirali na naslednji način:

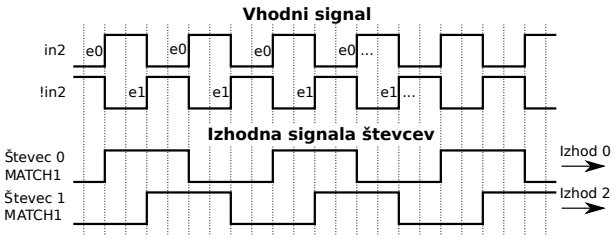
- Za vir ure/hitrosti smo uporabili eCAP modul v funkciji PWM generatorja s katerim smo generirali 50% PWM signal, ki mu lahko nastavljamo frekvenco. Ta modul smo uporabili saj ima omogoča 32 bitno nastavitev periode (najnižja frekvenca = $200MHz/2^{32} \approx 0,046$).
- Preko števca 0 in HLC-ja temu 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 13 in uporablja 1 LUT4 modul, 2 števca in HLC modul. Implemetirana je na naslednjih korakih, ki se navezujejo na poteke signalov s slike 13 in 14.

- 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” (Pogl. 3.4), 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 13: Shema boljše zasnove inkrementalnega dajalnika s pomočjo CLB koprocesorja



Slika 14: Potek signalov boljše zasnove inkrementalnega dajalnika s slike 13

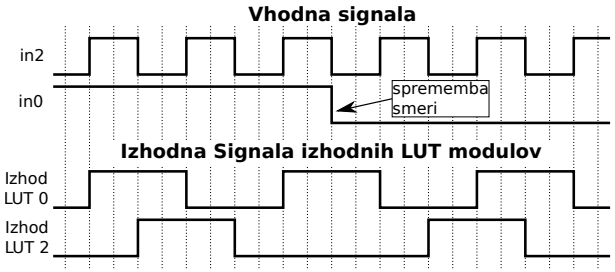
5.1.3 Obračanje smeri inkrementalnega dajalnika

V drugem primeru iz poglavja 5.1.2 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 15, 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.

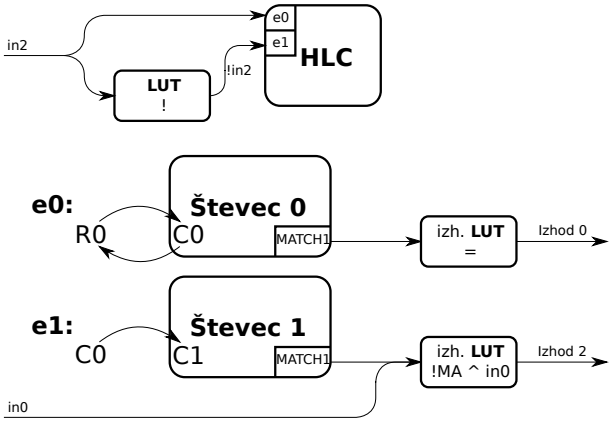


Slika 15: Prikaz izhodnih signalov iz CLB koprocesorja pri simulaciji inkrementalnega dajalnika, glede na simulirane vhodne signale

Celotna shema končne implementacije je prikazana na sliki 16.

5.2 Vhodni signali v CLB

vhodni signali v poglavju 5.1 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



Slika 16: Celotna shema končne implementacije simuliranega inkrementalnega dajalnika z uporabo CLB koprocesorja

5.1.2, 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.

5.2.1 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".

5.2.2 *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".

5.3 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

Že 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.

Vendar pa tudi CLB, kot vsaka stvar ne pride brez pomankljivosti. Skoraj vsaka "izboljšava" nad FPGA s seboj prinese tudi kakšno frustracijo za programerja; grafični vmesnih "CLB Tool" je na prvi pogled res prijazen, a se zaradi velike abstrakcije kaj hitro izgubimo med neštetimi nastavitvenimi meniji. Lahko si predstavljamo, da bi ob obsežnejših projektih bilo že zaradi nekakšne standardizacije bolj smiselno uporabiti FPGA.

Vendar pa je kljub pomankljivostim tovrstna izvedba dodatnega koprocesorja zelo smiselna. Znanje programiranja FPGA vezij je precej specifično, zato je kadra, ki bi imel visoko znanje o njih bore malo. Poleg tega pa je s stališča delodajalcev velikokrat bolj smiselno uporabiti rešitev, ki jo lahko implementira obstoječ kader, kot pa iskanje znanja izven podjetja.

7 Dodatno branje

Več informacij o sami implementaciji CLB je opisanih v [3]. Za migracijo obstoječih FPGA projektov na CLB pa je precej dober uvod vir [4].

Literatura

[1] *C2000™ Configurable Logic Block (CLB) introduction*. Texas Instruments Inc. 2021. URL: https://training.ti.com/c2000-configurable-logic-block-clb-introduction.

[2] *CLB Tool*. SPRUIR8A. Rev. A. Texas Instruments Inc. ZDA, sep. 2019. URL: https://www.ti.com/lit/ug/spruir8a/spruir8a.pdf?ts=1608498444631&ref_url=https%253A%252F%252Ftraining.ti.com%252Fclb-training-c2000-mcus.

[3] 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.

[4] 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.

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

[6] *TMS320F2837xD Dual-Core Microcontrollers Technical Reference Manual*. SPRUHM8I. Rev. 2019-9. Texas Instruments Inc. ZDA, dec. 2013. URL: https://www.ti.com/lit/ug/spruhm8i/spruhm8i.pdf?ts=1616431731212&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FTMS320F28377D.