16-Bitni procesor s memory mapped UART sučeljem

Dokumentacija

Matija Žagar 29.9.2021

Sadržaj

1. UVOD

Dizajnirani sklop je 16-bitni procesor s pristupom UART („Universal Asynchronous Receiver-Transmitter“) modulu za primanje i odašiljanje podataka preko UART protokola. Korištena je Harvardska arhitektura pa stoga procesor sadrži odvojene memorije za podatke i instrukcije. Procesor se programira korištenjem asemblerskog jezika čije instrukcije obavljaju sve zadatke potrebne za upravljanje registrima, izračunima te memorijom. UART modulu se pristupa kao i normalnoj memoriji s naznakom da se on nalazi na virtualnoj adresi s koje učitavamo primljene ili pišemo podatke za odašiljanje UART protokolom.

2. Pregled projekta

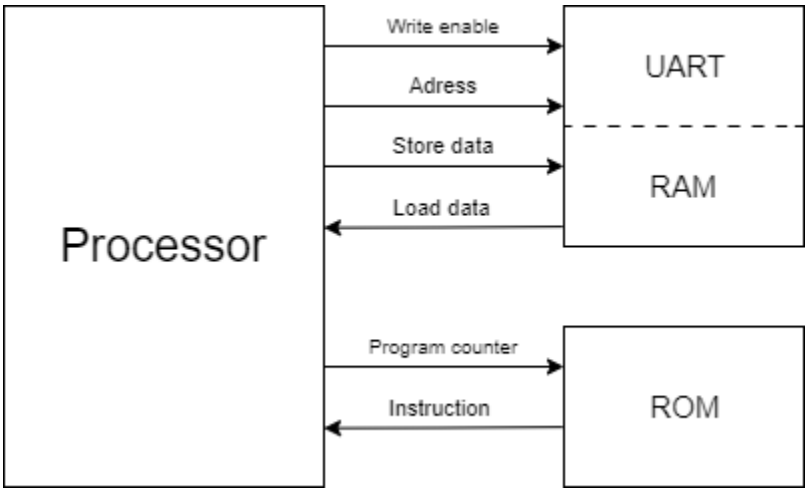
Projekt se sastoji od 4 glavne komponente: procesora ili središnje upravljačke jedinice (CPU), ROM memorije, RAM memorije te UART modula koji su povezani kao na slici 1.

ROM memorija služi za spremanje 15-bitnih instrukcija te iz njega CPU čita u ovisnosti o brojaču instrukcije (program counter). Jedini način pisanja u ovu memoriju je u VHDL modulu prije sinteze projekta jer sami procesor nema mogućnost spremanja novih instrukcija.

RAM memorija se koristi za spremanje podataka važnih programu koji se ponekad koriste jer za pristup njima treba utrošiti barem jedan procesorski ciklus.

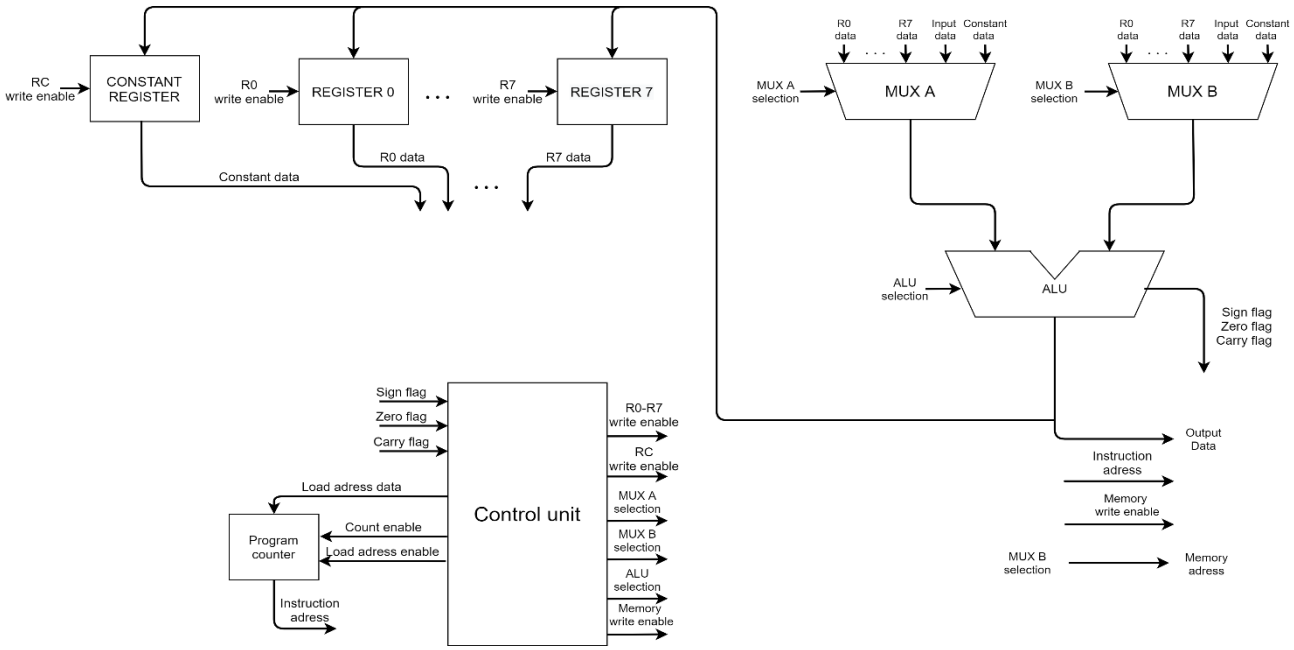
UART modul je procesoru način komunikacije sa bilo kojim vanjskim uređajem koji može odašiljati signal UART protokolom te omogućava širi raspon mogućnosti i interaktivnosti ovog projekta.

CPU ili procesor je glavni upravljački dio ovog projekta koji obrađuje instrukcije iz ROM-a te kontrolira UART modul



Slika 1. Shematski prikaz dizajniranog sklopa

2.1. Središnja upravljačka jedinica (CPU)



Slika 2.

Središnja upravljačka jedinica (CPU) je dio ovog sklopa koji upravlja izračunima i pisanjem po memoriji te čitanjem iste. U svakom procesorskom ciklusu pročita se jedna 15-bitna instrukcija iz ROM-a te se nakon toga izvršava. Izvršavanje instrukcije počinje izborom operanda pomoću multipleksera (MUX) A i B. Izlazi iz multipleksera su ulazi aritmetičko logičke jedinice (ALU). ALU izvršava operaciju navedenu u instrukciji i formira rezultat te statusne bitove ili zastavice (flag) koji opisuju rezultat. Zastavice označavaju da je rezultat nula (ZERO zastavica), da je rezultat negativan (SIGN zastavica) ili da je rezultat imao prijenos bit (CARRY zastavica). Nakon toga se rezultat prosljeđuje registrima opće namjene, registru za konstante, programskom brojaču te na izlaz procesora. CPU prema kodu instrukcije šalje signale dozvole zapisivanja rezultata. Na izlaz procesora se prosljeđuje 16-bitni izlazni podatak prema memoriji za podatke, 16-bitna adresa memorije za podatke, 16-bitna vrijednost programskog brojača kao adresa memorije za instrukcije te signal dozvole upisa u memoriju.

2.2. Skup procesorskih instrukcija

Dostupne procesorske instrukcije se dijele na aritmetičke instrukcije, instrukcije skoka te instrukcije za čitanje i pisanje po memoriji. S obzirom na to da je za opis sklopa korišten jezik za opis sklopovlja VHDL, instrukcije su definirane kao simboličke konstante radi lakše uporabe. Na taj način primjer instrukcije u ROM-u ima sljedeći oblik:

```
mov &R0 &R1 &N3
```

Znak '&' se koristi za konkatenaciju konstanti koje mogu biti instrukcije kao što su na primjer „mov“, „jmp“, „ld“ i druge. Konstante su također nazivi registara opće namjene numerirani od 0 do 7, oni bi se navodili na sljedeći način: „R0“, „R5“ itd. Koristi se još jedna posebna konstanta nazvana „N3“ koja se koristi da naznači da se taj dio instrukcije ne koristi, ali ti bitovi moraju postojati kako bi instrukcija bila zadane duljine. U instrukcijama skoka koristimo konstantne brojeve koji moraju biti duljine 9 bitova tj. 3 oktalna znaka.

2.2.1. Aritmetičke instrukcije

Aritmetičke instrukcije direktno računa aritmetičko logička jedinica. Kod instrukcije je sljedeći:

```
I I I I I I  Z Z Z  X X X  Y Y Y
```

Gdje slova 'I' predstavljaju 6-bitni kod instrukcije, a slova 'Z', 'X' i 'Y' predstavljaju adrese registara opće namjene. U svim instrukcijama se rezultat sprema u registar 'Z'. Registri 'X' i 'Y' se ponašaju kao ulazi (operandi). Ukoliko instrukcija koristi samo jedan operand, drugi operand se može definirati proizvoljno. Sintaksa „RZ <- [RX]“ označava da se sadržaj registra 'X' upisuje u registar 'Z'.

Tablica 1. Aritmetičke instrukcije

Operacija	Funkcija	Kod instrukcije
`mov Rz, Rx`	RZ <- [RX]	000000
`add Rz, Rx, Ry`	RZ <- [RX] + [RX]	000001
`sub Rz, Rx, Ry`	RZ <- [RX] - [RX]	000010
`and1 Rz, Rx, Ry`	RZ <- [RX] & [RX]	000011
`or1 Rz, Rx, Ry`	RZ <- [RX] | [RX]	000100
`not1 Rz, Rx`	RZ <- not [RX]	000101
`inc Rz, Rx`	RZ <- [RX] + 1	000110
`dec Rz, Rx`	RZ <- [RX] - 1	000111
`shl1 Rz, Rx`	RZ <- shl [RX]	001000
`shr1 Rz, Rx`	RZ <- shr [RX]	001001
`neg Rz, Rx`	RZ <- - [RX]	001010
`ashr Rz, Rx`	RZ <- ashr [RX]	001011

2.2.2. Instrukcije skoka

Instrukcije skoka su osnovni dio svakog programskog jezika koje omogućavaju neslijedno izvršavanja koda. Koristeći iste možemo postići bitne aspekte programa kao što su uvjetna grananja ili petlje. Kod instrukcije skoka ima sljedeći format:

```
I I I I I I  A A A A A A A A A
```

Gdje slova 'I' predstavljaju 6-bitni kod instrukcije, a slova 'A' predstavljaju 9-bitnu adresu na koju se skače. Statusni bitovi koje generira ALU se koriste kao uvjeti za skok. Statusni bit ZERO je označen sa slovom 'z', SIGN bit sa slovom 's' te CARRY bit slovom 'c' u nazivu instrukcije, a ako želimo da se skok dogodi u slučaju da taj određeni bit nije aktivan u nazivu se nalazi i slovo 'n'. Tako bi instrukcija za bezuvjetan skok na adresu 0 izgledala ovako:

```
jmp &"000000"
```

Instrukcija za skok uvjetno ako je zastavica ZERO aktivna je sljedeća:

```
jmpz &"000000"
```

Instrukcija za skok uvjetno ako zastavica ZERO nije aktivna izgledala bi na sljedeći način:

```
jmpnz &"000000"
```

Tablica 2. Instrukcije skoka

Operacija	Funkcija	Kod instrukcije
`jmp ADDR`	PC <- ADDR	010000
`jmpz ADDR`	if Z=1 PC <- ADDR	010001
`jmps ADDR`	if S=1 PC <- ADDR	010010
`jmpc ADDR`	if C=1 PC <- ADDR	010011
`jmpnz ADDR`	if Z=0 PC <- ADDR	010101
`jmpns ADDR`	if S=0 PC <- ADDR	010110
`jmpnc ADDR`	if C=0 PC <- ADDR	010111

2.2.3. Instrukcije za pristup memoriji

Kako bi pristupili RAM memoriji koristimo instrukcije STORE (spremi) i LOAD (učitaj). Kako bismo mogli definirati kojoj adresi RAM-a pristupamo u instrukciji ćemo navesti u kojem registru je spremljena adresa.

Instrukcija LOAD (kod „ld“) učitava vrijednost spremljenu na adresi RAM-a koja se nalazi u registru 'Y', u registar 'Z'. Instrukcija STORE (kod „st“) sprema vrijednost iz registra 'X' na adresu RAM-a zapisanu u registru 'Y'.

Tablica 3. Instrukcije za pristup memoriji

Operacija	Funkcija	Kod instrukcije
`ld Rz, Ry`	RZ <- [[RY]]	100000
`st Ry, Rx`	[RY] <- [RX]	110000

2.2.4. Instrukcije za unos konstanti

Kako bismo mogli koristiti korisničke konstante postoje naredbe za unos proizvoljne konstante. S obzirom na to da je instrukcija duga 15 bita, pri čemu je 6 bita rezervirano za kod instrukcije, proizlazi kako nije moguće s jednom instrukcijom upisati 16-bitni broj. Zato instrukcija LOAD CONSTANT (kod „ldconst“) ima sijedeći format:

I I I I I I B D D D D D D D

Gdje slovo 'I' označava 6 bitova instrukcije koja se odabire, slovo 'B' označava jedan kontrolni bit koji određuje hoće li bajt koji upisujemo biti gornji ili donji tj. upisujemo li bitove 0. do 7. (bit 'B' je postavljen na 0) ili bitove 8. do 15. (bit 'B' je postavljen na 1). Slovo 'D' označavaju 8 podatkovnih bitova (jedan bajt). Ako je kontrolni bit 'B' postavljen na '0', onda će se gornji bajt registra postaviti na 0, a donji na 8-bitni podatak iz instrukcije. Ako je kontrolni bit 'B' postavljen na '1', podatci iz instrukcije će se upisati u gornji bajt, a donji bajt registra će ostati nepromjenjen, na što se mora paziti ako ne želimo zadržati podatke u donjem bajtu. Instrukcija je napravljena na ovaj način tako da za upis konstanti za koje je potrebno 8 ili manje bitova za zapis nije potrebno koristiti dvije naredbe. Za spremanje ovih vrijednosti ne koriste se registri opće namjene, već posebni registar za konstante kojemu se ne može pristupiti na drugi način osim preko naredbi `ldconst` i `mvcreg`.

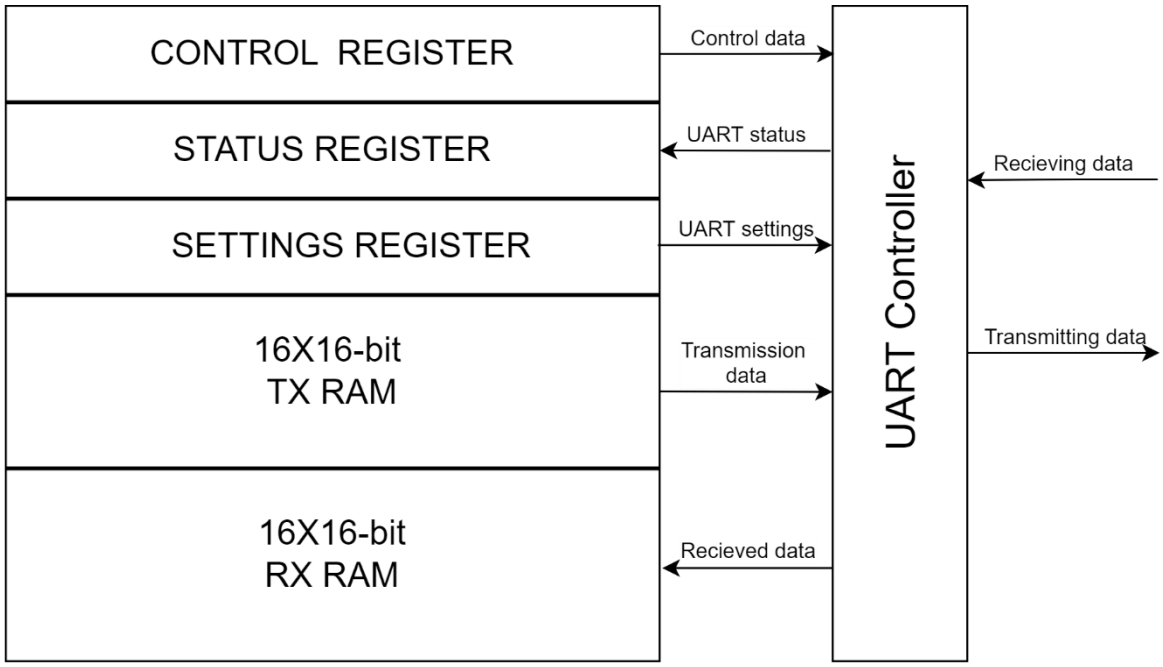
Kada smo u registar za konstante spremili željenu konstantu, premještamo ju u registar opće namjene naredbom `mvcreg` (premjesti konstantu u registar , eng. „move constant to register“), koja sprema vrijednost iz registra za konstante („RC“) u registar RX.

Tablica 4. Instrukcije za rukovanje konstantama

Operacija	Funkcija	Kod instrukcije
`ldconst lower/upper bit,DATA`	RC <- DATA	100110
`mvcreg Rx`	RX <- [RC]	100111

2.3. UART modul

Kako bi procesor mogao komunicirati s vanjskim sklopovima ima mogućnost pristupa modulu koji obavlja primanje i odašiljanje podataka UART protokolom.



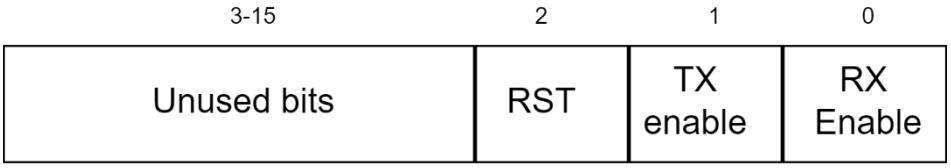
Slika 3. Dijagram UART modula

Koristeći već opisane instrukcije LOAD i STORE procesor UART modulom upravlja preko kontrolnog registra i registra postavki UART-a, te upisuje podatke za slanje, odnosno čita primljene podatke na dodijeljenim virtualnim adresama. UART kontroler prima podatke (ukoliko je aktivan bit dozvole primanja), sprema ih u RX RAM te odašilje podatke spremljene u TX RAM (ukoliko je aktivan bit dozvole odašiljanja). Procesor definira koliko podataka želi zadržati u RX RAM-u (postavljanjem vrijednosti u registru postavki), a kada se taj broj premaši, započnu se prepisivati najstariji podaci. Za slanje je također definiran dio registra postavki koji označava koliko dio podataka spremljenih u TX RAM želimo odaslati prije nego se počnu ponavljati, krenuvši od najranije odaslanog.

UART potokol koji se koristi za primanje i odašiljanje podataka je sljedeći: prvo se odašilje jedan Start bit koji ima vrijednost logičke '0', zatim se odašilje 8 podatkovnih bitova i na kraju jedan Stop bit koji ima vrijednost '1'. Ako ovaj standard nije zadovoljen podatak se ne može smatrati važećim.

2.3.1. Registri UART modula

Kontrolni registar UART modula se može čitati i pisati na virtualnoj adresi x“1000“ te ima sljedeću podjelu:



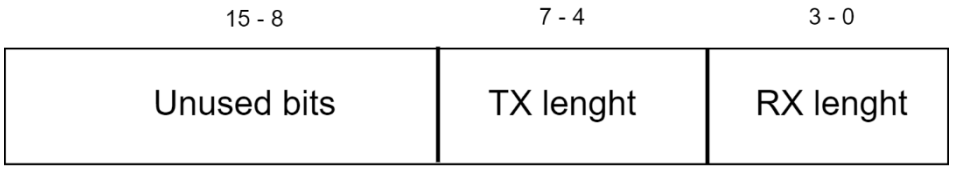
Slika 4. Značaj bitova kontrolnog registra

RX Enable ili dozvola primanja, ukoliko je aktivna, dopušta da sklop prima podatke te zapisuje u RX RAM.

TX Enable ili dozvola odašiljanja dopušta sklopu da odašilje podatke spremljene u TX RAM.

RST ili reset bit aktivan na logičku '1' postavlja brojače odašiljanja i primanja na nultu poziciju. Ovaj reset je tipa „Clear on write“ što znači da će, ukoliko aktiviran, već u slijedećem ciklusu sam postaviti na logičku '0'.

Registar postavki UART modula se može čitati i pisati te se nalazi na virtualnoj adresi x“1001“. Oblikovan je na slijedeći način:

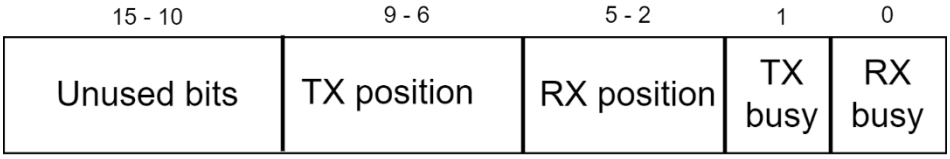


Slika 5. Značaj bitova registra postavki

RX length je 4-bitni broj koji označava koliko riječi RAM memorije se koristi. Najveća moguća veličina je 16, a najmanja 1. Kada se ispune sve riječi na koje je dopušteno pisati (ograničenje RX length) započinje se prepisivati podatke počevši od početka.

TX length je 4-bitni broj koji označava koliko prvih riječi od onih spremljenih u TX RAM će se odašiljati. Kada su sve riječi odaslane, ponovno se odašilje od početka TX RAM-a (ukoliko je aktivan signal dozvole upisa).

Statusni registar sadrži informacije o radu UART sklopa te se može samo čitati. Nalazi se na virtualnoj adresi x“1002“ te pohranjuje slijedeće podatke:



Slika 5. Značaj bitova registra postavki

RX busy ili prijemnik zauzet označava da se trenutno vrši primanje podatka.

TX busy ili odašiljač zauzet označava da se trenutačno odašilju podatci.

TX i RX position su 4-bitni brojevi koji označavaju koja riječ se trenutno odašilje (u slučaju TX position) ili gdje se po redu primljena riječ zapisuje slijedeće.

3. Program za računanje preko UART protokola

Koristeći sve navedene funkcionalnosti ovog sklopa, razvijen je program u asemblerskom jeziku koji je prethodno definiran. Ovaj program obrađuje jednostavne zahtjeve za izračunom primljene UART protokolom. Kada se program pokrene on čeka da se primi broj, operator te još jedan broj nad kojima će se izvršiti operacija. Kraj zahtjeva označava znak ';', te se nakon njegovog primitka započinje obrada i računanje. Kada je rezultat spreman, on se odašilje znak po znak UART protokolom na izlaz procesora. S obzirom na to da je procesor 16-bitni, maksimalni raspon brojeva koje može spremiti je od -32768 do 32767, ukoliko su brojevi ili rezultat izvan tog raspona odaslati će se netočan podatak.

Podržane operacije su zbrajanje (operator '+'), oduzimanje (operator '-'), množenje (operator '*') te dijeljenje (operator '/').

Primjer transakcije:

Na prijemniku UART modula procesora se primi slijedeći niz ASCII znakova:

```
1000/50;
```

Tada bi procesor obavio potreban izračun te na odašiljač odaslao UART protokolom sljedeći niz znakova:

4. Zaključak

Sklop, koji je ostvaren ovim projektom, omogućuje izvedbu korisničkih programa. Razvijeni asemblerski jezik vrlo je značajan za stvaranje apstrakcije oko implementacije instrukcija, što ujedno i približava programiranje ovog sklopa korisniku. Najveće ograničenje ovog procesora je 16-bitna duljina riječi što značajno ograničava korištenje istog za mnoge matematičke funkcije. Ovo ograničenje jednako utječe i na količinu radne memorije koju možemo imati te ju ograničava na 64 KB.

Iako razvijen i testiran na Zynq ZIBO-7000 FPGA razvojnom sustavu, ovaj sklop se može koristiti i s većinom FPGA ploča jer je opisan u jeziku za opis sklopovlja VHDL. Razvijeno rješenje je moguće jednostavno implementirati u vlastitom razvojnom okruženju te iskoristiti njegov potencijal.