Detekcija gresaka u digitalnim serijskim komunikacijama

Abstrakt

Serijska komunikacija je proces slanja podataka između dva kompjuterska sistema u kojem se svaki bit šalje pojedinačno preko istog komunikacijskog kanala. Ovo je u kontrastu sa paralelnim komunikacijama gdje se svi biti jednog podatka šalju u isto vrijeme i svaki bit ima svoj zasebni kanal (žicu). Serijske komunikacije su izuzetno korisne za slanje podataka na velike udaljenosti gdje cijena kablova ima značajnu ulogu. Primjer ovakve upotrebe su podvodni kablovi za internet koji se u nekim slučajevima prostiru preko cijelog okeana. Međutim serijske komunikacije nisu samo korisne u prenošenju podataka na velike udaljenosti i sve više se koriste u primjenama kao što su štampana kola zbog manje cijene u uporedbi sa paralenim komunikacijama. Pojava grešaka je neizbježna u svakom komunikacijskom sistemu, a pogotovo u prijenosu podataka na velike udaljenosti. S obzirom da skoro sva složenija tehnoligija ovisi od serijskih komunikacija mogučnost detektovanja grešaka je veoma važna.

Uvod

Postoji mnogo različitih metoda detekcije gresaka, ali se sve svode na jedan zajednički problem. Kako primatelj poruke moze znati da je došlo do greške ako ne zna orginalnu poruku. Ovaj problem se rješava tako što na kraj poruke dodamo neke informacije o poslatoj poruci koje primatelj provjerava kada primi cijelu poruku. Analogija ovog sistema u pravom životu bi bila kada bi na kraju rečenice dodali kojim jezikom pričamo. Tada bi osoba sa kojom pričamo mogla da provjeri da li sve riječi koje smo rekli spadaju u taj jezik. Normalno ovakav sistem ne bi bio praktičan u komputerskoj primjeni.

- Problem dva generala

U IT-u postoji poznat problem dva generala koji glasi ovako:
Dva generala se sa svojim vojskama nalaze na brdima koja okružuju dolinu. U sredini doline je trvrđava. Cilj generala je da zauzmu tvrđavu, ali to mogu postići jedino ako napadnu u isto vrijeme. Ako napadne samo jedan general on će biti sigurno pobijeđen, što znači da se dva generala moraju koordinisati tako da obadvojica mogu sa apsolutnom sigurnošću znati da će napasti u isto vrijeme. Njihov jedini način komunikacije je putem glasnika koji moraju ići nesigurnim putem na kojem postoji šansa da budu uhvaćeni, čime poruka ne bi bila dostavljena. Na koji način se ova dva generala mogu dogovoriti tako da obadvojica budu sigurni da će napasti u isto vrijeme.

Ovaj problem je nemoguće riješiti. Njegova svrha je da pokaže da nikada nećemo biti sigurni da je poruka koju pošaljemo sa jendig kompjuterskog sistema stigla do svoje destinacije. Zbog toga su potrebni sistemi koji će nas obavijestiti da je došlo do greške u prijenosu i povećati šansu da poruka dođe do svoje destinacije.

Fokus ovog rada će biti samo metode detektovanja grešaka, ne sistemi koju preuzimaju nakon što dođe do greške.

Provjera parnosti

Jedna od metoda za pronalaženje grešaka je provjera parnosti. U osnovi ovom metodom se provjerava parnost broja jedinica u poslanoj poruci. Recimo da šaljemo osam bitnu poruku. Od tih osam bita sedam bita je poruka, a osmi bit pretstavlja parnost poruke. To znači da ako je broj jedinica u poruci paran prvi ili zadnji bit će biti jedinica (ovo zavisi od protokola, to jeste da li jedinica pretstavlja paran ili neparan broj jedinica). Ovom metodom parnost poruke će se uvijek održati zavisno od toga da li je protokol “even” ili “odd” parity.

Primjer:
Koristit ćemo protokol koji provjera da li je broj jedinica neparan i ako jeste na kraj poruke dodaje jedinicu. Ovakav protokol se naziva even parity zato što broj jedinica u finalnoj poruci uvijek ostaje paran.

Poruka od 7 bita	Broj jedinica	Finalni bajt	Broj jedinica u finalnom Bajtu
0 0 0 0 0 0 0	0	0 0 0 0 0 0 0 0	0
1 1 1 1 1 1 1	7	1 1 1 1 1 1 1 1	8
1 1 0 0 1 0 1	4	1 1 0 0 1 0 1 0	4
1 1 1 0 1 0 1	5	1 1 1 0 1 0 1 1	6

Sl. 01. Tabela različitih vrijednost sa i bez parity bit-a

Kao što možemo vidjeti broj jedinica u finalnom bajtu će uvijek ostati paran. Zahvaljujući ovoj osobini potrebno je samo provjeriti parnost broja jedinica kako bi znali da li je došlo do greške.
Još jedna prednost ove metode je lakoća hardverske implementacije. Parity check se može napraviti koristeći samo jedan D flip-flop gate, što znači da je provjera jako brza.

Dok prednost provjere parnosti jeste njena jednostavnost i brzina, postoje i limitacije. Da bi ih mogli objasniti potrebno je uvesti pojam Hammingove udaljenosti. Hammingova udaljenost je broj bita koji se razlikuju u dvije poruke. Naprimjer u porukama 1**1**01 i 1**0**01 razlikuje se samo drugi bit te imaju Hammingovu udaljenost od 1, a poruke 1**10**1 i 1**01**1 u kojima se razlikuju drugi i treći bit imaju Hammingovu udaljenost od 2. Problem sa provjerom parnosti je što ako se promjene dva bita (ili bilo koji paran broj bitova) greška neće biti detektovana. Ovo znači da je minimalna Hammingova udaljenost između dvije poruke koja neće biti detektovana 2.

Iz ovog razloga razvijene su druge, efikasnije metode za detekciju grešaka.

Checksum

Još jedna metoda za detektovanje grešaka je checksum. Riječ checksum se odnosi na veliki broj različitih protokola. Osnovna ideja ove metode je sabiranje svih djelova poruke (bajtova, slova ili brojeva od kojih se ta poruka sastoji) i dodavanje tog zbira na kraj poruke tako da primaoc može da izvrši svoju kalkulaciju i provjeri da li je došlo do greške.

Najjednostavniji primjer checksum-a bi bio slučaj u kojem šaljemo neki broj i saberemo sve cifre tog broj, zatim primaoc uradi isto i provjeri da li se dobivene sume podudaraju. Očigledan problem ovog sistema je mogučnost da različiti brojevi daju ista rješenja. Primjer:

Poruka 1: 123
Checksum: 6

Poruka 2: 114
Checksum: 6

Iz ovog razloga su razvijene metode gdje se dobivene sume dodatno obrađuju.

Primjer primjene checksum-a su bar kodovi. U bar kodu checksum pretstavlja zadnja cifra koja se računa na sljedeći način: svaka cifra na neparnom mjestu se množi sa tri. Zatim se sve cifre sabiraju i suma se oduzima od sljedećeg najmanjeg broja djeljivog sa deset.

Sl. 02. Primjer bar koda sa checksum-om.
3*9 + 1*0 + 3*3 + 1*1 + 3*1 + 1*0 + 3*1 = 43, 50 - 43 = 7
https://barcode.tec-it.com/en/EAN8?data=9031101

Checksum također možemo gledati kao funkciju kojoj kao parametar dajemo jednu vrijednost, ona tu vrijednost preradi i vraća rezultat.. Ovakve funkcije se zovu hash funkcije i osmišljene se tako da zadovoljavaju određene kriterije.

Kriteriji hash funkcija:

1. Hash funkcije su determinističke. To znači da ćemo za istu unesenu vrijednost uvijek dobiti isti rezultat.
2. Dvije različite unesene vrijednost nikad ne smiju imati isti rezultat.
3. Hash funkcije ne smiju imati inverznu funkciju. Nakon što dobijemo rezultat hash funkcije iz tog rezultat ne smije biti moguće dobiti unesenu vrijednost.
4. Male promjene u unesenoj vrijednosti moraju rezultirati u potpuno različitoj izlaznoj vrijednosti (avalanche effect).
5. Rezultat hash funkcije mora biti iste veličine neovisno od veličine unesene vrijednosti.

Jednostavne hash funkcije kao što je primjer sa bar kodom se ne moraju strogo pridržavati ovih kriterija jer se najčešće koriste za sprječavanje ljudskih grešaka. Ovo je u kontrastu sa danas popularnim hash funkcijama kao SHA-1, SHA-2, MD5 itd. Svrha ovih hash funkcija, koje se još zovu kriptografske hash funkcije, osim integriteta podataka je i sigurnost. Kriptografske hash funkcije nam omogučavaju da provjerimo da li je neki file izmjenjen bez našeg znanja.

Dok ova metoda jeste izuzetno efektivna u detektovanju grešaka i pruža značajnu sigurnost, njena najveća mana je brzina. Za računanje hash funkcija potreban je velik broj kompjuterskih operacija što ih čini neprikladnim za korištenje u svrhu serijskih komunikacija gdje su brzina prijenosa podataka i energetska potrošnja najvažniji faktori. Iz ovog razloga checksum metoda se rijetko koristi za provjeru malih količina informacija, poput paketa koji služe za prenošenje podataka preko interneta. Najčešća upotreba checksum metode je provjera fajlova nakon što su već downloadovani ili za osiguravanje integriteta podataka na disku.

CRC

Cyclic redundancy check

CRC ili Cyclic redundancy check je cjerovatno najkorištenija metoda detekcije grešaka u serijskim komunikacijama, a razlog tome je što se koristi za provjeru svakog paketa podataka poslatog preko interneta. Prednost CRC-a je njegov odnos efikasnosti i jednostavnosti. CRC je dovoljno jednostavan da se može hardverski implementirati, ali opet ima odlične sposobnosti za detekciju grešaka.

Dok sam CRC jeste jednostavan za implementirati i hardverski i softverski daleko je od jednostavnog za objasniti i shvatiti.

CRC mozemo gledati kao posebnu podvrstu checksum-a. Još uvijek imamo funkciju kojoj kada damo neku vrijednost on vraća drugu vrijednost promijenjenu na predvidiv način. Čak šta više CRC je napravljen tako da poštuje sve iste kriterije kao HASH funkcije.

Kao što smo već vidjeli operacija sabiranja nije dovoljna da napravi efikasan algoritam za detekciju grešaka. Potrebno nam je nešto što daje više haotičan rezultat. To nešto je dijeljenje. Osnovna ideja CRC-a je da kada bilo koji broj podijelimo sa nekim drugim brojem i ostatak oduzmemo od orginalnog broja, razlika će biti djeljiva sa originalnim djeliocem.

Primjer:
10 / 3 = 3 (1)
(10 - 1) / 3 = 3 (0)

Dijeljenje u decimalnom sistemu je vrlo jednostavno, ali da vidimo kako bi ova ista kalkulacija izgledala u binarnom sistemu. Da bi to uradili morat ćemo se sjetiti kako da ručno dijelimo brojeve:

10 dec = 1010 bin
3 dec = 11 bin

```
 1 0 1 0 / 1 1 = 0 1 1 ----- Rezultat
-0 0 |  |  (0 * 11)            011 bin = 3 dec
 1 0 1 |
  -1 1 |  (1 * 11)
 0 1 0 0
    -1 1  (1 * 11)
  0 0 0 1 ------ Ostatak
                0001 bin = 1 dec
```

Ako sada ostatak oduzmemo od prvobitnog broja razlika će biti dijeljiva sa 11 (bin).

```
 1 0 1 0
-0 0 0 1
 1 0 0 1 ----- 1001 bin = 9 dec
```

```
 1 0 0 1 / 1 1 = 0 1 1 ---- Rezultat
-0 0 |  |  (0 * 11)            011 bin = 3 dec
 1 0 0 |
  -1 1 |  (1 * 11)
 0 0 1 1
    -1 1  (1 * 11)
  0 0 0 0 ------ Ostatak je 0
```

U pravoj CRC kalkulaciji iskoristiti ćemo činjenicu da je ostatak nula. Umjesto da kao što smo do sada radili na kraj poruke dodamo neke podatke koje ćemo kasnije porediti sa rezultatom funkcije izvršene na primaocu, jednostavno ćemo provjeriti da li je ostatak kalkulacije jednak nuli.

Jedan od problema sa kojima se susrećemo tokom klasičnog dijeljenja je što moramo prenositi brojeve. Na primjer kada oduzimamo 1 od 10 (binarno) proces ide ovako: prvo oduzmemo 1 od 0, ostaje nam 1, a pamtimo 1. Zatim oduzmemo 1 od 1 čime dobijemo 0 i rezultat je 01. 1 0

-0 1
0 1

Kao što znamo računske operacije poput dijeljenja kompjuterima ne predstavljaju problem, ali kada nam je prioritet brzina, a samim tim i smanjivanje broja operacija klasično dijeljenje postaje poprilično nepraktično.

Iz ovog razloga CRC koristi konačne skupove brojeva. Moj cilj će biti da ovu temu objasnim što jasnije bez da ulazim preduboko u oblast skupova brojeva.

Da bismo shvatili kako skup brojeva može biti konačan morat ćemo razmotriti skupove brojeva sa kojima smo već upoznati. Svrha brojeva je da sa njima manipulišemo i nad njima vršimo operacije kako bi riješili neki problem. Ta manipulacija brojeva se vrši po pravilima algebre čiji je cilj da uvijek daju predvidive rezultate. Da bi mogli ispoštovati pravila algebre brojevi koje koristimo također moraju pratiti određeni set pravila. Na primjer skup prirodnih brojeva kojem tu i tamo fali nekoliko cifara nam ne bi bio od velike koristi. Iz ovog razloga postoje aksiomi koji određuju strukturu skupa brojeva.
Ti aksiomi su sljedeći:

1) Asocijativnost:
    a) a + (b + c) = (a + b) + c
    b) a * (b * c) = (a * b) * c
2) Komutativnost:
    a) a + b = b + a
    b) a * b = b * a
3) Distributivnost:
    a) a * (b + c) = a * b + a * c
4) Neutralni element za sabiranje:
    a) a + 0 = a
5) Neutralni element za množenje:
    a) a * 1 = a
6) Invers za sabiranje:
    a) a + (-a) = 0
7) Invers za množenje:
    a) a * 1/a = 1

Kada imamo skup brojeva koji poštuje sve ove aksiome znamo da poštuje i pravila algebre i možemo ga koristiti kao bilo koji drugi skup brojeva.

Kao što vidimo nijedan od aksioma ne zahtijeva da skup brojeva bude beskonačan, to nam omogućava da za našu CRC kalkulaciju koristimo skup koji se sastoji od samo dva broja. Jedan i nula.

Ali prije nego što počnemo koristiti ovaj konačni skup brojeva moramo definisati kako u njemu rade algebarske operacije, sabiranje, oduzimanje, množenje i dijeljenje, a zatim provjeriti da li ovaj novi skup zadovoljava sve aksiome.

Algebarske operacije u skupu [0, 1]:

0 + 0 = 0	0 - 0 = 0
0 + 1 = 1	0 - 1 = 1
1 + 0 = 1	1 - 0 = 1
1 + 1 = 0	1 - 1 = 0

0 * 0 = 0	0 / 1 = 0
0 * 1 = 0	1 / 1 = 1
1 * 0 = 0	
1 * 1 = 1	

Neke stvari možda izgledaju neobično, kao na primjer 1 + 1 = 0, ali ako provjerimo svi aksiomi su zadovoljeni i time možemo zaključiti da je naš skup validan.

Jedan od glavnih razloga za korištenje ovog sistema je to što ako pogledamo tablicu sabiranja i oduzimanja shvatimo da su te operacije identične XOR operaciji. Ova činjenica za sada nije bitna ali je jako važna za hardversku implementaciju CRC-a.

Nakon ove digresije u polje matematike, vratimo se CRC-u. Kako nam konačan skup može pomoći da riješimo problem prenošenja brojeva tokom dijeljenja? Prvo ćemo vidjeti kako je taj problem nastao. Srž prenošenja brojeva je ustvari u tome da je u klasičnom skupu brojeva 0 - 1 = -1. Mi smo ovaj problem riješili tako što -1 više ne postoji i sada je 0 - 1 = 1. Ako se vratim na primjer oduzimanja 10 - 01 to izgleda ovako:

```
  1 0
-0 1
 1 1
```

Na prvi pogled ovo nema smisla. 11 (bin) je veće od 10 (bin). Kako onda oduzimanjem možemo dobiti veći broj? Odgovor je da u ovom novom sistemu nema smisla porediti brojeve po veličini. Iz 10 možemo dobiti 11 i oduzimanjem i sabiranjem istog broja, što

nas dovodi do zaključka da 11 nije ni veće ni manje od 10. Dakle jedini način da poredimo brojeve je po redu veličine.1 < 10 < 100.
Sa ovim dolazimo do dijeljenja. Dijeljenje je vrlo slično kao što je bilo i prije, razlikuje se samo po tome što je sada jedina kriterija za poređenje brojeva njihov red veličine.

Primjer od ranije:

```
 1 0 1 0 / 1 1 = 0 1 0 ----- Rezultat
-1 1 |  |  (1 * 11)              011 bin = 3 dec
 0 1 1 |
  -1 1 |  (1 * 11)
 0 0 0 0
     -0 0  (1 * 11)
  0 0 0 0
```

Kao što vidimo u novom sistemu 1010 je djeljivo sa 11. U klasičnom pogledu ovo nema smisla, 10 nije djeljivo sa tri, ali nama to nije bitno jer nas ne zanimaju vrijednosti koje dobijemo, već samo da su sva pravila algebre ispoštovana.

Sada kada razumijemo sistem računanja koji se koristi u CRC kalkulacijama možemo objasniti i kako CRC radi. CRC ustvari nije ništa više od tretiranja cijele poruke kao jedan broj i dijeljenja tog broja (koristeći prethodno objašnjena pravila). Ovo možda izgleda jednostavno ali većina svojstava CRC kalkulacije zavisi od toga koji dijelioc izaberemo. Ovaj dijelioc se naziva generator polynomial ili polly.

Prođimo sada kroz korake koji se dese prilikom slanja jedne poruke, generisanja CRC-a i provjere. Recimo da želimo poslati poruku “Hi”.

ASCII vrijednosti za slova ove poruke su: H - 0100 1000, i - 0110 1001.
Koristit ćemo osam bitni polly (dijelioc): 1001 1011.

Prvi korak je generisanje CRC-a (sve operacije se vrše po pravilima [1, 0] skupa brojeva):

1. Uzmemo sve bajtove poruke i pretvorimo ih u jedan broj
   H + i = 0100100001101001
2. Na kraj poruke dodajemo onoliko nula koliko polly koji koristimo ima cifara. U našem slučaju 8.
   Dakle naša proširena poruka izgleda ovako: 0100100001101001000000000

3. Sada poruku podijelimo sa poly-em koji koristimo i zadržimo ostatak, rezultat dijeljenja nije bitan. Ostak u našem slučaju je 11110101.
4. Na kraju ostatak koji smo dobili oduzmemo od proširene poruke. Rezultat izgleda ovako: 01001000011010011111010 1

Na prvi pogled zadnji korak izgleda neobično, jer smo rekli da ostatak oduzimamo ali smo ga dodali. Ali ako se podsjetimo po novim pravilima operacije oduzimanja i sabiranja su iste.

Sljedeći korak je provjera da li je došlo do greške u prijenosu. Kada primaoc dobije poruku sa CRC-om potrebno je samo da cijelu poruku podijeli sa istim polly-em sa kojim je CRC originalno izračunat. Ako je ostatak nula nije došlo do greške.

Praktičan rad

Plan

Praktični dio ovog rada će se sastojati od pravljenja modela serijske komunikacije i primjenjivanja prethodno spomenutih metoda za detekciju grešaka na taj model. Ovaj model će biti napravljen od dva arduino mikrokontrolera. Međusobno će komunicirati preko jedne data žice i jende clock žice. Za detekciju grešaka koristit ćemo prvo provjeru parnosti a onda CRC.

Hardver

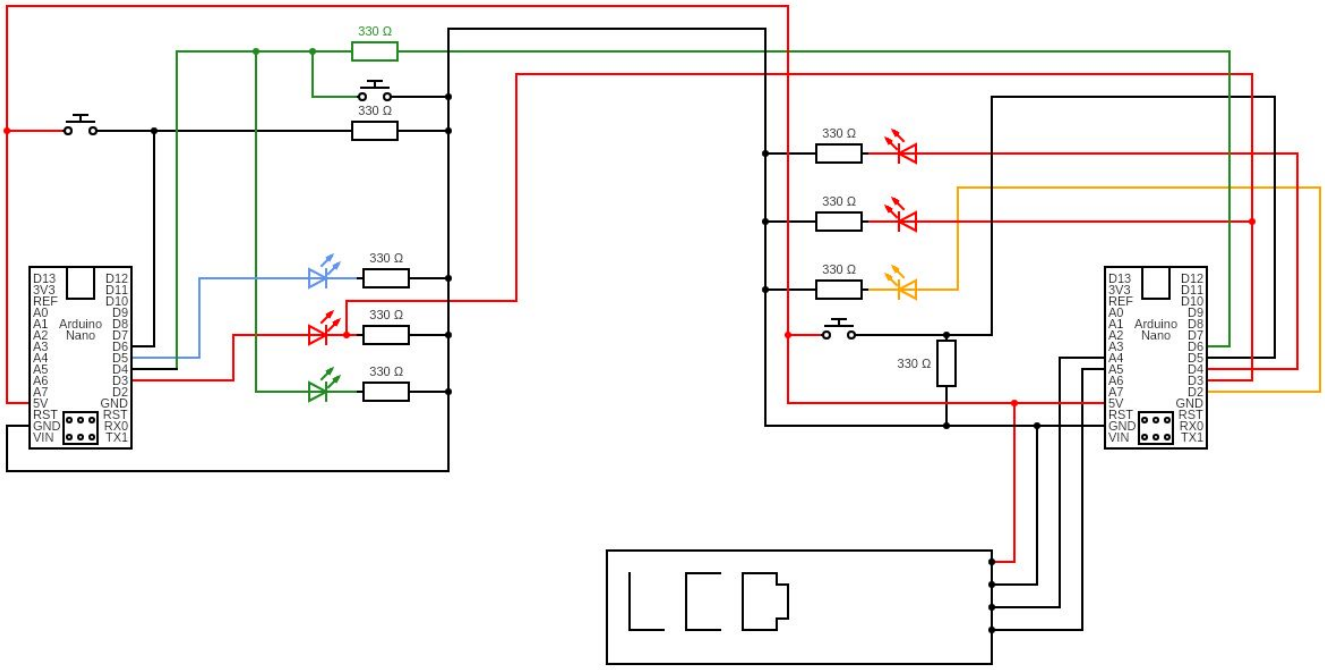
Za hardverski dio ovog rada koristim dva arduino nano v3 mikrokontrolera. Jedan ima ulogu transmitera, a drugi resivera.

- Transmiter

Sa strane transmitera su spojene tri LED diode i dva dugmeta. Funkcije dvije od tri LED diode su da pokazuju trenutno stanje data i click žica, a treća služi za indikaciju parnosti poruke. Funkcija prvog dugmeta je da započne proces slanja poruke, a drugo dugme služi da uvedemo grešku u poruku. Kada se pritisne ovo dugme će mehaničiki spustiti data žicu na nulu i onemogučiti slanje jedinica. Pošto grešku uvodimo mehanički ne postoji način da transmiter sazna da je došlo do greške, ovo čini cijeli proces vjerodostojnim serijskim komunikacijama u pravom svijetu.

- Resiver

Komponente sa resiverske strane su vrlo slične ali imaju malo drugačiju funkciju. Glavna razlika je što sa ove strane imamo LCD ekran (ovaj ekran kontrolišemo sa I2C kontrolnom pločicom) i samo jedno dugme umjesto dva. Na ekran prvo ispisujemo svaki bit poruke čim stigne, zatim nakon osmog bita ispišemo bajt koji tih osam bitova sačinjava. Ako poruka stigne bez greške bit će ispisana na ekranu, a ako ipak dođe do greške koju uspijemo detektovati na ekranu će biti ispisano ERROR.



Sl. 03. Šematski prikaz kola.
Slika je napravljena uz pomoč https://www.circuit-diagram.org

Softver

Za razvijanje koda koristio sam arduinov online IDE (integrated development environment), arduino create. Programski jezik koji se koristi za programiranje arduino mikrokontrolera je modifikovana verzija c++ osnovana na wirining projektu.

- Softverska implementacija provjere parnosti

Kao i hardver, softver se dijeli na transmiter i resiver. Prvo ćemo pogledati kako izgleda kod sa strane transmitera.

- **Kod transmitera**

Pogledajmo osnovnu strukturu programa.

```
void loop() {
  if (digitalRead(resetPin)){ // Dugme je pritisnuto
    while (digitalRead(resetPin)) {} // Dugme je pušteno
    digitalWrite(parityPin, LOW); // Restovati LED za indikaciju parnosti
    parityCheck = true;           // Postaviti parnost na parno (true)
prije slanja poruke
    sendMessage(message);         // Pošalji poruku

    parityCheck ? sendParityBit(true) : sendParityBit(false); // Pošalji bit za
parnost

    parityCheck ? digitalWrite(parityPin, HIGH) : digitalWrite(parityPin, LOW);
// Upali ili ugasi LED za parnost
  }
  if (Serial.available()) { // Dostupna nova poruka na serijskom portu
    readSerial(message); // Pročitaj novu poruku
  };
}
```

Sl. 04. void loop() funkcija transmitera

Ovo je funkcija koja će se izvršavati i ponavljati sve dok je program uključen, zbog toga nju koristimo isključivo za zvanje drugih funkcija koje onda izvršavaju svoje zadatke. Kao što vidimo ova funkcija čeka dok se ne pritisne dugme za slanje poruke. Kada je dugme pritisnuto poruka se šalje, a kada pošaljemo poruku iza nje šaljemo jedan bit koji pretstavlja parnost poruke.

Sljedeća funkcija koju posmatramo je sendMessage() funkcija:

```
void sendMessage(char *TXstring){
  for (int byteIndex = 0; byteIndex <= strlen(TXstring); byteIndex++) { // For koji prolazi kroz svaki karakter u poruci
    char currByte = TXstring[byteIndex];
    for (int bitIndex = 0; bitIndex < 8; bitIndex++) { //For koji prolazi kroz svaki bit u trenutnom bajtu
      bool currBit = (0x80 >> bitIndex) & currByte; //Ako je na bitIndex mjestu u trenutnom bitu 1 postaviti currBit na true
      if (currBit) { //Ako je trenutni bit 1 poslati 1
        digitalWrite(txPin, HIGH);
        pulseClock();
        parityCheck = !parityCheck;
      }
      else { //Ako je trenutni bit 0 poslati 0
        digitalWrite(txPin, LOW);
        pulseClock();
      }
    }
  }
}
```

Sl. 05. sendMessage() funkcija

Ova funkcija možda na prvi pogled izgleda poprilično komplikovano, ali ono što je najvažnije je da primjetimo na koji način šaljemo podatke. Kao što je ranije spomenuto za prijenos podataka koristimo dvije žice, jedna za datu (podatke), a druga za clock. Ovaj tip serijske komunikacije se naziva asinhrona komunikacija. Razlog tome je što koristimo clock žicu da bi smo resiveru rekli kada smo poslali neki podatak. U ovom sistemu svaki put kada na clock žicu stavimo pozitivan napon resiver će znati da je stigao jedan bit. U ovoj vrsti komunikacije brzinu prenosa podataka određuje razmak

između dva pulsa clock žice. Razlog što se zove asinhrona komunikacija je što razmaci između pulseva clock žice ne moraju biti jednaki.

- ## Kod resivera

Prijeđimo sada na kod resivera. Kao i kod transmitera prvo ćemo pogledati osnovnu strukturu koda u void loop() funkciji.

```
void loop() {
  if (clockIndicator) { // Detektovan clock puls
    updateLcd();
    clockIndicator = false;
  };
  if (endOfMessage && recievedParity) { // Čekati kraj poruke
    lcdPrint("        ", 0, 1); // Očistiti ekran
    parityCheck ? digitalWrite(parityPin, HIGH) : digitalWrite(parityPin, LOW);
// Inicirati parnost preko LED
    parityCheck ? lcdPrint("Parity: even",0,1) : lcdPrint("Parity: odd",0,1);
    if (parityCheck) { // Ako je parnost parna to znaci da je doslo do greške
      delay(1000);
      lcdPrint("        ", 0, 1);
      lcdPrint("Error       ", 0, 1);
      endOfMessage = false;
    }
    resetVariables();
  };
  if (digitalRead(resetPin)) { // Spremi se za novu poruku
    while (digitalRead(resetPin)) {};
    resetLcd();
  }
};
```

Sl. 06. void loop() funkcija na strani resivera

Ovako izgleda void loop() funkcija na strani resivera. U njoj možemo vidjeti tri if provjere. Krenimo od drugog if-a jer prvi treba posebno objašnjenje. U drugom if-u provjeravamo da li je poruka u potpunosti primljena (uključujući i bit koji označava parnost poruke). Ako je ovaj uslov ispunjen provjeriti ćemo da li je došlo do greške (Znamo da parnost poruke *zajedno sa parity bit-om* mora biti neparna tako da samo provjeravamo da li je parnost ukupne poruke parna), a zatim ispisujemo parnost na ekran i ažuriramo LED diodu za indikaciju parnosti.
Treći if jednostavno provjerava da li je pritisnuto dugme za resetovanje i ako jeste pripremi program za primanje nove poruke.

Vratimo se sada na prvi if. Kao što vidimo on provjerava da li je došlo do clock pulsa, ali kada je taj uslov ispunjen samo ažuriramo ekran i ništa više. To je zato što ovdje zapravo ne detektujemo puls clock žice. Šta se ustvari desi kada dođe novi clock puls je da se aktivira interrupt. Objašnjavanje interrupt-a u c++ je izvan teme ovog rada ali je dovoljno za znati da kada je clock puls detektovan sve u programu se prestane izvšavati i pozove se funkcija koja pročita bit i doda ga u poruku.

Ovo je osnovni princip rada jednog sistema serijskih komunikacija.

- Softverska implementacija CRC metode

Kod za CRC je gotovo isti kao za detekciju parnosti. Što se tiće prenosa podataka nema razlike. Jedina razlika je tome šta šaljemo nakon poruke. U Prošlom primjeru smo nakon poruke slali bit koji predstavlja parnost poruke. Ovaj put ćemo umjesto tog bita slati cijeli bajt koji će predstavljati CRC kalkulaciju poruke.

- **Kod Transmitera**

```
void loop() {


  if (digitalRead(sendButtonPin)) {  //Dugme je pritisnuto
    while (digitalRead(sendButtonPin)) {} //Dugme je pušteno
    crcValue = calcCrc(message, crcPoly);
    sendMessage(message); //Posšalji poruku
    sendCrc(crcValue);
  }


  if (Serial.available()) { //Dostupna nova poruka na serijskom portu
    readSerial(message); //Pročitaj novu poruku
  };


}
```

Sl. 07. void loop() funkcija na strani transmitera

Kao što vidimo ovaj kod je vrlo sličan kao kod sa provjerom panosti, možda čak i jednostavniji. Jedna važna razlika je što u ovom kodu pozivamo calcCrc() i sendCrc() funkcije. Ove funkcije rade tačno ono što njihovo ime kaže, calcCrc() funkcija vrši CRC kalkulacije na poruci koju joj proslijedim, a sendCrc() funkcija tu vrijednost pošalje resiveru.

- **Kod resivera**

Kao i kod za transmiter ovaj kod se ne razlikuje puno od koda za provjeru parnosti. U suštini jedina razlika je što sada greške tražimo preko CRC-a.

```
void loop() {
  if (clockIndicator) { // Detektovan clock puls
    if (!recievedCrc) {
      updateLcd();
    }
    clockIndicator = false;
  };

  if (digitalRead(resetPin)) { // Spremi se za novu poruku
    while (digitalRead(resetPin)) {};
    resetVariables();
    resetLcd();
  }

  if (recievedCrc) {
    lcdPrint("         ", 0, 1); // Očisti ekran
    if (!calcCrc(message, crcPoly)) {
      lcdPrint("CRC: Good", 0, 1);
    } else {
      lcdPrint("CRC: Error", 0, 1);
    };
    recievedCrc = false;
    clockIndicator = false;
  }
};
```

Sl. 08. void loop() funkcija sa strane resivera

Možemo primjetiti da ponovo imamo tri if provjere. Ovaj kod je u suštini analogan sa kodom za provjeru parnosti. Imamo tri provjere koje konceptualno provjeravaju iste stvari kao i u prošlom kodu. Važno je za primjetiti da CRC računamo nakon što stigne CRC vrijednost sa transmitera. Kada CRC vrijednost stigne dodajemo je na ostatak poruke i šaljemo u calcCrc() funkciju. Ova funkcija je potupuno ista na transmiteru i resiveru što znači da kada u nju ubacimo poruku sa prethodno izračunatom vrijednošću dodatom na kraj trebalo da dobijemo 0 (poruci oduzmemo ostatak od dijeljenja sa nekim brojem i onda opet podijelimo sa istim brojem). U suprotnom znamo da imamo grešku.

Zaključak

Metode detektovanja grešaka koje smo istražili u ovom radu nisu savršene. Kao što smo vidjeli sa pričom o dva generala kanali prijenosa podataka su po definiciji nesigurni, te uvijek postoji šansa da se negdje provuče jedinica koja je trebala biti nula ili obrnuto. Ali to je istina za bilo šta u stvarnom svijetu. Ipak zahvaljujuči ovim metodama svijet tehnologije u kojem danas živimo je moguć.Gotovo svaki podatak koji se kreće našim uređajima, bili to čipovi koji međusobno komuniciraju unutar naših mobitela ili podatci koje šaljemo serveru na drugom kraju svijeta, sa sobom nosi nekoliko bitova koji mu omogućuju da sigurno dođe do svoje destinacije. Za razliku od skoro svih drugih grana IT-a ova oblast nije nešto što je sklono naglim promjenama i razvoju (izuzetak oveme su kriptovalute koje su zasnovane na kriptografskim hash funkcijama). Metode koje danas koristimo u svrhu detektovanja grešaka su gotovo na teoretskom limitu onoga šta je moguće postići. Kako god uprkos ovome smatram da je važno razumjeti tehnologiju koju svakodnevno koristimo i kojom smo okruženi na ovom temeljnom nivou i shvatiti da to nije nedokučno znanje, već da se bazira na potpuno shvatljivim principima.

Literatura

https://www.sciencedirect.com/topics/engineering/hamming-distance - Distance and code performance

“Telecommunications Engineer's Reference Book” - DrM D Macleod MA PhD, 1993.
https://link.springer.com/referenceworkentry/10.1007%2F1-4020-0613-6_13637 - Parity Check, Martin H. Weik

https://www.lammertbies.nl/comm/info/crc-calculation - On-line CRC calculation and free library

https://web.archive.org/web/20180402205812/http://www.wolfgang-ehrhardt.de/crc_v3.html - A Painless Guide to CRC Error Detection Algorithms, Ross N. Williams

http://www.peterjockisch.de/texte/computerartikel/Kryptographische-Pruefsummen/Kryptographische-Pruefsummen_EN.html - “Practical Application of Cryptographic Checksums”, Peter Jockisch

Sadržaj