Enginyeria de Sistemes TIC

Xarxes de Comunicació

Pràctica 1 - Detecció d'errors

Francisco del Águila López

Setembre 2017
Escola Politècnica Superior d'Enginyeria de Manresa
Universitat Politècnica de Catalunya

1 Objectiu

L'objectiu d'aquesta pràctica és definir un mòdul en C que serveixi per fer una detecció d'errors. Aquest mòdul es farà servir per implementar un protocol d'enllaç fiable.

2 Codis detectors d'errors

Existeixen diversos mecanismes de detecció de errors en transmissions digitals. A [Wiki-Error] es troben classificats de la següent manera:

1. Codis de repetició
2. Bits de paritat
3. Checksums
4. Comprovació de redundància cíclica (CRC)
5. Funcions de hash de xifrat
6. Codis correctors d'errors

Aquesta classificació es de menys a més eficaços. També estan ordenats de manera creixent en complexitat. En la última posició es troben codis que a més de detectar errors, també són capaços de corregir-los.

A la pràctica s'ha de desenvolupar un codi detector basat en un Checksum [Wiki-Check] i basat en un CRC [Wiki-CRC]. La complexitat de l'algoritme de Checksum és molt

reduïda. Aquest algoritme es dissenyarà similar a l'algoritme de Checksum que es fa servir al protocol TCP i al protocol IP de Internet. En el cas del disseny de l'algoritme de CRC es farà us de les llibreries de C dels AVR [AVR-libc], en particular de la llibreria /util/crc16.h que facilitarà la feina.

3 Disseny del Checksum

3.1 Definició

El Checksum que es dissenyarà serà el que es fa servir a [Check-TCP/IP] amb la diferència que la mida serà de 1 Byte.

3.1.1 Càlcul del Checksum

Aquest algoritme consisteix en la suma de totes les paraules que formen el bloc de dades a protegir. L'ample de les paraules serà de 1Byte. Quan es produeix *carry* en cadascuna de les sumes que es realitzen, aquest *carry* es torna a sumar al resultat. Un cop acabades totes les sumes, s'obté un resultat de la mida d'un Byte. Amb aquest resultat es fa el complement a 1 (invertir els bits). El que s'obté correspon al valor del Checksum.

És fàcil adonar-se que si ara tornem a repetir tot el procés de suma del bloc de dades incloent en el procés aquest Byte extra de Checksum, el resultat de la suma serà de 0b11111111 (tot a 1). Això és el que permetrà a un receptor detectar si s'han produït errors en la comunicació del bloc de dades que s'ha transmès. Ja que, el transmissor, a part d'enviar el bloc de dades amb la informació que sigui, també enviarà al final aquest Byte extra de redundància (el Checksum).

3.1.2 Comprovació del Checksum

Un receptor que ha pactat amb el seu corresponent transmissor una detecció d'errors per Checksum de 1Byte, espera rebre un bloc de dades consistent en:

1. un missatge (de mida en Bytes variable)
2. un últim Byte consistent en el Byte de Checksum.

Per tant per comprovar si s'han produït errors en aquesta transmissió, agafarà tot aquest bloc de bytes rebut (incloent el Checksum) i realitzarà la suma de tots ells. Recordeu que si es produeix algun *carry* s'ha de tornar a sumar. Aquesta suma ha de donar un resultat de 0b11111111 (tot a 1) si no s'ha produït cap error. En el cas que s'hagi produït alguna modificació d'algun bit (error) s'ha d'esperar que aquesta suma deixi de donar 0b11111111 (tot a 1) i per tant es considera que hi ha errors.

3.1.3 Característiques particulars del Checksum d'aquesta pràctica

- La mida del Checksum és de 1Byte. Això vol dir que la quantitat de bits redundants que s'afegeixen són 8.
- La mida del bloc de dades amb la que es calcularà el Checksum serà variable. Aquest bloc de dades serà el que formarà la trama del protocol d'enllaç.
- El bloc de dades que protegirà el Checksum està format per caràcters que poden ser transmesos en Morse. Això vol dir que només s'admeten les lletres de A-Z i números de 0-9. Els caràcters especials que no formen part del codi Morse internacional així com els accents, no estan permesos. De la mateixa manera, no s'admeten lletres minúscules.
- La codificació dels caràcters admesos és ASCII, ja que la conversió a codi Morse es realitza a la capa física.
- El resultat del càlcul del Checksum dóna lloc a 1Byte que pot prendre qualsevol valor. Per tant, pot ser que no es trobi un possible caràcter morse vàlid per transmetre aquest Byte. Cal establir un mecanisme que garanteixi poder transmetre amb caràcters morse aquest Checksum calculat, sigui quin sigui el seu resultat.

Per garantir que el càlcul del Checksum es pugui codificar amb caràcters de Morse vàlids, es fa servir la codificació hexadecimal. Es a dir, aquest Byte es codificarà amb 2 caràcters ASCII en format hexadecimal. Els caràcters hexadecimals 0-F són caràcters vàlids per transmetre en Morse.

3.2 Implementació

La implementació del Checksum consisteix en dues funcions, una que calculi el valor del checksum, add_check() i una altre que comprovi si és correcte, check_is_ok().

3.2.1 Funció de càlcul

La implementació de la funció que calcula el Checksum es farà en llenguatge C. La funció Checksum ha de cumplir les següents condicions:

- El nom de la funció és add_check()
- Ha de tenir com a paràmetre d'entrada un *string* que conté el bloc de dades que es vol protegir amb el Checksum.
- Com a sortida, ha de donar el mateix *string* d'entrada modificat, havent afegit al final els dos caràcters que codifiquen en hexadecimal el Byte que s'ha calculat. Això implica que hem d'assegurar que al *string* que es passa com a paràmetre hi hagi espai suficient per poder afegir aquests caràcters al final.
- També ha de retornar específicament els dos caràcters que formen el Checksum.

3.2.2 Funció de comprovació

Per la implementació del mòdul de detecció d'errors, també es fa necessari l'existència d'una funció per comprovar el checksum. Aquesta funció comprova si un bloc de dades conté possibles bits erronis mitjançant la redundància afegida al final.

- El nom de la funció de comprovació és check_is_ok()
- El paràmetre d'entrada és un *string* que conté el bloc de dades + el Checksum al final
- El resultat de la funció serà un booleà (True vol dir que no s'han detectat errors).
- El *string* d'entrada ha de quedar modificat eliminant el Checksum del final.

Per implementar check_is_ok() cal fer servir com a funció auxiliar la funció add_check(). Per facilitar aquesta feina, cal definir una funció auxiliar get_redun() amb la finalitat de separar per una banda el string corresponent a les dades i per altra banda els 2 caràcters del final corresponents al byte de redundància. Aquesta funció compleix amb els següents requeriments:

- El paràmetre d'entrada és un string que conté la redundància al final. Recordar que aquesta redundància són 2 caràcters que codifiquen en format hexadecimal el Byte corresponent a la redundància.
- Com a sortida, ha de donar per una banda el mateix string d'entrada però sense la redundància i per altre banda, els 2 caràcters corresponents a aquesta redundància.

Combinant get_redun() i add_check() és trivial implementar check_is_ok().

La definició de les funcions auxiliars byte2hex() i hex2byte() que tenen com a finalitat transformar un byte en 2 caràcters corresponents a la seva representació hexadecimal i a l'inrevés, us poden ajudar a una correcta estructuració del codi.

4 Disseny del CRC

El funcionament d'un CRC està descrit a [Wiki-CRC]. Al igual que en el cas del Checksum, l'algoritme per afegir un CRC a un *stream* de bytes o per comprovar si s'han produït errors en aquest *stream* és el mateix. Poden haver diferents mides de bits afegits per garantir la detecció d'errors. Les mides més comuns són de 16 bits, però en el cas de la pràctica farem servir una mida de CRC de 8bits. A major mida de CRC, més capacitat de detecció d'errors.

La llibreria /util/crc16.h ofereix 3 funcions de càlcul per 3 variants de CRC de 16 bits i una funció de càlcul per el CRC de 8 bits. La funció

```
static __inline__ uint8_t _crc_ibutton_update (uint8_t __crc, uint8_t __data)
```

accepta com a paràmetres 2 bytes. __data és el byte que s'ha d'anar introduint corresponent al *stream* que es vol processar. __crc és el valor inicial o bé el valor calculat

anterior del candidat a ser els bits corresponent al CRC. El valor retornat és el nou valor del CRC calculat. Això vol dir que aquesta funció s'ha d'anar cridant successivament fins processar tot el *stream* corresponent. El valor inicial amb el que s'ha de començar és zero.

Per calcular el CRC s'ha de processar el *stream* de Bytes. El valor final retornat és el valor del CRC.

Per comprovar el CRC s'ha de processar un *stream* de Bytes que contindrà el *stream* original i a l'últim Byte el CRC correcte. Si el valor retornat final és tot zeros implica que no s'han detectat errors. Recordeu, que en el nostre cas, el *stream* que rebem té l'últim byte desdoblat en dos caràcters. Això és així degut al requeriment que aquest CRC s'ha de poder transmetre amb codificació Morse i per tant fent servir els caràcters vàlids Morse. Al igual que en el cas del Checksum, per garantir que es pugui codificar amb caràcters de Morse vàlids, es fa servir la codificació hexadecimal. Aquest Byte es codificarà amb 2 caràcters ASCII en format hexadecimal. Els caràcters hexadecimals 0-F són caràcters vàlids per transmetre en Morse.

4.1 Implementació

De la mateixa manera que en el cas del Checksum, necessitem d'una funció per al càlcul global del CRC, add_crc() i d'una funció encarregada de la seva comprovació, crc_is_ok().

4.1.1 Funció de càlcul

La implementació de la funció que calcula el CRC es farà en llenguatge C. La funció CRC ha de complir les següents condicions:

- El nom de la funció és add_crc()
- El paràmetre d'entrada és un *string* que conté el bloc de dades que es vol protegir amb el CRC.
- Com a sortida, ha de donar el mateix *string* d'entrada modificat, havent afegit al final els dos caràcters que codifiquen en hexadecimal el Byte que s'ha calculat. Això implica que hem d'assegurar que al *string* que es passa com a paràmetre hi hagi espai suficient per poder afegir aquests caràcters al final.
- També ha de retornar específicament els dos caràcters que codifiquen en hexadecimal el CRC.
- Aquesta funció utilitza de manera auxiliar

```
_crc_ibutton_update ()
```

4.1.2 Funció de comprovació

Aquesta funció serà equivalent a la del Checksum.

- El nom de la funció de comprovació és crc_is_ok()
- El paràmetre d'entrada és un *string* que conté el bloc de dades + el Byte de CRC al final, codificat en hexadecimal
- El resultat de la funció serà un booleà (True vol dir que no s'han detectat errors).
- El *string* d'entrada ha de quedar modificat eliminant el CRC del final.

De la mateixa manera que en el cas anterior, la funció crc_is_ok() ha de fer servir de manera auxiliar add_crc(). La funció get_redun() és independent de quin tipus de redundància s'aplica, per tant, també és útil en aquesta ocasió. Implementar crc_is_ok() fent servir add_crc() i get_redun() és trivial. Les funcions auxiliars byte2hex() i hex2byte() són igualment útils en aquest cas.

5 Estudi previ

1. Definiu com han de ser els prototipus de les funcions que s'han d'implementar. En el cas que calgui definir algun tipus de dades, doneu la seva definició.
2. Definiu què conté el fitxer de capçalera del mòdul error_morse.h.

6 Treball pràctic

1. Dissenyeu les funcions auxiliars d'aquest mòdul i comproveu el seu correcte funcionament amb algun programa de test.
2. Dissenya un programa que s'executi al teu PC de manera que utilitzant l'entrada estàndard calculi el Checksum d'un bloc de caràcters. S'ha d'ignorar els caràcters que no siguin vàlids. El delimitador de bloc de caràcter serà el retorn de carro. A la sortida estàndard s'ha de donar el bloc d'entrada amb el Checksum afegit. El delimitador serà també el retorn de carro.
3. Dissenya un programa que s'executi al teu PC de manera que utilitzant l'entrada estàndard comprovi que el Checksum d'un bloc de caràcters és correcte. El delimitador de bloc és el retorn de carro. La sortida s'ha d'indicar si és correcta o no.
4. Dissenya un programa que s'executi a l'AVR que faci el mateix que el programa 2. Considera que l'entrada i sortida estàndard és el port sèrie.
5. Dissenya un programa que s'executi a l'AVR que faci el mateix que el programa 3. Considera que l'entrada i sortida estàndard és el port sèrie.

6. Dissenya un programa que s'executi a l'AVR que faci el mateix que el programa 2 però fent servir la detecció amb CRC. Considera que l'entrada i sortida estàndard és el port sèrie.
7. Dissenya un programa que s'executi a l'AVR que faci el mateix que el programa 3 però fent servir la detecció amb CRC. Considera que l'entrada i sortida estàndard és el port sèrie.

Referències

[Wiki-Error] `http://en.wikipedia.org/wiki/Error_detection`

[Wiki-Check] `http://en.wikipedia.org/wiki/Checksum`

[Wiki-CRC] `http://en.wikipedia.org/wiki/Cyclic_redundancy_check`

[AVR-libc] `http://www.nongnu.org/avr-libc/user-manual/group__util_ _crc.html`

[Check-TCP/IP] `http://en.wikipedia.org/wiki/IPv4_header_checksum`