Übungen zu Architektur Eingebetteter Systeme

Blatt 4

22.05.2009

Teil 1: Grundlagen

1.1: Grundlagen des Cyclic redundancy code (CRC)

Im Gegensatz zum Parity-Check, der nur einfache Bit-Fehler erkennen kann, können mit dem CRC-Check auch viele Mehrfachfehler sicher oder doch zumindestens mit einer hohen Wahrscheinlichkeit erkannt werden. Das Verfahren basiert auf Polynomarithmetik. Eine n-stellige Nachricht wird als Polynom $(k-1)$-ten Grades aufgefasst, wobei jedes Bit der Nachricht ein Koeffizient des Polynoms darstellt. Vorteil ist, das dafür eine Modulo-2-Arithmetik verwendet werden kann. D.h. nach jedem Rechenschritt wird Modulo 2 gerechnet. Dadurch sind bei Additionen und Subtraktionen im Binären keine Überträge zu berücksichtigen. Beispiel:

$$\begin{array}{rr} 10011011 & 10011011 \\ +\ \underline{11001010} & -\ \underline{11001010} \\ 01010001 & 01010001 \end{array}$$

Additionen und Subtraktionen führen somit zum gleichen Ergebnis. Die Verknüpfung der einzelnen Bits kann mit xor erfolgen. Die Division erfolgt korrespondierend zur klassischen Division, mit dem Unterschied das bei den einzelnen Subtraktionen modulo 2 gerechnet wird. Dadurch fallen Überträge weg und die Berechnung wird wesentlich einfacher. Beispiel:

$$\begin{array}{l} 111010100/101 = 1101110 \\ \text{-}\underline{101} \\ \quad 100 \\ \quad \text{-}\underline{101} \\ \quad\quad 011 \\ \quad\quad \text{-}\underline{000} \\ \quad\quad\quad 110 \\ \quad\quad\quad \text{-}\underline{101} \\ \quad\quad\quad\quad 111 \\ \quad\quad\quad\quad \text{-}\underline{101} \\ \quad\quad\quad\quad\quad 100 \\ \quad\quad\quad\quad\quad \text{-}\underline{101} \\ \quad\quad\quad\quad\quad\quad 010 \\ \quad\quad\quad\quad\quad\quad \text{-}\underline{000} \\ \quad\quad\quad\quad\quad\quad\quad 10 = \text{Rest} \end{array}$$

Für den CRC-Check wird ein Bit-String (oder Polynom) der Länge k gewählt. Dabei muss das erste und das letzte Bit eine 1 sein. Ein Beispiel ist 10011. Dieser Bit-String wird als Generator G bezeichnet. Die eigentliche Nachricht n (bzw. ihr Polynom $N(x)$) wird um $(k-1)$ Stellen nach links geshiftet, also n wird mit2^{k-1} multipliziert (bzw. $N(x) \cdot x^{k-1}$). Diese geshiftete Polynom wird mit Modulo 2 Arithmetik durch G geteilt. Durch die einfache Berechnung der Subtraktionen ist die Division auch einfach zu berechnen. Bei der Division bleibt ein Rest $R < G$ übrig. Von der geshifteten Nachricht wird dieser ermittelte Rest abgezogen (was in modulo 2 gleichbedeutend ist mit addiert). Damit wird die Nachricht durch G teilbar. Dabei werden nur die zugefügten Stellen modifiziert, die Bitfolge der eigentlichen Nachricht bleibt unangetastet. Diese Nachricht kann gesendet werden. Zur Verifikation der Nachricht auf der Empfängerseite ist diese durch G zu teilen. Bleibt ein Rest, ist die Übertragung fehlerhaft.

Hier nochmals zusammengefasst die einzelnen Schritte zur Berechnung der Checksumme:

1. Gegeben eine Nachricht M und ein CRC-Generator G mit Länge k.
2. Shifte die Nachricht M um $k-1$ Bits nach links (füge $k-1$ Nuller an)
3. Dividiere die neue Nachricht durch den Generator (mit Modulo 2 Arithmetik)
4. Addiere (bzw. Subtrahiere, ist hier das gleiche) den Rest zur Nachricht dazu (Modulo 2 Arithmetik!)
5. Ergebnis ist die zu sendende modifizierte Nachricht inklusive Checksumme

Diese Nachricht kann folgendermaßen geprüft werden:

1. Dividiere modifizierte Nachricht durch den Generator
2. Ist Rest ungleich Null, ist die Nachricht fehlerhaft.

Das Verfahren soll zunächst in C Programmiert werden. Dazu sind zwei Routinen zu schreiben, eine zum Erzeugen der um die Prüfsequenz erweiterte Nachreicht, eine um eine gegebene Nachricht zu verifizieren. Ein Ansatz basiert auf einfachen logischen Operation. Abbildung 1 zeigt das Prinzip. Dabei wird die obige Division im Modulo 2 System nachgebildet. Interessant ist, dass das eigentliche Ergebnis der Division nicht interessiert. Die Nachricht ist schrittweise Bit für Bit abzuarbeiten. Es bietet sich an ein Schieberegister zu verwenden bzw. in C nachzubilden. Dieses Register hat die Länge des Generators. In dieses Register wird nun die Nachricht Bit für Bit hineingeschoben, angefangen vom obersten Bit der Nachricht. Sobald das oberste Bit des Registers 1 ist, wird das komplette Register mit dem Generator Bitweise mit XOR verknüpft (in C gibt es einen entsprechenden Operator). Dann wird das nächste Bit der Nachricht eingeschoben, das erste Bit fällt dann aus dem Register heraus.

Dieses Bit ist immer 0, da das oberste Bit des Generators immer 1 ist, und eine XOR Verknüpfung nur stattgefunden hat, wenn dieses Bit 1 war.

Es zeigt sich somit, das das erste Bit des Generators, das ja immer 1 ist, weggelassen werden kann. Dann muss allerdings auch das Register um das oberste Bit gekürzt

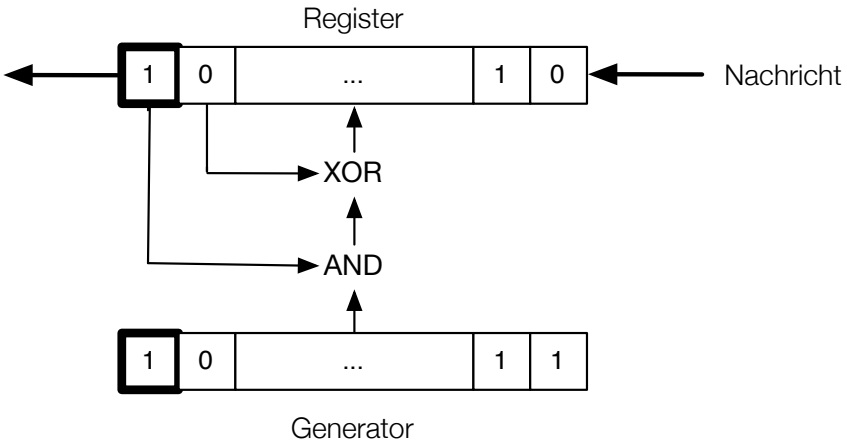

Abbildung 1: Prinzip des CRC-Checkers mit Hilfe eines Schieberegisters

werden. Dann entscheidet natürlich nicht mehr das oberste Bit im Register, ob eine xor-Verknüpfung stattfindet, sondern das Bit was gerade aus dem Register herausgeschoben wurde.

Nach der Nachricht ist noch die entsprechende Anzahl Nullen in das Register zu schieben (siehe Schritt 2 des Verfahrens). Dann steht im Register der Rest, dieser kann an die Nachricht angefügt werden.

Teil 2: Aufgaben

Aufgabe 1: CRC-Programm in C

Bereiten Sie für die Übungen ein C-Programm vor, dass eine binär-codierte Nachricht einließt und die CRC Prüfsumme generiert. Außerdem soll das Programm eine Nachricht mit angefügter Prüfsumme auf Fehler prüfen können, d.h. ob die Nachricht fehlerfrei übertragen wurde. Für die Ein- bzw. Ausgabe können Sie die Bibiothek `stdio.h` und die Befehle `scanf` sowie `printf` verwenden.

Zur Minimierung des Schreibaufwandes können Sie von einem festen Generatorpolynom $G = 10011$ ausgehen und die Nachricht konstant auf 12-Bit setzten. Beachten Sie, dass bei der Prüfung eine Nachricht 16-Bit einlesen werden müssen. Sie sollten bei der Programmierung Schieberegister verwenden.

Zum Test Ihres Programms können Sie folgende Nachrichten verwenden (die letzten 4 Bit sind die Prüfsumme):

- 101010101010 0101
- 101110101101 0011

- 011010110101 1010
- 011010111111 0111

Aufgabe 2: CRC-Programm auf Nios II laden

Laden Sie nun Ihr C-Programm auf den Nios II. Sie müssen zuerst den Softcoreprozessor wieder über Quartus II auf das Board spielen. Unter Quaturs II können Sie direkt den Programmer öffnen und dort Ihre Datei `lights.sof` hinzufügen. Mit `Start` wird dann der FPGA programmiert.

Nun können Sie, wie auf dem letzten Übungsblatt, mit der Programmierung unter dem Programm Nios II fortfahren.

Teil 3: Lösungen

Aufgabe 1: C-Programm zur Generierung/ Prüfung von CRC

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define POLY 19  // Festes Generator Polynom 10011 in DEC
4
5  int crc = 0; // Integer als Schieberegister (Verwendung der letzten 5 Bit)
6
7  int main(void)
8       {
9       char whattodo = ' ';
10      while((whattodo != 'G') & (whattodo != 'P'))  //CRC erzeugen oder prüfen
11      {
12      printf("\nCRC (G)enerieren oder (P)ruefen? ");
13      scanf("%c",&whattodo);
14 }
15 int count;
16      int datastream[16];
17      if (whattodo == 'G')
18      {                                 //Bei Erzeugung sind die letzten 4 Elemente 0
19           count=12;
20           datastream[12] = 0;
21           datastream[13] = 0;
22           datastream[14] = 0;
23           datastream[15] = 0;
24      }
25      else
26      {
27           count=16;
28      }
29      printf("\nBitte Daten eingeben (%d Bit)\n",count); //Eingabe der Daten
30      int i;
31      for(i = 0; i < count; i++) //Bei Erzeugung 12, bei Prüfung 16
32      {
33           printf("%2d> ",i+1);
34           scanf("%d", &datastream[i]);
35      }
36      printf("Datenstrom: "); //Datenstrom ausgeben
37      for(i=0; i<16; i++)
38      {
39           printf("%d",datastream[i]);
40      }
41      printf("\n");
42      int j;
43
44      for(j=0; j<5; j++) //Schieberegister mit den ersten 5 bit des Datenstroms fuellen
45      {
46           crc<<=1;
47           if(datastream[j]==1) crc |= 1;
48      }
49      int k;
50      for (k=j; k<=16; k++) //Array (Datastream) durchlaufen
51      {
52           if (crc & 16) // Wenn das erste Bit eine 1 ist
53           {
54                crc ^= POLY; //Dann CRC XOR mit dem Polynom
55           }
```

```
56            crc<<=1; //Register schieben
57            if(datastream[k]==1) crc |= 1; //und naechstes Bit holen
58        }
59        crc &= 15;
60        if (whattodo == 'P') //Ausgabe bei Pruefung
61        {
62            if(crc == 0)
63            {
64                printf("Datenstrom nicht fehlerhaft\n");
65            }
66            else
67            {
68                printf("Datenstrom fehlerhaft , CRC Pruefung nicht bestanden\n");
69            }
70        }
71        else
72        {
73            printf("Das CRC des Datenstroms ist: ");  //Ausgabe bei Generierung
74            int l;
75            if(crc & 8) printf("1"); else printf("0");
76            if(crc & 4) printf("1"); else printf("0");
77            if(crc & 2) printf("1"); else printf("0");
78            if(crc & 1) printf("1"); else printf("0");
79            printf("\n");
80            printf("Damit ist die Nachricht mit Pruefsumme: ");
81            for(l=0; l<12; l++) {printf("%d",datastream[l]);}
82            if(crc & 8) printf("1"); else printf("0");
83            if(crc & 4) printf("1"); else printf("0");
84            if(crc & 2) printf("1"); else printf("0");
85            if(crc & 1) printf("1"); else printf("0");
86            printf("\n");
87
88        }
89    return EXIT SUCCESS;
90    }
```