DETECÇÃO DE ERROS EM MENSAGENS

CHECAGEM CRC

Leonardo Trevisan Gandolfo e Victor Gouveia de Menezes Lyra

Equipe de Controle

Recife
2019

Sumário

1 Introdução **1**
1.1 O que é o CRC 1
1.2 Por que usar o CRC 1
1.3 Como funciona o CRC 1

2 Implementação do CRC **3**
2.1 Implementação no código do transmissor e receptor 3

1 Introdução

1.1 O que é o CRC

O CRC (*Cyclic Redundancy Check*), verificação cíclica de redundância, é um método de detecção de erros que podem ocorrer em uma cadeia de dados. Sendo assim, é possível garantir a integridade dos dados enviados a partir de um sistema de transmissão.

É amplamente utilizado pela facilidade de implementação e de análise matemática. Sua estrutura consiste de uma divisão polinomial, cujo resultado é anexado ao final da mensagem. Quando a mensagem é recebida, a mesma divisão é efetuada e dependendo do resultado dessa operação, é possível detectar erros.

Vale salientar que existe a possibilidade de erros não serem detectados, o que é esperado, visto que esta é a natureza fundamental da verificação de erros. Quanto a isso, o indicado é que polinômios primitivos sejam utilizados como geradores, o que garante um grau de confiabilidade maior.

1.2 Por que usar o CRC

Durante a transmissão entre os rádios dos robos e o computador foi detectada uma grande quantidade de erros, que podem ser provenientes do interfaceamento entre o computador e a comunicação serial ou da distância e interferência entre os rádios. Por isso, fez-ze necessária a implementação de mais medidas de segurança.

O motivo da escolha do CRC foi por sua simplicidade e eficiência, assim como o fato de sua aplicação intendida ser muito parecida ao aplicado neste caso.

1.3 Como funciona o CRC

Existem diversos tipos de CRC, classificados de acordo com a quantidade de bits, assim como o polinômio gerador.

O gerador será nada mais que o divisor da operação descrita a seguir.

No exemplo utilizaremos uma mensagem de 14 bits com um CRC de 3 bits. Para isso o polinômio deve ser de 3ª ordem, que possui quatro coeficientes e precisa ser escrito em binário. No caso escolhemos "1011". Dessa forma o resultado do calculo terá 3 bits.

Codificaremos a seguinte mensagem:

$$\boxed{11010011101100} \tag{1}$$

$$\begin{array}{l} 11010011101100000 \\ 1011 \\ \hline 01100011101100000 \end{array}$$

Note que na primeira linha colocamos a mensagem deslocada para direita com 3 zeros (tamanho do CRC), abaixo o polinômio gerador. A operação realizada é um OU-exclusivo bit-a-bit (XOR). Os bits restantes são diretamente copiados para o resultado daquele passo. O divisor é então deslocado para a direita em 1 bit e o processo é repetido até que divisor alcance o ultimo bit da mensagem.

A seguir vemos o calculo completo:

$$
\begin{array}{l}
11010011101100000 \\
1011 \\
\hline
01100011101100000 \\
\ \ 1011 \\
\hline
00111011101100000 \\
\ \ \ 1011 \\
\hline
00010111101100000 \\
\ \ \ \ 1011 \\
\hline
00000001101100000 \\
\ \ \ \ \ \ 1011 \\
\hline
00000000110100000 \\
\ \ \ \ \ \ \ 1011 \\
\hline
00000000011000000 \\
\ \ \ \ \ \ \ \ 1011 \\
\hline
00000000001110000 \\
\ \ \ \ \ \ \ \ \ 1011 \\
\hline
00000000000101000 \\
\ \ \ \ \ \ \ \ \ \ 1011 \\
\hline
00000000000000100
\end{array}
$$

Paramos a operação quando obtemos um resto de 3 bits. Perceba que a operação é feita de forma que o divisor esteja sempre abaixo do primeiro bit 1 do dividendo (por que o quociente da operação para aquele passo seria 0), como ocorre no caso.

Sendo assim, temos em mão o CRC, neste caso "100", com isso, o anexamos ao fim de mensagem, da seguinte forma:

$$\boxed{11010011101100100} \tag{2}$$

Após a transmissão da mensagem acima, o receptor realizará o mesmo processo para validá-la:

$$
\begin{array}{l}
11010011101100100 \\
1011 \\
\hline
01100011101100000
\end{array}
$$

A operação segue como a anterior, teremos dois casos:

- **Caso o resto seja ZERO:** a mensagem foi transmitida sem erro e pode ser usada.
- **Caso o resto seja DIFERENTE DE ZERO:** houve erro durante a transimssão da mensagem e ela não pode ser utilizada.

2 Implementação do CRC

A utlização do CRC no projeto do F-180 será dada em 3 etapas, ele será incluido na mensagem na interface entre o computador e a comunicação serial da TIVA, o que é feito hoje através de um programa em Java. Os trabalhos em uma interface feita em C++ serão iniciados em breve. As duas etapas restantes se referem aos códigos "CodigoTransmissor" e "CodigoRobo" que são códigos implementados nas TIVAS conectadas aos rádios transmissor e receptor, sucessivamente.

Para isso foi utilizada a biblioteca **Fast CRC**. Ao analisar os tempos de cada modo de CRC 16 bits foram encontrados os seguintes resultados:

CCITT	->	20 microsegundos
MCRF4XX	->	20 microsegundos
MODBUS	->	24 microsegundos
KERMIT	->	20 microsegundos
XMODEM	->	16 microsegundos
X.25	->	20 microsegundos

Dessa forma, foi decidido utilizar o XMODEM pois obteve nos testes o melhor resultado de tempo.

2.1 Implementação no código do transmissor e receptor

Ao anexar o CRC no final da mensagem na primeira etapa basta que o resultado da operação seja zero nas etapas seguintes para que ela seja validada.

Com isso em mente, foi criada uma função *bool* que retorna *true* caso o resultado for zero e *false* caso o resultado for diferente de zero.

```
bool crc_check(char msg[SIZE_MSG]){
    uint8_t msg_crc[SIZE_MSG];
    for(int i=0; i < 7; i++){
        msg_crc[i] = msg[i];
    }

    return (CRC16.xmodem(msg_crc, sizeof(msg_crc)) == 0);
}
```

É importante notar que a operação feita pela biblioteca *Fast CRC* só pode operar em *arrays* uintx_t e como a mensagem transmitida pelo rádio é de tipo *char* é necessário fazer a conversão.

Com o resultado desta função sabemos se a mensagem é segura, caso seja, a mensagem é utlizada normalmente, caso contrário, ela é descartada.