NVS- Projekt 2020/21, 5AHIF

MLT3- Simulator

Maurice Putz

8. Jänner 2021

Inhaltsangabe

1 Beschreibung

Der MLT-3 Simulator dient einer Simulierung der Leitungskodierung MLT-3. Vom Sender werde zuerst ASCII-Zeichen ins Dezimal System ungewandelt, dann weiter in binären Code und dann mittels der MLT-3 Signalform welche aus den binären Teilen besteht, übertragen. Die MLT-3 codierte Datenfolge wird nun beim Empfänger zurück ins binäre System übersetzt, danach ins dezimale und schlussendlich wieder als ACII-Zeichen dekodiert ausgegeben.

Das Programm gibt diesen Vorgang standardgemäß auf der Konsole als Tabelle formatiert aus. Es exisiteren weitere Optionen und Funktionen, welche in Abschnitt "Bedienanleitung", genauer beschrieben sind.

1.1 MLT-3 Zusammenfassung

MLT steht für Multilevel Transmission Encoding, die drei am Ende steht für "3 levels", also die drei Spannungsformen welche eine MLT-3 kodierte Datenfolge haben kann. Diese drei Spannungen, was die Folge zu einem Ternären Signal macht, werden in einer Folge meisten als +, 0 und - dargestellt, wie man gut in Abb. 1 erkennen kann.

MLT-3 ändert den Datenstrom bei jeder logischen Eins, bei einer logischen Null ändert sich der Zustand der Leitung nicht und das gleiche Zeichen wird einfach ein weiteres Mal übernommen. Dies bringt vor allem der Vorteil, dass die Kodierung viel weniger Bandbreite als andere wie zum Beispiel NRZ (No Return to Zero) benötigt, da eben nur bei einer Änderung mit einer logischen Eins, die Spannung geändert wird.

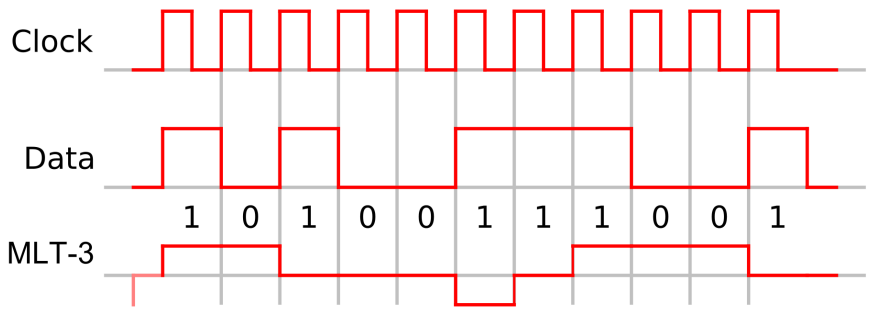

Abb. 1: Clock dient zur Markierung der Übergangszeitpunkte

Die Spannung ändert sich wie bereits erwähnt jedes Mal wenn eine logische Eins verabreitet wird. Diese Änderung variert folgender Reihenfolge:

1. Von 0 auf +
2. Von + auf 0
3. Von 0 auf -
4. Von - auf 0

Diese vier Schritte wiederholen sich jedes Mal bei einer logischen Eins in genau dieser Reihenfolge.

2 Bedienanleitung

ASCII data to send	Binary	MTL-3	Binary encoded from MLT-3	Data received in ASCII
a	01100001	0+00000-	01100001	a
d	01100100	0+000---	01100100	d
d	01100100	0+000---	01100100	d
a	01100001	0+00000-	01100001	a

Abb. 2: Ausgabe mit Aufruf .mlt3send dsad

Durch den bloßen Aufruf des Programms, wie zum Beispiel ./mlt3send, wird der Prozess ohne speziellen Funktionen gestartet. Eine zufällige Anzahl von ASCII-Zeichen wird zufällig oft (zwischen 1 und 127 mal) auf der Konsole als Tabelle formatiert ausgegeben. Die Tabelle zeigt das gewählte ASCII Zeichen, die binäre Kodierung dazu, dann den MLT-3 Übertrag, dann diesen als binär wieder zurückübersetzt und schließlich in der letzten Spalte, das dekodeirte ASCII-Zeichen wieder. (Abb. 2)

Wenn mann beim Aufruf des Programms, wie zum Beispiel ./mlt3send dsad eingibt, werden die einzelnen Zeichen hinter dem Programmnamen überprüft, ob sie in ASCII enthalten sind und im positiven Fall, wird das Programm genau wie bei einem einfachen Aufruf gestartet. Nur diesmal werden bloß die ASCII-Zeichen, welche hinter dem Programmnamen stehen zufällig oft (zwischen 1 und 127 mal) übertragen.

- Eingabe: ./mlt3send dsad
- Ausgabe: [Siehe Abb. 2]

```
MLT-3 Encoding
Usage: ./mlt3send [OPTIONS] [input_characters]

Positionals:
  input_characters TEXT       Given characters will be random times send over with MLT-3    Example: "./mlt3send asdf"

Options:
  -h,--help                   Print this help message and exit
  -o,--filechar TEXT          Writes the ouput of MLT-3 process with given characters as markdown in a file
  -f,--file                   Writes the ouput of MLT-3 process as markdown in a file
  -t,--time                   Time measurement of sending until receiving data
  -c,--color                  Standard table output with color
  -a,--allowed                Show allowed character for input
```

Abb. 3: Hilfe des Programms in der Kommandozeile

Option **-h,–help**:
Zeigt die Hilfe für das Programm an, liefert das Menü wie in Abb. 3 zurück.

- Eingabe: ./mlt3send -h
- Ausgabe: [Siehe Abb. 3]

Option **-o,–filechar TEXT**:
Startet das Programm und schreibt das Ergebnis als Markdown in eine Datei, Speicherort der Datei wird nach Aufruf auf der Konsole ausgegeben. Erwartet ASCII kodierbare Zeichen als Eingabe. Beispiel:

- Eingabe: ./mlt3send -o asdf
- Ausgabe:

  [NOTE]
  The result is saved to the location: /home/maurice/Desktop/mlt3.mkd
  It is formated as markdown.

Flag **-f,–file**:
Startet das Programm mit zufällig gewählten ASCII-Zeichen und schreibt sie als Markdown in eine Datei, Speicherort der Datei wird nach Aufruf auf der Konsole ausgegeben. Beispiel:

- Eingabe: ./mlt3send -f
- Ausgabe:

  [NOTE]
  The result is saved to the location: /home/maurice/Desktop/mlt3.mkd
  It is formated as markdown.

Flag **-t,–time**:
Startet das Programm und gibt das Ergebnis auf der Konsole aus. Zusätzlich wird am Ende die gemessene Zeit in Nano und Mikrosekunden ausgegeben. Der Grund für die beiden Einheiten ist, dass manchmal, wenn zufällig und ein Zeichen übertragen wird, oder der Computer sehr schnell ist, auf dem das Programm ausgeführt wird, dies unter einer Mikrosekunde passiert.

Die gemessene Zeit wird mit Start des Sende-Threads angefangen, und endet mit der Verarbeitung aller Daten durch den Empfängerthread. Die Ausgabe in die Konsole oder das schreiben in eine Datei wird nicht mitgemessen, jedoch die Erstellung der Tabelle für die weitere Ausgabe schon.

Beispiel:

- Eingabe: ./mlt3send -t
- Ausgabe:

  (Tabelle mit den Werten wie in Abb. 2)
  .
  .
  .
  Elapsed time in nanoseconds : 1058770 ns
  Elapsed time in microseconds : 1058 µs

Flag **-c,–color**:
Startet das Programm normale mit zufällig gewählten ASCII Zeichen und gibt die Tabelle färbig aus.
Achtung: kann je nach Terminal-Anwendung und Computer zu schlechter Performance bei der Ausgabe führen. Die Geschwindigkeit der Übertragung wird allerdings nicht beeinflusst.

- Eingabe: ./mlt3send -c
- Ausgabe: Tabelle wie in Abb. 2 nur farbig

Flag **-a,–allowed**:
Damit kann man alle erlaubten ASCII Zeichen für das Programm sich anzeigen lassen. Einige ASCII Zeichen wie zum Beispiel 1 - SOH (start of heading), in C++ nicht über die Kommandozeile gewünscht verarbeitet werden kann. Alle Zahlen und Buchstaben und die meisten welche in ASCII enthalten sind, werden unterstützt.

- Eingabe: ./mlt3send -a
- Ausgabe: Tabelle, wo auf der linken Seite Dezimal-Werte stehen und auf der rechten Seite die jeweiligen zugerhörigen ASCII Zeichen.

3 Umsetzung

Das Programm legt nach dem Parsen ein Objekt, einer selbst erstellten Klasse einer Thread-Safe Queue an. Anfangs war nicht sicher welchen Typ der Queue unterstützen sollte, demnach wurden Templates verwendet. Die Implementierung der Thread-Safe Queue Klasse befindet sich im indlude Verzeichnis in der Datei queue.h. Sie wurde im Header definiert, da die Verwendung von Templates keine Aufteilung der Implementation in eine .cpp Datei zuließ.

Danach wird ein Thread sender erstellt:

```
1 thread sender{send_data_tf, random_tf(ref(main_table),
2              input_chars), ref(q), ref(main_table)};
```

Dieser Thread ruft eine Funktion:

```
1 void send_data_tf(vector<int> data_to_send,
2                   Queue<string>& queue, Table& tab)
```

Diese Funktion bekommt einen vector vom Typ int, eine Referenz zur vorhin angelegten Queue vom Typ string und eine Referenz zu einer vorhin angelegten Tabelle aus der externen Bibliothek tabulator.h. Die Funktion liefert nichts zurück, der vector enthält die zufällig erstellten ASCII-Zeichen in Dezimaldarstellung. Diese integer, welche einen Wert zwischen 33 und 127 haben können (alle zulässigen ASCII-Zeichen), werden in der Funktion zuerst auf einen Hilfsvector in ihrer binären Darstellung gepusht und dann mittels einer Funktion als MLT-3 kodiert auf den Queue als string gepusht.

Die Funktion dazu:

```
1 string convert_to_mlt3(bitset<8> binary_block)
```

Diese liefert einen string zurück und benötigt eine bitset mit 8 Stellen als Parameter. In der Funktion send_data_tf(...), wird diese Funktion hier für jedes Zeichen einmal aufgerufen. Der zurückgelieferte string, welcher zum Beispiel so aussehen kann: 0+00—0, wird auf die Queue gepusht.

Der Thread sender benutzt um die Werte für den ersten Paramter Funktion send_data_tf(...) zu bekommen selbst auch eine Funktion:

```
1 vector<int> random_tf(Table& tab, string allowed="")
```

Diese Funktion liefert den Vektor vom Typ int zurück welcher in der Funktion send_data_tf(...) benötigt wird. Sie bekommt selbst eine Referenz auf die Tabelle für die Ausgabe später mit und einen string names "allowed". Dieser ist per default ein Leerstring. Falls dem Programm beim Start entsprechende Zeichen mitgegeben werden, wird nur aus dieser Zeichenmenge dann eine zufällige Anzahl davon in den Vektor eingefügt. Insgesamt können insgesamt Zeichen zwischen 1 und 127 mal in den Vektor eingefügt werden, dies hat keinen speziellen

Grund, man könnte theoretisch das obere Limit an Zahlen auch bis zum maximalen Vektor-Größe Limit setzen. Jedoch wäre das nicht sinnvoll, da die Ausgabe in der Kommandozeile sehr lange dauern würde und andere Probleme mit dem Speicher auftreten würden, bei Interesse sind dazu weiter unten Links.

Die maximale vector Größe könnte man wie folgt feststellen:

```
std::vector<int> myvector;
std::cout << "max_size: " << myvector.max_size();

//Moegliche Ausgabe:

max_size: 1073741823
```

Nach diesem Prozess wird ein andere Thread gestartet:

```
thread receiver{decode, ref(q), ref(main_table)};
```

Dieser Thread namens receiver, ruft folgende Funktion auf:

```
void decode(Queue<string>& queue, Table& tab)
```

Diese Funktion decode(...), liefert nichts zurück. Ihre Aufgabe ist, den Queue in einer while-Schleife abzuarbeiten, jedes Element zuerst von MLT-3 mittels einer Hilfsfunktion ins binäre zurück, und dann schließlich als ASCII-Zeichen zu übersetzen.

Die Hilfsfunktion:

```
decrypt_from_mlt3(string mlt3_data, Table& tab, int table_col_cnt)
```

Sie wird für jedes Element im Qeue einmal aufgerufen, der Aufruf geschieht in der vorherig dargestellten Funktion decode(...). Sie bekommt beim Aufruf die Referent auf die Tabelle von der Bibliothek tabulator.h mit, damit sie die Werte in die jeweillige Spalte und Reihe eintragen kann, da das Element nach der Abarbeitung vom Queue gelöscht wird. Weiters bekommt die Funktion natürlich einen string names mlt_data mit, welchen sie zuerst ins binäre, und dann mittels einem sstringstream aus der standard-Bibiothek, in eine bitset mit acht Stellen unwandelt und schließlich zurück in eine ASCII-Zeichen.

3.1 Verwendete Externe Klassen

- rang-library: für die farbige Ausgabe der Tabelle
- CLI11-library: für die Verarbeitung von Eingabe und Options- und Funktionsargumenten
- spdlog-library: für das Logging
- tabulate-library: für die Formatierung der Ausgabe als Tabelle

4 Quellen

- https://de.wikipedia.org/wiki/MLT-3-Code
- https://de.wikipedia.org/wiki/Ternäres_Signal
- https://www.cplusplus.com/reference/vector/vector/max_size/
- Link zu Limit von vektoren: https://stackoverflow.com/questions/32316346/limit-on-vectors-in-c
- Folie 21. "Encoding and Decoding" aus dem NVS Unterricht