**IP-DVB Multiplexer/Demultiplexer**

May 28, 2021

Ali Taatian

**1) MUX.cpp (multiplexer)**

In this multiplexer, 6 different classes have been used:

* **Data-buffer**: which performs data reading from the desired source (file, source, IP or Ethernet, . . .) and controls the source data. In the test mode, data is called from the file "in.txt.” This is done by **take-byte-from-file()** method.

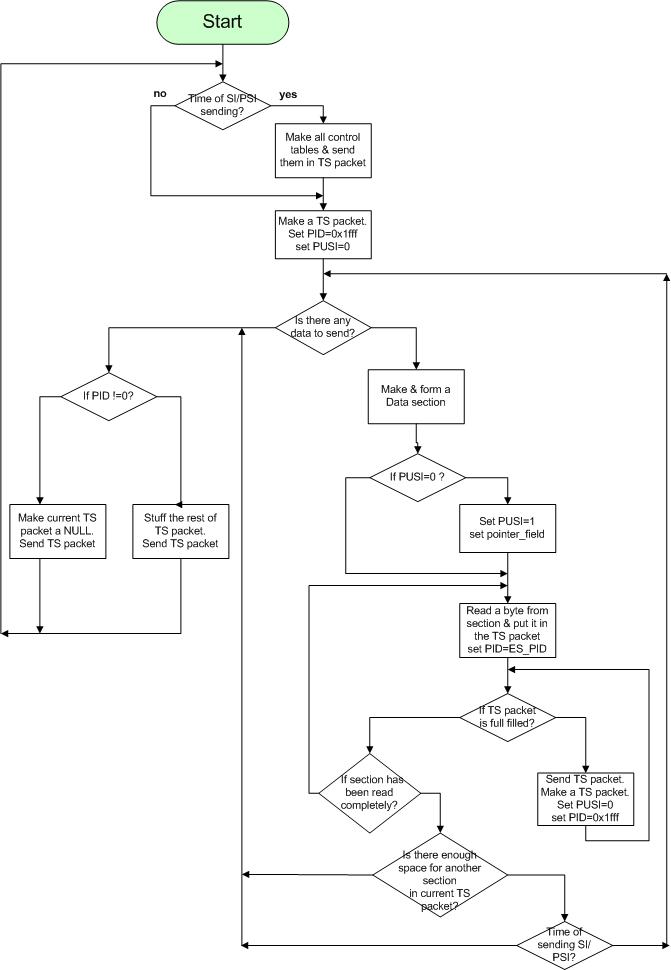
**1-1) Classes constructing “Section”:**

These classes have their own buffer, and each **section** that is created is completely placed in their buffer.

* **Data-sec**: This class is tasked with producing data-section in DSM-CC format. And with the **fill-from-data-buffer()** method, it performs this action and values the different fields.
* **PAT-SEC**: This class is responsible for building the **PAT** **section**. In this PAT, a program (number 0x0001) is set and its PMT is allocated. In addition, the 0x0000 number program is also listed for NIT.
* **PMT-SEC**: This class is responsible for making **PMT Section**. For our service, IP datagram, there is only one Elementary Stream. And Stream-type=0x0d is used.
* **SDT-SEC**: Used to produce **SDT Section**. In this **section**, **data-broadcast-descriptor** is sent.
* **NIT-SEC**: Used to produce **NIT section**. (Under Schedule No. 0x0000)
* **TSpacket**: This class makes 188 bytes TS packets. In this way, PAT, PMT, SDT or Data Sections are inserted in these TS packets, according to standard 1-13818. Then each TS that is made complete is sent (written) to the outflow file by **Send-ts()** method. The exit file is called "mux.txt". When the whole source is read, the **Enough** flag is set, TS packet is no longer produced, and the program ends.

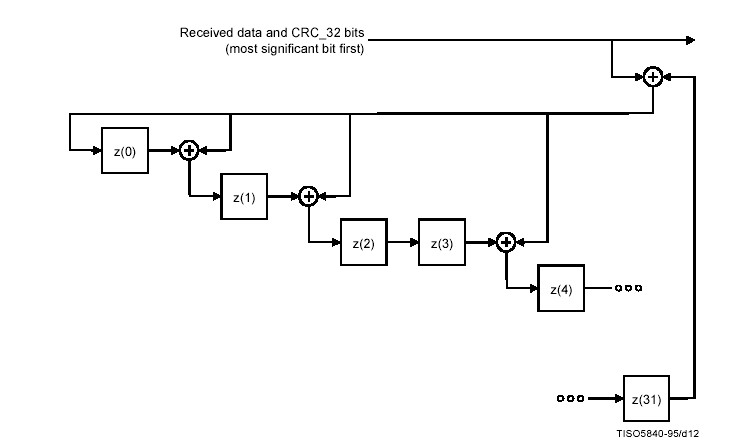
The method of sending data is that at the beginning of the steam, all of each PSI or SI table to be sent, such as PAT, PMT, SDT and NIT, are sent. Then sending **Data** begins. At the same time, after sending a few data sections, **SI** and **PSK** tables will be sent again. **SI-sending-interval** commands the start of the sending. For example, it can send all SI and PSI tables once for every 50 **data.section**s.

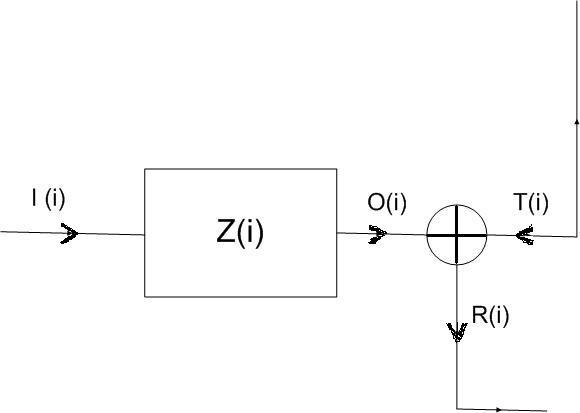
**1-2) MUX flowchart**



**1-3) CRC-32 encoder/decoder**

According to annex A of standard 13818-1, Generating Functions are used. The algorithm mentioned in this annex is for the decoder. But it can also be used as an encoder. In this way, before entering the 4 bytes of CRC-32, the **z(i)** outputs must be read. In this case, this output will be the same as the CRC-32. According to the following diagram, **z(i)**sare delay units.





Therefore, if we consider the two times t and t+1, the following algorithm will be executed by as t increases:

P(t)= s(t) ^ O[31] (t)

T(t)= p(t) \* Poly

R(t)= O(t) ^ T(t)

I [k+1] (t) = R[k] (t)

I [0] (t) = p(t)

O(t+1)= I(t)

with initial value: O(t+0) = 1.

**Poly** is a 32-element array which is filled with the polynmial = 0x4C11DB7 and is fixed. In order to determine that the CRC is correct in the decoder, the value O[k] for all Ks must be zero after the arrival of the last bit of the string (i.e. the last bit of CRC-32).

In the encoder, after entering all string bits, the value of O[k], for k= 0 to 31, shows the values of CRC-32 bits.

**2) DEMUX.cpp (demultiplexer)**

The classes used in this demultiplexer are as follows:

* **TSpacket**: creates TS packets and is filled by **readTS-fromfile()**. (From the input file, mux.txt)
* **PAT-SEC**: fills the PAT Section from the existing TS using **fillfromTS()** method, and its fields are set when completed (checked by **PAT-is-made()** method),.
* **PMT-SEC**: creates the **PMT** table.
* **Data-Sec**: creates the **Data Section,** and writes its content to demux.txt using **FillToFile()** method.
* **Program**: holds the parameters of the program (e.g. **PMT-PID**), and is set by a **PAT-SEC** object.

**2-1) Executing the code**

First, the code searches TS packets with PID = 0 to find **PAT**. After constructing **PAT**, it searches PMT to find **ES-PID**. After obtaining the parameters of the stream from the **SI/PSI** tables, the code starts reading **Data Section**s and writes the containing data to the output file.

When reading **Data Section**, the code first reads the **LLC\_SNAP\_flag** and writes the data (IP datagram) to the output file based on the existence of LLC encapsulation.

Meanwhile, while reading the data, if in the stream, it reaches to **SI/PSI** tables once again, it reads them and runs Update, if changed.