American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS)

ISSN (Print) 2313-4410, ISSN (Online) 2313-4402

© Global Society of Scientific Research and Researchers

http://asrjetsjournal.org/

A Singular Function Algorithm for Packing the Protocol Description Unit Packet

Kumbirai T. Mukosera[a]*, Gilford Hapanyengwi[b], Ben Nyambo[c], Emmanuel Mashonjowa[d]

[a,b,c]*Computer Science Department, Faculty of Science, University of Zimbabwe, Harare, Zimbabwe*

[d]*Physics Department, Faculty of Science, University of Zimbabwe, Harare, Zimbabwe*

[a]*Email: ktmukosera@gmail.com*, [b]*Email: babaticha@gmail.com*

[c]*Email: bennynyambo@gmail.com*, [d]*Email: emash@science.uz.ac.zw*

Abstract

A new sequential singular function algorithm for packing Protocol Description Unit packets in the Short Message Mobile Originated Point-to-Point service is proposed. The single function algorithm simplifies the Short Message Entities programming where ATtention Commands are used to link Switched on Mobile Stations with Service Centers. This is in mode 0 of the short message service transmission. This paper shows that by using, only one single string function, the "*Mid*" or "*Substring*" function, it is still possible and is simpler to encode all the metadata contained in the Transport Protocol Data Unit frame of the Submit Protocol Description Unit packet. Our results show that, this proposed algorithm, always outputs 100% perfectly accurate packets for any combination of header and user data payload submitted in hexadecimal octets or decimal semi-octet format. It is thus possible for programmers not only to manipulate the distinct fields of the header data but also to modify the metadata using that single function only.

Keywords: Algorithm; AT+CMGF=0; Mid String; PDU; SMS SUBMIT; Substring.

1. Introduction

Short Message Point-to-Point Services (SMS) is an integral facility of the second generation (2G) Global System for Mobile (GSM) and its subsequent evolution generations [1]. It does not only give users a substitute way of transmitting in a cellular communication platform like GSM but it can at times be the only available option due to its low demand for strong signal strength to communicate [2].

---

* Corresponding author.

Its operation between a Short Message Entity (SME) and a Short Message Service Centre (SMSC) is managed by an inbuilt asynchronous modem that is controlled via AT (ATtention) commands [3].

1.1. The Text Mode and Protocol Description Unit (PDU) mode

Only two modes of sending or receiving SMS exist on the GSM platform when using an AT commanded ETSI compliant asynchronous GSM modems for data uplink or downlink [4]. These two methods are, *"the Text mode"* and *"the Protocol Description Unit (PDU) mode"*. The *text mode* is the simplest to use and it is just but a higher level of the encoding of the bit streams in the PDU packet. It is activated when an "AT+CMGF=1" command is send to the GSM modem [5]. Unfortunately, the text mode, though easy to program, is limited in flexibility when it comes to how it can structure the different elements of the SMS content. Moreover, there is no guarantee that the text mode will be supported by any given mobile phone [5]. This is unlike with the PDU format which is mandatorily supported on any SME by GSM / ETSI standards [1]. Furthermore, alphabets of the text mode can differ and therefore the displayed messages can display differently from the original message which does not happen with the PDU format [6]. The contents are thus standardized when extracted under any operating system environment. PDU mode also gives the capability to specify the desired destination SME number [4], the period the SMS can be stored in the network suppose the SME device is not reachable and request for delivery reports can also be encoded in the message content [7]. Similarly, a message received in PDU format can display other specifications like the sending SME date and time, number and type of content so that the decoding process can be as accurate as intended. The PDU mode is activated when an "AT+CMGF=0" command is send to the GSM modem. Given the foregoing facts then, of the two methods of sending SMS from an active SME via an SMSC that are available, the most comprehensive is the Protocol Description Unit (PDU) format [5]. The PDU mode therefore, can be visualized as a management mode of all SMS data elements. It is more structured in the construction of the entity of the message when either sending or receiving an SMS. This is why of the two modes, this research prefers to focus on the PDU packet in more depth. As such, in this research we try to come up with only one function that will attempt to make it is as equally easy to implement the PDU packet from source code, as it is with the text mode to handle. In this process however, we still want to leave the programmer with the rich features inherent in the PDU frame design at the same time.

2. Review of the PDU packet

The SMS submit PDU / Short Message Mobile Originated Point-to-Point (SMMO) [8] packet has two main parts which are the Service Centre Address (SCA) and the Transport Protocol Data Unit (TPDU). TPDU are the transport layer datagrams that move data packets between the SMEs and the SMSCs. It is mandatory for a response message to be sent back whenever a TPDU moves between these two systems [9]. The relationship between the SMS-SUBMIT PDU and the SCA and TPDU is given in figure 1 below. Not all SME support the full SMS submit PDU format. Some support only the TPDU format where there is no SCA in the PDU packet [3]. These SME will use the SCA which will have been stored in the SIM card by the network operator. This difference then leads to two types of PDU packets.

The packet from the SME which supports only TPDU format would have the SCA part replaced with a 00

Hexadecimal octet [4].

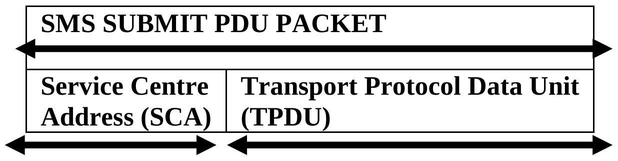

SMS SUBMIT PDU PACKET
Service Centre Address (SCA)
Transport Protocol Data Unit (TPDU)

Figure 1: SMS SUBMIT PDU structure

A typical encoded PDU packet which supports SCA would look like the one below.

07916213111902F111000C916273335366000000AA0AE8329BFD4697D9EC37

A similar encoded PDU packet which does not support SCA and has TPDU data only is as below

0011000C916273335366000000AA0AE8329BFD4697D9EC37

If the two PDU packet types are compared it is noticed that only starting from the right side going to the left are they alike. This is true going all the way to the header byte of the second PDU. The metadata in the first PDU packet with both an SCA and a TPDU frame can be broken down into the following segments as in table 1.

Table 1: PDU metadata segments

Segment	**Data**	**Field Name**	**Meaning**
1	07	Length of SMSC	There are 7 Octets to follow
2	91	Type of SMSC	SMSC is in international format
3	6213111902F1	SMSC Number	The SMSC number is 26311191201
4	11	First Octet	Defines other settings
5	00	Message ID (TP-MR)	The ID auto generated for that SMS by modem
6	0C	Sending Number Length	The sending number is 12 digits long
7	91	Sending Number format	91 Means international format
8	627333536600	Sender Number	Sender Number is +263733356600
9	00	Protocol Identifier TP-PID	Indicates network protocol and nature of message
10	00	Data Encoding Scheme TP-DCS	00 means user data encoding scheme
11	AA	Message validity TP-VP	Will hold message for maximum amount of time possible
12	0A	TP-UDL	Size of user data in this case 10 characters
13	E8329BFD4697D 9EC37	TP-UD	The message encoded "*hellohello*"

3. The single function *Mid string* method

The ***Mid*** function is defined by Microsoft Corporation as a function that returns a string that will be containing a required specified number of characters from an initial string [10]. The structure of the ***Mid*** function as defined in the Microsoft Developer Network (MSDN) is as follows

Public Shared Function Mid(ByVal Str As String, ByVal Start As Integer, Optional ByVal Length As Integer) As String

If for instance, the ***Mid*** function is called with the value of '***Str***' being = '**abcdefghij**', and the value of '***Start***' being an integer = 3 and the value of '***Length***' being an integer = 5. The function will return the character string '**cdefg**'. This is because it would have moved 3 places into the '***Str***' string thus submitted. These 3 places from the start of the string will get us to the character "**c**" in the string "**abcdefghij**". The function then takes the next 5 subsequent characters inclusive as defined by the '***Length***=5' passed to the function. The function therefore returns a string which has the 5 characters '**cdefg**'. In different programming platforms this function may have different names or other alternatives but the syntax is always the same. For instances this ***Mid*** function is also equivalent to the *substring* function (denoted as "***substr***") in some variants of the C programming languages. The ***substr*** function however, maintains the same number of arguments and syntax in these languages as those defined for the ***Mid*** function [10]. It also returns the same value results as the ***Mid*** function for similar argument value calls. We will therefore continue to use the ***Mid*** terminology as the common reference to such all functions in their respective languages. So, in order to simplify the process of encoding the full PDU packet, this, ***Mid*** function is the only function that we will use to encode the PDU packet. By using only this one function it will be easier to construct the metadata segments of the PDU packet each time that one is programming a code for sending SMS. This is unlike in the cases where several functions are used to create each individual segment of the PDU packet. Several individual functions are used since the data formats are different for each field and one has to use an appropriate function that handles that field format appropriately. However in this proposed case we propose that it is possible to articulate all the fields by just the one versatile function together with our proposed algorithm.

4. The flow chart of the algorithm

Figure 2 shows the flow chart of the algorithm that is proposed. The flow chart will start by receiving a number of the intended recipient of the message. It will also concurrently receive the text of the message that needs to be send. Once it has received those two parameters, the flow control shows what validation decisions will be made on the data that will have been input in the algorithm. The flow control assumes that the recipient number that is entered is a valid mobile phone or ME number and it is in the international format. It also assumes that the input text characters are all from the 7 bit GSM character set. Since the characters are all assumed from the 7 bit GSM alphabet the algorithm does not articulate issues of validating them. To also make the PDU packet simpler it will be assumed that the text to be used as input is less than 160 characters long. This will avoid having to accommodate the splitting of the two PDU packets so that they are sent as two packets which are later joined at the Short Message Mobile Terminated Point-to-Point (SMMT) end.

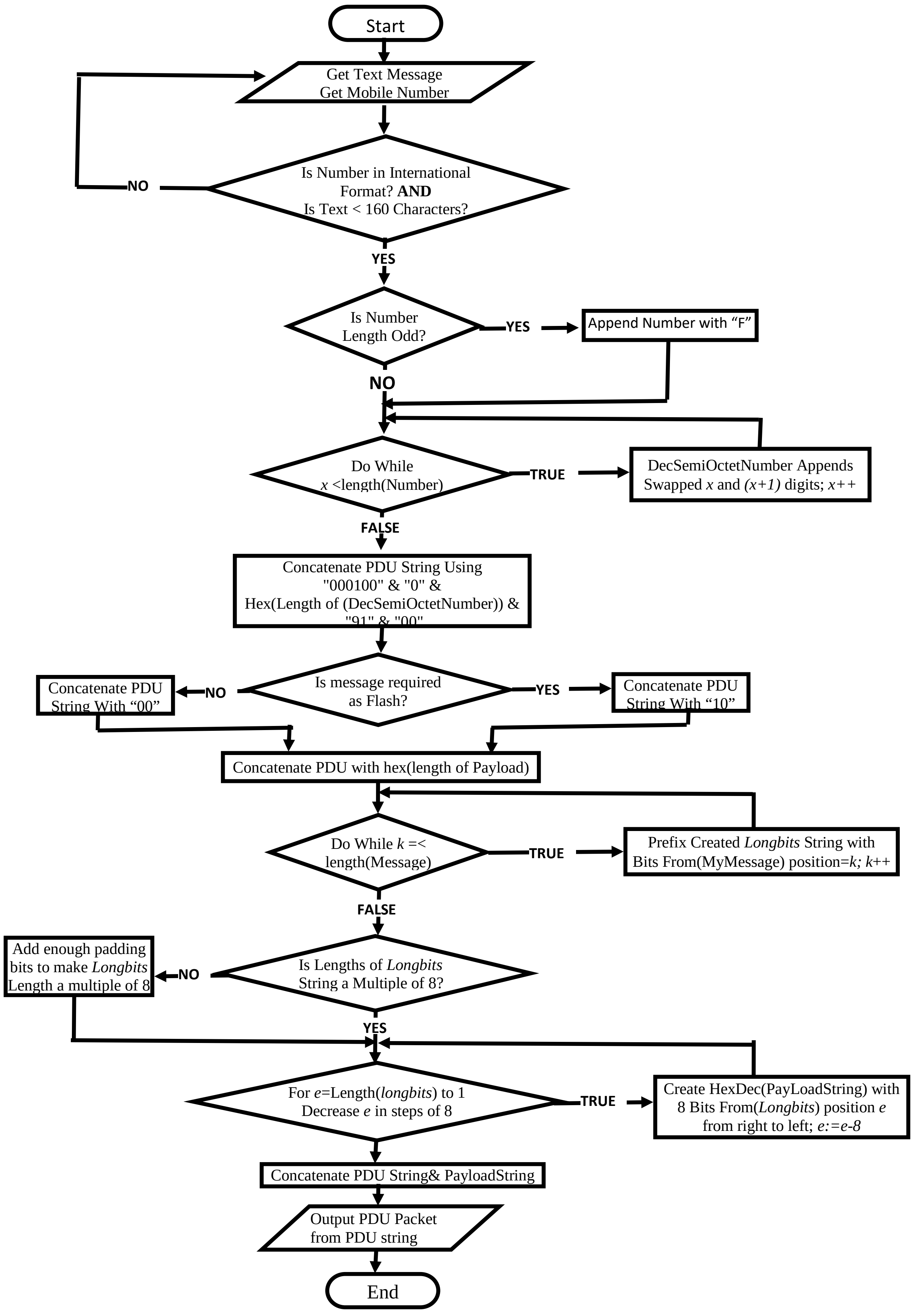

Start
Get Text Message
Get Mobile Number
Is Number in International Format? AND Is Text < 160 Characters?
NO
YES
Is Number Length Odd?
YES
Append Number with "F"
NO
Do While x <length(Number)
TRUE
DecSemiOctetNumber Appends Swapped x and (x+1) digits; x++
FALSE
Concatenate PDU String Using "000100" & "0" & Hex(Length of (DecSemiOctetNumber)) & "91" & "00"
Is message required as Flash?
NO
Concatenate PDU String With "00"
YES
Concatenate PDU String With "10"
Concatenate PDU with hex(length of Payload)
Do While k =< length(Message)
TRUE
Prefix Created Longbits String with Bits From(MyMessage) position=k; k++
FALSE
Is Lengths of Longbits String a Multiple of 8?
NO
Add enough padding bits to make Longbits Length a multiple of 8
YES
For e=Length(longbits) to 1 Decrease e in steps of 8
TRUE
Create HexDec(PayLoadString) with 8 Bits From(Longbits) position e from right to left; e:=e-8
Concatenate PDU String& PayloadString
Output PDU Packet from PDU string
End

Figure 2: Flow chart for the singular function PDU packing algorithm sequence without an SCA

5. Implementation of the algorithm using only the *MID* function

In the example to follow, we will have as inputs to our algorithm, the text **“*hellohello*”** as the intended message and the number **“263733356600”** as the mobile number that we intend to receive the message. We will illustrate how our PDU string is built up by tracing the contents of a continual concatenation of the string variable which we will call, ***PDUStringVar***. This variable is first initialized as an empty string and should end up containing our PDU packet after tracing through the algorithm. The steps of the algorithm will follow the sequence of the flow chart of figure 2. We will be building the PDU version without a SCA as this will be taken from the register of the modem and will have been pre saved by AT commands. So the ***PDUStringVar*** will be assigned the string **[000100]**. The first “00” will mean that there is no SCA number in our PDU packet. The next “01” is the PDU header. If we needed to set a message validity time we would set this value to “11” instead. The next “00” will form the message reference number. This can be any number and so we will set it to 00. So at this stage the contents **PDUStringVar= [000100]**

Step 1: Since the number 263733356600 has 12 characters that means it is “even”. For even numbers there will be no need to put the half octet “F” before preparation of the decimal semi-octet number field. If a field which is represented in decimal semi-octets exists, it may have complete octets if the length of the number is even or can have one half octet if the length is odd [13]. At the same time this number is assumed to be in international format which means it is in format “91”. The length of 12 characters is represented by hex 0xC. So we will now concatenate our PDU with the bytes **[0C] and [91]** the length of the number and the number format respectively. At the end of this step the contents ***PDUStringVar*=[**000100**0C91]**

Step 2: In this step there will be a loop through the digits of the number submitted and it is changed from ‘decimal’ to ‘decimal semi octets’ using only the ***Mid*** function again.

```
Do While x < Length of Num In Dec
    Num in Dec Semi Octets := concatenation of Mid(Num in Dec Semi Octets, x + 1, 1) & Mid(Num In Dec, x, 1)
    x:=x+ 2
Loop
```

As there is loop through the decimal number submitted, the decimal semi octets numbers are built by continuously concatenating with 2 returned instances of the ***Mid*** function. In both encapsulated calls of the function, the original number in decimal is passed as the contents of ***Str*** parameter each time. However, in the first instance of the ***Mid*** function, there is a call by *value* of *Start* having a value of *(x+1)* while in the second instance there is a *call by value* of *Start* with the value of *(x)* only. The value of *Length* is constant as 1 in both instances and the value of *x* will increment in steps of 2 each time. This ***Mid*** function then has the purpose of converting the decimal number from ‘decimal’ to ‘decimal semi octets’ ready for encoding it. After running through this sequence we will have the contents of *Num in Dec Semi Octets*=**[627333536600]**. This is concatenated to the already existing contents of ***PDUStringVar*****=[**000100**0C91]**. The new contents of the ***PDUStringVar*** *will become***=[**0001000C91**627333536600]**.

Step 3 Here we check if the user wants the message to reflect as a flash message or not at the SMMT end. If it is not required as a flash we concatenate with a **[00]** and if it is required to flash we concatenate with a **[10]**. We assume here that we do not want a flash message therefore we will post a **[00]**. However, before we do the concatenation, at this stage we will need to set **[00]** on the protocol identifier segment to indicate that we do not require a higher level protocol. If we would want to specify how long our message could stay on the SMSC for retry we would set the next flag to say [AA] provided in the PDU header we would have set [11]. Since our PDU header is a [01], we will not have a validity period. This will mean that the contents of the ***PDUStringVar*** *will become***=[**0001000C91627333536600**0000].**

Step 4 Here the payload which is the hex length of the message is determined in hex octets and the value of the payload is again appended to the PDU packet build up. Our test text is "*hellohello*" and has a hex payload of 0x0A. The field is supposed to have 2 character so we will append a 0 so that it fills up the trailing bit to become **[0A]**. At the end of this step our ***PDUStringVar*** *will become***=[**00010000C916273335366000000**0A]**

Step 5 In this step, a continuous string variable declared as *LongLongBits* is created. It will contain a series of all binary representations of the 7 bit character table equivalent of the characters in the original message *(hellohello)* but in reverse order. The coding of this operation using only the ***Mid*** function is outlined below.

```
For x = 1 To Length of (Original Message)

LongLongBits := BinaryDigits of (Ascii(Mid(Original Message, x, 1))) & LongLongBits

Next
```

The *LongLongBits* string is concatenated each time by appending it from the beginning to the end instead of appending it at the end. Since the value of *Start* position is always decreasing as it is passed to ***Mid*** as *x*, this effectively rearranges the order of the bits in the string originally occurring from left to right so that they are now from right to left. After running this loop the contents of LongLongBits will be

[1101111110110011011001100101110100011011111110110011011001100101110100 0]

Step 6 : In this step there is only need to add padding bits to the resulting string of bits.

```
LongLongBits = Create a String(8 – Length of(LongLongBits) Mod 8, "0") & LongLongBits
```

Here the *LonglongBits* string is prefixed by as many padding "0" bits as there are in the value of the remainder of dividing the whole string by a full octet of bits=8. This will ensure that the length of our *Longlongbits* is a multiple of 8. If there are up to 8 zeros then a padding bit was not needed and the contribution of the 8 zeros is 0 and is automatically ignored since they form a high order of zeros in the most significant part of the byte. That is to say 0x0000 0000F is equivalent to 0x000F which is equivalent to just 0xF. After this execution the contents of *LongLongBits* will be:

[**00**110111111101100110110011001011101000110111111101100110110011001011101000]

As can be seen this is again after addition of the padding bits which in this case are only 2 zeros.

Step 7: In this step, each of the set of octets computed in Step 4 and contained in the *LonglongBits* variable is given its decimal value using the ***Mid*** function before being converted to its hexadecimal-octets equivalent. Another variable, OctetString is used to concatenate all the ensuing hexadecimal-octets into one continuous string. This OctetString is the one which will give the final TP-UD frame of the PDU packet. The algorithm to do this is given below.

```
For x = Length of (LongLongBits) To 1 in Steps of (-8)
    {
    DecNum = 0
    DecNum = DecNum + value of (Mid(LongLongBits, x + 0 - 7, 1)) * ($2^7$)  }
    DecNum = DecNum + value of (Mid(LongLongBits, x + 1 - 7, 1)) *          } Upper Octet Nibble    ($2^6$)
    DecNum = DecNum + value of (Mid(LongLongBits, x + 2 - 7, 1)) *                                  ($2^5$)
    DecNum = DecNum + value of (Mid(LongLongBits, x + 3 - 7, 1)) * ($2^4$)
    DecNum = DecNum + value of (Mid(LongLongBits, x + 4 - 7, 1)) * ($2^3$)  }
    DecNum = DecNum + value of (Mid(LongLongBits, x + 5 - 7, 1)) *          } Lower Octet Nibble    ($2^2$)
    DecNum = DecNum + value of (Mid(LongLongBits, x + 6 - 7, 1)) *                                  ($2^1$)
    DecNum = DecNum + value of (Mid(LongLongBits, x + 7 - 7, 1)) * ($2^0$)
        If DecNum <= 15 Then OctetString = OctetString & "0" & Hex(DecNum) Else OctetString = OctetString & Hex(DecNum)
}
Next
```

In the algorithm above, the weighting of octet bits to their decimal equivalent can be simplified by running it through a loop that iterates 8 times. Each time, a variable *y* can be used in the loop to decrease the value of *Start* that will be passed to the ***Mid*** function. The simpler statement to replace the 8 statements above is given below.

*DecNum := DecNum + Val(Mid(LongLongBits, x + y - 7, 1)) * 2*$^{(7-y)}$

The last execution is to concatenate the *OctetString*, which apparently is the last field of the PDU packet with the rest of the PDU string that was being built. After execution this *OctetString* will have the contents **[E8329BFD4697D9EC37]**

These contents are then suffixed to the ***PDUStringVar*** *which was now* **=[**0001000C9162733353660000000A**]**. At this stage we will now have as the contents of our full packet of ***PDUStringVar*** as below

[0001000C9162733353660000000A**E8329BFD4697D9EC37]**. This is the required PDU packet for sending the message ***"hellohello"*** to the number ***"263733356600"*** under the stated options. It can be observed that this packet was built each time by a call to only 1 function which was the ***Mid*** string function.

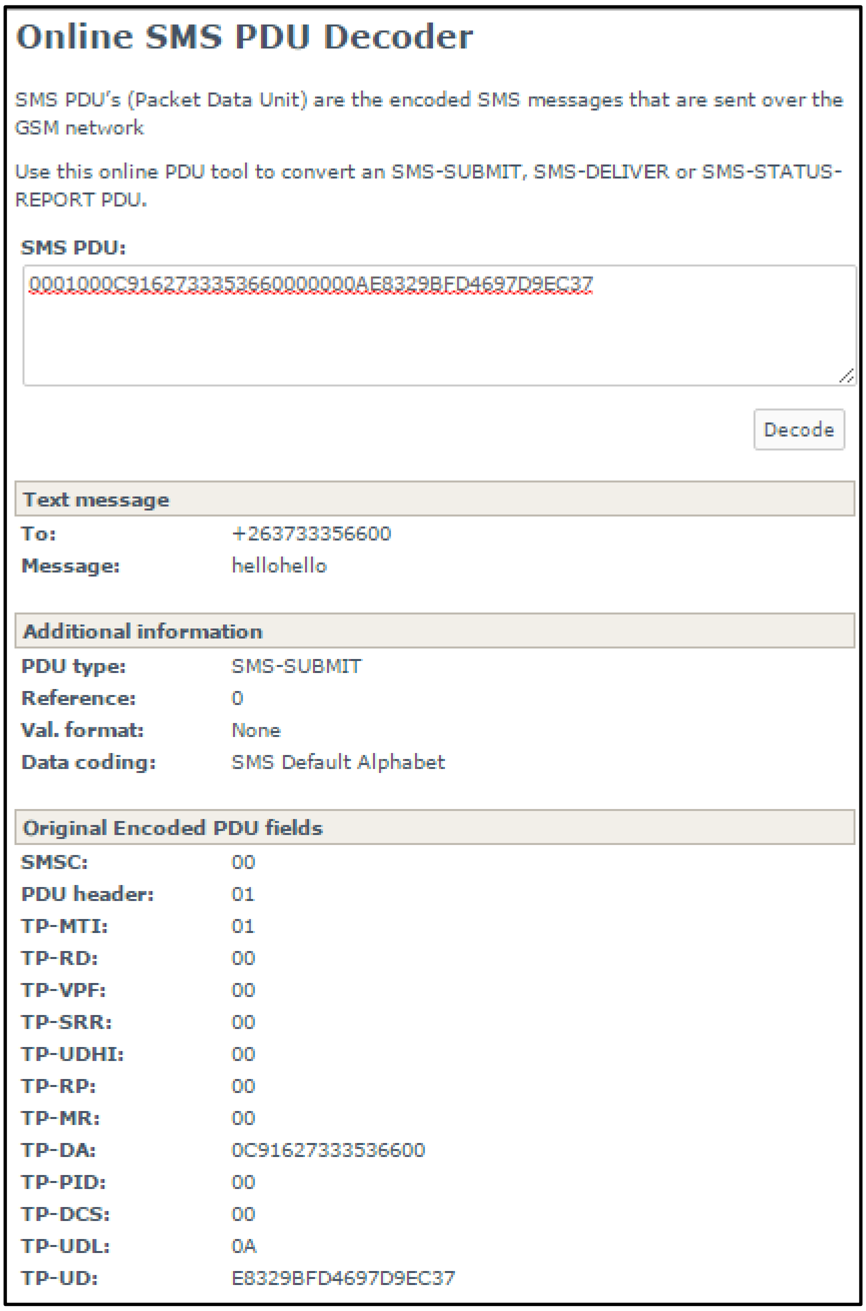
6. Results and discussion

We analyse our results in the ultimate accuracy of encoding the required PDU packet according to the ETSI standards. To make this accuracy analysis aspect, we feed similar required messages and numbers into at least two other algorithms that are found online. These generate PDU packets which we will then compare with ours. We start by testing whether the constructed packet is valid by posting it online on the following address https://www.diafaan.com/sms-tutorials/gsm-modem-tutorial/online-sms-pdu-decoder/

The results of the analysis by this online tool are shown in Figure 3. It is shown by this result that primarily all the fields and segments of the SMS submit framework are 100% as intended.

Table 2: Results of the created SMS SUBMIT PDU by our singular function and those from online utilities

#	ALGORITHM USED	SMS SUBMIT PACKET GENERATED
1	Our Singular Function Algorithm	0001000C9162732137875900003754741914AFA7C76B9058FEBEBB41E6371EA4AEB7E173D0DB5E9683E8E832881DD6E741E4F7D90582C564335ACD76C3E500
2	http://www.smartposition.nl/resources/sms_pdu.html	0011000C916273213787590000AA3754741914AFA7C76B9058FEBEBB41E6371EA4AEB7E173D0DB5E9683E8E832881DD6E741E4F7D90582C564335ACD76C3E500
3	http://rednaxela.net/pdu.php	0001000C9162732137875900003754741914AFA7C76B9058FEBEBB41E6371EA4AEB7E173D0DB5E9683E8E832881DD6E741E4F7D90582C564335ACD76C3E500
4	http://rednaxela.net/pdu.php	0011000C916273213787590000AA3754741914AFA7C76B9058FEBEBB41E6371EA4AEB7E173D0DB5E9683E8E832881DD6E741E4F7D90582C564335ACD76C3E500



Online SMS PDU Decoder

SMS PDU's (Packet Data Unit) are the encoded SMS messages that are sent over the GSM network

Use this online PDU tool to convert an SMS-SUBMIT, SMS-DELIVER or SMS-STATUS-REPORT PDU.

SMS PDU:

```
0001000C9162733353660000000AE8329BFD4697D9EC37
```

Text message

To:	+263733356600
Message:	hellohello

Additional information

PDU type:	SMS-SUBMIT
Reference:	0
Val. format:	None
Data coding:	SMS Default Alphabet

Original Encoded PDU fields

SMSC:	00
PDU header:	01
TP-MTI:	01
TP-RD:	00
TP-VPF:	00
TP-SRR:	00
TP-UDHI:	00
TP-RP:	00
TP-MR:	00
TP-DA:	0C91627333536600
TP-PID:	00
TP-DCS:	00
TP-UDL:	0A
TP-UD:	E8329BFD4697D9EC37

Figure 3: Analysis from [https://www.diafaan.com] of the PDU packet just created by our singular function

We create a longer SMS SUBMIT PDU packet with a different mobile number being [**+263712737895**] and the message [**The quick brown fox jumps over the lazy dog. 0123456789**]. Our results are compared in table 2.

It is observed from table 2 that our generated PDU packet (#1) is exactly equivalent to that of (#3). On the other hand packet (#2) is also equivalent to packet number (#4). However, in essence these packets achieve the same thing. The packets differ because #3 & #4 of the online generated had selected active the sending of *validity periods* of the SMS in the SMSC whereas ours (#1) and (#3) this was not preferred. This validity period option is activated when the second byte in the PDU is set as "11". In this case one will have also to put the validity period just before the "payload" bytes. The results therefore show that our proposed singular function algorithm packs perfect PDU packets 100% all the time.

References

[1] B. K. Siang, A. R. Bin Ramli, V. Prakash, and S. A. R. Bin Syed Mohamed, "SMS gateway interface remote monitoring and controlling via GSM SMS," in 4th National Conference on Telecommunication Technology, NCTT 2003 - Proceedings, 2003, pp. 84–87.

[2] G. Gu and G. Peng, "The survey of GSM wireless communication system," in Proceedings of ICCIA 2010 - 2010 International Conference on Computer and Information Application, 2010, pp. 121–124.

[3] F. Hillebrand, F. Trosby, K. Holley, and I. Harris, Short Message Service (SMS): The Creation of Personal Global Text Messaging. 2010.

[4] a. Festag and S. Hess, "ETSI technical committee ITS: news from european standardization for intelligent transport systems (ITS)- [global communications newsletter]," IEEE Commun. Mag., vol. 47, no. June 2009, pp. 1–4, 2009.

[5] J. Brown, B. Shipman, and R. Vetter, "SMS: The Short Message Service," Computer (Long. Beach. Calif)., vol. 40, no. 12, pp. 106–110, 2007.

[6] J. G. Caudill, "Mobile Computing : Parallel developments," Int. Rev. Res. Open Distance Learn., vol. 8, no. 2, p. 13, 2007.

[7] Z. Qian, D. Luo, and S. Wu, "Analysis and design of a mobile forensic software system based on AT commands," in 2008 IEEE International Symposium on Knowledge Acquisition and Modeling Workshop Proceedings, KAM 2008, 2008, pp. 597–600.

[8] R. Teymourzadeh, S. A. Ahmed, K. W. Chan, and M. V. Hoong, "Smart GSM based home automation system," in Proceedings - 2013 IEEE Conference on Systems, Process and Control, ICSPC 2013, 2013, pp. 306–309.

[9] G. Le Bodic, Mobile Messaging Technologies and Services: SMS, EMS and MMS: Second Edition. 2005.

[10] S. Attaway, "Chapter 7 - String Manipulation," in Matlab (Third Edition), 2013, pp. 235–263.