colorForth: The Next Generation

2023 Aug 12

Howerd Oakford www.inventio.co.uk

Abstract

colorForth : What is it for? Why am I still developing it after more than 20 years?
Because it is interesting - here are some new ideas that I intend to implement :

I need more colours!

All colorForths, from Chuck's original in 2001 to cf2023, use a token format with 4 bits for the "colour" and 28 bits for a Shannon-Fano compressed name. The "colours" are :

```
actionColourTable:    ;    * = number
    dd colour_blank   ; 0     extension token, remove space, do not change the colour
    dd colour_yellow  ; 1     yellow "immediate" word
    dd colour_yellow  ; 2  *  yellow "immediate" 32 bit number in the following pre-parsed cell
    dd colour_red     ; 3     red forth wordlist "colon" word
    dd colour_green   ; 4     green compiled cf2023
    dd colour_green   ; 5  *  green compiled 32 bit number in the following pre-parsed cell
    dd colour_green   ; 6  *  green compiled 27 bit number in the high bits of the token
    dd colour_cyan    ; 7     cyan macro wordlist "colon" word
    dd colour_yellow  ; 8  *  yellow "immediate" 27 bit number in the high bits of the token
    dd colour_white   ; 9     white lower-case comment
    dd colour_white   ; A     first letter capital comment
    dd colour_white   ; B     white upper-case comment
    dd colour_magenta ; C     magenta variable
    dd colour_silver  ; D
    dd colour_blue    ; E     editor formatting commands
    dd colour_black   ; F
```

I need at least one more bit, so that Magenta variables can have more than 28 bits of Shannon-Fano compressed name.

The current colorForth token format :

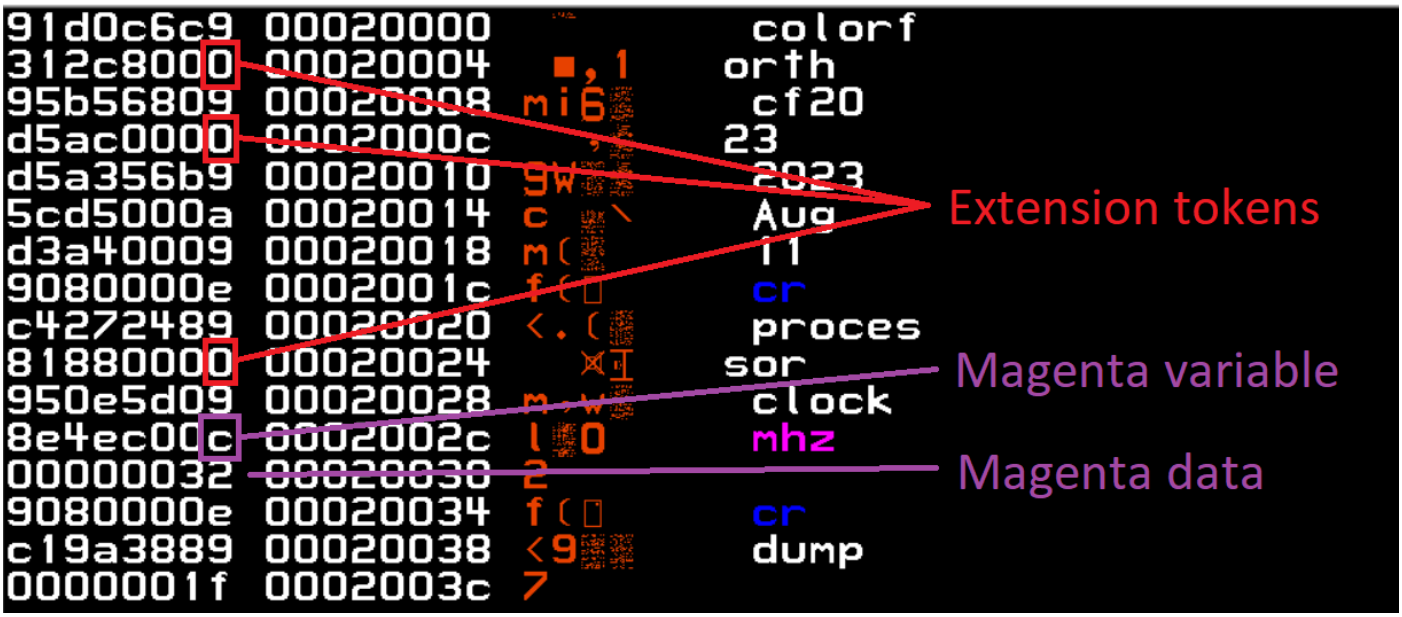
Colour			Name	
4 bits			28 bits Shannon-Fano / UTF-8	

Maybe it is time for an update…

Extension Tokens

Longer names are supported by the Extension “colour”, they backspace to remove the space that was added when displaying the previous token, then continue with the same colour as the previous token.

Typing ’64 block dump’ shows the start block, and how the white lowercase (colour 9) comment “colorforth” requires two tokens, “colorf” and “orth”.
The Magenta variable (colour hex c) uses the next address in the block to store its data.
It is impossible to know if the Magenta data is really data, or an extension token for the Magenta variable’s name, so Magenta variables are currently implemented without extension tokens.
If you define a Magenta variable with a name that does not fit into one token, it will not work.



Attempting to define a Magenta variable called “longname” gets a far as “longna”, then the extension token containing “me” gets interpreted as the Magenta data.
($8a000000 = -1979711488). Just don’t do it!

Encapsulation and Structure

This problem is actually an example of a larger set of problems to do with encapsulation – how do you define the beginning and end of a sequence of tokens?
For example, a string expressed as a sequence of tokens.

The solution is to add an Extension bit to the colour part of each token.

This means that you lose a bit from the name, leaving only 27 bits instead of 28.
So “rshift” no longer fits into one token, the digit ‘1’ is the ‘t’, so we could rename it “rshif”.

But why stop there?
By taking another bit from the name we could have 2 bits for “structure” :

0 0	on	carry on
0 1	dn	drop down a level
1 0	up	jump up a level
1 1	end	end of sequence

This would allow structured data such as JSON and XML files to be defined.

And, by taking another two bits we can add Version Control:

0 0	both versions
0 1	my version
1 0	the other version
1 1	deleted token (displayed as blank)

This leads to the new colorForth token format :

Colour	Structure	Version	Name	
4 bits	2 bits	2 bits	24 bits Shannon-Fano / UTF-8	

This also conveniently maps to one byte for the “colour” and meta-data and 3 for the name.

Reducing the number of bits for the name means that more tokens will require extension tokens, and this will increase the code size.

Currently, red colorForth tokens, that define a new word, only use the first token to compare with names in the wordlists. This means that names such as “myvalue1” and “myvalue2” clash – they appear to have the same name, even though their extension tokens in the source block are different.

Adding an Extension bit would make it easier to implement a comparison using more than one token. This would increase the size of the wordists in memory, and slow down the search for a word in the wordlists – an interesting tradeoff.

Structure

Each colorForth source block is an unstructured array of 32 bit tokens.
Following one of the key principles underlying colorForth, everything focuses on the token :
The editor reads it and displays it, the compiler reads it and compiles it, the interpreter reads it and interprets it, all depending on its “colour”.
Adding two bits for “Structure” allows the array of tokens to be structured :

0 0	on	carry on
0 1	dn	drop down a level
1 0	up	jump up a level
1 1	end	end of sequence

For example, for storing a JSON file (From : https://www.guru99.com/json-tutorial-example.html) :

```
{
  "student": [

     {
        "id":"01",
        "name": "Tom",
        "lastname": "Price"
     },

     {
        "id":"02",
        "name": "Nick",
        "lastname": "Thameson"
     }
  ]
}
```

Using a notation for each colorForth token [structure | **name**]

```
[on|stude] [end|nt]
    [dn|id]
        [up|01]
    [dn|name]
        [up|Tom]
    [on|lastn] [dn|ame]
        [up|Price]
    [up|]
    [dn|id]
        [up|01]
    [dn|name]
        [up|Nick]
    [on|lastn] [dn|ame]
        [on|Thame] [up|son]
    [up|]
```

Each new indentation in the JSON file is represented by a dn (down) token. Names that require more than one token use the “on” structure until the name is finished, then they can go up, down, or end this part of the structure.
A string would be a “dn” token, some “on” tokens and an “up” token.

Version Control

Again, following one of the key principles underlying colorForth, everything focuses on the token :

0 0 both versions
0 1 my version
1 0 the other version
1 1 deleted token (displayed as blank)

Bob and I want to work on our latest colorForth project.
To implement colorForth version control, tokens are marked as either “my” or “other” tokens.
Say “my” tokens are in blocks 256 to 511, and Bob’s “other” tokens are in blocks 1256 to 1511, copied from Bob’s computer.

Editing

We start off with the same code, and I edit the name of a token from “rshift” to “rshif”.
I mark the original token as “other”, and insert my edited token marked with “my”.

Pressing the F2 button toggles the Editor between displaying “my” an the “other” tokens, only one or the other is displayed, so I can see the change flashing.

Inserting

When I insert a new token, I mark it as “my” token, and also add the same token marked as “deleted”. This means that pressing F2 displays either my new token, or a blank space of the same width, so that the rest of the tokens in the block, after my edit, are not shifted around.

Deleting

Similarly to inserting a new token, deleting a token means marking it as “deleted”, again so the rest of the tokens are not shifted around, and repeatedly pressing F2 highlights only the important differences.

Pushing to Bob

At some point I will want to show Bob what I have done, and vice-versa, so “my” and the “other” code can be compared. When the version we want to keep has been selected, the Version Control fields can be replaced by “both”, and tokens marked as “deleted” can be actually deleted, and the blocks can be exported to Bob and anyone else who is interested.

Please note that both the Structure and Version Control ideas are as yet unproven.

Watch this space!

Cheers,

Howerd howerd@inventio.co.uk 2023 Aug 12