Reconfigurable Computing For Embedded System Devices

This project aims to be a quick prototyping unit, in spirit of the mbed platform itself. The setup consists of a mbed board, a LCD and few other components that demonstrate that how easy it is to develop a prototype using this platform. This project uses a scripting language called Forth for its scripting purpose. Forth is a structured, stack based computer language. It was designed by Chrales H. Moore. Forth is an interpreted computer language. It uses post fix method for performing arithmetic's.

Introduction to Forth:
Every Forth code consists of what is known as words. Words can be thought of as function calls. Words usually take their arguments from the stack and store the result back onto the stack. Take for example the '+' operator, which also is a word in the Forth language. The '+' operator takes next on stack and top of stack as its argument and performs the addition on them to store back the result onto the stack.

```
3 2 + . <Cr>
5
```

When you enter 3 and 2, it is pushed onto the stack which is called as the parameter stack. All operations in Forth language are performed on this stack. '+' is an operator, which takes Top of stack (TOS) element and Next on stack (NOS) as two parameters and leaves the result on top of stack. The dot (.) operator Pops out the TOS and displays it. The OK message is printed out by the interpreter to indicate that the interpreter is ready for the next input. Similarly, you can perform '-', '*' and '/' on stack. All arithmetic operations must be in Post Fix format.

Every valid token that is not a number, is a word in Forth language. Even the operators, +, -, *, and / are all words. Forth maintains an internal data structure called dictionary in which these words are stored. You can define you're own word which then becomes a part of the dictionary. In other words, it becomes a part of the Forth Language. You can create a word by entering compile mode while in interpreter mode. You enter compile mode by entering colon operator (:) which in itself is a word which switches the forth interpreter to compiled mode. The
definition of a word should end with semi colon (;). Now you can use your newly created word just as any other Forth word. If you want to define a word which computes the square of a number, you could enter the following in the interpreter.

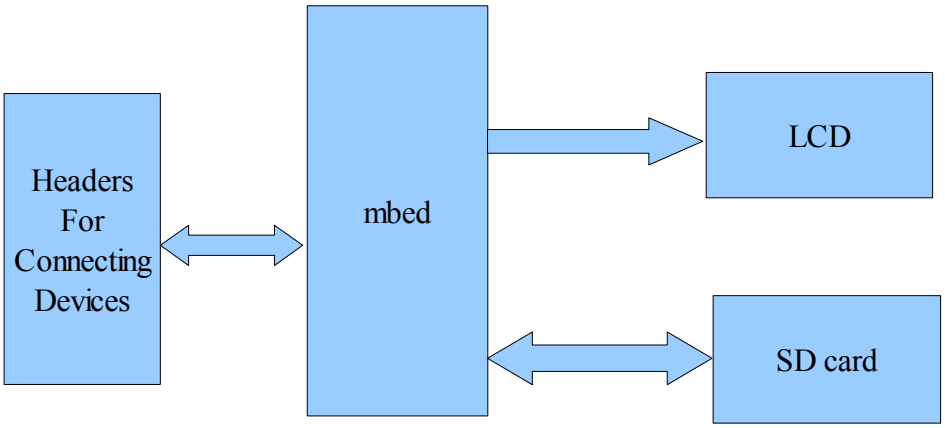
```
: square dup * . ;
```

This word when executed, would take TOS and duplicate it, multiply these two numbers and then pop out the result using dot operator. The first used in 'square' is dup, this word copies the top of stack and leaves TOS with another copy of what was on TOS . The second word is the multiplication and third one is the dot operator. The dot (.) operator Pops out the TOS and displays it. Below is 'square' word usage demonstration:

```
4 square <cr>
16
```

The Reconfigurable computing system on mbed consists of a simple, minimal GUI system that would help accelerate quick prototyping. The GUI system supports buttons, static texts, progress bars and bitmap images. Words for controlling and reading I/O ports have been included any additional functionalities can be Incorporated easily into this system. You can load files from the Forth scripts from the SD card as well (stored in Unix file format newline=\n). Forth scripts can also access ports. You can set a port pin by using word 'DigitalOut' which expects port position in Top of stack and port digital value next in stack. This word internally uses object of type DigitalOut.

The set up:
The hardware consists of a mbed module, an LCD, a touch screen controller, a SD card holder and a few other components.The LCD module is based on ili931 controller. 16-bit RGB data format is used for drawing images on the LCD. All the functions related to LCD can be found in lcd.c file under the 'GUI' folder. The drawing to LCD happens in non buffered mode. The SD card is used to store Forth scripts which can be loaded and executed.



Usage:
Let us start with the simple hello world program. Lets define a word called hello-world which will output string "Hello world" on the serial console.

```
: hello-world." Hello world " ;
```

The 'hello-world' word uses the string operator " ." " which displays its argument string onto the serial console. The word is invoked at an interactive terminal by entering hello-world in serial terminal.

```
hello-world <Cr>
```

The next example uses the GUI interface available to build a simple application that displays a message whenever click on a button.

```
: hello." You clicked on hello  " Cr ;
: exit_loop exit_ml ;
30 40 1 crt_btn "hello" "Hello" hello
100 40 2 crt_btn "exit" "Exit" exit_loop
show
```

```
ml      \ enter the main loop
```

First, we define a word called “hello” and “exit_loop”. “hello” word prints “you clicked on hello” on the serial console and the “exit_loop” causes the execution to move out of the main loop(explained later). Next, we create a button using “crt_btn” word. This word accepts x coordinate position, y coordinate position and id from the stack the other arguments are name, caption of the button and call back word respectively. Whenever there is an event on the button, the call back word is executed. The call back word here happens to be hello which was defined earlier. Similarly the “Exit” button is constructed. Once the GUI layout is defined we need to display it on the LCD this is done by using the word show. This word takes no argument and displays the GUI elements as defined in the script. The event handling is done by the “ml” word which stands for main loop. This word is responsible for executing the callback words. The mainloop can be exited using the word “exit_ml” which happens to be a part of definition “exit_loop” word. Hence clicking on “Exit” button would cause the execution to come out of mainloop and enter into interactive mode. Below is a picture of the LCD displaying the GUI elements as described in the script above.