# Pattern Matching

You are a celebrated astronaut in the Indian Space Program. You are on a mission to visit the planet Htrae and communicate with the sentient species there. As the captain and flight computer engineer on this mission, your job is to keep the crew and all the electronics on the spaceship safe and in order.

Unfortunately, as soon as you land on Htrae, your crew is captured and taken into custody (Obviously, they are scared of the species which is destroying its home planet with global warming). Stuck in a high-tech cell, you and your crew start searching for ways to get out. Taking pity on you and on the benefit of the doubt that you did not contribute to global warming, the Alien overlord gives you a huge file with lots of digital keys in each line. She says that the file is valid for 24 hours and the right key is a pattern that matches the regular expression - `(1|0)*1(1|0)[20]`[1] and that you get one chance at opening the digital lock failing which you will forever be doomed in the cell. Clearly, as the flight computer engineer, you have the coolest computer with an FPGA chip in it. The Linux enthusiast in you reminds you of the famous "grep" command. However, this takes a large amount of time and RAM to find the pattern in the file because of the file size. Since your cool computer has an FPGA, use your digital design skills to quickly find the correct key.

You are supposed to fill in the module "`regex`" located at "`code/regex.v`". A testbench file has been provided to you. Use any tools you are familiar with (such as `Iverilog`[2], `GTKWave`[2], `ModelSim` etc.) to demonstrate operation of logic implemented in structural, gate level Verilog or VHDL.

## Tips and help from an empathetic alien

1. A subset of keys containing the correct key from the original file exists in "code/`input-data-file.txt`". Your simulations can handle this in a reasonable amount of time. Just run the "`make`" command to check your solution.

2. Please refer to section 3 except 3.4 in the paper titled "*Fast Regular Expression Matching using FPGAs*", by Dr. Reetinder Sidhu and Dr. Viktor K Prasanna. Figures 2, 3, 4, 5, and 7 will help you understand how to implement an NFA in logic.

3. Under the "`examples`" directory, you will find implementations for different regular expressions. Steps to run the examples are:
   - $ `make`
   - Execute any of the binary files to simulate and record the waveforms. Eg:
     - `$ ./zero_dot_one`
   - Open "`regex.vcd`" file using `GTKWave`
     - `$ gtkwave regex.vcd`

---

[1] `'[]'` is used to represent the number of repetitions. Thus, the expanded regular expression for `(1|0)*1(1|0)[20]` is `(1|0)*1(1|0)(1|0)(1|0)(1|0)(1|0)(1|0)(1|0)(1|0)(1|0)(1|0)(1|0)(1|0)(1|0)(1|0)(1|0)(1|0)(1|0)(1|0)(1|0)(1|0)`

[2] Installation command: Use `apt` and `yum` for debian and RHEL based machines respectively. Eg: `$ sudo apt install iverilog gtkwave / $ sudo yum install iverilog gtkwave.`