2150 Shattuck Ave, Penthouse, C/O SkyDeck,
Berkeley, California 94704-1347, USA
support@grovf.com

GROVF
Accelerating BigData Analytics

Regular Expression Demo Project Description Using PCIe (regexp-pcie)

Document version: 1.0

2150 Shattuck Ave, Penthouse, C/O SkyDeck,
Berkeley, California 94704-1347, USA
support@grovf.com

GROVF
Accelerating BigData Analytics

Table of Content

2150 Shattuck Ave, Penthouse, C/O SkyDeck,
Berkeley, California 94704-1347, USA
support@grovf.com

GROVF
Accelerating BigData Analytics

Introduction

Demo Project regexp-pcie is designed to showcase the functionality and maximum bandwidth of regexp-ip.

The scope of delivery includes:

- Bitstream for Alveo U200
- Library for working with Alveo U200
- Demo applications

Regular Expression IP Description (regexp-ip)

regexp-ip is designed to search in the text for matches against the provided regular expression rules. The implementation is fully pipelined and runs at a 400MHz frequency. regexp-ip accepts AXI-Stream with text (16 bytes per clock cycle) and on each clock cycle produces a 16-bit vector output indicating the presence (‘1’) or absence (‘0’) of a match against a given regular expression. It should be noted that regexp-ip finds only the fact of coincidence with the regular expression as well as the last character of this match, regexp-ip does not find the beginning of a match. The maximum throughput of a single core is 50 Gbps.

For regexp-ip to work, you must configure it through the AXI4-LITE interface. After entering regular expressions, a necessary set of parameters is formed by the "Regular Expression Compiler" on the host side, which is provided in source code.

List of supported characters:

. - Any Character Except New Line
\d - Digit (0-9)
\D - Not a Digit (0-9)
\w - Word Character (a-z, A-Z, 0-9, _)
\W - Not a Word Character
\s - Whitespace (space, tab, newline)
\S - Not Whitespace (space, tab, newline)
[] - Matches Characters in Brackets
[^] - Matches Characters NOT in Brackets
| - Either Or
( ) - Group
* - 0 or More
+ - 1 or More
? - 0 or One
{3} - Exact Number
{3,4} - Range of Numbers (Minimum, Maximum)

2150 Shattuck Ave, Penthouse, C/O SkyDeck,
Berkeley, California 94704-1347, USA
support@grovf.com

\b - Word Boundary
\B - Not a Word Boundary
^ - Beginning of a String
$ - End of a String

Anchors symbols such as \b, \B, ^ and $ are not supported in regexp-ip itself but are supported by the software compiler, which replaces them with other characters from the list of supported, and that does not affect the fact of detection specified regular expression in the text.

The "{min, max}" construct supports a maximum of 16 repetitions. If the max field has a value greater than 16, it will be perceived as infinitely large. That said, an expression such as {3, 17} means that the number of repetitions will be from 3 to infinity.

In this implementation, the symbol "|" supports up to 4 choices. For example, given expression is valid: (aaa | bb | ccc | dd).

The compiler breaks the inputted regular expression into so-called "sub-regexps." A sub-regexp is a string from a set of characters or their classes to which the same quantifiers (such as '{}', '*', '+', '?'), brackets or conditional statements (such as a symbol '|') applies.

In fact, regexp-ip is a collection of sequentially connected IPs (subregexp-ip) for sub-regexp processing. Adjusting the maximum number of sub regular expressions in subregexp-ip, you can change the amount of resources consumed in the FPGA to the prejudice of the maximum length of the supported regular expression.

For this implementation, the maximum size of the sub regular expression is 4 characters. This implementation supports 24 sub regular expressions. In the case of error detection during the regular expression processing, the compiler returns extended string messages for a sufficient understanding of errors and solutions.

Examples of how the regular expression is divided into sub regular expressions

Example 1.
The following regular expression: \d\d\d-\d\d\d-\d\d\d\d is divided into sub regular expressions in the following way:
\d\d\d-
\d\d\d-
\d\d\d\d

Example 2.
The following regular expression: [A-Z][a-z]+\s[A-Z][a-z]+$ is divided into sub regular expressions in the following way:
[A-Z]
[a-z]+

2150 Shattuck Ave, Penthouse, C/O SkyDeck,
Berkeley, California 94704-1347, USA
support@grovf.com

\s[A-Z]
[a-z]+
$

Example 3.
The following regular expression: \d\d\d\s\w+\sSt., \w+ [A-Z][A-Z] \d* is divided into sub regular expressions in the following way:
\d\d\d\s
\w+
\sSt.,
<SPACE>
\w+
<SPACE>[A-Z][A-Z]<SPACE>
\d*

Regular Expression Project Description

The Regexp-PCIe project is meant for demonstration purposes, in order, to show the functionality of regexp-ip features as well as maximum bandwidth. The regexp-ip is controlled by the host using the SDK, which is provided in the source code. The reference examples of how to use the software SDK is shown in the demo applications, also provided in source codes.

The project workflow is the following: The text is sent to the RAM on the Alveo U200, after which the host application gives a start command. In the next step, the text is read from the RAM and passed to regexp-ip input via AXI-Stream. The results are saved in the RAM on the Alveo U200. Then the host application can fetch these results from RAM on Alveo U200.

As stated earlier, regexp-ip runs at 50 Gbps. In order to achieve 100 Gbps bandwidth for the regexp-pcie demo project, two instances of regexp-ips were integrated into the project. Each of them is running with the maximum bandwidth of 50 Gbps, in parallel, both connected to their appropriate strip of the RAM.

The first instance of regexp-ip supporting 24 sub regular expressions (each of them having 4 symbols) processes the 96 character length regular expression and occupies SLR0 on the Alveo U200. The second instance of a regexp-ip with the same characteristics occupies the SLR2 of the FPGA. SLR1 is free for other functionalities.

The regexp-ip, which accepts 16 chars per clock cycle over AXI-Stream, saves the output results in the following way:

<48 bits> - The ID of the inputted 16 char string block
<16 bits> - the vector of matches on the given input text block. “1” means there is a match in that position.

The example of how to convert the output vector of regexp-ip to the numeric positions is given in the example regexp-simple with source codes.

2150 Shattuck Ave, Penthouse, C/O SkyDeck,
Berkeley, California 94704-1347, USA
support@grovf.com

GROVF
Accelerating BigData Analytics

The library to work with Alveo U200 and demo applications

The library to work with the Alveo U200 and demo applications are provided with source codes written in C++. CMake and Boost c++ libraries is needed for building the applications.

Demo Applications

regexp-simple - a simple application showcasing the functionality of the regular expression IP core. It is a console application, and it is controlled by the command line parameters. To get the available options for the application, simply run the application in the command line with the following parameter <--help>. This application accepts a file with the text and a file with the regular expression. The results will be shown after the application finishes the execution. Also, the application displays the results of matching with changed colors (red colors where matching happened).

regexp-bw - a simple application showcasing the processing of large log files. The application divides the text files into two parts and simultaneously runs this data on two different Regexp-IPs. The application also measures the bandwidth of processing.

For the regexp-simple application, in the folder build, are a set of testbenches, which can be run using scripts ./tb1.sh, ./tb2.sh, etc.

Driver and OS

To work with Regexp-PCIe project, you need to install into the system the XDMA driver from the Xilinx Github repository: https://github.com/Xilinx/dma_ip_drivers

The applications, drivers and the project were tested on the Ubuntu 18.04 OS/Kernel version 5.3