

String.h Accelerator +

GROUP MEMBERS: David Tougaw & Matthew Bowen

Introduction

This project is to propose a DE2-115 String.h accelerator with add-on functions. The idea is to implement most string manipulation/examination functions in a hardware component that is interfaced with the Avalon Bus and the DE2-115 Computer System. The speedup of hardware vs. software will be compared.

The list below includes the most common string manipulations from cstring.h as well as several **additions (+)**. A select few of these functions will be implemented in hardware.

- | | | |
|------------|------------------|---------------|
| • Compare | • +StringToFloat | • +ToLower |
| • Search | • +Replace | • +ToUpper |
| • IndexOf | • +Remove | • +StartsWith |
| • +Reverse | • +Frequency | • +Normalize |

We decided to implement such component because the standard String.h library is very limited and lacks functionalities.

Strings are very common data types, so creating an accelerator would be beneficial for those fields where text manipulation is very important and can take time.

Implementation

For such hardware component, we decided for an Avalon Memory-Mapped custom peripheral. Our first design had a FIFO buffer to store input letters and reduce address space utilization, but due to timing issues related to the Avalon interconnect we decided to use a larger region of the address space based on a **MAX_BLOCKS** and **ADDRESS_BITS** parameters.

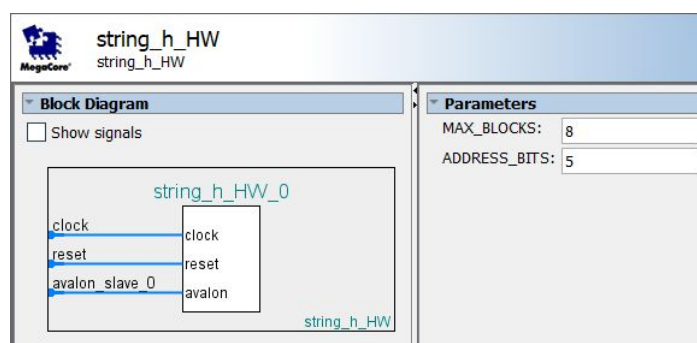


Figure 0. HW Component Block Symbol and Parameters

Developer View

String.h Hardware + (Default 32 Byte Strings)

MAX_BLOCKS = 8										
Reg #	Register	Offset	31	30	29	...	13 - 6	5-2	1	0
Register 0	Status/Control	0x0	Unused			...	Length	Index	Go	Done
Register 1	String A [0]	0x4	B[31]	B[30]	B[29]	B[1]	B[0]
...										
Register 8	String A [7]	0x20	B[31]	B[30]	B[29]	B[1]	B[0]
Register 9	String B [0]	0x24	B[31]	B[30]	B[29]	B[1]	B[0]
...										
Register 16	String B [7]	0x54	B[31]	B[30]	B[29]	B[1]	B[0]

Figure 1. Developer View of String.h Hardware +

Size of component can be specified during the addition of the component to the computer system by specifying **MAX_BLOCKS** parameter. Furthermore, **ADDRESS_BITS** will allow the scaling of the component's **ADDRESS SPACE** to fit specified number of words.

Status/Control register corresponds to register 0.

StringA ranges from register 1 to register MAX_BLOCKS and StringB ranges from register MAX_BLOCKS + 1 to 2*MAX_BLOCKS .

Default value of MAX_BLOCKS is 8 where each input string has 32 bytes of data (Figure 1).

Status/Control register has the following fields:

- **Length [13:6]** - Specifies the length of StringB needed for Search.
- **Index [5:2]** - Specifies the function that needs to be computed on inputted string(s).
- **Go [1]** - When set to 1, hardware module will start execution of A/B Type function. Go bit needs to be deasserted to reset peripheral.
- **Done [0]** - Set to 1 when hardware module has finished execution and result can be read. Reset of bit occurs when unused index is written to Status/Control register.

Functions

String.h Hardware + implements two types of functions: **A Only** and **A/B Type**.

A Only Type

A Only functions are performed on StringA input only. **Execution begins** as soon as go bit is set. **Done** bit is set when entire execution is complete. **Result** can be read from StringA register.

The following functions are of A Only Type:

A Type Only Functions		
Function	Index	Description
ToUpper	1	Converts StringA alphabetic characters to UPPERCASE
ToLower	2	Converts StringA alphabetic characters to LOWERCASE
Reverse	3	Reverse StringA

Table 1. A Type Only Functions

A/B Type

A/B Type functions are performed on StringA and StringB inputs. **Execution begins** when Go bit is set. **Done** bit is set when entire execution is complete. Go bit needs to be deasserted to reset peripheral. **Result** of such functions can be read from Register MAX_BLOCKS + 1.

The following functions are of A/B Type:

A/B Type Functions		
Function	Index	Description
Compare	0	Compares stringA with StringB. Returns 1 when (StringA == StringB)
Search	4	Search for StringB (Length required) in StringA and return starting index. If not found, returns 0xFF

Table 2. A/B Type Functions

Simulation

The String Hardware peripheral underwent thorough testing before being implemented and tested on the DE2-115 board. Multiple test cases were performed for each function as shown below. **All test cases were successful.**

Testing was also performed on a FIFO module. The tests were successful, however, due to unresolved reading issues the FIFO was not implemented.

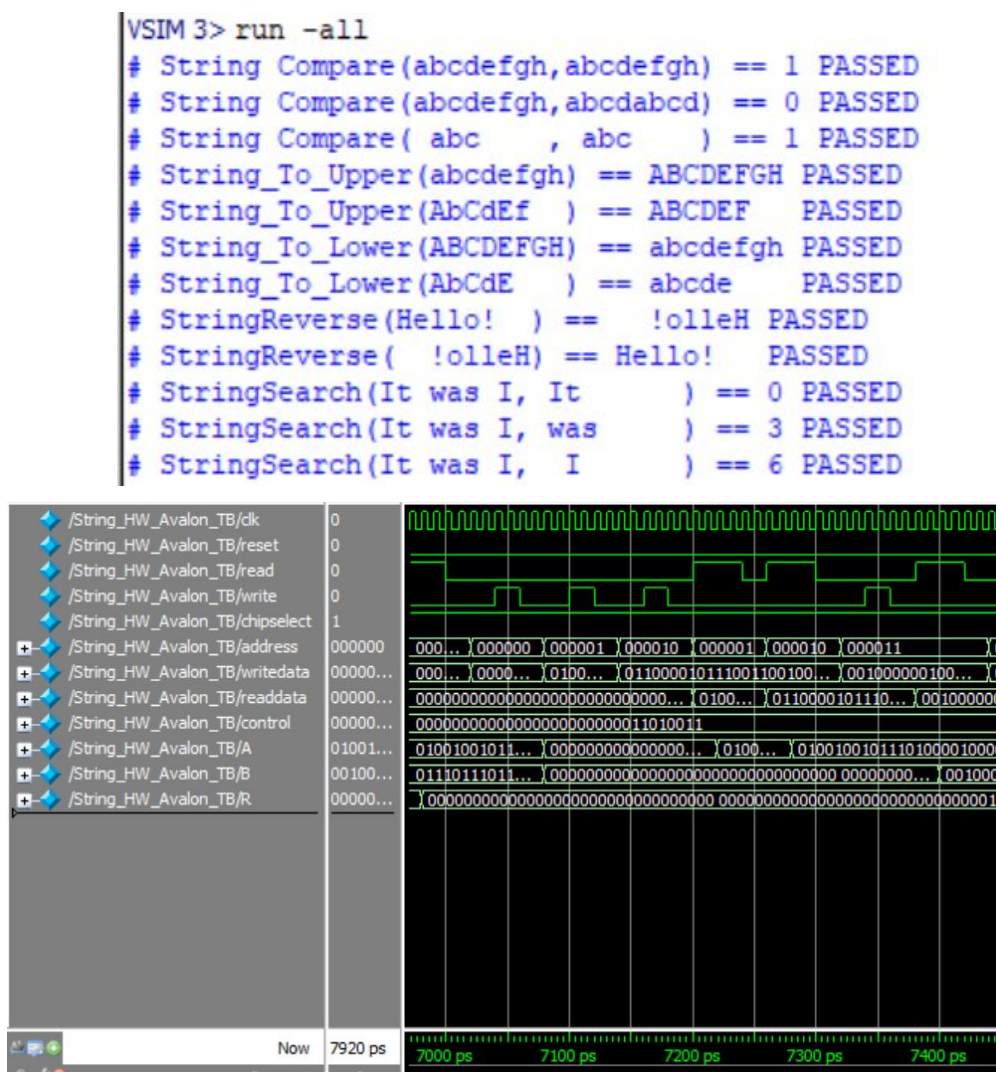


Figure 2. Testbench and Wave Output

DE2-115 Computer System Output

Compare

```

#### string.h vs String HW + peripheral ####
####   BY D. Tougaw & Matthew Bowen   ####
Index  Function
#      TEST READ/WRITE AVALON
0      Compare
1      ToUpper
2      ToLower
3      Reverse
4      Search
Select function [Index]: 0
===== StringCompare(str1, str2) =====
String A: LYLA----LYLA----LYLA----LYLA----
String B: LYLA----LYLA----LYLA----LYLA----
===== StringCompare(str1, str2) =====
SW EQUAL
HW EQUAL
Software CC = 1837      ET = 36.740000 us
Hardware CC = 226      ET = 4.520000 us
Speedup = 8.128319
=====
Any char to continue..|

#### string.h vs String HW + peripheral ####
####   BY D. Tougaw & Matthew Bowen   ####
Index  Function
#      TEST READ/WRITE AVALON
0      Compare
1      ToUpper
2      ToLower
3      Reverse
4      Search
Select function [Index]: 0
===== StringCompare(str1, str2) =====
String A: LYLA----LYLA----LYLA----LYLA----
String B: LYLA-----LYLA----LYLA----
===== StringCompare(str1, str2) =====
SW NOT EQUAL
HW NOT EQUAL
Software CC = 994      ET = 19.880000 us
Hardware CC = 226      ET = 4.520000 us
Speedup = 4.398230
=====
Any char to continue..|

```

Figure 3. Compare SW&HW Function Output

ToUpper

```

#### string.h vs String HW + peripheral ####
####   BY D. Tougaw & Matthew Bowen   ####
Index  Function
#      TEST READ/WRITE AVALON
0      Compare
1      ToUpper
2      ToLower
3      Reverse
4      Search
Select function [Index]: 1
===== StringToUpper(str) =====
String A: lylatagssongdamptynecapebarnflow
Read A: LYLA
Read A: TAGS
Read A: SONG
Read A: DAMP
Read A: TYNE
Read A: CAPE
Read A: BARN
Read A: FLOW
===== StringToUpper(str) =====
Software CC = 4764      ET = 95.280000 us
Hardware CC = 235      ET = 4.700000 us
Speedup = 20.272340
=====
Any char to continue..|

```

Figure 4. ToUpper SW&HW Function Output

ToLower

```

#### string.h vs String HW + peripheral ####
####   BY D. Tougaw & Matthew Bowen   ####
Index  Function
#      TEST READ/WRITE AVALON
0      Compare
1      ToUpper
2      ToLower
3      Reverse
4      Search
Select function [Index]: 2
===== StringToLower(str) =====
String A: LYLAtagsSONGdamptyneCAPEBARNflow
Read A: lyla
Read A: tags
Read A: song
Read A: damp
Read A: tyne
Read A: cape
Read A: barn
Read A: flow
===== StringToLower(str) =====
Software CC = 3519      ET = 70.380000 us
Hardware CC = 238      ET = 4.760000 us
Speedup = 14.785714
=====
Any char to continue..|

```

Figure 5. ToLower SW&HW Function Output

Reverse

```

#### string.h vs String HW + peripheral ####
####   BY D. Tougaw & Matthew Bowen   ####
Index  Function
#      TEST READ/WRITE AVALON
0      Compare
1      ToUpper
2      ToLower
3      Reverse
4      Search
Select function [Index]: 3
===== Reverse(str) =====
String A: LYLAtagsSONGdamptyneCAPEbarnFLOW
String A SW Reversed: WOLFnrabEPACenytpmadGNOSsgatALYL
Read A: WOLF
Read A: nrab
Read A: EPAC
Read A: enyt
Read A: pmad
Read A: GNOS
Read A: sgat
Read A: ALYL
===== Reverse(str) =====
Software CC = 4225      ET = 84.500000 us
Hardware CC = 235      ET = 4.700000 us
Speedup = 17.978723
=====
Any char to continue..|

```

Figure 6. Reverse SW&HW Function Output

Search

```

#### string.h vs String HW + peripheral ####
####   BY D. Tougaw & Matthew Bowen   ####
Index  Function
#      TEST READ/WRITE AVALON
0      Compare
1      ToUpper
2      ToLower
3      Reverse
4      Search
Select function [Index]: 4
===== Search(strA,strB) =====
String A: LYLAtagsSONGdamptyneCAPEBARNflow
String B: tags
===== Search(strA,strB) =====
SW FOUND at pos: 4
HW FOUND at pos: 4
Software CC = 6302      ET = 126.040000 us
Hardware CC = 238      ET = 4.760000 us
Speedup = 26.478992
=====
Any char to continue..|

```

Figure 7. Search SW&HW Function Output

Compilation Report

The String Hardware peripheral used the following resources as shown in figure 8:

Flow Summary	
<<Filter>>	
Flow Status	Successful - Mon Dec 09 23:33:06 2019
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	String_HW_Avalon
Top-level Entity Name	String_HW_Avalon
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	812 / 114,480 (< 1 %)
Total registers	371
Total pins	74 / 529 (14 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

Figure 8: Compilation report of String Hardware peripheral

Conclusion

After several testing cases, the following performances were recorded for SW and HW implementation:

ET = Execution Time	CC = Clock Cycles				
Function	SW CC	HW CC	SW ET	HW ET	Speedup
Compare	2268.5	226	4.54E-05	4.52E-06	10.038
ToUpper	4764	235	9.53E-05	4.70E-06	20.272
ToLower	3519	238	7.04E-05	4.76E-06	14.786
Reverse	4225	235	8.45E-05	4.70E-06	17.979
Search	6302	238	1.26E-04	4.76E-06	26.479
AVERAGE SPEEDUP					17.911

Table 3. SW vs HW Performance

From these results, we can clearly see that **String.h Hardware Accelerator +** is 17.9 faster on average against String.h or comparable software implementations.

```

1  /*
2  * #####
3  * CPE423
4  * String Hardware Avalon INTERFACE-----
5
6  / _____ \
7  \ _____ /
8  / _____ \
9  \ _____ /
10 / _____ \
11 \ _____ /
12 / _____ \
13 \ _____ /
14 / _____ \
15 \ _____ /
16 / _____ \
17 \ _____ /
18 / _____ \
19 \ _____ /
20 / _____ \
21 \ _____ /
22 / _____ \
23 \ _____ /
24 / _____ \
25 \ _____ /
26 / _____ \
27 \ _____ /
28 / _____ \
29 \ _____ /
30 / _____ \
31 \ _____ /
32 / _____ \
33 module String_HW_Avalon #(MAX_BLOCKS = 2, ADDRESS_BITS = 5)
34     (input logic clk, reset, read, write, chipselect,
35      input logic [ADDRESS_BITS - 1:0] address,
36      input logic [31:0] writedata,
37      output logic [31:0] readdata
38      );
39
40     logic write_reg_A, write_reg_B, write_reg_Control;
41     logic read_reg_A, read_reg_B, read_reg_Control, read_reg_Result;
42
43     logic [31:0] Control;
44
45     logic start, done;
46     logic [3:0] index;
47     logic [7:0] length;
48
49     /* ----- Control/Status Flags ----- */
50     assign read_reg_Control = (address == 0) && read && chipselect;
51     assign write_reg_Control = (address == 0) && write && chipselect;
52
53
54     /* ----- StringA Flags -----
55     * ADDRESS 1 - MAX_BLOCKS
56     */
57     logic [0:MAX_BLOCKS-1] [31:0] StringA;
58     assign write_reg_A = (address >= 1) && (address <= MAX_BLOCKS) &&
59     write && chipselect; // Write Register Flags
60     assign read_reg_A = (address >= 1) && (address <= MAX_BLOCKS) &&
61     read && chipselect && ~done; // Read Register Flags
62
63     /* ----- StringB Flags -----
64     * ADDRESS 9 - 2*MAX_BLOCKS
65     */

```

```

65     assign write_reg_B      = (address > MAX_BLOCKS) && (address <=
(MAX_BLOCKS+MAX_BLOCKS)) && write && chipselect;    // Write Register Flags
66     assign read_reg_B      = (address > MAX_BLOCKS) && (address <=
(MAX_BLOCKS+MAX_BLOCKS)) && read && chipselect;    // Read Register Flags
67
68     /* ----- Result Flags -----
69     * ADDRESS 9 - 2*MAX_BLOCKS
70     */
71     logic [0:MAX_BLOCKS-1] [31:0] Result;
72     assign read_reg_Result  = (address >= 1) && (address <= MAX_BLOCKS) && read &&
chipselect && done;    // Read Register Flags
73
74
75     // Instantiate String_HW module
76     String_HW U0(.clk(clk),
77                 .reset(reset),
78                 .go(start),
79                 .index(index),
80                 .length(length),
81                 .A(StringA),
82                 .B(StringB),
83                 .done(done),
84                 .Result(Result)
85                 );
86
87     // Control Register bits
88     assign Control[0] = done;    // Output
89     assign start      = Control[1];    // Input
90     assign index      = Control[5:2];    // Input
91     assign length     = Control[13:6];    // Input
92
93
94     // Process Read & Write Commands StringA and StringB
95     always_ff@(posedge clk)
96     begin
97         if (reset) begin
98             StringA      <= '{default:32'h0};    // initialize StringA to NULL Chars
99             StringB      <= '{default:32'h0};    // initialize StringB to NULL Chars
100            Control[31:1] <= 0;    // Reset Control Register
101            end
102            // StringA & StringB Read/Write
103            else begin
104                if (write_reg_Control)    Control[31:1] <= writedata[31:1];
// WRITE Control/STATUS REGISTER (ignore bit 0: done)
105                else if (read_reg_Control) readdata <= Control;
// READ Control/STATUS REGISTER
106                else if (write_reg_A)    StringA[address - 1] <= writedata;
// WRITE TO StringA
107                else if (read_reg_A)    readdata <= StringA[address - 1];
// READ FROM StringA
108                else if (write_reg_B)    StringB[address - MAX_BLOCKS - 1] <= writedata;
// WRITE TO StringB
109                else if (read_reg_B)    readdata <= StringB[address - MAX_BLOCKS - 1];
// READ FROM StringB
110                else if (read_reg_Result) begin
111                    readdata <= Result[address-1];
// READ FROM RESULT
112                    StringA <= 0;
// Reset String A
113                    StringB <= 0;
// Reset String B
114                end
115            end
116        end
117    endmodule

```

```

1  /*
2  * #####
3  * CPE423
4  * String_HW
5  * David Tougaw and Matthew Bowen
6
7  ( _
8  \ _
9  \ _
10 \ _
11 \ _
12 \ _
13 \ _
14 * 12/9/2019
15 * -----
16 * String.h Hardware Accelerator +
17 * -----
18 * Dev BOARD => Altera DE2-115
19 * -----
20 * 0) Wait for go
21 * 1) Go to string function depending on index value
22 * 2) Perform Computation
23 * 3) Wait in Done state until go bit reset
24 * #####
25 */
26 module String_HW #(MAX_BLOCKS = 2)
27     (input logic clk, reset, go,
28      input logic [3:0] index,
29      input logic [7:0] length,
30      input logic [0:MAX_BLOCKS*4-1][7:0] A, B,
31      output logic done,
32      output logic [0:MAX_BLOCKS*4-1][7:0] Result
33     );
34     parameter RESET=4'd0, S1=4'd1, S2=4'd2,
35                S3=4'd3, S4=4'd4, S5=4'd5,
36                S6=4'd6, S7=4'd7, S8=4'd8,
37                S9=4'd9, DONE =4'd10;
38
39     logic [3:0] state, nextstate;
40     logic found;
41     integer i, j, count, string_index; // Counter variables
42
43     always_ff @(posedge clk)
44         if (reset) state <= RESET; // synchronous Reset
45         else state <= nextstate;
46
47     always@(posedge clk) begin
48         case(state)
49             // RESET State
50             RESET: begin
51                 done <= 0; // Reset done flag
52                 nextstate <= S1; // Go to wait state
53                 count <= 0; // Reset count
54                 Result <= '{default:8'h0}; // Reset result
55             end
56             // Wait for go signal
57             S1: begin
58                 done <= 0; // Reset done flag
59                 i <= 0; // Reset counter i
60                 j <= 0; // Reset counter j
61                 found <= 0; // Reset found flag
62                 Result <= '{default:8'h0}; // Reset result
63                 if (go)
64                     nextstate <= S2; // Go flag set, go to Function
65                     Selection
66             end
67             S2: begin
68                 done <= 0; // Reset done flag
69                 i <= 0; // Reset counter i
70                 j <= 0; // Reset counter j
71                 found <= 0; // Reset found flag
72                 Result <= '{default:8'h0}; // Reset result
73                 if (go)
74                     nextstate <= S3; // Go flag set, go to Function
75                     Selection
76             end
77             S3: begin
78                 done <= 0; // Reset done flag
79                 i <= 0; // Reset counter i
80                 j <= 0; // Reset counter j
81                 found <= 0; // Reset found flag
82                 Result <= '{default:8'h0}; // Reset result
83                 if (go)
84                     nextstate <= S4; // Go flag set, go to Function
85                     Selection
86             end
87             S4: begin
88                 done <= 0; // Reset done flag
89                 i <= 0; // Reset counter i
90                 j <= 0; // Reset counter j
91                 found <= 0; // Reset found flag
92                 Result <= '{default:8'h0}; // Reset result
93                 if (go)
94                     nextstate <= S5; // Go flag set, go to Function
95                     Selection
96             end
97             S5: begin
98                 done <= 0; // Reset done flag
99                 i <= 0; // Reset counter i
100                j <= 0; // Reset counter j
101                found <= 0; // Reset found flag
102                Result <= '{default:8'h0}; // Reset result
103                if (go)
104                    nextstate <= S6; // Go flag set, go to Function
105                    Selection
106            end
107            S6: begin
108                done <= 0; // Reset done flag
109                i <= 0; // Reset counter i
110                j <= 0; // Reset counter j
111                found <= 0; // Reset found flag
112                Result <= '{default:8'h0}; // Reset result
113                if (go)
114                    nextstate <= S7; // Go flag set, go to Function
115                    Selection
116            end
117            S7: begin
118                done <= 0; // Reset done flag
119                i <= 0; // Reset counter i
120                j <= 0; // Reset counter j
121                found <= 0; // Reset found flag
122                Result <= '{default:8'h0}; // Reset result
123                if (go)
124                    nextstate <= S8; // Go flag set, go to Function
125                    Selection
126            end
127            S8: begin
128                done <= 0; // Reset done flag
129                i <= 0; // Reset counter i
130                j <= 0; // Reset counter j
131                found <= 0; // Reset found flag
132                Result <= '{default:8'h0}; // Reset result
133                if (go)
134                    nextstate <= S9; // Go flag set, go to Function
135                    Selection
136            end
137            S9: begin
138                done <= 0; // Reset done flag
139                i <= 0; // Reset counter i
140                j <= 0; // Reset counter j
141                found <= 0; // Reset found flag
142                Result <= '{default:8'h0}; // Reset result
143                if (go)
144                    nextstate <= DONE; // Go flag set, go to Done state
145                    Selection
146            end
147            DONE: begin
148                done <= 1; // Done flag set
149                nextstate <= S1; // Go to wait state
150            end
151        endcase
152    end
153 endmodule

```

```

66         nextstate <= S1;           // Wait for go
67     end
68
69     // Read index for computation
70     S2: begin
71         case (index)
72             0: nextstate <= S3;     // String Compare
73             1: nextstate <= S4;     // String To Upper
74             2: nextstate <= S5;     // String To Lower
75             3: nextstate <= S6;     // String Reverse
76             4: nextstate <= S7;     // String Search
77             default: nextstate <= RESET; // Invalid index
78         endcase
79     end
80
81     // String Compare [index = 0]
82     S3: begin
83         if (A == B)
84             Result <= 1;           // Strings Equal, return 1
85         else
86             Result <= 0;           // Strings Not Equal, return 0
87
88             nextstate <= DONE;
89
90         end
91
92     // String To Upper [index = 1]
93     S4: begin
94         for (i = 0; i < MAX_BLOCKS*4; i = i+1)
95             if (A[i] >= "a" && A[i] <= "z") // if character is lowercase
96                 Result[i] <= A[i] - 32;     // Convert to uppercase
97             else
98                 Result[i] <= A[i];           // Unchanged
99
100             nextstate <= DONE;
101         end
102
103     // String to Lower [index = 2]
104     S5: begin
105         for (i = 0; i < MAX_BLOCKS*4; i = i+1)
106             if (A[i] >= "A" && A[i] <= "Z") // if character is uppercase
107                 Result[i] <= A[i] + 32;     // Convert to lowercase
108             else
109                 Result[i] <= A[i];           // Unchanged
110
111             nextstate <= DONE;
112         end
113
114     // String Reverse [index = 3]
115     S6: begin
116         for (i = 0; i < MAX_BLOCKS*4; i = i+1)
117             Result[i] <= A[MAX_BLOCKS*4-1 - i]; // Reverse String
118
119             nextstate <= DONE;
120         end
121
122     // String Search [index = 4]
123     S7: begin
124         if (found) begin
125             Result <= string_index; // String found, assign Result
126                                     = string starting location
127             nextstate <= DONE;       // Go to Done state
128         end
129         else
130             Result <= 256;           // Default "Not Found" value

```

```

131         if (i < MAX_BLOCKS*4 && ~found) begin
132             if (B[j] == A[i]) begin
133                 if (j == 0)           // First character correct
134                     string_index <= i; // Mark starting location of
                                         string
135                 j <= j + 1;           // Increment j for every
                                         correct character in sequence
136
137                 if (j == length-1)    // If number of correct
                                         characters in sequence = length
138                     found <= 1;       // String has been found
139             end
140         else
141             j = 0;                    // If character in sequence is
                                         incorrect, reset sequence counter j
142
143             i <= i + 1;                // Go to next character in string
144         end
145     else
146         nextstate <= DONE;
147     end
148
149     // DONE State.
150     DONE: begin
151         done <= 1;                    // Done flag set
152
153         // Wait until go signal is deasserted
154         if (~go)
155             nextstate <= S1;          // Go to Wait state
156         else
157             nextstate <= DONE;        // Stay in Done state
158         end
159
160     default: nextstate <= RESET;
161 endcase
162 end
163 endmodule

```



```

1  /*
2  * #####
3  * CPE423
4  * FINAL PROJECT - String.h Hardware Accelerator +
5
6  / _____ \
7  \ _____ /
8  / _____ \
9  \ _____ /
10 / _____ \
11 \ _____ /
12
13 * David Tougaw
14 * 12/9/19
15 * -----
16 * D. Tougaw & Matthew Bowen
17 * -----
18 * Dev BOARD => Altera DE2-115
19 * DE2-115 Computer System + Custom Avalon comp.
20 * -----
21 * JTAG_UART used for terminal inputs and outputs
22 * -----
23 * #####
24 */
25 // include files
26 #include "address_map_nios2.h"
27
28 // needed for printf
29 #include <stdio.h>
30 #include <string.h>
31 #include <stdint.h>
32
33 // INTERVAL TIMER
34 #define CLOCK_RATE 50000000.0
35
36 #define BUFFER_SIZE 64
37
38 // STRING.h HW Macros
39 #define READ_CONTROL_STATUS *(String_HW_ptr)
40 #define WRITE_CONTROL_STATUS *(String_HW_ptr)
41 #define CLEAR_CONTROL_STATUS *(String_HW_ptr) = 0;
42 #define MAX_BLOCKS 8
43
44 // INDEXES
45 #define TEST -13
46 #define COMPARE 0
47 #define TOUPPER 1
48 #define TOLOWER 2
49 #define REVERSE 3
50 #define SEARCH 4
51
52 /* function prototypes */
53 char get_char( void );
54 uint32_t get_uint( void );
55 void put_char( char c );
56 uint32_t pow(uint32_t n, char p);
57 uint32_t stringToInt32bit(char buffer[], unsigned lastIndex);
58 unsigned int inputParamTerminal(char buffer[]); // Retrieves string input
59 // from terminal
60 void start_timer();
61 uint32_t snapshot_timer();
62 void get4Chars(char* string, char *out, int index);
63 void get4CharsInt(uint32_t value, char *out);
64 void pointer4CharsInt(uint32_t value, char * out);
65 void clearTerminal();
66 void SwapValue(char *a, char *b);

```

```

66
67 void stringHWCalls(uint32_t index,char* stringA,char* stringB, char length);
68
69 /* String SW prototypes */
70 void strToUpper(char* string, int length);
71 void strToLower(char* string, int length);
72 void strReverse(char* string, char length);
73
74 // POINTERS
75 volatile uint32_t * TIMER_ptr = (uint32_t *)TIMER_BASE;
76 volatile uint32_t * String_HW_ptr = (uint32_t *)String_HW_BASE;
77
78 char length = 64;
79 char test[] = "lylatagssongdamptynecapebarnflowonceafanjohnleadkokodirtgeekhaul"; //
double quotes add null terminator
80
81 char cmp_1 [] = "LYLA----LYLA----LYLA----LYLA----";
82 char cmp_2 [] = "LYLA----DAVE----LYLA----LYLA----";
83 char str1_UPPER_rev[] = "LYLAtagsSONGdamptyneCAPEbarnFLOW";
84
85 char str1_UPPER[] = "LYLAtagsSONGdamptyneCAPEBARNflow";
86 char find [] = "tags";
87
88 void main() {
89
90     uint32_t ticksHW,ticksSW;
91
92     while(1){
93         char str1[] = "lylatagssongdamptynecapebarnflow"; // double quotes add null
terminator
94         char str2[] = "onceafanjohnleadkokodirtgeekhaul"; // double quotes add null
terminator
95         char out [4]; // temp var
96         // reset for next round
97         ticksSW=0;
98         ticksHW=0;
99         // MENU ECHO
100        printf("\n#### string.h vs String HW + peripheral ####\n");
101        printf("##### BY D. Tougaw & Matthew Bowen #####\n");
102        printf("%-7s%-25s\n", "Index", "Function");
103        printf("%-7s%-25s\n", "#", "TEST READ/WRITE AVALON");
104        printf("%-7s%-25s\n", "0", "Compare");
105        printf("%-7s%-25s\n", "1", "ToUpper");
106        printf("%-7s%-25s\n", "2", "ToLower");
107        printf("%-7s%-25s\n", "3", "Reverse");
108        printf("%-7s%-25s\n", "4", "Search");
109        printf("Select function [Index]: ");
110        inputParamTerminal(out);
111        putchar('\n');
112        // input ASCII to number
113        char index = out[0] - 48;
114        switch(index) {
115            case TEST: {
116                printf("\n#### TEST READ/WRITE AVALON ####\n");
117                // WRITE 4 char blocks to HW module
118                char k;
119                printf("Control/Status: %x\n",READ_CONTROL_STATUS);
120
121                // Write StringA and StringB
122                for(k=0; k < length/4; k++)
123                {
124                    get4Chars(test,out, k);
125                    *(String_HW_ptr + k + 1) = *((uint32_t *)(out));
126                    if((k+1) <= 8)
127                        printf("Write A: %s \n",out);
128                    else

```

```

129         printf("Write B: %s \n",out);
130     }
131
132     // TEST CONTROL READ/WRITE
133     WRITE_CONTROL_STATUS = 0xFEED0000;
134     printf("WRITE Control/Status: %x\n",0xFEED0000);
135
136     // PRINT TO CONSOLE INT TO CHAR
137     printf("Control/Status: %x\n",READ_CONTROL_STATUS);
138     for(k=0; k < length/4; k++)
139     {
140         uint32_t val;
141         val = *(String_HW_ptr + k + 1);
142         //printf("Read HEX: %x \n",val);
143         if((k+1) <= 8)
144             printf("Read A: ");
145         else
146             printf("Read B: ");
147         putchar(val & 0x000000FF);
148         putchar((val & 0x0000FF00) >> 8);
149         putchar((val & 0x00FF0000) >> 16);
150         putchar((val & 0xFF000000) >> 24);
151         putchar('\n');
152     }
153     CLEAR_CONTROL_STATUS;
154 }break;
155 case COMPARE: {
156     printf("===== StringCompare(str1, str2) =====\n");
157     char k;
158     // Write StringA
159     for(k=0; k < MAX_BLOCKS; k++)
160     {
161         get4Chars(cmp_1,out, k);
162         *(String_HW_ptr + k + 1) = *((uint32_t *) (out));
163         //printf("WRITE A: %s \n",out);
164     }
165     printf("String A: %s \n",cmp_1);
166     printf("String B: %s \n",cmp_2);
167     // Write StringB
168     for(k=0; k < MAX_BLOCKS; k++)
169     {
170         get4Chars(cmp_2,out, k);
171         *(String_HW_ptr + k + MAX_BLOCKS + 1) = *((uint32_t *) (out));
172         //printf("WRITE B: %s \n",out);
173     }
174     // WRITE INDEX and GO BIT
175     // SOFTWARE CC
176     start_timer();
177     WRITE_CONTROL_STATUS = 0b00010; // index = 0, go = 1
178     // POLL for DONE BIT
179     while(!(READ_CONTROL_STATUS & 1));
180
181     ticksHW = snapshot_timer();
182
183     uint32_t resHW = *(String_HW_ptr + MAX_BLOCKS);
184
185     // CLEAR CONTROL/STATUS REGISTER
186     CLEAR_CONTROL_STATUS;
187
188     // SOFTWARE CC
189     start_timer();
190     uint32_t resSW = strcmp(cmp_1,cmp_2);
191     ticksSW = snapshot_timer();
192
193     /***** StringCompare Display Code *****/
194     printf("===== StringCompare(str1, str2) =====\n");

```

```

195         //printf("Software strcmp(%s, %s) = ",str1, str1);
196         if (!resSW) printf("SW EQUAL \n");
197         else         printf("SW NOT EQUAL \n");
198         //printf("Hardware strcmp(%s, %s) = ",str1, str1);
199         if (resHW)   printf("HW EQUAL \n");
200         else         printf("HW NOT EQUAL \n");
201         printf("Software CC = %-8d ET = %-5f\n",ticksSW,ticksSW/CLOCK_RATE*1000000);
202         printf("Hardware CC = %-8d ET = %-5f\n",ticksHW,ticksHW/CLOCK_RATE*1000000);
203         printf("Speedup = %-8f\n",ticksSW*1.0/ticksHW);
204         printf("=====\n");
205
206     }break;
207     case TOUPPER: {
208
209         printf("===== StringToUpper(str) =====\n");
210         char k;
211         // Write StringA
212         for(k=0; k < MAX_BLOCKS; k++)
213         {
214             get4Chars(str1,out, k);
215             *(String_HW_ptr + k + 1) = *((uint32_t *)(out));
216         }
217         printf("String A: %s \n",str1);
218         // WRITE INDEX and GO BIT
219         // HARDWARE CC
220         start_timer();
221         WRITE_CONTROL_STATUS = ((uint32_t) index << 2) | 2;
222
223         // POLL for DONE BIT
224         while(!(READ_CONTROL_STATUS & 1));
225
226         ticksHW = snapshot_timer();
227
228         // SOFTWARE CC
229         start_timer();
230         strToUpper(str1,32);
231         ticksSW = snapshot_timer();
232
233         // WRITE READ Result 32 bits (4chars) at a time
234         for(k=0; k < MAX_BLOCKS; k++)
235         {
236             uint32_t val;
237             val = *(String_HW_ptr + k + 1);
238             printf("Read A: ");
239
240             putchar(val & 0x000000FF);
241             putchar((val & 0x0000FF00) >> 8);
242             putchar((val & 0x00FF0000) >> 16);
243             putchar((val & 0xFF000000) >> 24);
244             putchar('\n');
245         }
246         // CLEAR CONTROL/STATUS REGISTER
247         CLEAR_CONTROL_STATUS;
248         /***** String to Upper Display Code *****/
249
250         printf("===== StringToUpper(str) =====\n");
251         printf("Software CC = %-8d ET = %-5f\n",ticksSW,ticksSW/CLOCK_RATE*1000000);
252         printf("Hardware CC = %-8d ET = %-5f\n",ticksHW,ticksHW/CLOCK_RATE*1000000);
253         printf("Speedup = %-8f\n",ticksSW*1.0/ticksHW);
254         printf("=====\n");
255
256     }break;

```

```

257 case TOLOWER: {
258     printf("===== StringToLower(str) =====\n");
259     char k;
260     // Write StringA
261     for(k=0; k < MAX_BLOCKS; k++)
262     {
263         get4Chars(strl_UPPER,out, k);
264         *(String_HW_ptr + k + 1) = *((uint32_t *)(out));
265     }
266     printf("String A: %s \n",strl_UPPER);
267     // WRITE INDEX and GO BIT
268     // HARDWARE CC
269     start_timer();
270     WRITE_CONTROL_STATUS = ((uint32_t) index << 2) | 2;
271
272     // POLL for DONE BIT
273     while(!(READ_CONTROL_STATUS & 1));
274
275     ticksHW = snapshot_timer();
276
277     // SOFTWARE CC
278     start_timer();
279     strToLower(strl,32);
280     ticksSW = snapshot_timer();
281
282     // WRITE READ Result 32 bits (4chars) at a time
283     for(k=0; k < MAX_BLOCKS; k++)
284     {
285         uint32_t val;
286         val = *(String_HW_ptr + k + 1);
287         printf("Read A: ");
288
289         putchar(val & 0x000000FF);
290         putchar((val & 0x0000FF00) >> 8);
291         putchar((val & 0x00FF0000) >> 16);
292         putchar((val & 0xFF000000) >> 24);
293         putchar('\n');
294     }
295
296     // CLEAR CONTROL/STATUS REGISTER
297     CLEAR_CONTROL_STATUS;
298     /***** String to Lower Display Code *****/
299
300     printf("===== StringToLower(str) =====\n");
301     printf("Software CC = %-8d ET = %-5f\n",ticksSW,ticksSW/CLOCK_RATE*1000000);
302     printf("Hardware CC = %-8d ET = %-5f\n",ticksHW,ticksHW/CLOCK_RATE*1000000);
303     printf("Speedup = %-8f\n",ticksSW*1.0/ticksHW);
304     printf("===== \n");
305
306 }break;
307 case REVERSE: {
308     printf("===== Reverse(str) =====\n");
309     char k;
310     // Write StringA
311     for(k=0; k < MAX_BLOCKS; k++)
312     {
313         get4Chars(strl_UPPER_rev,out, k);
314         *(String_HW_ptr + k + 1) = *((uint32_t *)(out));
315     }
316     printf("String A: %s \n",strl_UPPER_rev);
317     // WRITE INDEX and GO BIT
318     // HARDWARE CC
319     start_timer();
320     WRITE_CONTROL_STATUS = ((uint32_t) index << 2) | 2;

```

```

321
322 // POLL for DONE BIT
323 while(!(READ_CONTROL_STATUS & 1));
324
325 ticksHW = snapshot_timer();
326
327 // SOFTWARE CC
328 start_timer();
329 strReverse(str1_UPPER_rev,32);
330 ticksSW = snapshot_timer();
331 printf("String A SW Reversed: %s \n",str1_UPPER_rev);
332
333 // WRITE READ Result 32 bits (4chars) at a time
334 for(k=0; k < MAX_BLOCKS; k++)
335 {
336     uint32_t val;
337     val = *(String_HW_ptr + k + 1);
338     printf("Read A: ");
339
340     putchar(val & 0x000000FF);
341     putchar((val & 0x0000FF00) >> 8);
342     putchar((val & 0x00FF0000) >> 16);
343     putchar((val & 0xFF000000) >> 24);
344     putchar('\n');
345 }
346
347 // CLEAR CONTROL/STATUS REGISTER
348 CLEAR_CONTROL_STATUS;
349 /***** String Reverse Display Code *****/
350
351 printf("===== Reverse(str) =====\n");
352 printf("Software CC = %-8d ET = %-5f\n",ticksSW,ticksSW/CLOCK_RATE*1000000);
353 printf("Hardware CC = %-8d ET = %-5f\n",ticksHW,ticksHW/CLOCK_RATE*1000000);
354 printf("Speedup = %-8f\n",ticksSW*1.0/ticksHW);
355 printf("===== \n");
356 }break;
357 case SEARCH: {
358     printf("===== Search(strA,strB) =====\n");
359     char k;
360     // Write StringA
361     for(k=0; k < MAX_BLOCKS; k++)
362     {
363         get4Chars(str1_UPPER,out, k);
364         *(String_HW_ptr + k + 1) = *((uint32_t *)(out));
365     }
366     printf("String A: %s \n",str1_UPPER);
367
368     // Uncomment and handle more than 4 chars with loop
369     // get4Chars(find,out, 0);
370
371     // Write StringB
372     printf("String B: %s \n",find);
373     *(String_HW_ptr + 9) = *((uint32_t *)(find));
374
375     uint32_t len = 4;
376
377     // WRITE INDEX and GO BIT
378     // HARDWARE CC
379     start_timer();
380     WRITE_CONTROL_STATUS = (len << 6) | ((uint32_t) index << 2) | 2;
381
382     while(!(READ_CONTROL_STATUS & 1));
383
384

```

```

385         ticksHW = snapshot_timer();
386
387         // SOFTWARE CC
388         start_timer();
389
390         char *ptr = strstr(str1_UPPER, find);
391         char res = (ptr == NULL) ? -1 : ptr - str1_UPPER;
392
393         ticksSW = snapshot_timer();
394         uint32_t resHW;
395         resHW = *(String_HW_ptr + MAX_BLOCKS);
396
397         // CLEAR CONTROL/STATUS REGISTER
398         CLEAR_CONTROL_STATUS;
399         /***** String Search Display Code *****/
400
401         printf("===== Search(strA,strB) =====\n");
402         if (res>=0) printf("SW FOUND at pos: %d \n",res);
403         else printf("SW NOT FOUND \n");
404         if (resHW != 0xFF) printf("HW FOUND at pos: %d \n",resHW);
405         else printf("HW NOT FOUND \n");
406         printf("Software CC = %-8d ET = %-5f\n",ticksSW,ticksSW/CLOCK_RATE*1000000);
407         printf("Hardware CC = %-8d ET = %-5f\n",ticksHW,ticksHW/CLOCK_RATE*1000000);
408         printf("Speedup = %-8f\n",ticksSW*1.0/ticksHW);
409         printf("===== \n");
410     }break;
411     // DEFAULT CASE when UNKNOWN INDEX
412     default: {
413         printf("\nOption not handled.\n");
414     }break;
415
416 }
417
418 printf("Any char to continue..");
419 // WAIT FOR ANY KEYBOARD INPUT
420 inputParamTerminal(str2);
421
422 // CLEAR TERMINAL
423 clearTerminal();
424 }
425 }
426
427 /*****
428  * StringReverse Function reverses string
429  *****/
430 void strReverse(char* string, char length)
431 {
432     // Swap character starting from two
433     // corners
434     for (int i = 0; i < length / 2; i++)
435         SwapValue(&string[i], &string[length - i - 1]);
436 }
437
438 /*****
439  * StringToUpper Function converts inputted string to uint32_teger 32 bit
440  *****/
441 void strToUpper(char* string, int length)
442 {
443     for (int i = 0; i < length; i++)
444         if (string[i] >= 'a' && string[i] <= 'z')
445             string[i] -= 32;
446 }
447
448 /*****

```



```

449     * StringToLower Function converts inputted string to uint32_teger 32 bit
450     *****/
451 void strToLower(char* string, int length)
452 {
453     char result[4];
454     for (int i = 0; i < length; i++)
455         if (string[i] >= 'A' && string[i] <= 'Z')
456             string[i] += 32;
457 }
458
459 /*****
460  * SwapValue Function
461  *****/
462
463 void SwapValue(char *a, char *b) {
464     char t = *a;
465     *a = *b;
466     *b = t;
467 }
468
469 /*****
470  * clearTerminal()
471  *****/
472 void clearTerminal()
473 {
474     putchar(0x1B);
475     putchar('[');
476     putchar('2');
477     putchar('J');
478 }
479
480 /*****
481  * get4Chars(char index)
482  * index specifies 4 char blocks
483  *****/
484 void get4Chars(char* string, char *out, int index)
485 {
486     out[0]=string[0+4*index];
487     out[1]=string[1+4*index];
488     out[2]=string[2+4*index];
489     out[3]=string[3+4*index];
490 }
491
492 /*****
493  * get4CharsInt(char index)
494  *****/
495 void get4CharsInt(uint32_t value, char *out)
496 {
497     //char out [4];
498     out[0]=(char)(value & 0xFF000000) >> 24;
499     out[1]=(char)(value & 0x00FF0000) >> 16;
500     out[2]=(char)(value & 0x0000FF00) >> 8;
501     out[3]=(char)(value & 0x000000FF);
502     //return (uint32_t)out;
503 }
504
505 /*****
506  * print4CharsInt(char index)
507  *****/
508 void pointer4CharsInt(uint32_t value, char * out)
509 {
510     //char out [5];
511     out[0]=(char)(value & 0xFF000000) >> 24;
512     out[1]=(char)(value & 0x00FF0000) >> 16;
513     out[2]=(char)(value & 0x0000FF00) >> 8;
514     out[3]=(char)(value & 0x000000FF);

```

```

515     out[4]=(char) 0x0A; // NULL CHAR
516     //return out;
517 }
518
519
520 /*****
521  * inputParamTerminal Function
522  *****/
523 unsigned int inputParamTerminal(char buffer[])    // Retrieves string input from
terminal
524 {
525     char in_char;
526     unsigned int num, i;
527     num = 0; i = 0;
528
529     in_char = get_char();
530     while(in_char != '\r' && in_char != '\n')    // Wait until character entered
thats not ENTER
531     {
532
533         if (in_char != '\0')    // If not NULL,
534         {
535             if (in_char == 0x08) // backspace
536             {
537                 if (i > 0) // Only backspace if there are characters in buffer
538                 {
539                     i--;
540                     printf("%c",in_char);
541                     buffer[i] = 0x00; // Delete previous char from buffer (NUL
character)
542                 }
543             }
544             else
545             {
546                 if (i < BUFFER_SIZE)
547                 {
548                     printf("%c",in_char);
549                     buffer[i] = in_char; // Add char to buffer
550                     i++;                // Increment counter
551                 }
552             }
553         }
554         in_char = get_char();
555     }
556     return i; // return length of string
557 }
558
559 /*****
560  * start_timer Function
561  *****/
562
563 void start_timer(){
564     volatile int * TIMER_ptr = (int *)TIMER_BASE;
565     *(TIMER_ptr + 2) = 0xFFFF;    // Reset timer to 0xFFFF
566     *(TIMER_ptr + 1) = 0x4;        // Set start bit
567 }
568
569 /*****
570  * snapshot_timer Function
571  *****/
572
573 uint32_t snapshot_timer()
574 {
575     *(TIMER_ptr + 4) = 0x1; //dummy write to snap_low
576     return 0xFFFF - *(TIMER_ptr + 4);
577 }

```

```

578
579 /*****
580  * stringToInt32bit Function converts inputted string to uint32_teger 32 bit
581  *****/
582
583 uint32_t stringToInt32bit(char buffer[],unsigned lastIndex)
584 {
585     uint32_t n=0;
586     int k;
587     for(k=0; k<lastIndex;k++) {
588         n+=(buffer[k]-0x30)*pow(10,lastIndex-k-1); // ASCII to int
589         buffer[k] = 0x00; // clear buffer
590     }
591
592     return n;
593 }
594
595 /*****
596  * POW Function
597  *****/
598
599 uint32_t pow(uint32_t base, char p)
600 {
601     uint32_t result=1;
602     char exp;
603     for(exp=p; exp>0;exp--)
604         result = result*base;
605
606     return result;
607 }
608
609
610 /*****
611  * Subroutine to read a character from the JTAG UART
612  * Returns \0 if no character, otherwise returns the character
613  *****/
614 char get_char( void )
615 {
616     volatile int * JTAG_UART_ptr = (int *) JTAG_UART_BASE; // JTAG UART address
617     int data;
618     data = *(JTAG_UART_ptr); // read the JTAG_UART data register
619     if (data & 0x00008000) // check RVALID to see if there is new data
620         return ((char) data & 0xFF);
621     else
622         return ('\0');
623 }
624
625 /*****
626  * Subroutine to send a character to the JTAG UART
627  *****/
628 void put_char( char c )
629 {
630     volatile int * JTAG_UART_ptr = (int *) JTAG_UART_BASE; // JTAG UART address
631     int control;
632     control = *(JTAG_UART_ptr + 1); // read the JTAG_UART control register
633     if (control & 0xFFFF0000) // if space, write character, else ignore
634         *(JTAG_UART_ptr) = c;
635 }

```