Example parser using the Hotstate machine

Parsing is to “analyze (a string or text) into logical syntactic components, typically in order to test conformability to a logical grammar.” (google)

The parser takes text and only passes characters that match a letter on a list to the output. Behind each grouping we insert a carriage return (0x0a) into the text.

In the file ascii_def_in.h are a set of defines that match the ascii character standard.

```
#define _z  ((din0 == 0) & (din1 == 1) & (din2 == 0) & (din3 == 1) &\
(din4 == 1) & (din5 == 1) & (din6 == 1) & (din7 == 0))


#define lower_case \
(_a|_b|_c|_d|_e|_f|_g|_h|_i|_j|_k|_l|_m|_n|_o|_p|_q|_r|_s|_t|_u|_v|_w|_x|_y|_z)
#define upper_case \
(_A|_B|_C|_D|_E|_F|_G|_H|_I|_J|_K|_L|_M|_N|_O|_P|_Q|_R|_S|_T|_U|_V|_W|_X|_Y|_Z)
```

```
#include "ascii_defs_in.h"

// for debugging in software
extern void _user();

#define bool _Bool

bool read_data= 0;
bool write_data = 0;
bool next_data = 0;

bool din0;
bool din1;
bool din2;
bool din3;
bool din4;
bool din5;
bool din6;
bool din7;
bool next_wrap_around;
bool valid;

void main() {
    _user(0); // set input
  // get rid of non-letters in front
  while (!(lower_case | upper_case)) {
```

```
  if (valid == 1) {
    read_data = 1, next_data = 1;
    read_data = 0, next_data = 0;
   }
  else read_data = 0, write_data = 0, next_data = 0;
  }
  // do the rest
  while(1) {
  if (valid == 1) {
    _user(1); // set input
    if ((lower_case | upper_case)) {
      write_data = 1, read_data = 1, next_data = 0;
      write_data = 0, read_data = 0;
    }
    else if (next_wrap_around == 0) {
      next_data = 1;
    }
    else {
      read_data = 1;
      read_data = 0;
    }
   }
  else read_data = 0, write_data = 0, next_data = 0;
  }
}
```

The _user() function is used to manipulate input to the state machine

```
// Copyright (c) 2022 HotWright Inc.
// Created Wed Nov 30 04:34:44 2022
#include "ascii_defs_in.h"
// This is the function _user called out in parser.c

#define bool _Bool

// Output States

extern bool read_data;
extern bool write_data;
extern bool next_data;

// Input Variables

extern bool din0;
extern bool din1;
extern bool din2;
extern bool din3;
extern bool din4;
extern bool din5;
extern bool din6;
extern bool din7;
extern bool next_wrap_around;
extern bool valid;
```

```
valid = 1;
int i = 0;
void _user(int number) {
// make sure this is here. It keeps a state external to the machine
next_wrap_around = next_data;
switch (number) {

        case 0:{ if (i++ < 4) DOT; else {i = 0;  _a; break;}
        case 1:{ i++; if (i==1) _a; if (i == 2) _B; break;}

        }
}
```

Once we have debugged the program using gdb we can start to work on simulating the design.

There are several options that will create testbenches. I suggest –T –S which initializes everything inside the state machine itself.

make tbs //this makes a standalone testbench where the memory files are passed directly to the SystemVerilog code.

This will create some files: parser_tb.v, parser_smdata.mem, parser_vardata.mem, and parser_template.v

While developing your code make sure to setup your tool chain so you can compile the code and not have to copy files around. That's why you have a user.v file. You can compile your code and overwrite the *_tb.v file without losing the functionality you have in the user.v file. Remember if you change the C code by adding variable names you will have to add them manually to your user.v file. When you use an option to create a testbench the compiler will create a user.v file if none exists.

The target board is the Xilinx Kria Vision Starter board running the Ubuntu 22.04.1 OS.

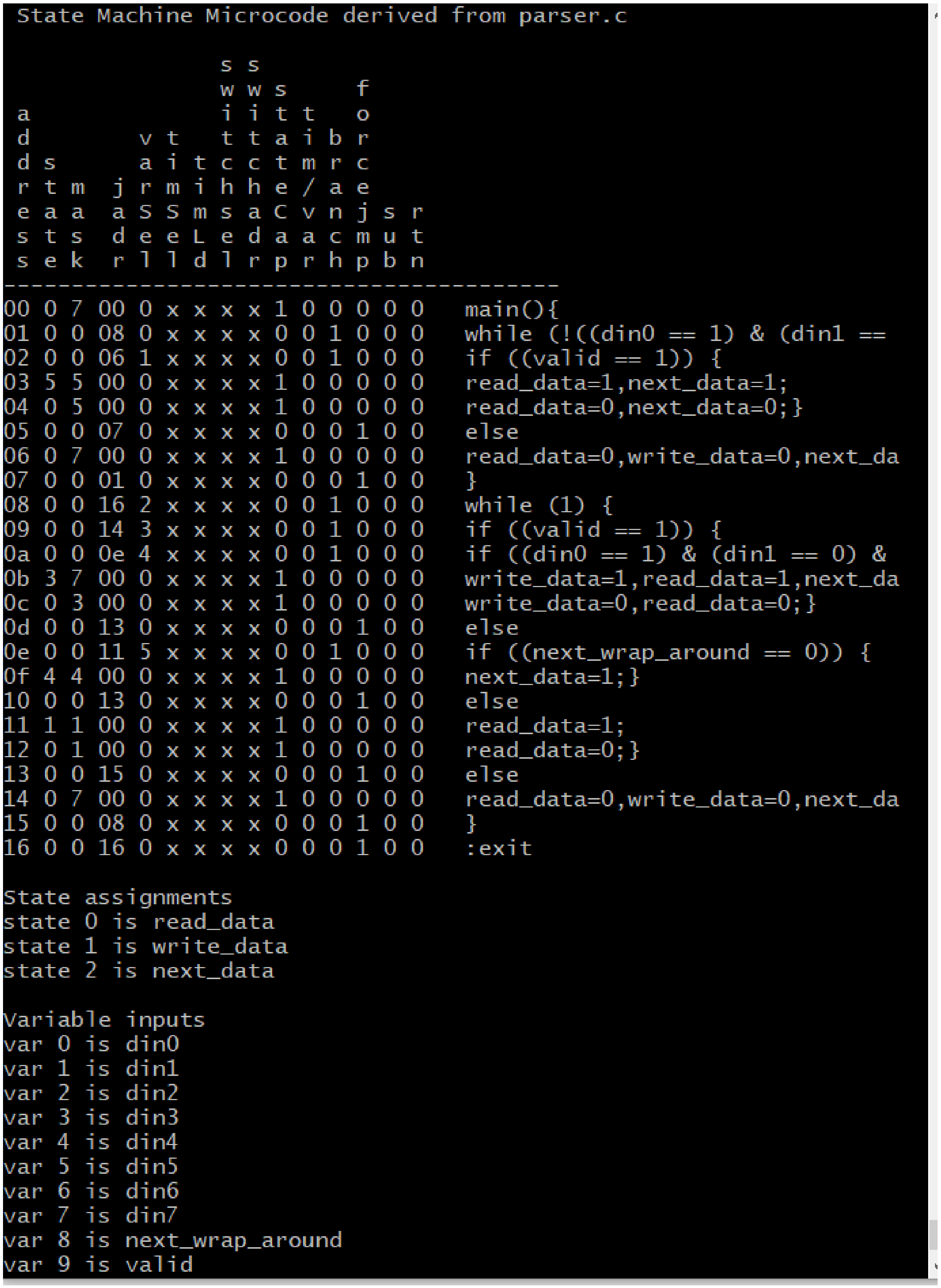
In hotstate/examples/parser you will find parser.c, _user.c, and user.v There are two directories, parser and parser_tb.

Type

hotstate parser.c

This will create a makefile. Edit the makefile and change –A to –a as the equations in the "if" statements are over 1000 characters long.

Type “make alz”. The analyze output will guide you in debugging. The address is the debug_adr in the simulations



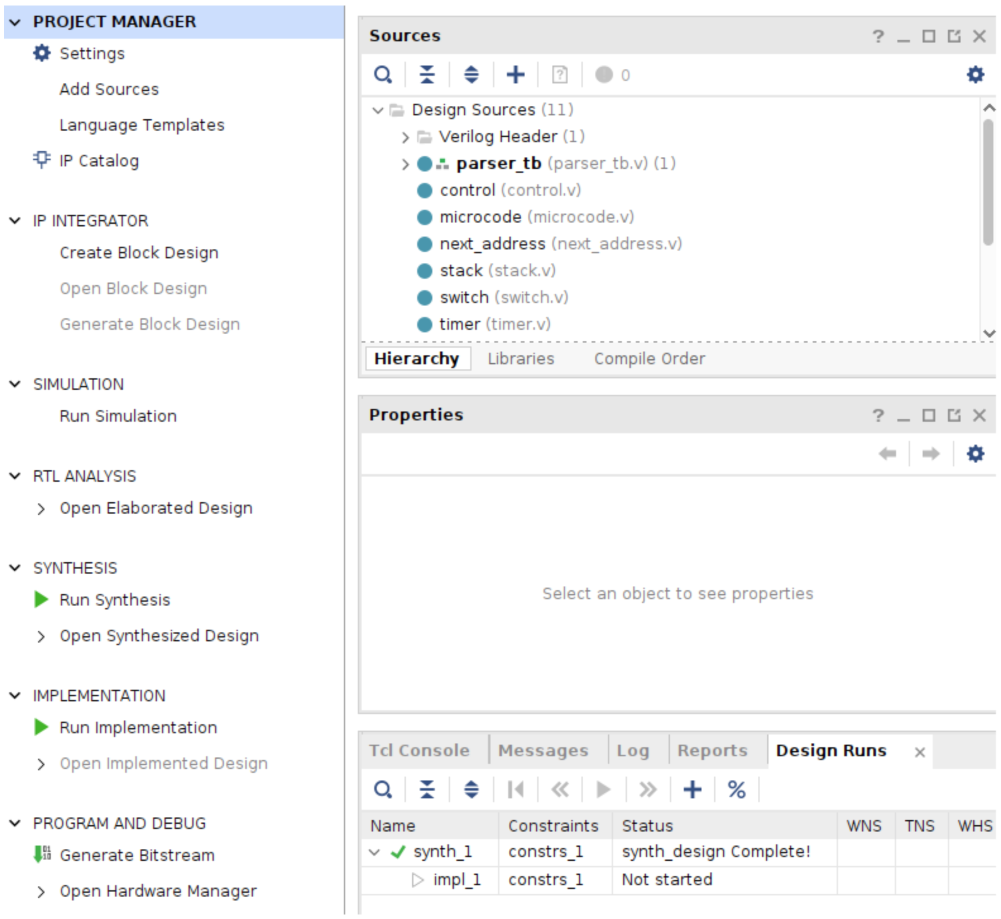
```
 State Machine Microcode derived from parser.c

                s s
                w w s     f
 a              i i t t   o
 d        v t   t t a i b r
 d s      a i t c c t m r c
 r t m  j r m i h h e / a e
 e a a  a S S m s a C v n j s r
 s t s  d e e L e d a a c m u t
 s e k  r l l d l r p r h p b n
-----------------------------------------
00 0 7 00 0 x x x x 1 0 0 0 0 0   main(){
01 0 0 08 0 x x x x 0 0 1 0 0 0   while (!((din0 == 1) & (din1 ==
02 0 0 06 1 x x x x 0 0 1 0 0 0   if ((valid == 1)) {
03 5 5 00 0 x x x x 1 0 0 0 0 0   read_data=1,next_data=1;
04 0 5 00 0 x x x x 1 0 0 0 0 0   read_data=0,next_data=0;}
05 0 0 07 0 x x x x 0 0 0 1 0 0   else
06 0 7 00 0 x x x x 1 0 0 0 0 0   read_data=0,write_data=0,next_da
07 0 0 01 0 x x x x 0 0 0 1 0 0   }
08 0 0 16 2 x x x x 0 0 1 0 0 0   while (1) {
09 0 0 14 3 x x x x 0 0 1 0 0 0   if ((valid == 1)) {
0a 0 0 0e 4 x x x x 0 0 1 0 0 0   if ((din0 == 1) & (din1 == 0) &
0b 3 7 00 0 x x x x 1 0 0 0 0 0   write_data=1,read_data=1,next_da
0c 0 3 00 0 x x x x 1 0 0 0 0 0   write_data=0,read_data=0;}
0d 0 0 13 0 x x x x 0 0 0 1 0 0   else
0e 0 0 11 5 x x x x 0 0 1 0 0 0   if ((next_wrap_around == 0)) {
0f 4 4 00 0 x x x x 1 0 0 0 0 0   next_data=1;}
10 0 0 13 0 x x x x 0 0 0 1 0 0   else
11 1 1 00 0 x x x x 1 0 0 0 0 0   read_data=1;
12 0 1 00 0 x x x x 1 0 0 0 0 0   read_data=0;}
13 0 0 15 0 x x x x 0 0 0 1 0 0   else
14 0 7 00 0 x x x x 1 0 0 0 0 0   read_data=0,write_data=0,next_da
15 0 0 08 0 x x x x 0 0 0 1 0 0   }
16 0 0 16 0 x x x x 0 0 0 1 0 0   :exit

State assignments
state 0 is read_data
state 1 is write_data
state 2 is next_data

Variable inputs
var 0 is din0
var 1 is din1
var 2 is din2
var 3 is din3
var 4 is din4
var 5 is din5
var 6 is din6
var 7 is din7
var 8 is next_wrap_around
var 9 is valid
```

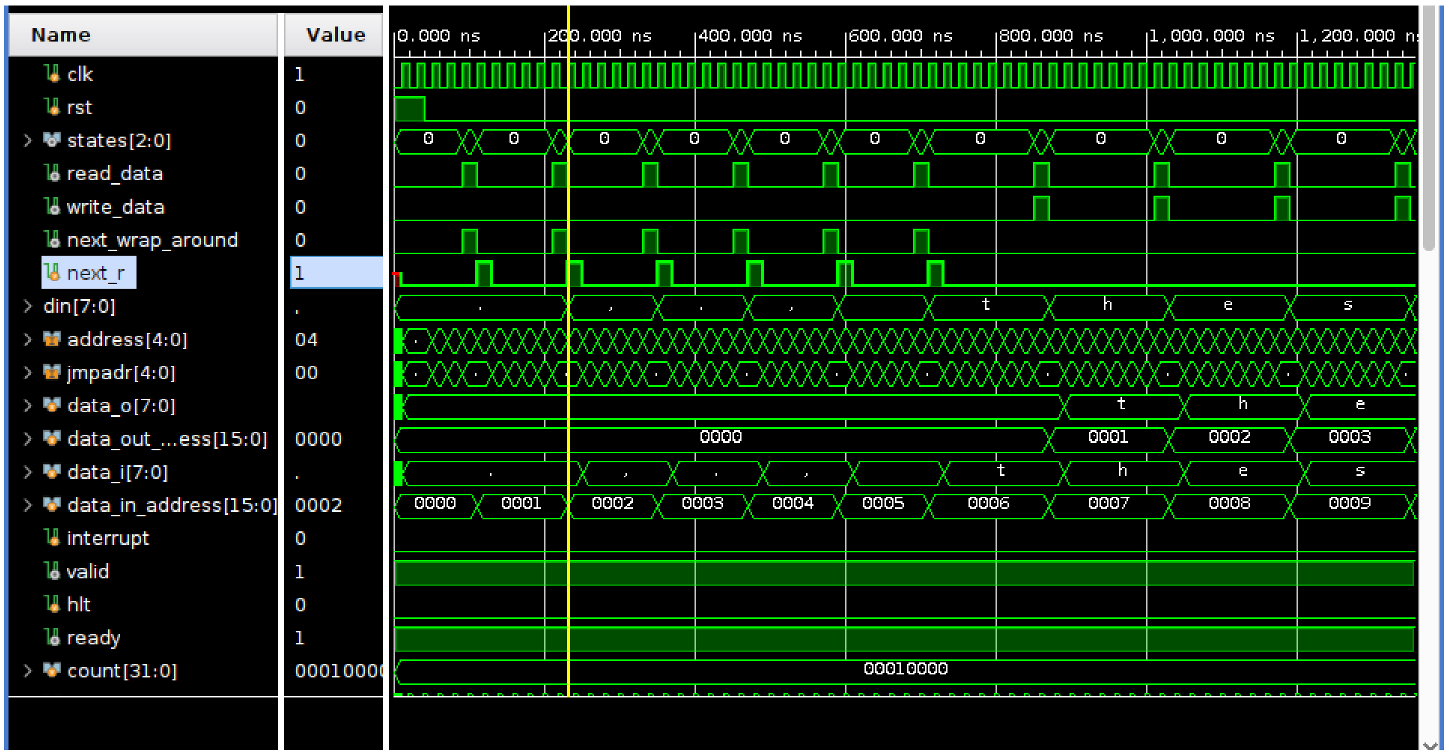
Open the Vivado project file parser/parser_tb/parser_tb.xpr, you'll see something like this.



Design Sources (11)
- Verilog Header (1)
- parser_tb (parser_tb.v) (1)
- control (control.v)
- microcode (microcode.v)
- next_address (next_address.v)
- stack (stack.v)
- switch (switch.v)
- timer (timer.v)

Name	Constraints	Status	WNS	TNS	WHS
synth_1	constrs_1	synth_design Complete!			
impl_1	constrs_1	Not started			

Click on "Run Simulation" and pick "Run Behavioral Simulation" and you'll see



Name	Value
clk	1
rst	0
states[2:0]	0
read_data	0
write_data	0
next_wrap_around	0
next_r	1
din[7:0]	.
address[4:0]	04
jmpadr[4:0]	00
data_o[7:0]	
data_out_...ess[15:0]	0000
data_i[7:0]	.
data_in_address[15:0]	0002
interrupt	0
valid	1
hlt	0
ready	1
count[31:0]	00010000

In hotstate/examples/parser/parser_tb/parser_tb.sim/sim_1/behav/xsim there are 2 files. They are text.txt and parse.out. text.txt is the input file and you can put any ascii file here. If the file is larger than 2000 characters you will have to edit the user.v file and adjust this line if (data_in_address > 2000) begin

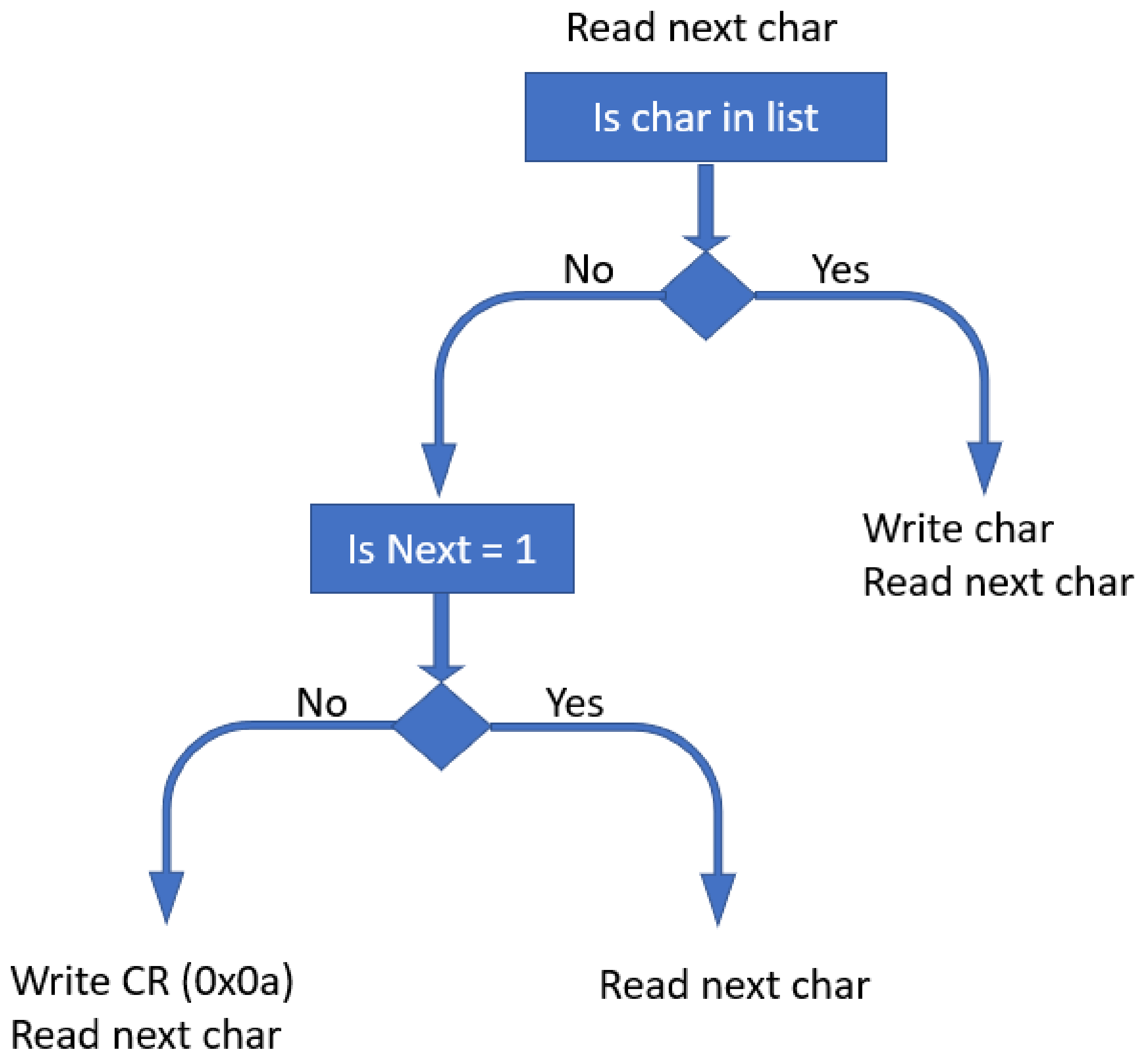
Real Hardware

The parser was implemented on a Xilinx Kria Vision Starter board. Vivado IP Integrator was used enter the hardware design.

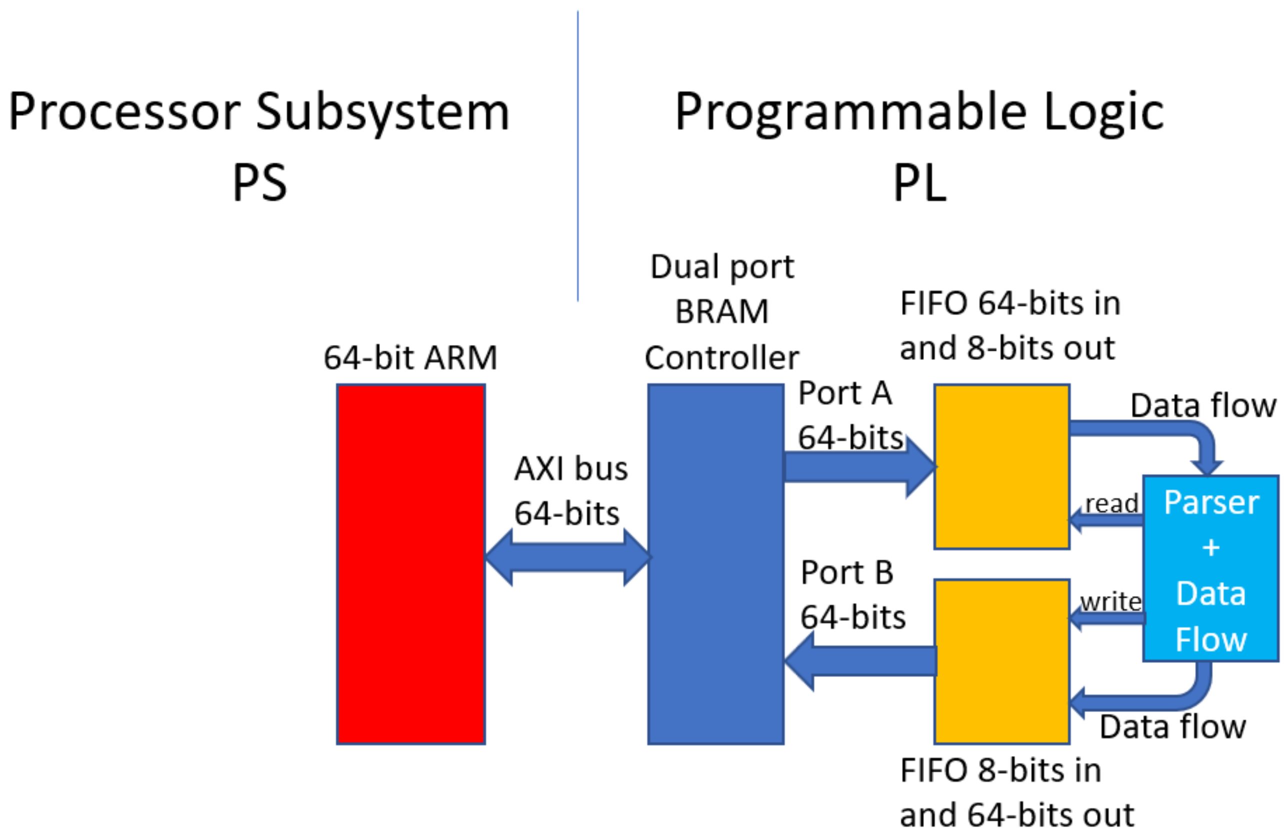
Theory of Operation:

A parser takes a string of characters and tests if a character is in a specified list. This is the algorithm.

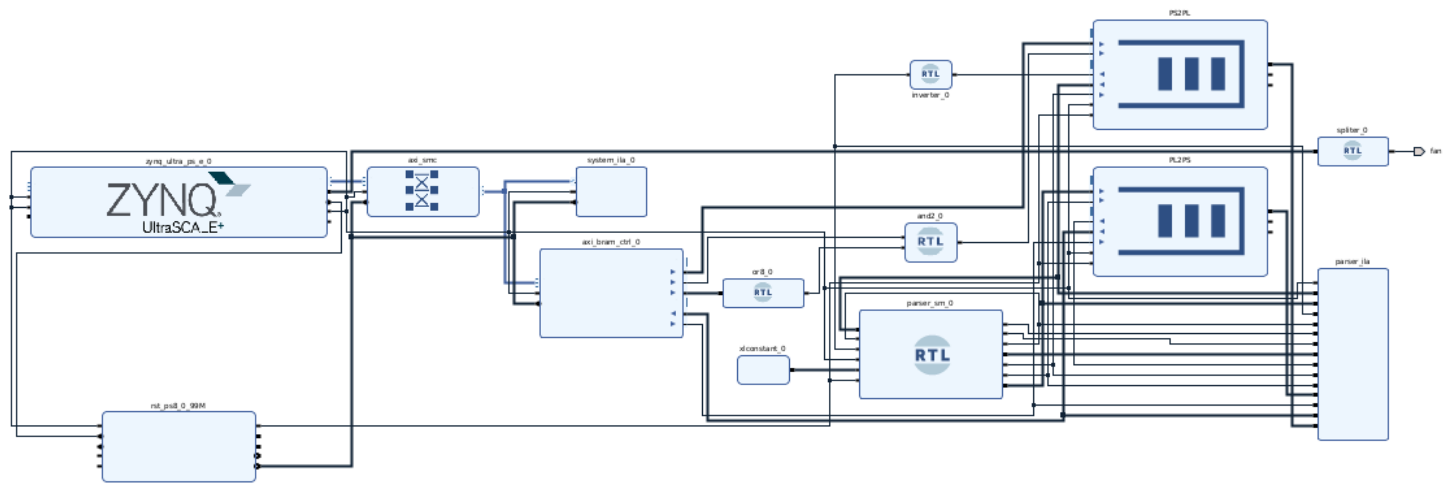
1) Test if the input matches the list. If it does write char and get next char
2) If it does not match the list check to see if the previous char did not match
3) If the previous char did not match the list, read next char
4) If the previous char did match the list, write a carriage return and read next char



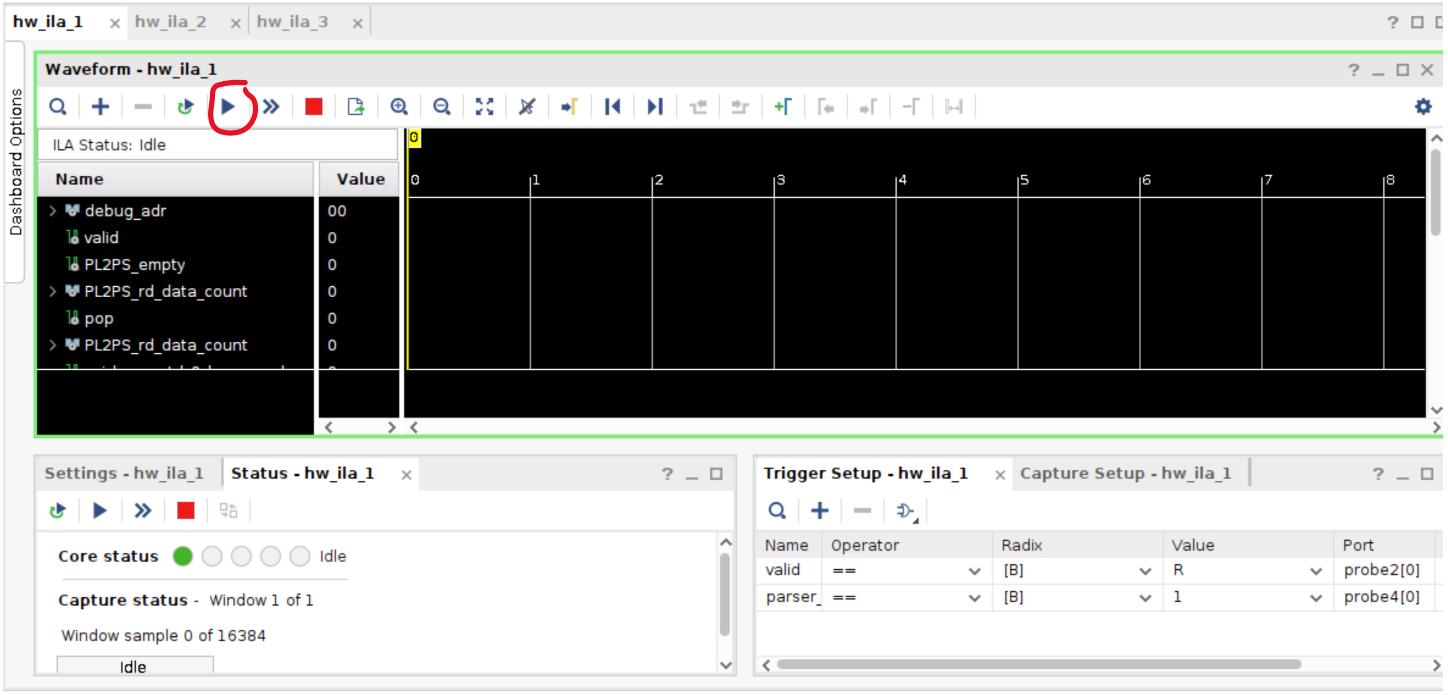
Hardware overview



Open the design at hotstate/examples/parser/parser. Click on “Open Block Design” and you’ll see this. The file that has the state machine is called parser_ip.v. The simulated testbench used a memory model but it was better to use a FIFO model to feed the parser.



This project has the bitstream all ready to go so open the hardware manager and auto connect to the Kria. Then click "program device". A panel will pop up and you can click on "Program". After the device is programmed, you will see 3 ILAs. One is for the top level of the parser, one is for the AXI bus, and one is for signals internal to the hotstate machine



ILA Status: Idle

Name	Value
debug_adr	00
valid	0
PL2PS_empty	0
PL2PS_rd_data_count	0
pop	0
PL2PS_rd_data_count	0

Core status Idle

Capture status - Window 1 of 1

Window sample 0 of 16384

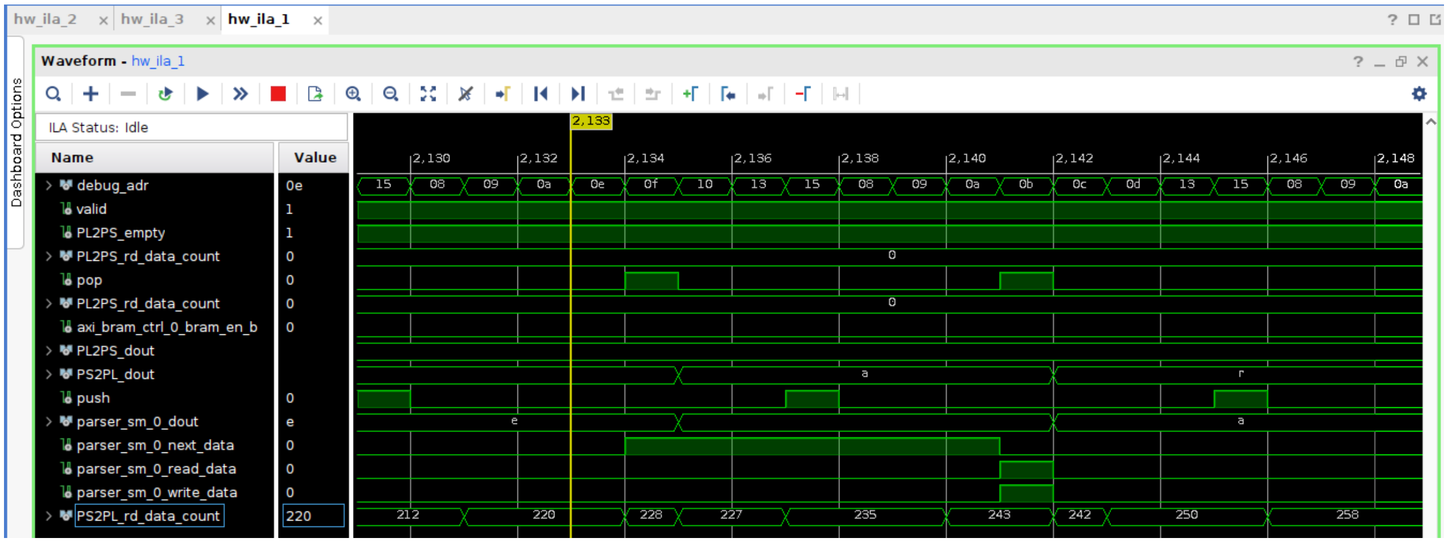
Idle

Name	Operator	Radix	Value	Port
valid	==	[B]	R	probe2[0]
parser_	==	[B]	1	probe4[0]

Click on the trigger button for any or all the ILAs. Now log into the Kria. You need to transfer the files in hotstate/examples/parser/parser_arm_code.

Run the command sudo ./parser-app

You'll see the following wave form from hw_ila_1



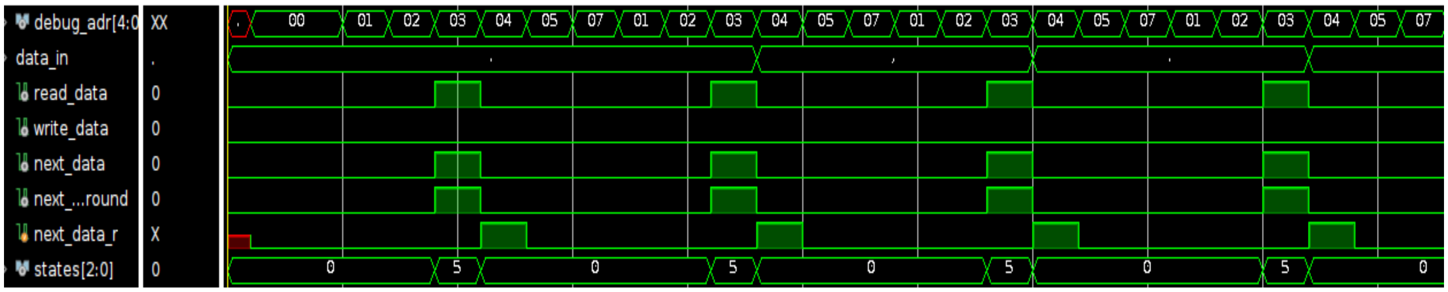
ILA Status: Idle

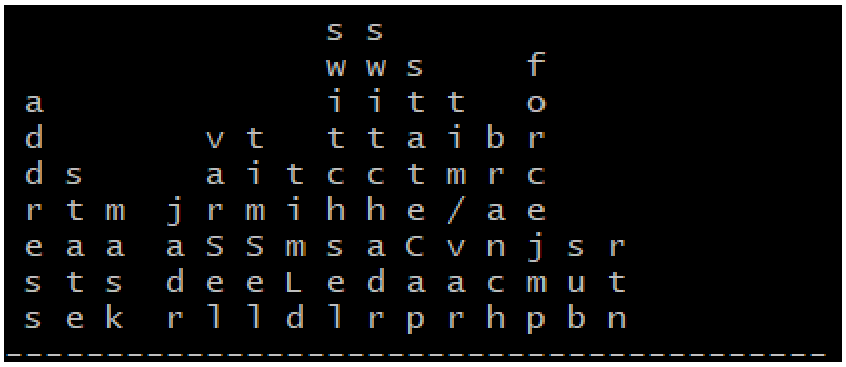
Name	Value
debug_adr	0e
valid	1
PL2PS_empty	1
PL2PS_rd_data_count	0
pop	0
PL2PS_rd_data_count	0
axi_bram_ctrl_0_bram_en_b	0
PL2PS_dout	
PS2PL_dout	
push	0
parser_sm_0_dout	e
parser_sm_0_next_data	0
parser_sm_0_read_data	0
parser_sm_0_write_data	0
PS2PL_rd_data_count	220

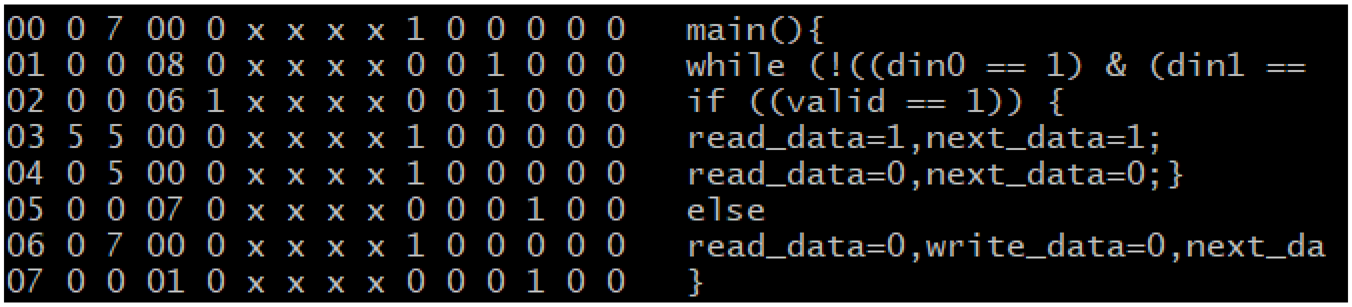
On the Arm where you ran the parser-app you'll find a file called "results.txt"

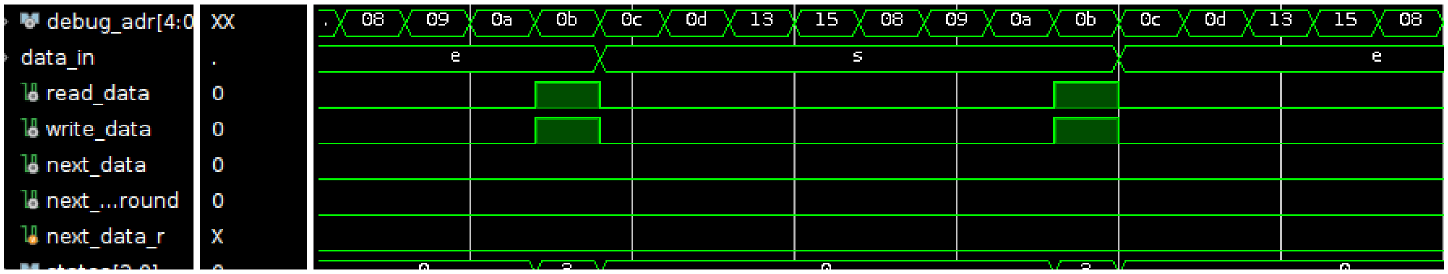
Match the debug address against the line of code in the analysis. Although the -O does not affect the output in the case of the parser it can be very important in other programs.

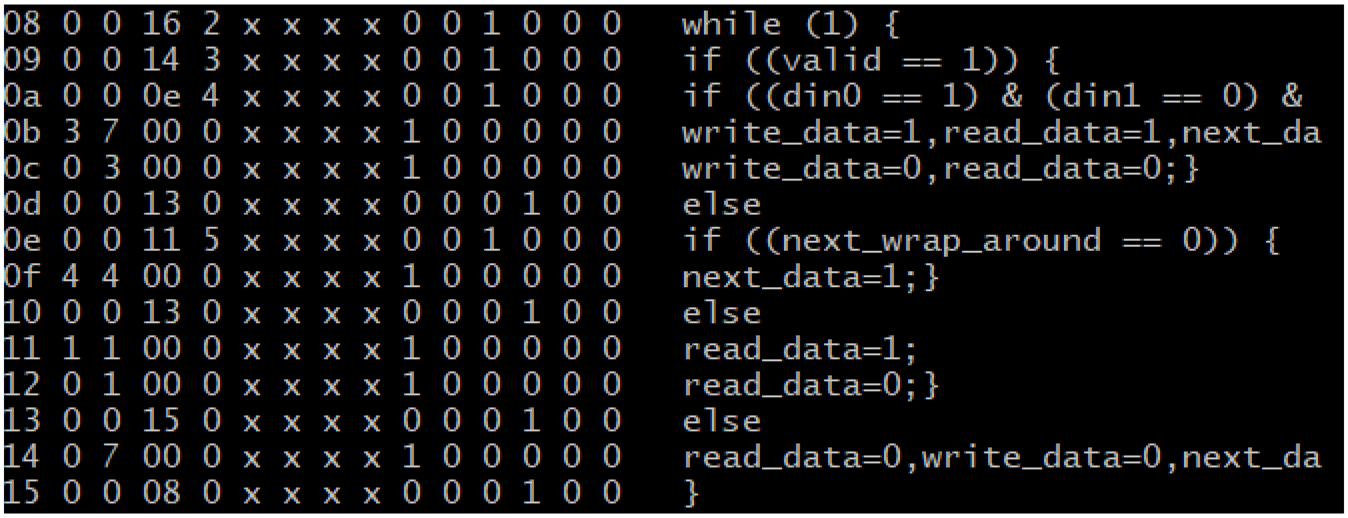
Hardware run:




```
              s s
              w w s     f
a             i i t t   o
d       v t   t t a i b r
d s     a i t c c t m r c
r t m j r m i h h e / a e
e a a a S S m s a C v n j s r
s t s d e e L e d a a c m u t
s e k r l l d l r p r h p b n
-------------------------------------------
```


```
00 0 7 00 0 x x x x 1 0 0 0 0 0   main(){
01 0 0 08 0 x x x x 0 0 1 0 0 0   while (!((din0 == 1) & (din1 ==
02 0 0 06 1 x x x x 0 0 1 0 0 0   if ((valid == 1)) {
03 5 5 00 0 x x x x 1 0 0 0 0 0   read_data=1,next_data=1;
04 0 5 00 0 x x x x 1 0 0 0 0 0   read_data=0,next_data=0;}
05 0 0 07 0 x x x x 0 0 0 1 0 0   else
06 0 7 00 0 x x x x 1 0 0 0 0 0   read_data=0,write_data=0,next_da
07 0 0 01 0 x x x x 0 0 0 1 0 0   }
```




```
08 0 0 16 2 x x x x 0 0 1 0 0 0   while (1) {
09 0 0 14 3 x x x x 0 0 1 0 0 0   if ((valid == 1)) {
0a 0 0 0e 4 x x x x 0 0 1 0 0 0   if ((din0 == 1) & (din1 == 0) &
0b 3 7 00 0 x x x x 1 0 0 0 0 0   write_data=1,read_data=1,next_da
0c 0 3 00 0 x x x x 1 0 0 0 0 0   write_data=0,read_data=0;}
0d 0 0 13 0 x x x x 0 0 0 1 0 0   else
0e 0 0 11 5 x x x x 0 0 1 0 0 0   if ((next_wrap_around == 0)) {
0f 4 4 00 0 x x x x 1 0 0 0 0 0   next_data=1;}
10 0 0 13 0 x x x x 0 0 0 1 0 0   else
11 1 1 00 0 x x x x 1 0 0 0 0 0   read_data=1;
12 0 1 00 0 x x x x 1 0 0 0 0 0   read_data=0;}
13 0 0 15 0 x x x x 0 0 0 1 0 0   else
14 0 7 00 0 x x x x 1 0 0 0 0 0   read_data=0,write_data=0,next_da
15 0 0 08 0 x x x x 0 0 0 1 0 0   }
```