FPGA exercise

8 July 2018
9 July 2018 (revised)

NOMACHI, Masaharu
Osaka University, Japan

Zhen-An LIU
Institute of High Energy Physics, China

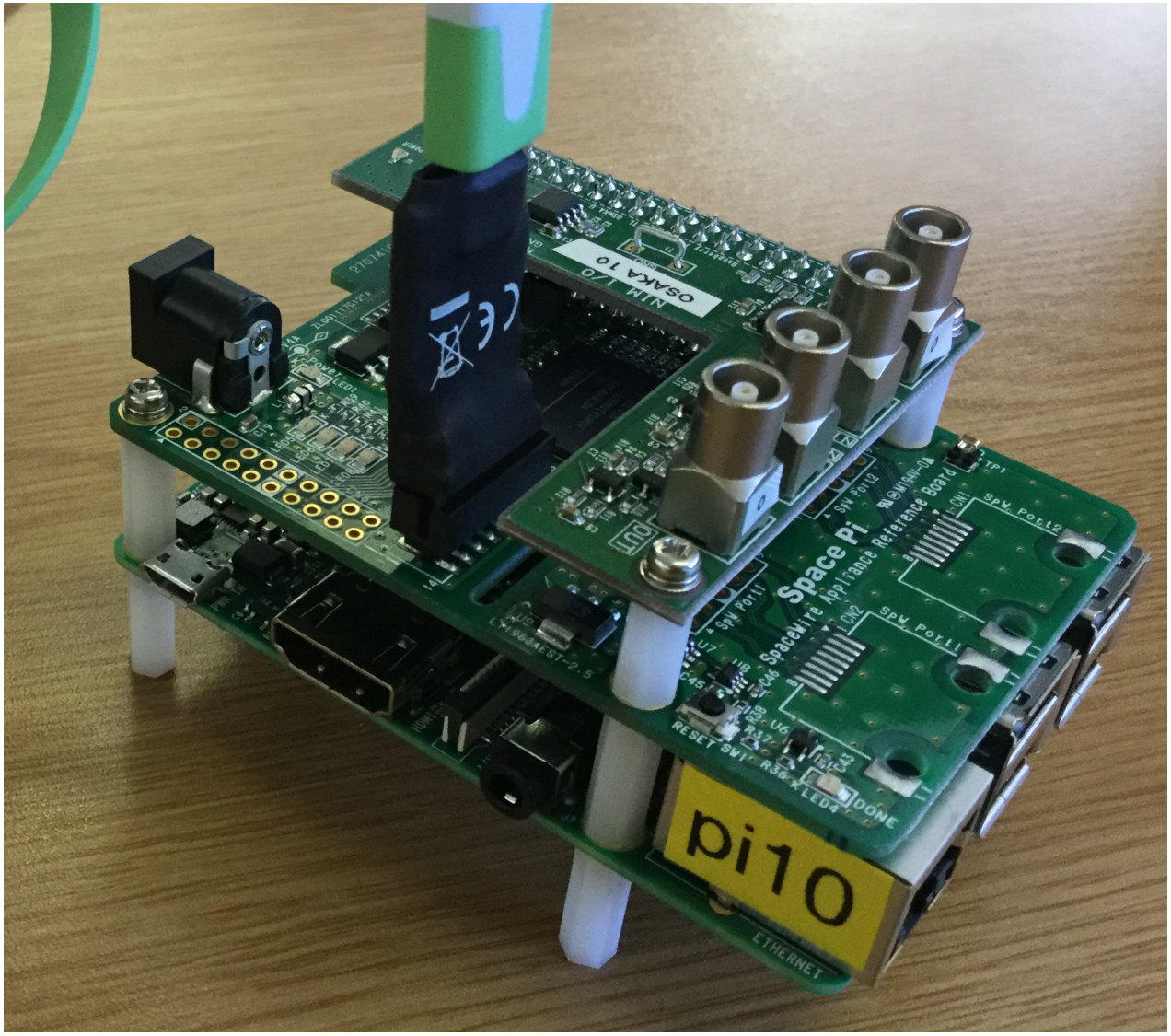
Outline

- Hardware Setup for the exercises
  - Key module
  - System setup
  - Communication with PI
  - Configuration with Web Browser
- Exercises

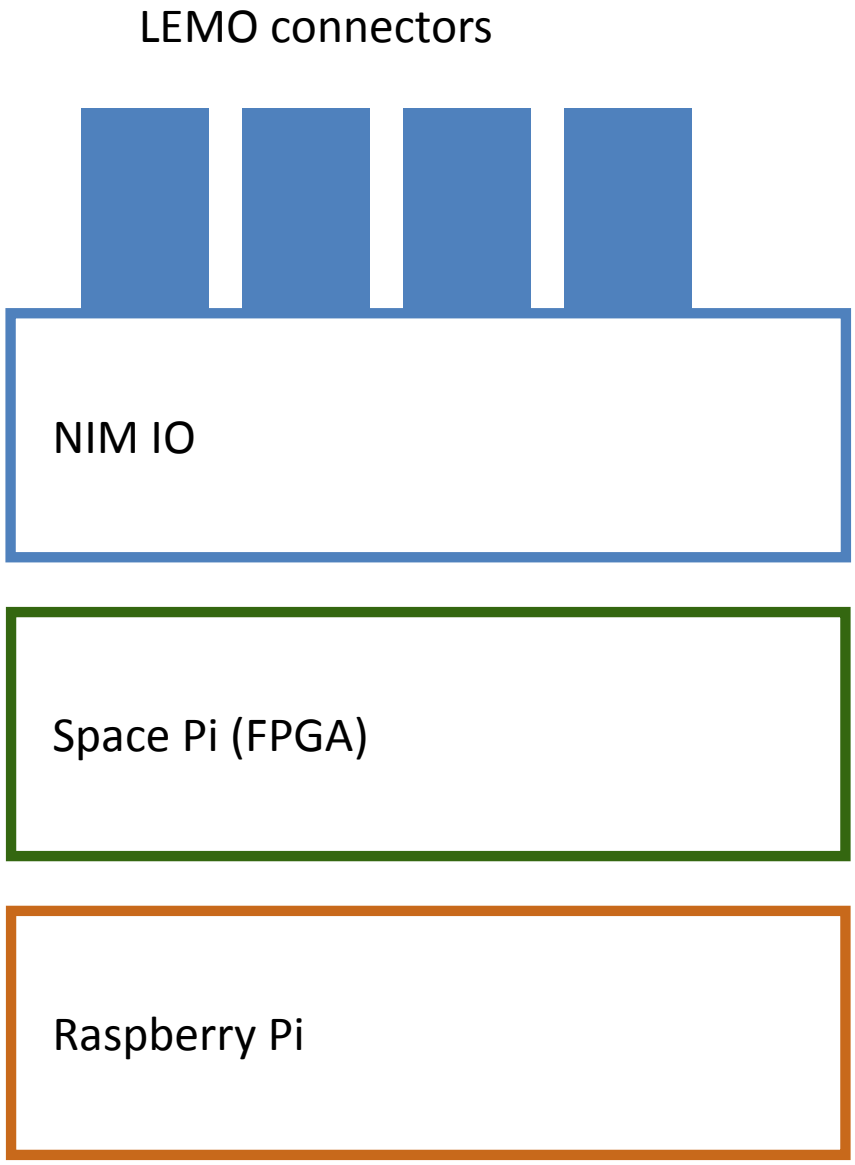
Hardware Setup for the exercises

Key Module (Raspberry Pi + Space Pi)

- This is the Key module we use for the exercises



More about the Module



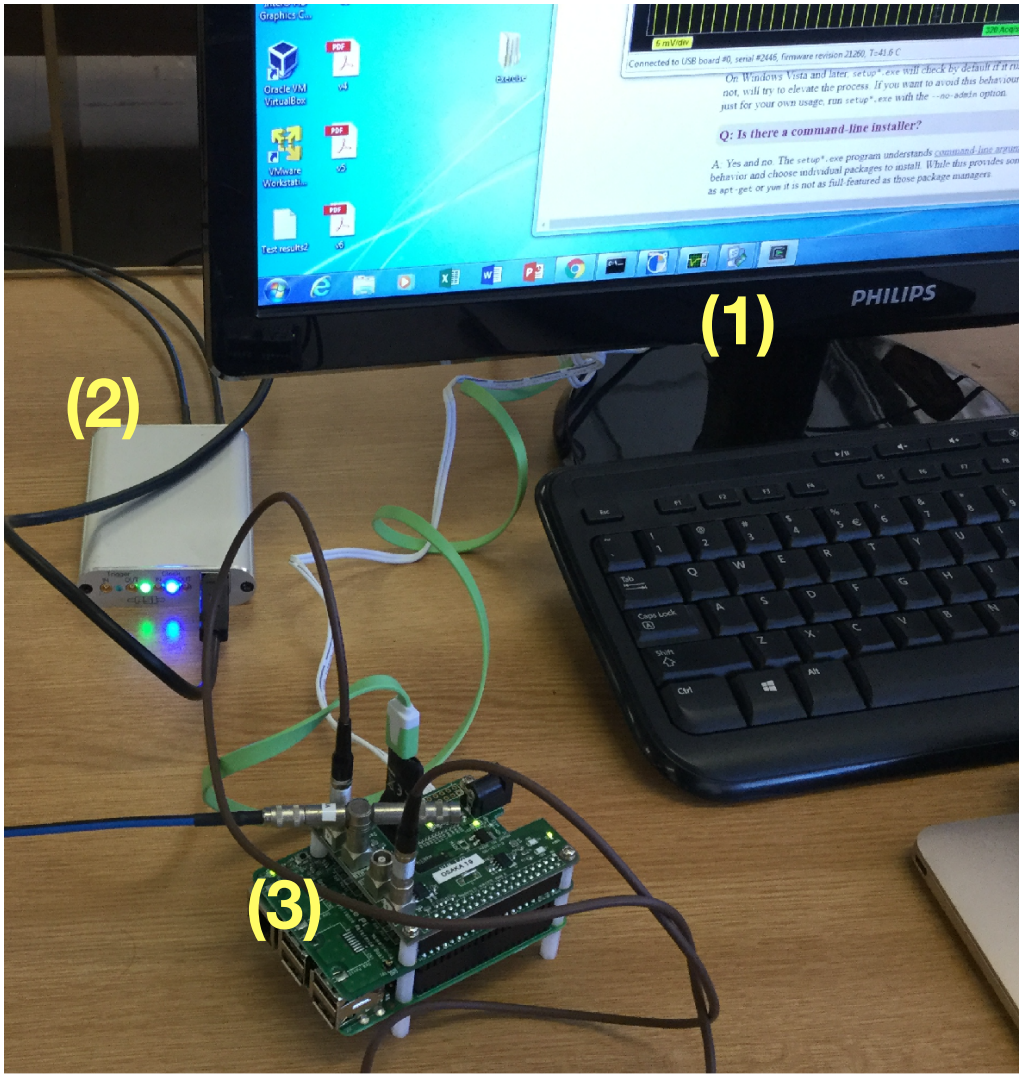
- Remarks:

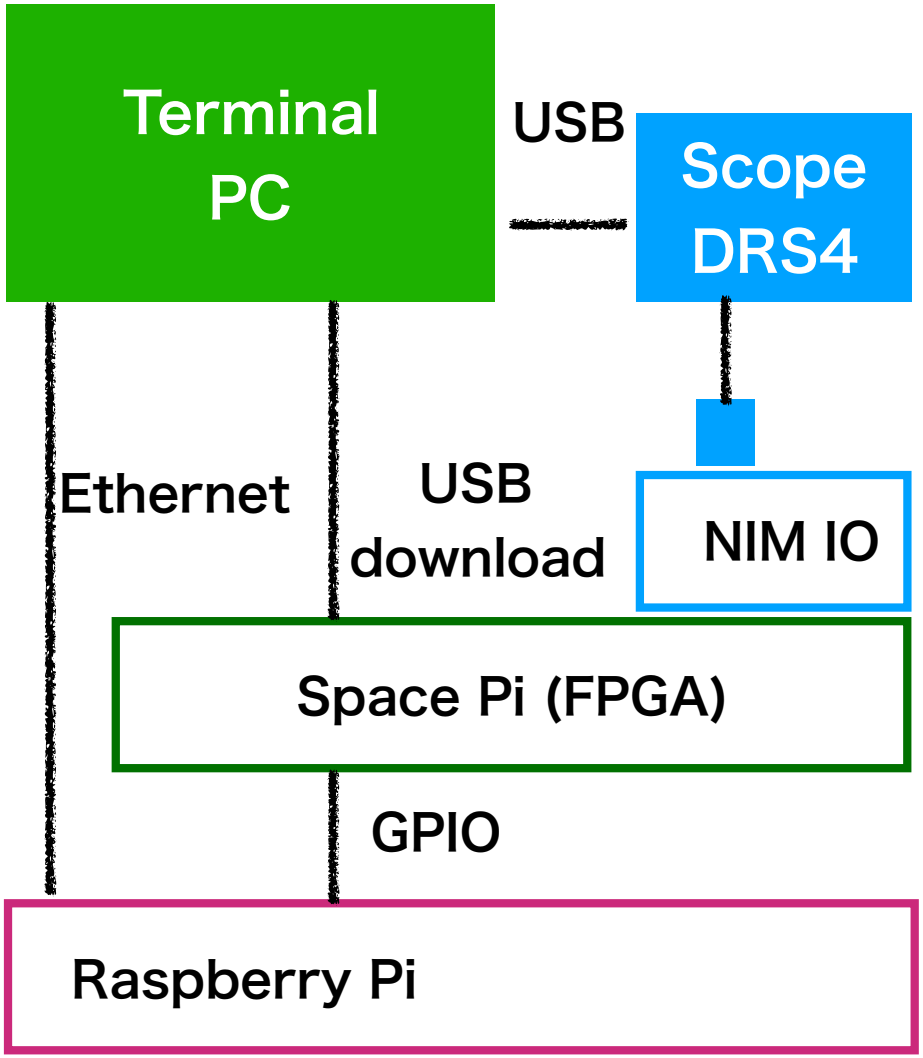
1. Conductor is not covered.
2. Be careful not to make short circuit.
3. conductor (such as cable) must be at the distance.

Touching board may cause problem(Stain reduce isolation, conductivity, etc.)

System Setup

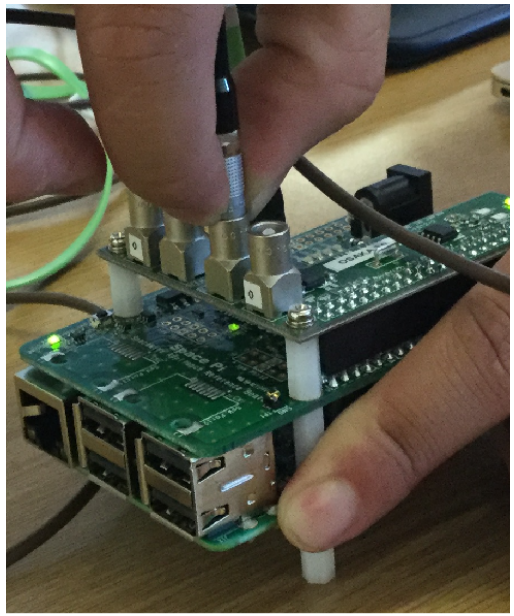
1.Terminal, 2.Scope, 3.Key Module

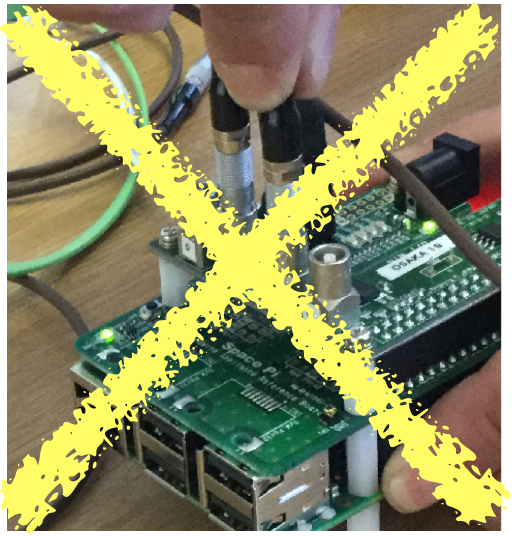




Cautions

- How to Plug and Unplug the LEMO connectors





Power Connection

- Two options to power up the system
  - 5V USB power line to Raspberry Pi
  - 5V Adaptor line to the SpacePI
- Please Remember to connect only one power line
- On the exercise, we feed power on Raspberry Pi.

Communication with PI

- From the Linux Window
  - Click on the Application, then
  - place the cursor on Favorite
  - Click on the Terminal, then
  - Type in ssh –l pi pi**xx**.tlabs.ac.za
    - In Pi**xx**, "**xx**" is ID number of your Raspberry Pi
  - Type in password CapeTown, then you are in Pi**xx**

- Work on PIxx (Start Web server)
  - cd ~/fpgatutor/Server-GPIO
  - ./server-GPIO

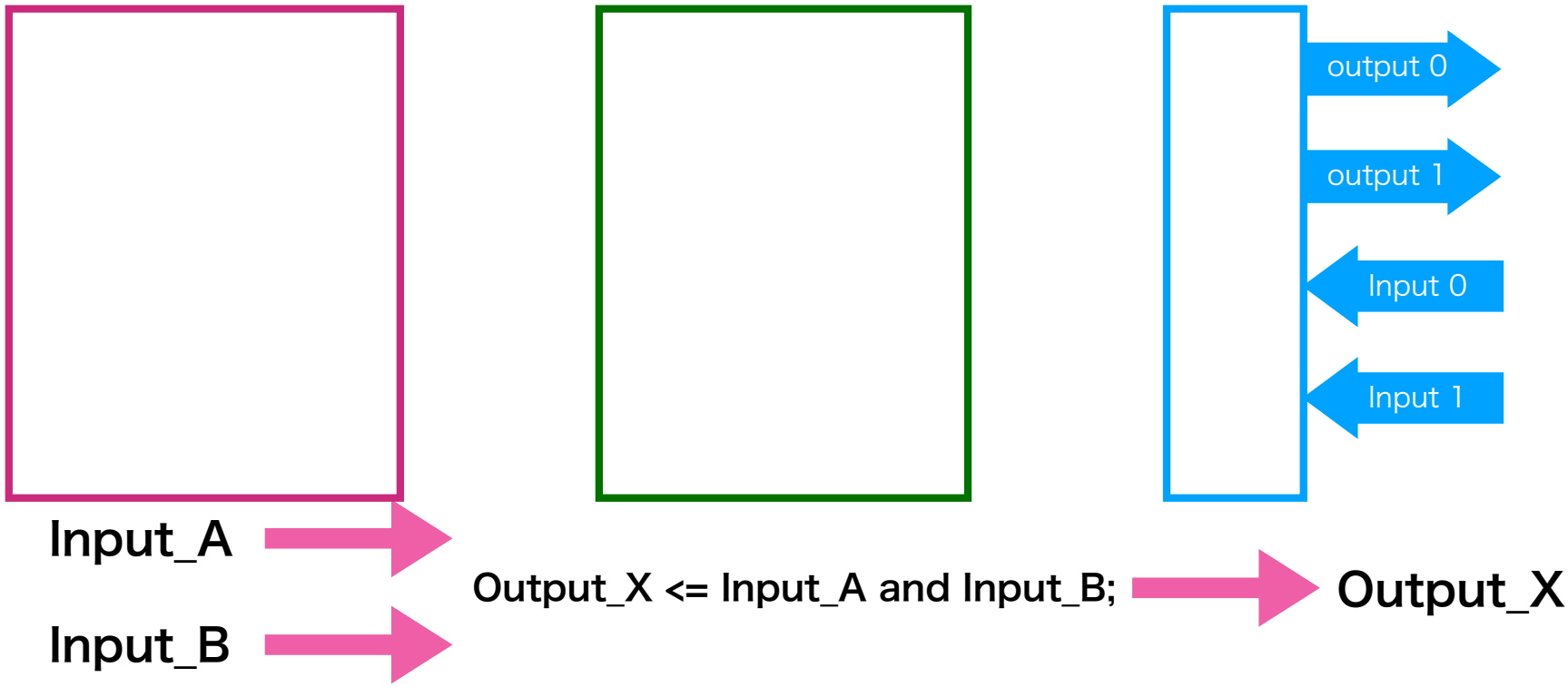
Configuration with Web Browser

- From the Linux Window
  - Click on the Application, then
  - place the cursor on Favorite
  - Click on the Firefox Web Browser, then
  - Type in http://pi**xx**.tlabs.ac.za:8080/

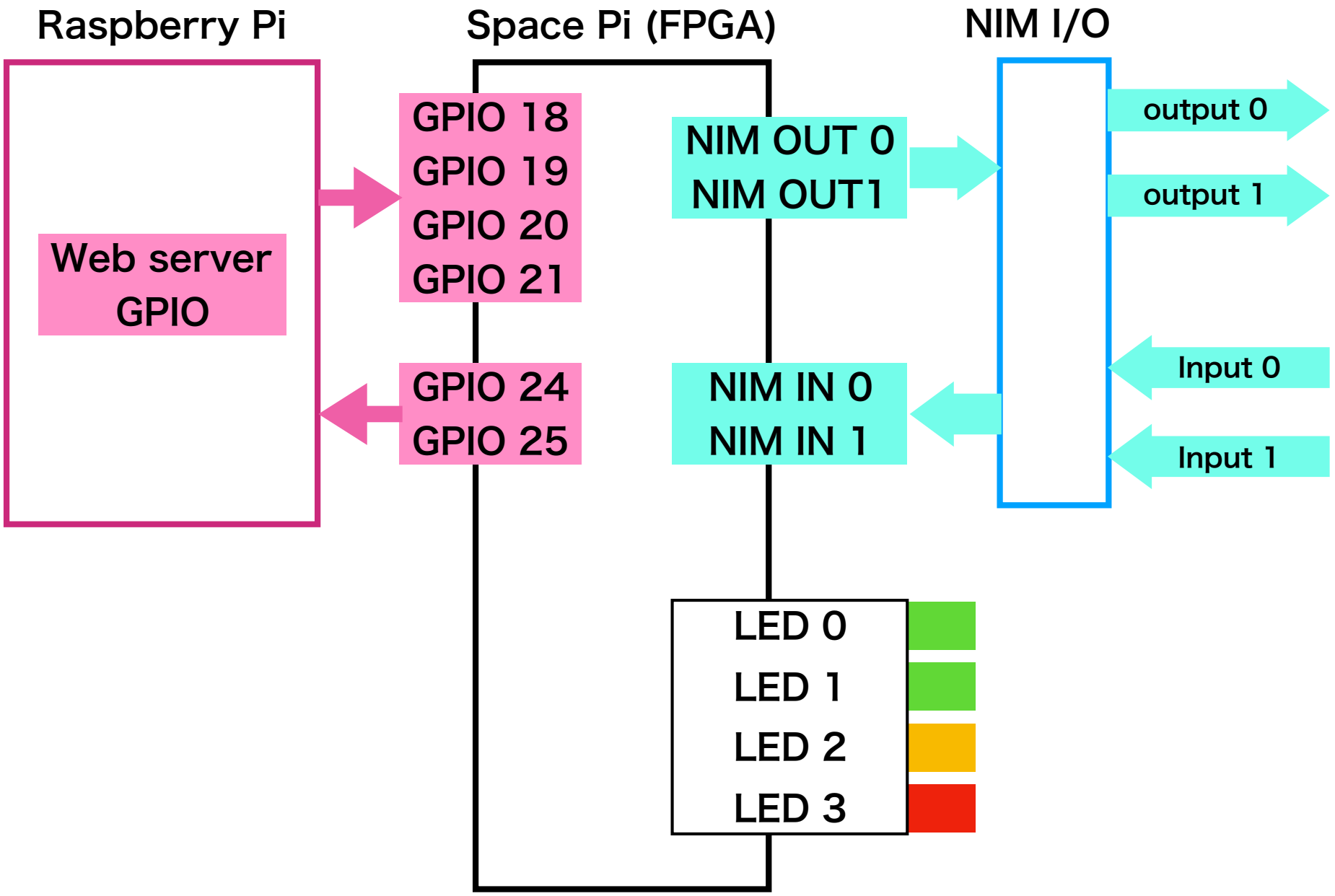
  - Change the state of GPIO output, press setup button.

Signal Input/Outputs

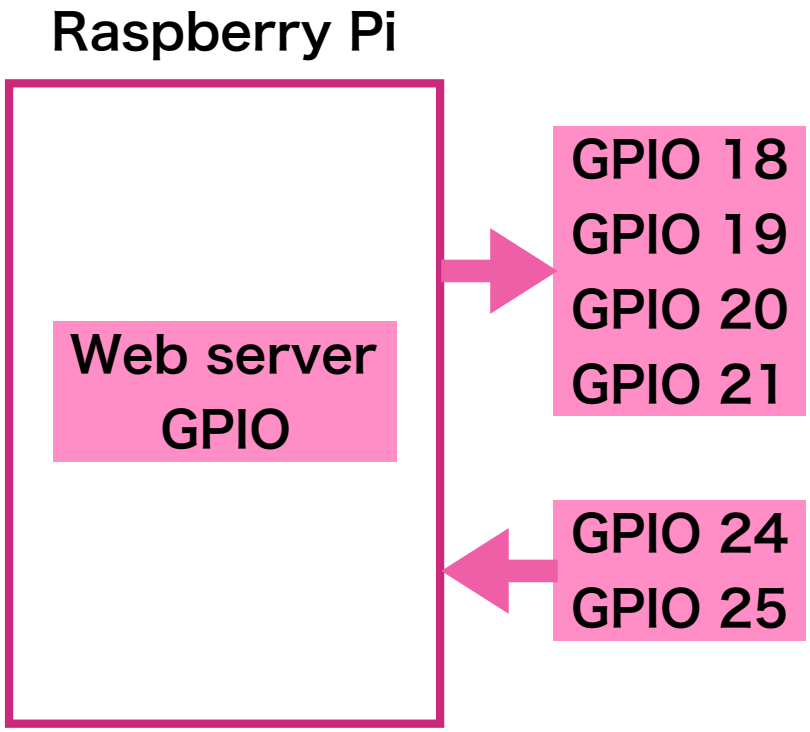
- Two Inputs
- Two Outputs



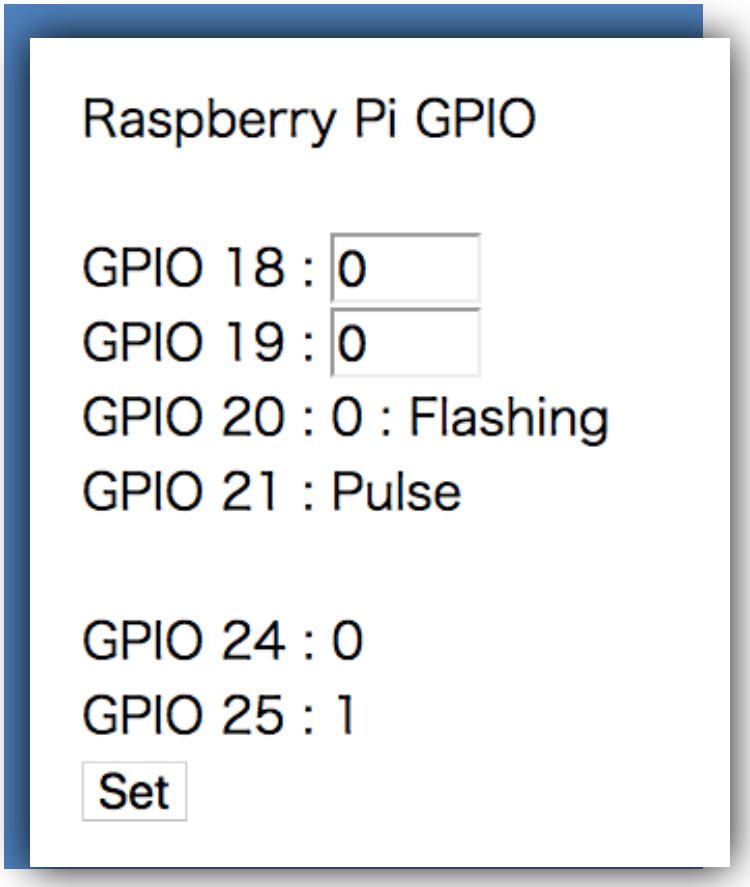
Connection Details



Web server configuration



Flashing
Pulse

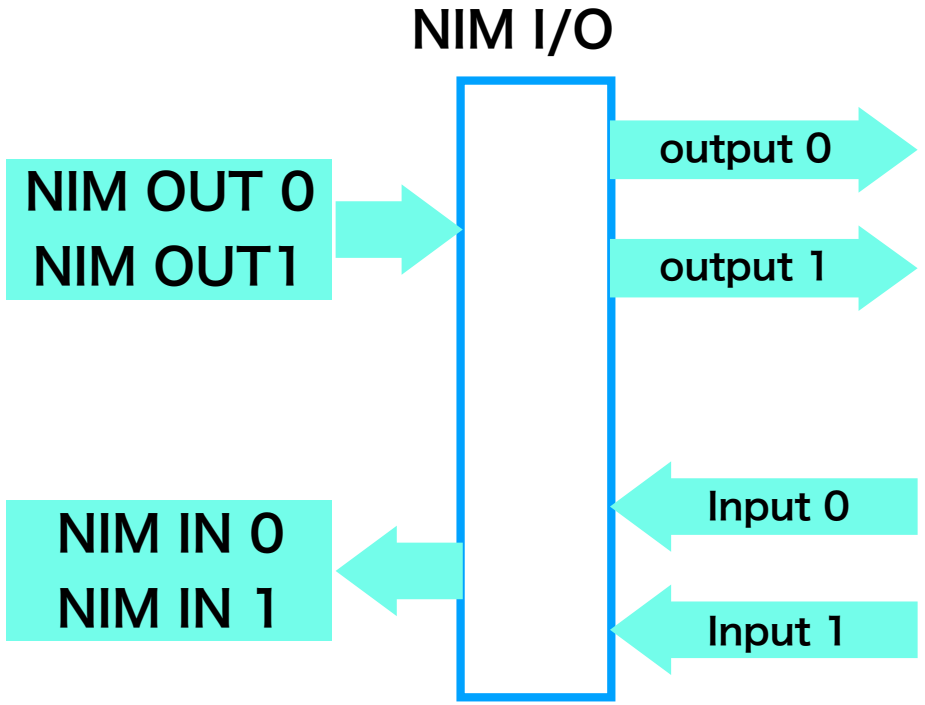


@ Raspberry Pi
type
cd fpgatutor/Server-GPIO
./server-GPIO

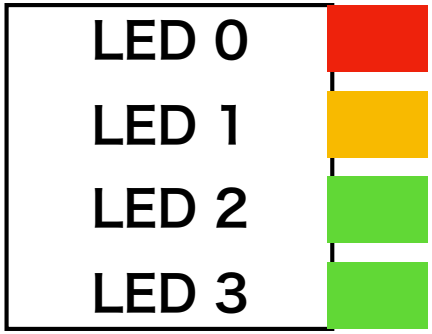
@ PC Web Browser
access
http://Pi**xx**.tlabs.ac.za:8080/GPIO.html

IO

NIM logic
‘1’ = 16mA current sink
(-16mA x 50 ohm = -800mV)
‘0’ = off
(0mA x 50 ohm = 0V)



LED
‘0’ = sink current to 0V
(LED is ON)
‘1’ = sink to 3.3V = no sink
(LED is OFF)



Starting ISE

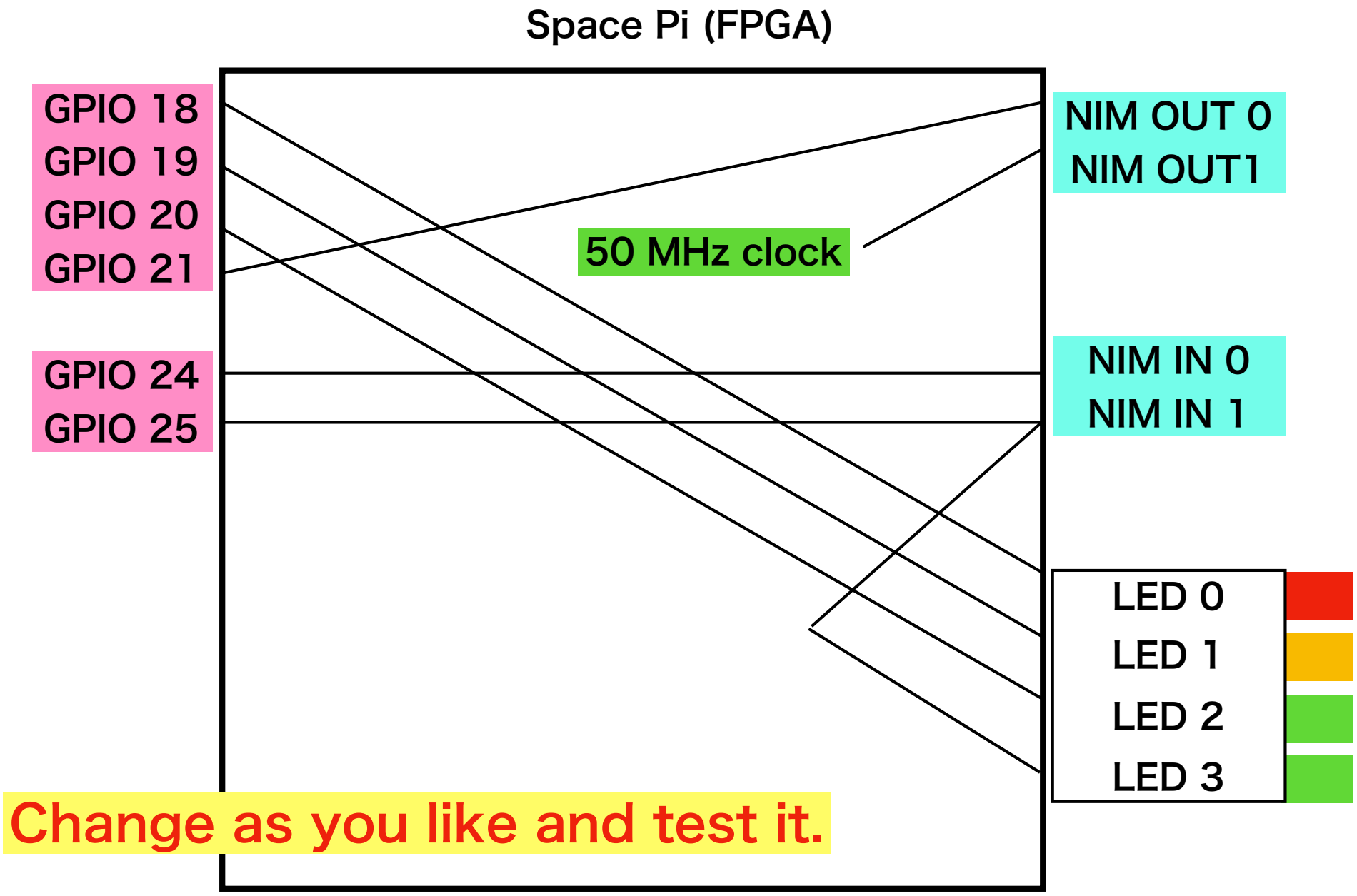
- Starting ISE by
  - Find the Vmware Workstation
    - Click on it to have a Workstation window.
    - Click on the Red Hat Enterprise Linux 7
    - Click on the Remind Me later and wait until the SCIENTIFIC LINUX comes up
    - Click on iThemba as username and input RTschool2018 as password to login
- Launch the PlanAhead 14.7
  - Click on Applications on the windows corner, and then
  - Click on the Terminal, and then type in
  - ./startXilinx.sh
- Open a project by
  - Click on the Open Project
  - Click on Desktop
  - Click on Exercise
  - Take one Exercise (Exercise_1, Exercise_2 and so on)
- Click on Exercise_1 Icon then you are in the exercise

- Tutorial of ISE can be found on the Web.
- https://www.xilinx.com/support/documentation/sw_manuals/xilinx13_2/ise_tutorial_ug695.pdf
- Project is already prepared on the exercise.
- Please refer HDL part.

Exercise 1

Learn How to program FPGA.
How to connect input and output.

Exercise example



```
entity Exercise_1 is
    Port (
    GLOBAL_RESETn     : in    std_logic;
    FPGA_LED          : out   std_logic_vector (3 downto 0);
    NIM_in            : in    std_logic_vector (1 downto 0);
    NIM_out           : out   std_logic_vector (1 downto 0);
    RaspPi_GPIO18     : in    std_logic;
    RaspPi_GPIO19     : in    std_logic;
    RaspPi_GPIO20     : in    std_logic;
    RaspPi_GPIO21     : in    std_logic;
    RaspPi_GPIO24     : out   std_logic;
    RaspPi_GPIO25     : out   std_logic;
    FPGA_CLK_50MHz    : in    std_logic
);
end Exercise_1;
```

Practice

Ex1) Which color of LED is flashing? Change color.

Ex2) Watch pulsed output on the scope.
Measure the pulse width.

Ex3) On/Off LED flashing according to GPIO 18

Ex4) Get “Flashing” from your neighboring Raspberry Pi and make “AND” with your “flashing”. Connect them to LEDs.

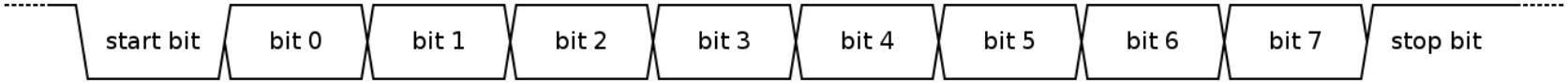
Exercise 2

Scheduled operation
How to send data to Raspberry Pi
Use Serial data link

ASCII code

How to send data from FPGA to Raspberry Pi?

Universal asynchronous receiver-transmitter

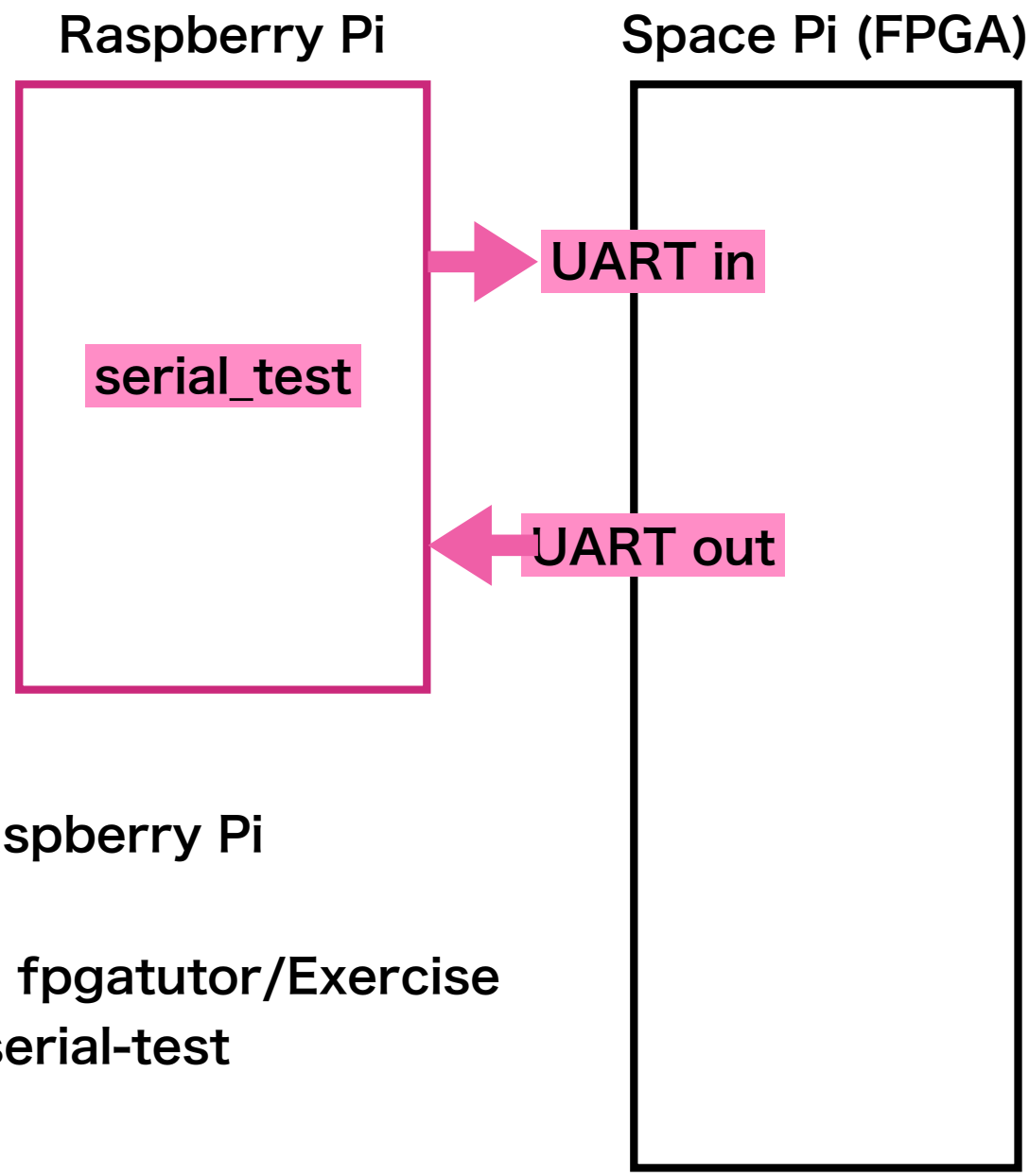


Baud rate

The number of digit in one second
10 digit (8 + start + stop)
for one character

Default in Raspberry pi
= 115200

upper 4 bit \ lower 4 bit	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL



@ Raspberry Pi

type

```
cd fpgatutor/Exercise
./serial-test
```

Sequence control

Control the actions by schedule.
wake up 6 am
eat breakfast 7am
and so on

```
signal t_count      : std_logic_vector(19 downto 0); -- clock count
signal b_count      : std_logic_vector(15 downto 0); -- bit count

constant baud     : integer := 115200;
constant f_clock  : integer := 50000000;
constant t_tick     : integer := f_clock/baud;
```

Tick for one digit = 50M/115.2 k = 434.03
Scheduled by tick count
0 = start bit, 1 = bit 0, and so on

```
PROCESS (clock)
BEGIN
    IF clock'EVENT AND clock = '1' THEN
        IF t_count = t_tick THEN
            t_count <= X"00000";
            tick <= '1';
        ELSE
            t_count <= t_count + 1;
            tick <= '0';
        END IF;
    end if;
end process;
```

In the “process”

connection is determined sequentially.

Connection works simultaneously

Not a sequential operation

```
PROCESS (clock)
BEGIN
        IF clock'EVENT AND clock = '1' THEN
                IF tick = '1' THEN
          b_count <= b_count+1;
                        CASE b_count IS
                                WHEN X"0000" => s_out <= '0';
                                WHEN X"0001" => s_out <= ASCII_char(0);
                                WHEN X"0002" => s_out <= ASCII_char(1);
                                WHEN X"0003" => s_out <= ASCII_char(2);
                                WHEN X"0004" => s_out <= ASCII_char(3);
                                WHEN X"0005" => s_out <= ASCII_char(4);
                                WHEN X"0006" => s_out <= ASCII_char(5);
                                WHEN X"0007" => s_out <= ASCII_char(6);
                                WHEN X"0008" => s_out <= ASCII_char(7);
                                WHEN OTHERS => s_out <= '1';
                        END CASE;

                END IF;
        END IF;
end process;
```

Practice

Ex1) Change character to send.

Ex2) Change frequency to send.

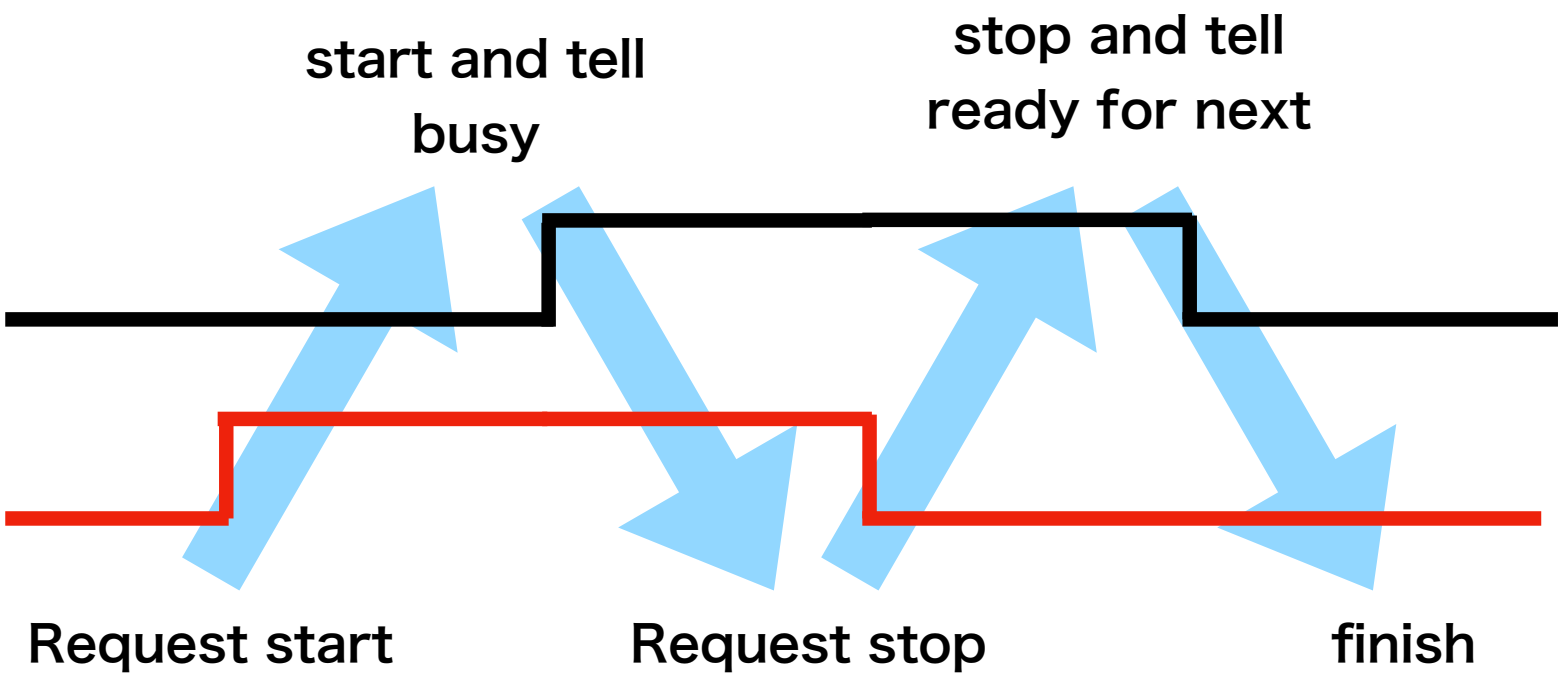
Ex3) Set Baud rate to 20MHz, see “s_out” on the scope. You don’t need to run serial_test. It does not work on such high speed.

Ex4) Change Character dynamically. Say, “0-1-2-3-4…”

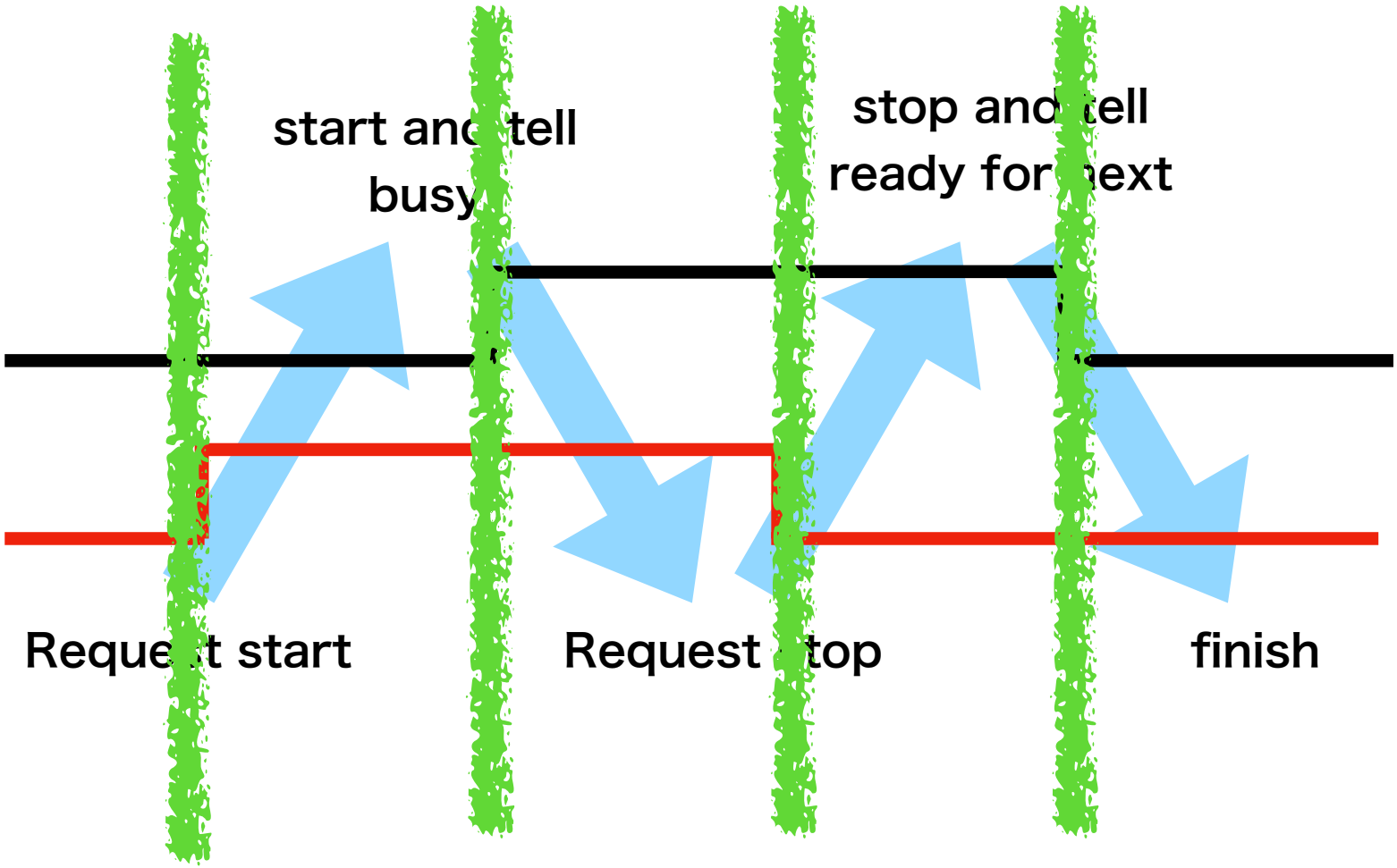
Exercise 3

State machine
Hand shake (ping pong)
How to control the operation by Raspberry Pi

hand shake


start and tell
busy
stop and tell
ready for next
Request start
Request stop
finish

hand shake

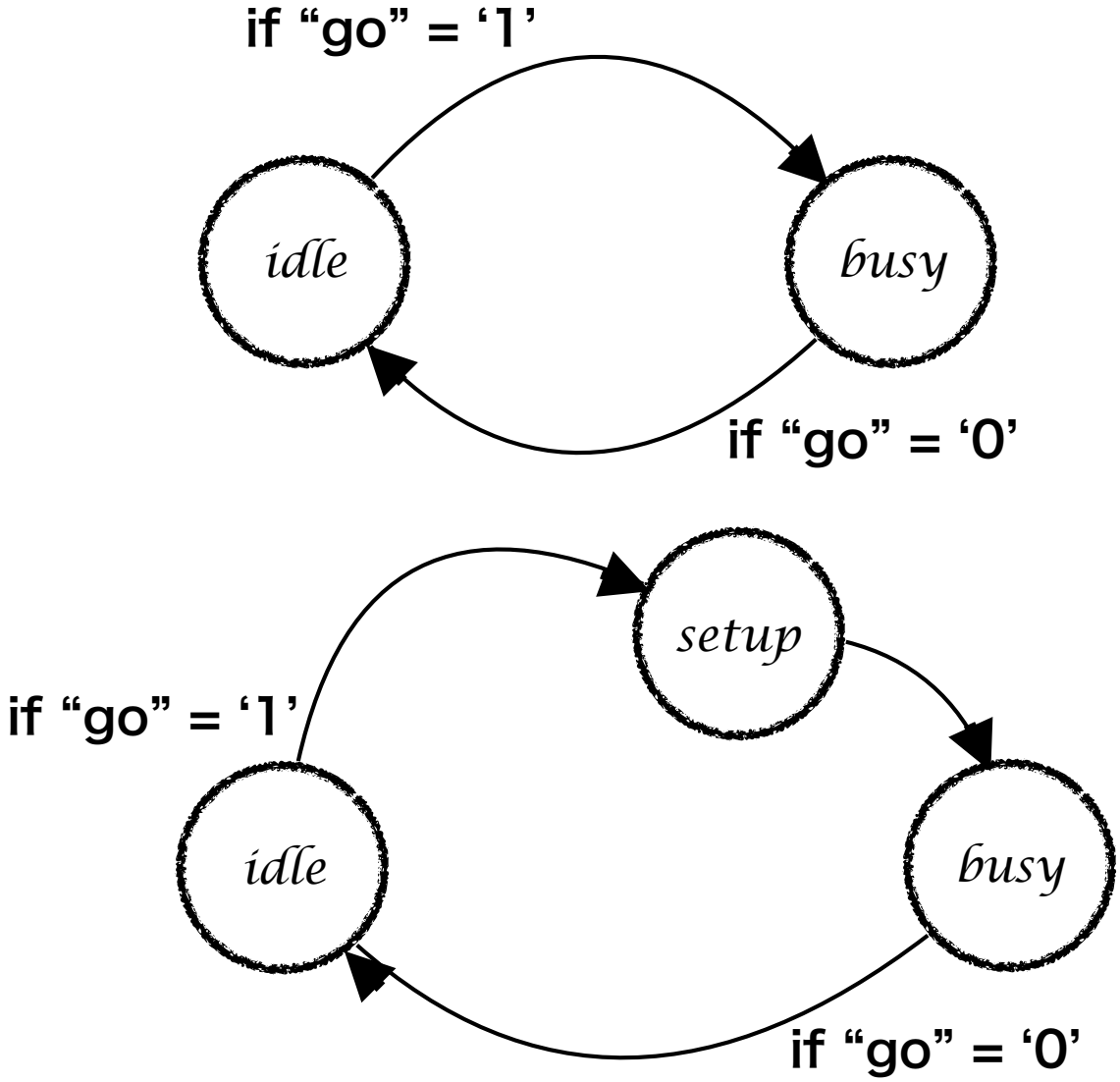


4 states exists.

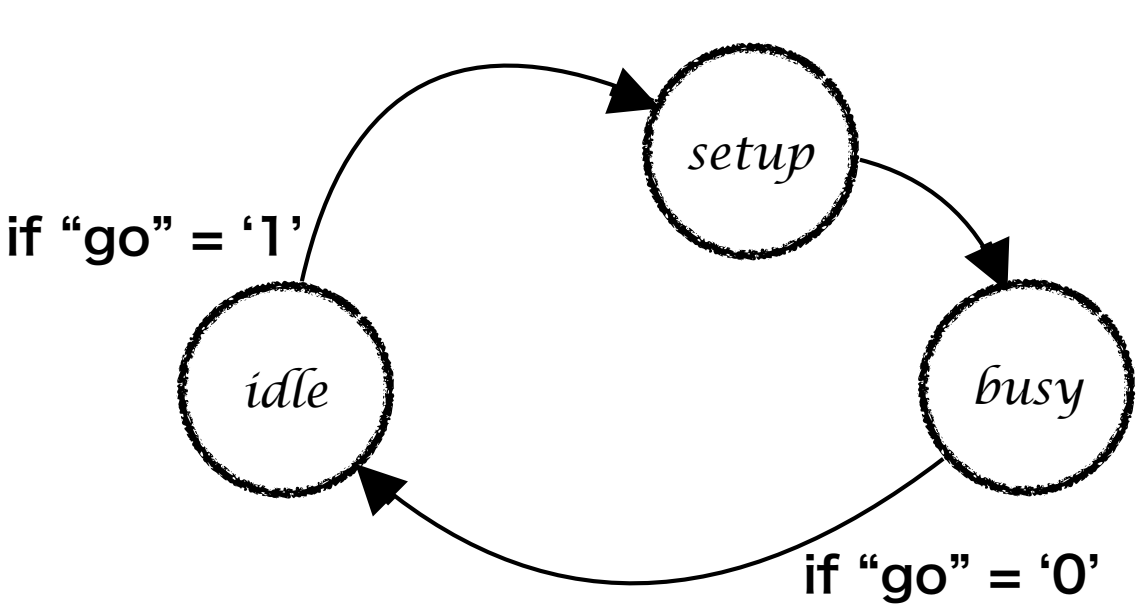
2 signals represent 4 states uniquely.

2 states on each side.

states on the FPGA



3 states need 2 bit state variable



One odd state may exist.
It is not reachable (true?)

Stupid state assignment
Bad example

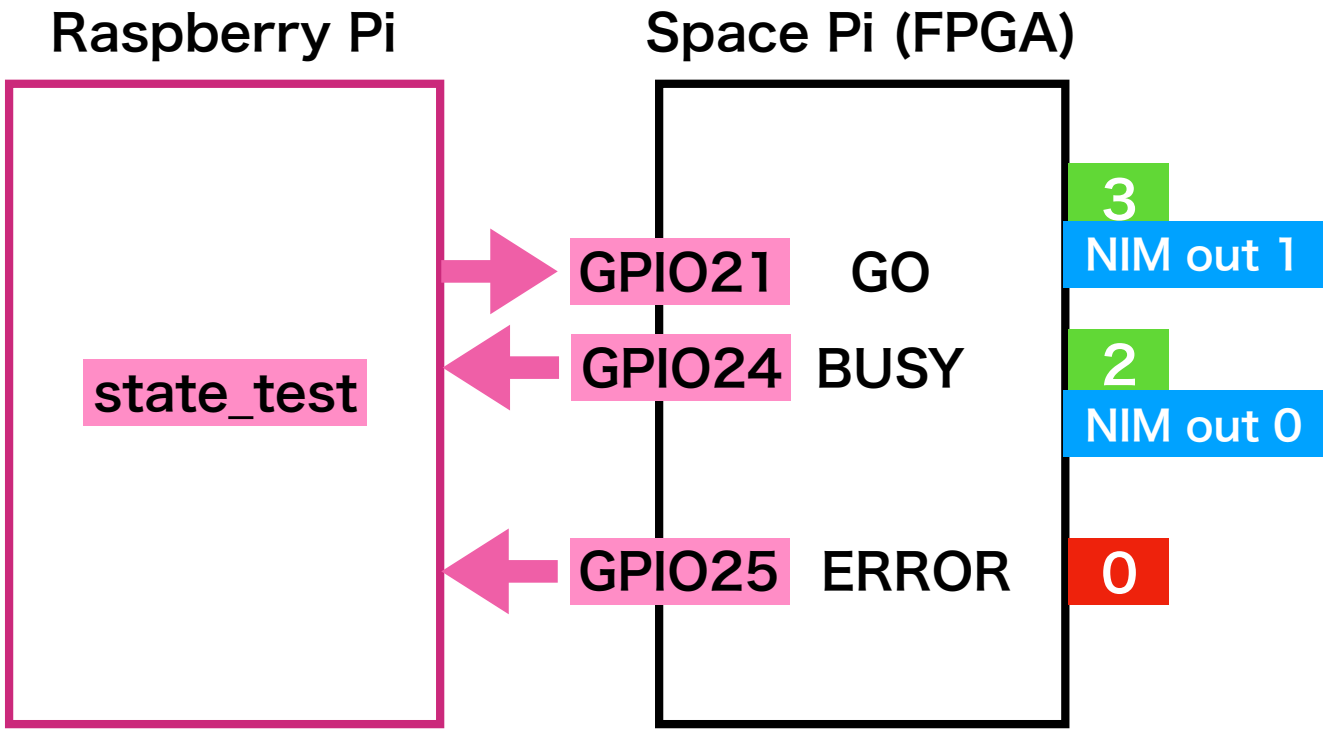
```
CONSTANT idle_state  : STD_LOGIC_VECTOR(2 DOWNTO 0) := B"000";
CONSTANT setup_state : STD_LOGIC_VECTOR(2 DOWNTO 0) := B"110";
CONSTANT busy_state  : STD_LOGIC_VECTOR(2 DOWNTO 0) := B"101";
CONSTANT error_state : STD_LOGIC_VECTOR(2 DOWNTO 0) := B"011";
```

Why Stupid ? To make error for the exercise, there are several stupid codes! (State assignment is one of them!)

The example causes Error.

Please exam the reason of the error and the frequency of the error.

There are several possibility to fix it. Which one is the best way?



@ Raspberry Pi

type

cd fpgatutor/Exercise

./handshake-test “n”

Test hand shake “n” times.

Practice

Ex1) Measure the frequency of error. How does is distribute? How about the average?

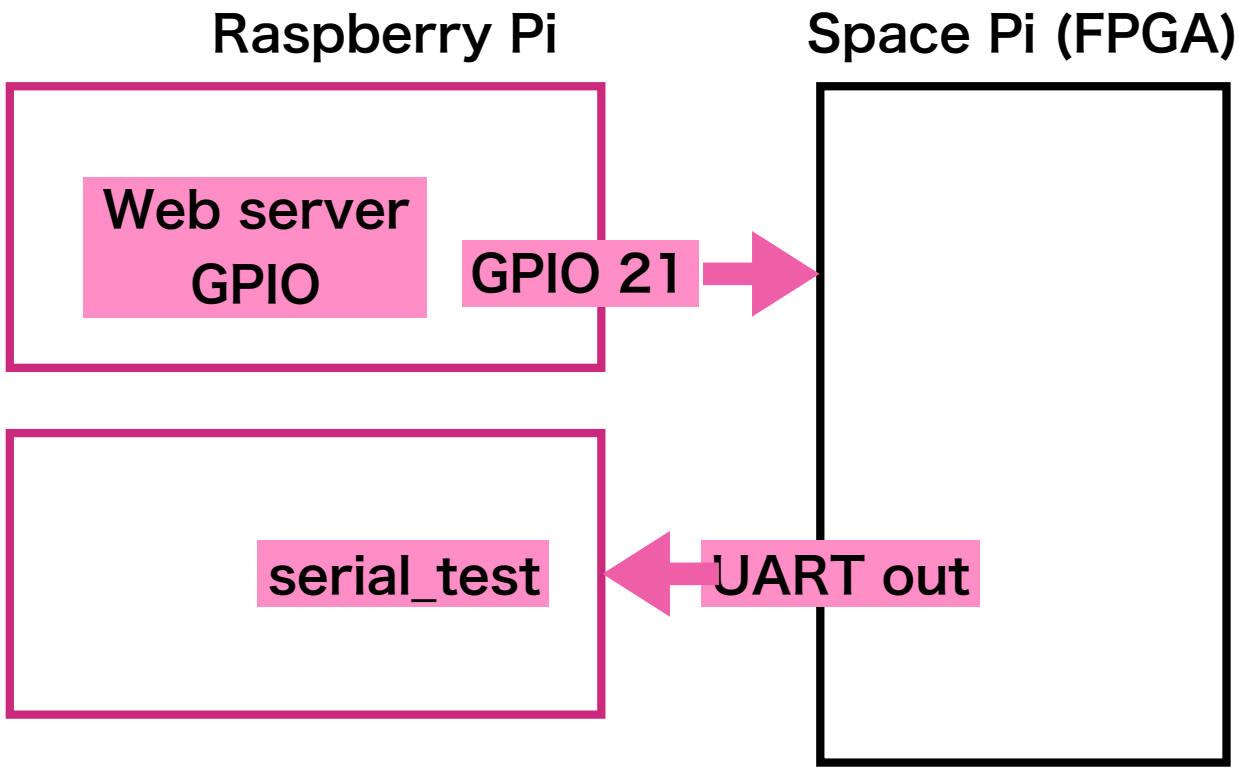
Ex2) Changing the state ID, error may disappear. However, it just hide the problem. It may depend on the environments. Someday, (in the most important mission) it may appear again. “Hidden problem is most dangerous bug”. You must solve the problem. Not hide the problem.

Answer is “Synchronize external signal”.

Please explain how the error occurred, how it is hidden, and how it is solved.

Exercise 4

FPGA send Data on request



Open two terminal windows

@ Raspberry Pi
type
cd fpgatutor/Exercise
./serial-test

@ Raspberry Pi
type
cd fpgatutor/Server_GPIO
./server-GPIO

@ PC Web Browser
access
http://pi**xx**.tlabs.ac.za:8080/GPIO.html

Practice

Ex1) Get pulse from your neighbor. Use it to trigger the transmission.

Ex2) On receiving pulse from your Pi, send “My”. On receiving pulse from your neighboring Pi, send “His/ Her”.

EX3) Error may not happen, but bug is hidden. Please fix it.

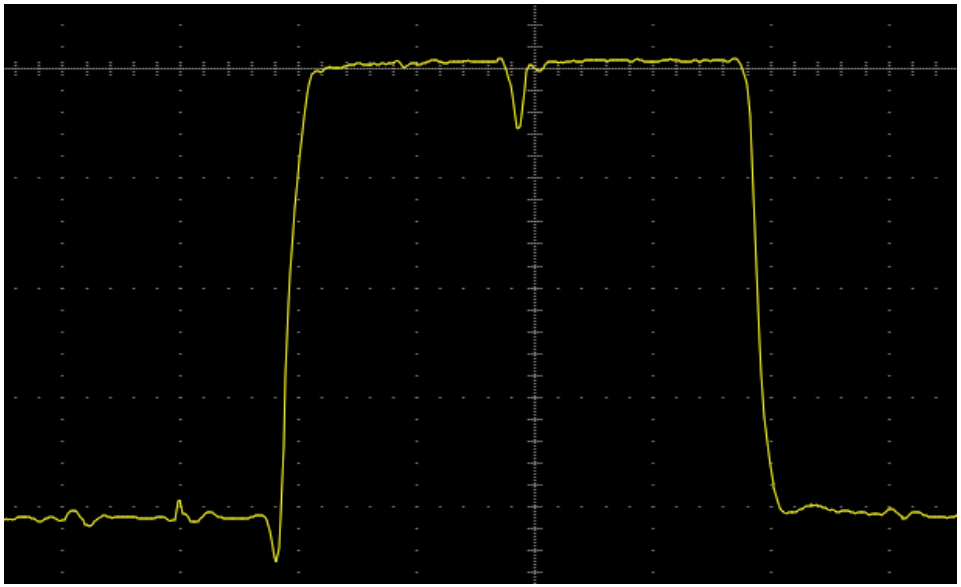
Exercise 5

See the hazard on the scope.

Hazard / Glitch

A hazard/glitch is a momentary unwanted transient output.

They occur due to unequal propagation delays along different signal paths.



Hazard is very short pule.
NIM output driver cannot be so fast to show correctly.
However, you will see the dip.

Practice

Ex1) Observe NIM out 1 with scope. Logically, there should not have dip. Explain, why dip (hazard) happen.

Ex2)It can be solved by synchronizing the output. Observe NIM out 1 with scope. Explain the reason.

Exercise 6

UART output (Character)

Use the same setup as Exercise 2
A,B,C…..Z will be sent.

Exercise 7

UART output (Hexadecimal)

Hexadecimal number is sent.
12345 <CR><LF> is sent.
Use the same setup as Exercise 2

Exercise 8

Measure the pulse width.

Resolution is 20ns.
Use tdc-test.c
It generate a pulse on GPIO 21 and get measured pulse width in Hexadecimal number.

- Time is measured by counting clock signal.
- On “start”, start count.
- On “stop”, stop count.
- The result is sent to Raspberry Pi.

- Measure the interval of
  - Two NIM inputs
  - Pulse width of one NIM signal
  - Execution time of Raspberry Pi
  - As you like ……..

Pin	GPIO	FPGA	Usage (Pi)	Usage (I/O)	
27	0	E1			
28	1	G3			
3	2	R1	I2C		
5	3	P1	I2C		
7	4	N1			
29	5	D1		NIM out 1	
31	6	C1		NIM out 2	
26	7	H2	SPI		
24	8	J3	SPI		
21	9	H1	SPI		
19	10	J1	SPI		
23	11	F1	SPI		
32	12	F2		NIM in 1	
33	13	B1		NIM in 2	
8	14	R2	UART Tx		in to FPGA
10	15	P2	UART Rx		out from FPGA
36	16	E2			
11	17	M1			
12	18	N3			To FPGA
35	19	B2			To FPGA
38	20	C2			Flash
40	21	C3			Pulse
15	22	K1			
16	23	M2			
18	24	L3			From FPGA
22	25	K2			from FPGA
37	26	A2			
13	27	L1			