Reading Keyboard Scan Codes Through the PS/2 Interface on the NEXYS3 board

TSTE12

Computer Technology, ISY

Linköpings Universitet

August 28, 2020

Table of Contents

1. Reading Keyboard Scan Codes Through the PS/2 Interface on the NEXYS3 board

1.1 Background

The PC/AT keyboard transmits data in a clocked serial format consisting of a start bit, 8 data bits (LSB first), an odd parity bit and a stop bit. The clock signal is only active during data transmit. The generated clock frequency is usually in the range 10 - 30 kHz. Each bit should be read on the falling edge of the clock.

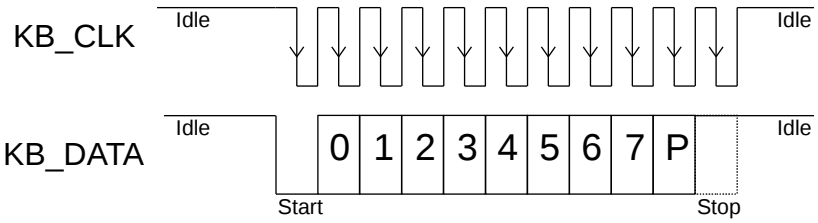


Figure 1: Keyboard transmission waveform

The above waveform represents a one byte transmission from the keyboard. The keyboard generally changes its data line on the rising edge of the clock as shown in the diagram. The data line only has to be valid on the falling edge of the keyboard clock. The Least Significant Bit is always sent first.

When a key is pressed, the keyboard transmits a 'make' code consisting of an 8-bit 'scan' code denoting the key pressed and, when the key is released, a 'break' code. The 'break' code (key released) consists of the same 8-bit scan code preceded by a special code - ’F0’H.

The keyboard handles combinations with control keys as SHIFT, CTRL, ALT, etc as two separate key presses, i.e., SHIFT-MAKE, 'A'-MAKE, SHIFT-BREAK, 'A'-BREAK. The 'A' scan code (’1C’H) is the same for both the shifted and unshifted state. To determine whether the 'A' scan code is interpreted as 'A' or 'a', the PC must keep track of the presence or absence of a prior SHIFT-MAKE.

Read more about the AT keyboard in appendix 1, “PC AT keyboard ref”.

1.2 Assignment

Create a circuit that accepts scan codes from a keyboard attached to the PS/2 interface of the NEXYS3 Board. The binary pattern of the scan code is displayed on the LEDs above the switches on the board. In addition, if a scan code for one of the '0'-'9' keys arrives, then the numeral will be displayed on the right 7-segment LED display of the NEXYS3 Board. The system should be clocked using the 100 MHz oscillator on the NEXYS3 board.

The lab group number should be displayed on the two leftmost 7-segment displays of the NEXYS3 Board.

Create also a testbench that checks if the keyboard decoder works, testing at least two different key presses. The testbench must report back to the simulation user in plain text if the test worked or not after the simulation stops.

1.3 Problem definition

Here are listed the definitions that is needed to complete the design.

1.3.1 Port definitions

The inputs and outputs of the circuit are defined as follows:

Table 1: Port interface description

Name	Type	Range	Description
kb_data	IN	std_logic	The scan code bits enters through this input.
kb_clk	IN	std_logic	The keyboard clock signal enters through this input.
sys_clk	IN	std_logic	The system clock (100 MHz) used to clock the system.
sys_rst	IN	std_logic	Reset system when sys_rst = ‘1’, connected to center button BTNS on the board.
db	OUT	std_logic_vector (7 downto 0)	Drive the LEDs above the switches on the NEXYS3 board. Output all data bits except the startbit, parity and stopbit received from the PS/2 keyboard
seg	OUT	std_logic_vector (7 downto 0)	Drive the segments of the 7-segment digits on the NEXYS3 Board.
ans	OUT	std_logic_vector (3 downto 0)	Select which 7-segment digit to drive on the NEXYS3 Board.

1.3.2 Signal definitions

Within the main body of the architecture section, these signals should be added. Note that more signals than the ones listed here is probably needed.

Table 2: Description of signals declared within the architecture

Name	Range	Description
rsb	std_logic_vector (6 downto 0)	Received digit 7-segment pattern
left_digit	std_logic_vector (7 downto 0)	7-segment pattern to show on left digit
middleleft_digit	std_logic_vector (7 downto 0)	7-segment pattern to show on middle left digit
middleright_digit	std_logic_vector (7 downto 0)	7-segment pattern to show on middle right digit
right_digit	std_logic_vector (7 downto 0)	7-segment pattern to show on right digit

1.3.3 Process definitions

Within the main body of the architecture section, these processes should be implemented:

sync_keyboard

Synchronize the external inputs (kb_data and kb_clk) to the system clock. Creates an internal copy of the keyboard signals that only changes values on the positive edge of the system clock.

detect_falling_kb_clk

Creates an output (edge_found) which is one during one system clock cycle when a falling edge of the synchronized kb_clk signal has been found. This can be done by comparing the current value of the synchronized kb_clk signal with the old value (stored in the previous clock period) of the synchronized kb_clk signal. Remember that the output signal from this process should not be used as a clock, as this would create a design with multiple clock domains.

convert_scancode

When edge_found is one, this process shifts the data bit on the kb_data input into the most significant bit of a 10-bit shift register. After 11 keyboard clock cycles, the lower 8 bits of the register will contain the scan code, the upper 2 bits will store the stop and parity bits, and the start bit will have been shifted through the entire register and discarded.

The current value in the shift register is applied to the LEDs above the switches. Since the LEDs are active-high, a segment will light for every '1' bit in the shift register. The LED segment will show some flickering as the shift register contents change.

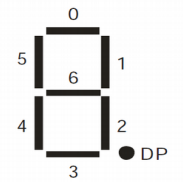


Figure 2: 7-segment display index to segment mapping

Table 3: Scan codes for the alpha numerics 0 to 9

KEY	SCANCODE	rsb
1	00010110	1111001
2	00011110	0100100
3	00100110	0110000
4	00100101	0011001
5	00101110	0010010
6	00110110	0000010
7	00111101	1111000
8	00111110	0000000
9	01000110	0010000
0	01000101	1000000
else		0000110

If the scan code in the shift register matches the codes for the digits 0-9, then the rightmost 7-segment LED display will be activated to display the corresponding digit. The segments are active-

low, and will light whenever the corresponding bit is '0'. If the scan code does not match one of these codes, the letter 'E' is displayed.

drive_7_segment

This process (available at /courses/TSTE12/material/Lab_Keyboard/drive_7_segment.vhdl) drives the 7-segment display of the NEXYS3 board. This process makes the interface to the 7-segment display compatible with the interface on the DE2-115 board used in later labs. The display on the NEXYS3 is multiplexed, and the process will alternately turn on one of the 7-segment digits. This is done fast enough to make it appear as if 4 different digits showing concurrently on the display.

The process uses the four vectors left_digit, middleleft_digit, middleright_digit and right_digit as source for the pattern to show on the 4-digit 7-segment display.

1.3.4 Tips and tricks

To avoid a lot of trouble in the design phase a few hints are useful:

- Do not use multiple clock definitions, i.e., rising_edge() or the corresponding 'EVENT definition can only be used on one signal. This signal can in this case only be the system clock signal named sys_clk.
- Do not use 'EVENT (except the above), 'LAST_VALUE, falling_edge or rising_edge (except the case above).
- It is ok to have the LED:s flicker while the data is shifted in when a key is pressed.

1.4 Interface

The synthesis tool needs to know how the FPGA is connected to the external components. An attribute description included in the VHDL description will be used for this purpose. How to include this into the design will be demonstrated later in section 2.3 Adding the pin attributes.

```
type mentor_string_array is array (natural range <>, natural range <>) of character;
attribute ARRAY_PIN_NUMBER : mentor_string_array;
attribute PIN_NUMBER : string;
attribute PULL : string;
attribute PIN_NUMBER of kb_clk : signal is "L12";
attribute PULL of kb_clk : signal is "PULLUP";
attribute PIN_NUMBER of kb_data : signal is "J13";
attribute PULL of kb_data : signal is "PULLUP";
attribute PIN_NUMBER of sys_clk : signal is "V10";
attribute PIN_NUMBER of sys_rst : signal is "B8";
attribute ARRAY_PIN_NUMBER of db : signal is
("T11","R11","N11","M11","V15","U15","V16","U16");
attribute ARRAY_PIN_NUMBER of ans : signal is ("N16","N15","P18","P17");
attribute ARRAY_PIN_NUMBER of seg : signal is
("M13","L14","N14","M14","U18","U17","T18","T17");
```

Please note that each string in a pin number attribute list must be of equal length. This is fixed by adding one space at the end of the string before “ if necessary. This is VHDL language requirement. Do not forget this in the project designs!

This attribute description is prepared in a text file and can be copied from

/courses/TSTE12/material/Lab_Keyboard/keyboard_NEXYS3_attributes.txt

1.5 Requirements

Here are the requirements to pass the laboratory:

- Implement the design using hdl_designer.
  - Declare a symbol and corresponding VHDL view.
- Design a test bench to verify the design.
  - The testbench should be built as a structural top level, i.e. block diagram in hdl_designer, and instantiate the keyboard symbol as a component.
  - The testbench should also include a stimuli and a control block in the same block diagram as the keyboard reader. The stimuli block generates the input data to the keyboard. The control block analyzes the design outputs.
  - The testbench has to implement at least two different scancodes (at least one being a digit key). The control block should check the output after each scancode have been generated (db, ans and seg).
  - The testbench should indicate each test case with correct or wrong behavior in the transcript window of the simulator. Take a look at the the ASSERT statement in VHDL. The user of the testbench should be able to load the testbench, type the command “run -all”, and then see three messages. The simulation should automatically stop when completed. The first two messages should indicate if the test of a key worked or not, and the final message stating that the testbench have completed all tests. The messages printed in the modelsim window could for example be

```
Key1 test ok
Key2 test ok.
All tests done, stopping simulation
```

- Synthesize the design and demonstrate the function on the NEXYS3 board.
- Summarize a number of key figures for the design.
  - How much of the total available design space is used?
  - What is the maximum clock frequency allowed for the design?

2. Design

It is here assumed that the reader have a basic knowledge of hdl_designer. The experience in the hdl designer intro in the tutorial of this course should be sufficient, available on the course material page on the web.

To enable one group working together on the same design, we prepared the setup. Follow the steps below to start the tool start and get the file permissions set correctly.

2.1 Introduction

Your designs should preferably be saved in a new project library directory.

1. Start the hdl designer tool.

   Load the course module TSTE12 and start a mentorskal

```
module load courses/TSTE12
mentorskal &
```

   Start hdl_designer in the mentorskal window using

```
TSTE12lab &
```

Once HDL Designer have started, do the following:

2. Open the design manager project view by selecting the Project tab at the lower left of the window.
3. Create a new project called LABS by selecting File→New→Project. Set the project name to LABS, the default working library name LAB1, and make sure the Directory in which your project folder will be created is /courses/TSTE12/labs/labgrp<nn>/LABS. Replace <nn> with your labgroup number.
4. Open/explore the library. Continue the work in this library and complete the exercise. If you have not opened the LAB1 library then you either double click on the library name or use right mouse button to access the popup menu for the library and choose explore.

2.2 Design entry

Implement the design according to the description and requirements found in section 1. The tutorial material describes how to use the HDL designer tool.

2.3 Adding the pin attributes

The symbol pin attributes are necessary in the synthesis step. Here it is described how to include the prepared pin attributes from section 1.4 Interface into the keyboard design.

1. Create the symbol for the design.
2. Open the symbol editor window
3. Locate "User:" under Declarations in the window. Place the mouse pointer over the text. Double-click using left mouse button on the text.
4. Open the pin attributes text file in a text editor.
5. Copy and paste the pin attributes from the text file into the declarations dialog box.
6. Press Ok and the attributes should appear under User declarations.
7. Save the symbol.
8. Finish the design with ports and behavioral view.

The symbol and interface is now complete. Continue to add the behavior of the design and validate the functionality.

2.4 Simulating the design

The created model should be simulated to verify that the design is correct. Modelsim can be executed from within the hdl_designer environment. Stimuli for test of the design can be created using a testbench, which is the prefered way.

During debugging may the repeated change of the testbench code followed by compile and simulate be a bit time consuming. Another approach is therefore to use the interactive debugging in Modelsim. You may then find the following useful:

Add stimulis to kb_clk and kb_data and validate the function. These can either be written into the Modelsim window or put in a macro file. The following example simulate the scan code for the ´4´ key (remember that the keyboard should only generate a clock when there are bits to send):

```
force  -repeat 10 sys_clk 0 0, 1 5
force sys_rst 1 0, 0 100
force kb_clk 1 25us, 0 100us, 1 150us, 0 200us, 1 250us, 0 300us, 1 350us, 0 400us, 1
450us, 0 500us, 1 550us, 0 600us, 1 650us, 0 700us, 1 750us, 0 800us, 1 850us, 0
900us, 1 950us, 0 1000us, 1 1050us, 0 1100us, 1 1150us
force kb_data 1 25us, 0 75us, 1 175us, 0 275us, 1 375us, 0 475us, 0 575us, 1 675us, 0
775us, 0 875us, 0 975us, 1 1075us
run 1200 us
```

These commands can also be put in a macro file to simplify repeated tests. Use a text editor to create the file and put the commands to be executed in it. It is important to keep all text on each force command in one single line for each force.

2.5 Creating the testbench

Here we describe briefly the steps to create a testbench. In order to write the testbench, check again the requirements in section 1.5.

1. Create a new symbol for the testbench design.
2. Open the symbol editor window
3. Open it as a new block diagram. This should open a new block diagram window.
4. Instantiate the keyboard reader as a component. Press the green block entry in the toolbars menu, i.e. , "add component".
5. Now a window with available components appear. Drag and drop the keyboard symbol on the testbench block diagram window.
6. Continue and add the stimuli block and control block to the schematic, using blue boxes. Connect these blocks to the keyboard.
7. Add behavior to the control and stimuli blocks.
8. Validate the testbench design. Important now is to choose generate and compile through components.
9. Validate the design.

1.1 Synthesis flow (using Precision)

1.1.1 Synthesize the Design

The design flow buttons include several steps in one command. There are flows defined for simulation and synthesis. The synthesis assume a validated and compiled design.

Select the keyboard design unit in the design browser.

Start synthesis by choosing the synthesis flow button. You find this next to the compile and generate toolbar buttons.

The Precision Synthesis settings appear. Make sure to change the following in the setup.

Technology:	Xilinx/SPARTAN6
Device:	6SLX16CSG324
Speed Grade:	-3
Design frequency:	100 MHz

Run Options:	Run Integrated Place and Route should be selected
Implementation options	Single Implementation Mode

Press ok. The synthesis tool is now started.

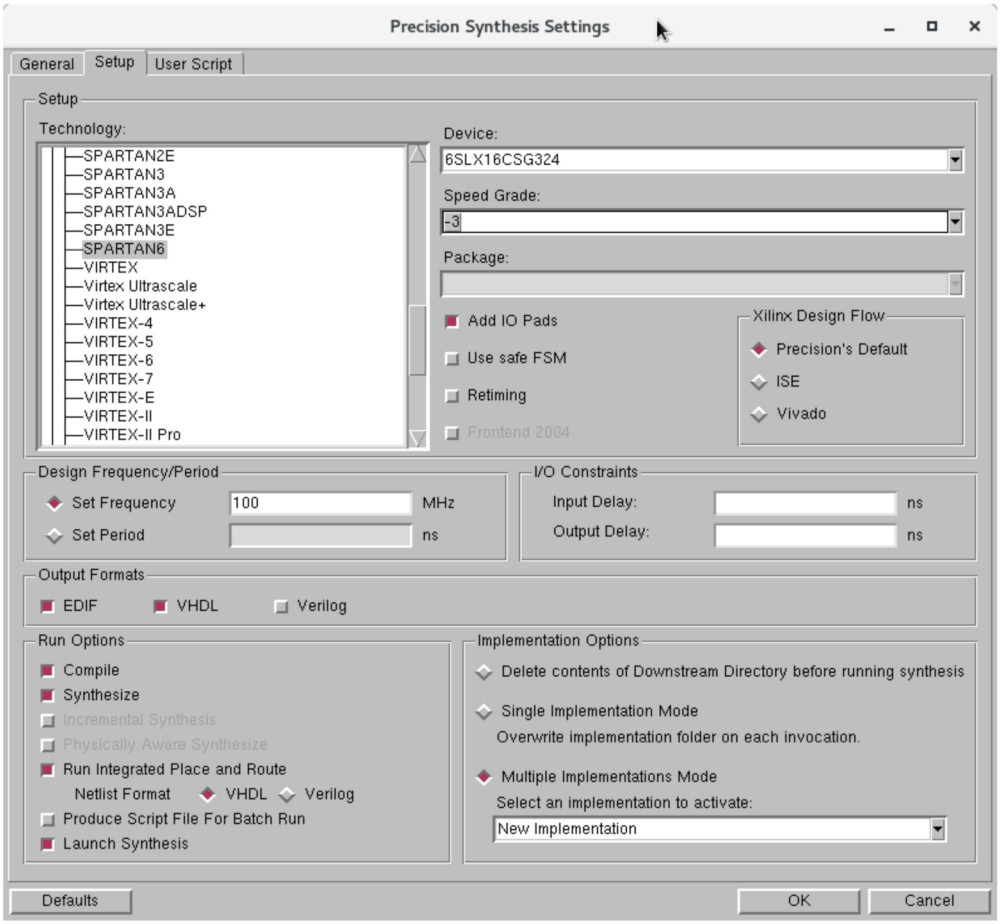


Figure 3: Synthesis settings dialog window

The synthesis now starts and generates a netlist, various reports including used pins, timing estimates etc. There is however not yet any bit FPGA configuration file created.

In the precision tool select the configuration menu using tools→place and route options→Integrated place and route. Set the “Generate Bitgen File” checkbox at the bottom of the windows, followed by OK, see Figure 4.

The next step is to rerun the place and route part of the synthesis. This is done by first selecting the ISE column on the left side of the window, and then press the place and route icon, see Figure 5.

Once the place and route step has been run, a bit FPGA configuration file is added to the synthesis results. The current settings (see implementation options in the lower right part of Figure 3). The synthesis results are located in a subfolder to /courses/TSTE12/labs/labgrp<nn>/LABS/LAB1/ps. The bit programming file is located in a subdirectory inside the synthesis folder (ps).

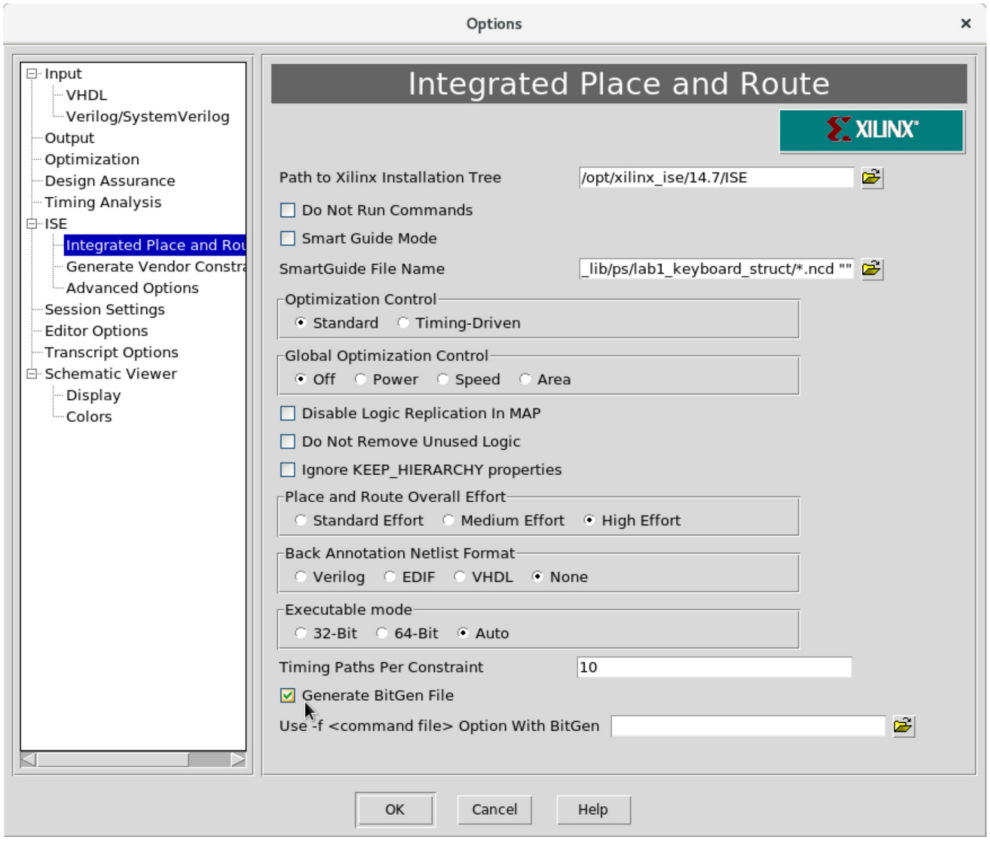


Figure 4: Setting the bitgen option

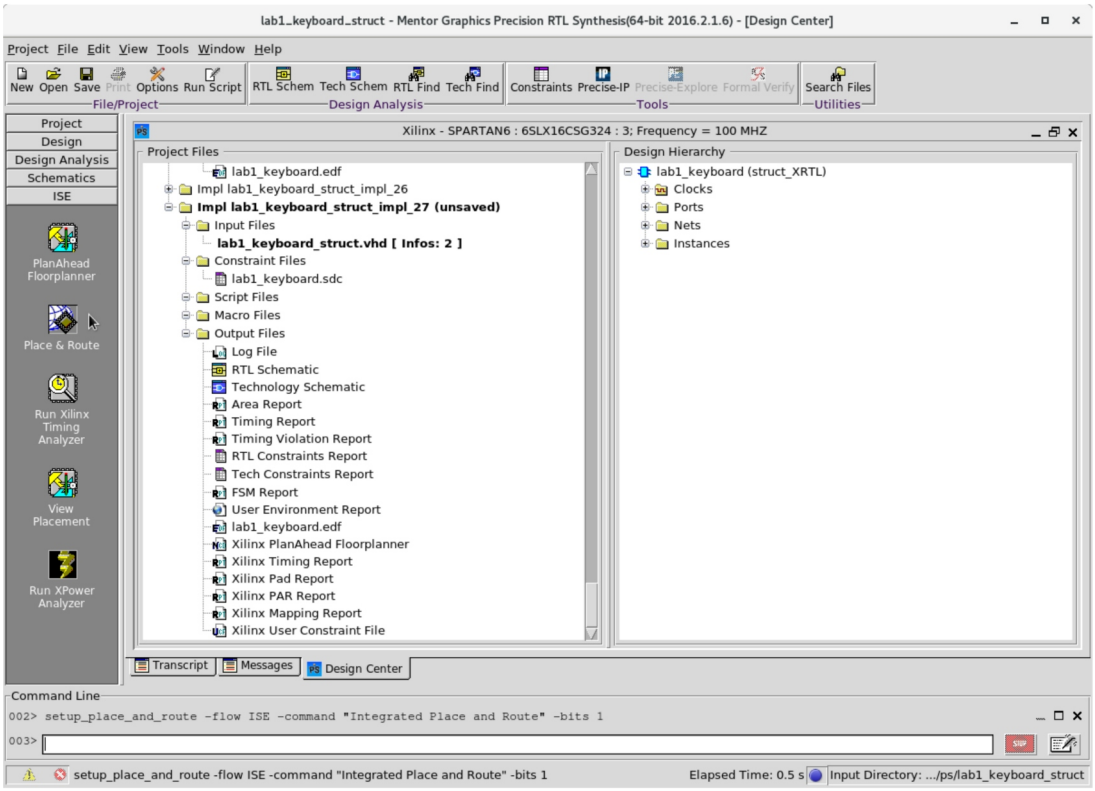


Figure 5: Rerun place and route by selecting ISE column menu

1.1.2 Results

A lot of reports has again been generated. Answer the following questions by looking at the design center tab in the precision tool. Look especially at the area report and the timing report:

Resource	Used	Available	Utilization	
IOs	.....	.....	.....	%
LUTs	.....	.....	.....	%
Registers	.....	.....	.....	%
Memory Bits	.....	.....	.....	%

1.2 Downloading the Design

The NEXYS3 board is always programmed from the computer it is attached to using an USB connection. The programming is carried out using a special programming software from the board vendor.

1.2.1 Programming the FPGA

These steps are for programming the NEXYS3 FPGA board.

1. Open a terminal window
2. Locate the .bit file using cd, ls and in some cases find (use ‘man find’ to understand how to use it) and position yourself in that folder.
3. Verify that the board is attached (and make sure it is power on).
   djtgcfg enum
   This should find one Nexys3 board
4. Program the board using the command
   djtgcfg prog -i 0 -d Nexys3 -f yourdesignname.bit
   The new design will be downloaded within seconds.
5. Try out the hardware. Verify that it is your group number showing on the leftmost 7-segment LEDs.
6. Press keys on the keyboard and observe the results on the LED displays.

1. Appendix: PC AT keyboard ref

A.1 Description of keyboard

The original keyboard design had a single chip microprocessor, but now a customized controller chip is used. This keyboard controller chip takes care of all keyboard matrix scanning, key de-bouncing and communications with the computer, and has an internal buffer if the keystroke data cannot be sent immediately. The PC motherboard decodes the data received from the keyboard via the PS/2 port using interrupt IRQ1.

The one thing that these keyboards do not generate is ASCII values. With a typical AT keyboard having more than 101 keys, a single byte could not store codes for all the individual keys, plus these keys along with shift, control, or alt, etc. Also for some functions there is no ASCII equivalent, for example 'page up', 'page down', 'insert', 'home', etc.

When the keyboard controller finds that a key is being pressed or released it will send this keystroke information, known as scan codes, to the PC motherboard. There are two different types of scan codes - make codes and break codes.

The communications protocol is bi-directional, but here we only discuss the keyboard to host part.

A.1.1 make code

A make code is sent whenever a key is pressed or held down. Each key, including 'shift', 'control' and 'alt', sends a specific code when pressed. Cursor control keys, 'delete', 'page up', 'page down', 'ins', 'home' and 'end', send extended make codes. The make code is preceded by 'E0'h to indicate an extended code. The only exception is the 'pause' key that starts with a unique 'El'h byte.

A.1.2 break code

A break code is sent when a key is released. The break code is the make code preceded by 'F0'h byte. For extended keys the break code has an 'E0'h preceding the 'F0'h and make code value. The only exception is the 'pause' key as it does not have a break code and does not auto-repeat when held down.

A.1.3 key code

Every key is assigned its own unique code so that the host computer processing the information from the keyboard can determine exactly what happened to which key simply by looking at the scan codes received. There is no direct relationship between the scan code generated by a particular key and the character printed on the key top.

The set of make and break codes for each key comprises a scan code set. There are three standard scan code sets -numbered 1, 2, and 3 - stored within the keyboard controller. Scan code set 1 is retained for compatibility for older IBM XT computers. Scan set 3 is very similar to the set 2 but the extended codes are different. Scan code set 2 is the default for all AT keyboards and all scan codes discussed here are from this set.

A.1.4 scan code

If, for example, you press 'shift' and 'A' then both keys will generate their own scan codes, the 'A' scan code value is not changed if a shift or control key is also pressed. Pressing the letter 'A' generates 'lC'h make code and when released the break code is 'F0'h, 'lC'h.

Pressing 'shift' and 'A' keys will generate the following scan codes :

The make code for the 'shift' key is sent '12'h.

The make code for the 'A' key is sent 'lC'h.

The break code for the 'A' key is sent 'F0'h, 'lC'h.

The break code for the 'shift' key is sent 'F0'h,' 12'h.

If the right shift was pressed then the make code is '59'h and break code is 'F0'h, '59'h.

By analysing these scan codes the PC software can determine which key was pressed. By looking at the shift keystroke the software can distinguish between upper and lower case.

A.1.5 Keyboard serial data

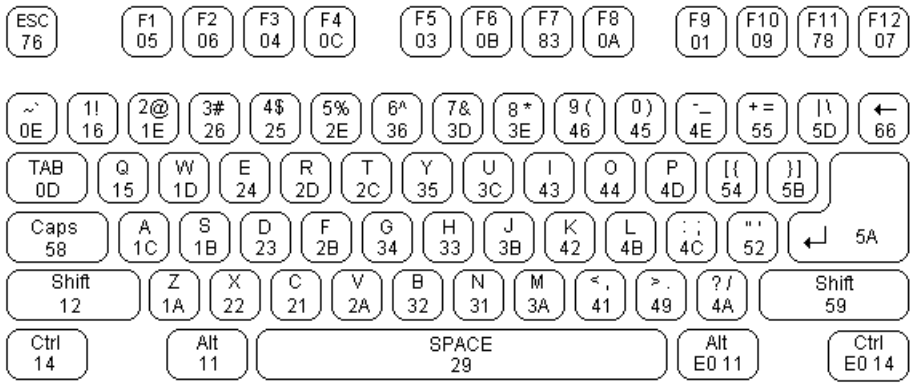
The AT keyboard transmission protocol is a serial format, with one line providing the data and the other line providing the clock. The data length is 11 bits with one start bit (logic 0), 8 data bits (lsb first), odd parity bit and a stop bit (logic 1). The clock rate is approximately 10 to 30kHz and varies from keyboard to keyboard.

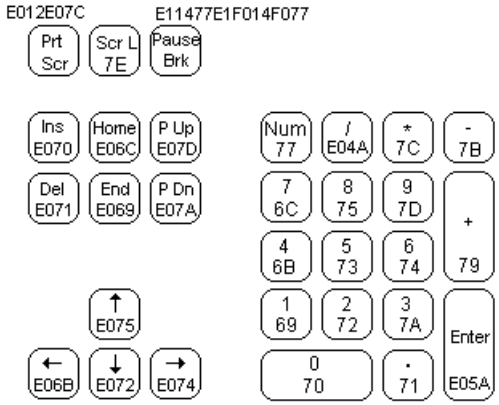
A.2 Scancodes and commands

A.2.1 Keyboard layouts with codes

Scan code tables

Keyboard alpha numeric scan codes





Key	Make	Break	
A	'lC'h	’F0’h	‘1C’h
B	'32'h	'F0’h	'32'h
C	'21'h	'F0’h	'21'h
D	'23'h	'F0’h	'23'h
E	'24'h	'F0’h	'24'h
F	'2B'h	'F0’h	'2B'h
G	'34'h	'F0’h	'34'h
H	'33'h	'F0’h	'33'h
I	'43'h	'F0’h	'43'h
J	'3B'h	'F0’h	'3B'h
K	'42'h	'F0’h	'42'h
L	'4B'h	'F0’h	‘4b'h
M	'3A'h	'F0’h	'3A'h
N	'31'h	'F0’h	‘31'h
O	'44'h	'F0’h	'44'h
P	'4D'h	'F0’h	'4D'h
Q	'15'h	'F0’h	‘15'h
R	'2D'h	'F0’h	'20'h
S	'IB'h	'F0’h	‘1B'h
T	'2C'h	'F0’h	'2C’h
U	'3C'h	'F0’h	'3C’h
V	'2A'h	'F0’h	'2A'h
W	'ID'h	'F0’h	‘1D'h
X	'22'h	'F0’h	'22'hhttps://www.win.tue.nl/~aeb/linux/kbd/scancode

Key	Make	Break	
		s.html	
Y	'35'h	'F0’h	'35'h
Z	'lA'h	'F0’h	‘1A'h
1	'16'h	'F0’h	‘16'h
2	'lE'h	'F0’h	‘1E'h
3	'26'h	'F0'h	'26'h
4	'25'h	'F0'h	'25'h
5	'2E'h	'F0'h	'2E'h
6	'36'h	'F0'h	'36'h
7	'3D'h	'F0'h	'3D'h
8	'3E'h	'F0'h	'3E'h
9	'46'h	'F0'h	'46'h
0	'45'h	'F0'h	'45'h

A.3 Further reading

A.3.1 Web references

https://www.avrfreaks.net/sites/default/files/PS2%20Keyboard.pdf

https://www.win.tue.nl/~aeb/linux/kbd/scancodes.html