

# EECS2070 02

# Digilent Basys3 FPGA Board

## Part 3

Ref: Digilent Basys3™ FPGA Board Reference Manual

黃稚存



國立清華大學  
資訊工程學系

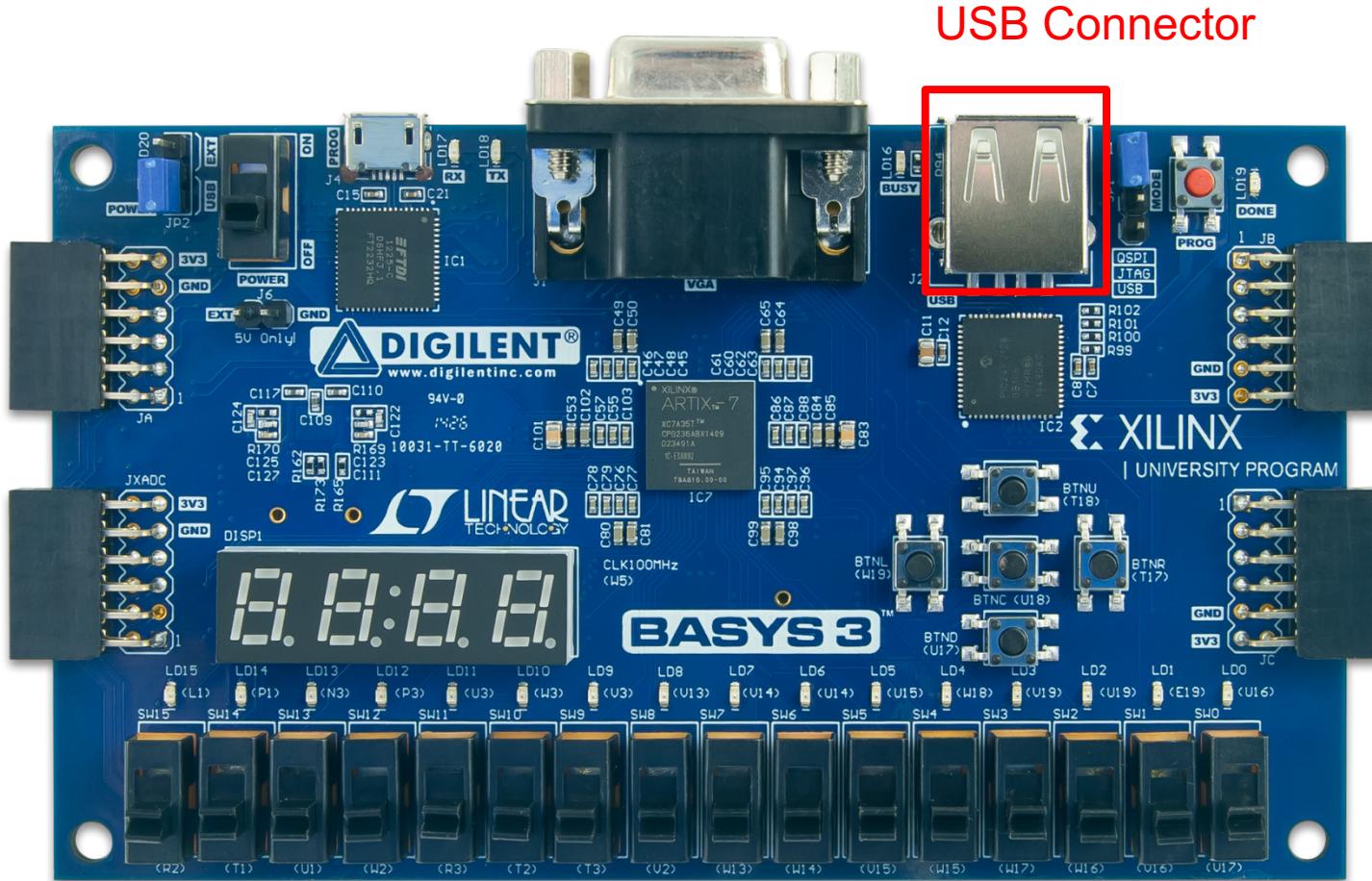
Lecture 04

# 聲明

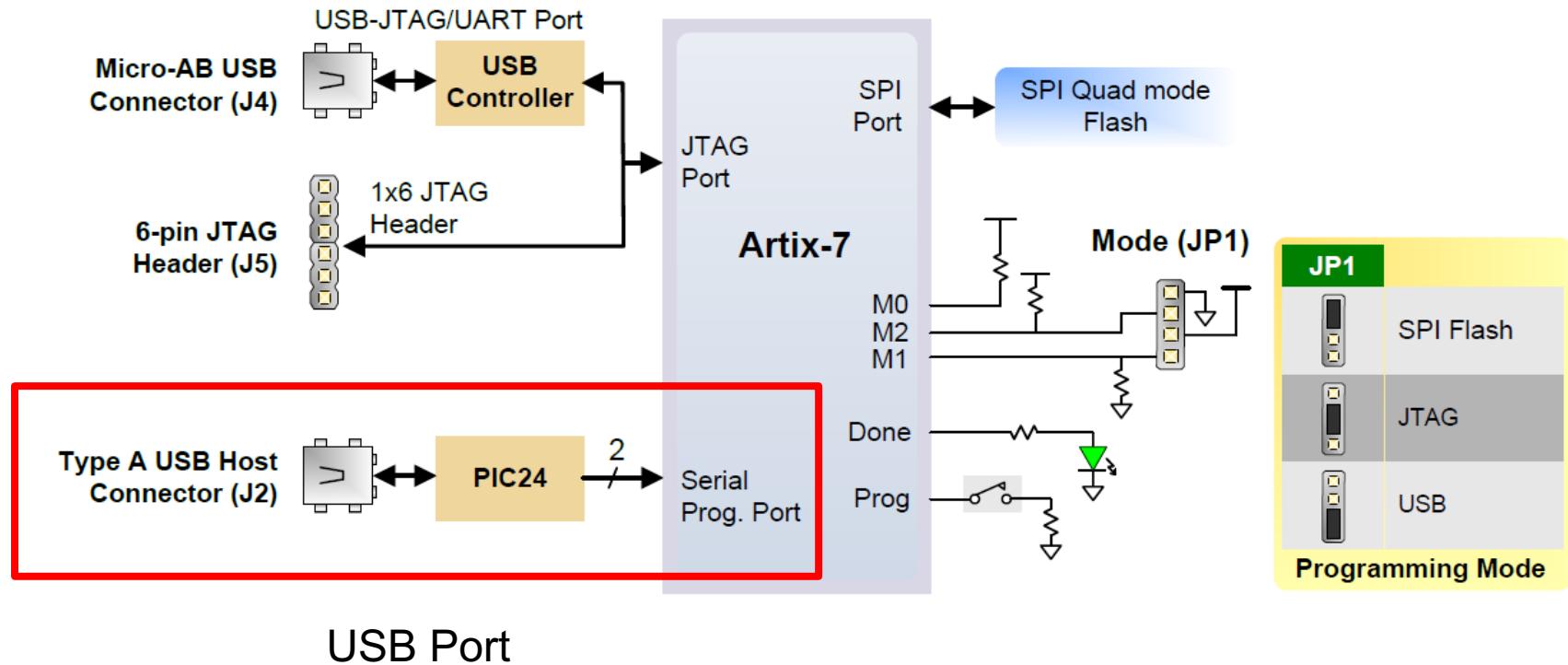
- 本課程之內容（包括但不限於教材、影片），僅供修課學生個人合理使用，非經授課教師同意，不得以任何形式轉載、重製、散布、公開播送、出版或發行本影片內容（例如將課程內容放置公開平台上，如 Facebook, Instagram, YouTube, Twitter 等等）。如有侵權行為，需自負法律責任。

# Basic Protocols of Keyboard Control

# USB HID (Human Interface Device) Host (1/3)

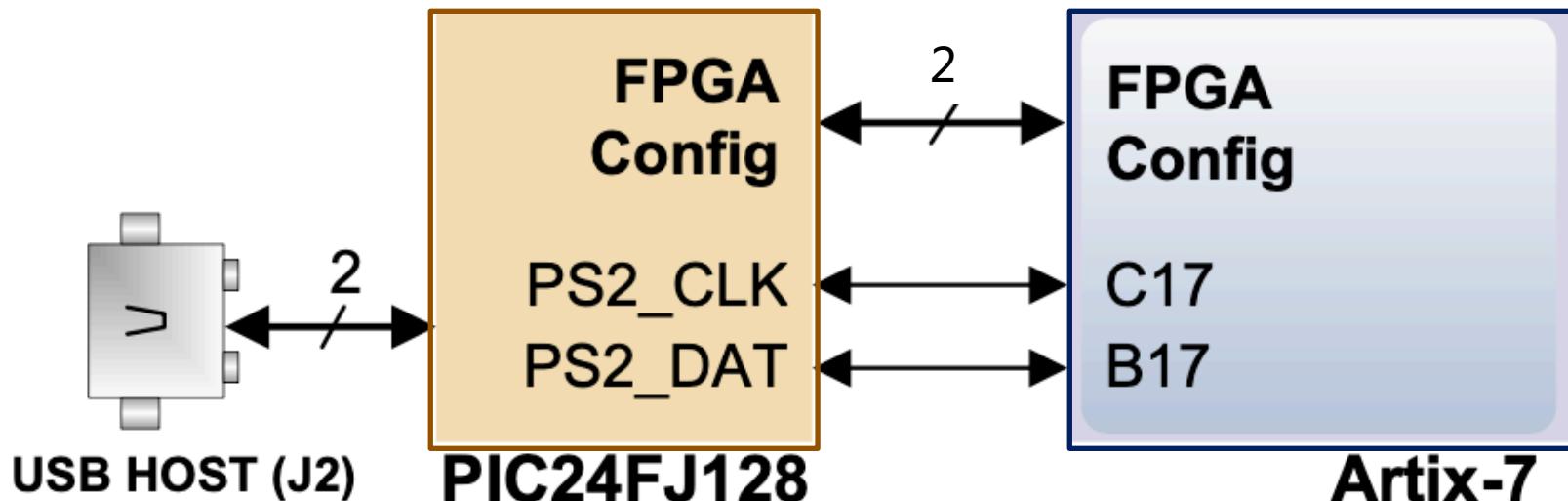


# USB HID Host (2/3)



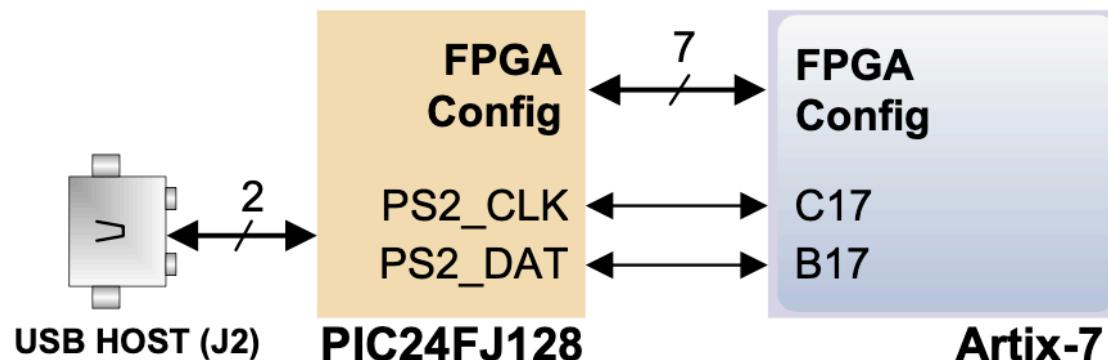
# USB HID Host (3/3)

- The USB port is controlled via a PIC24 chip
  - ◆ The PIC24 chip translates USB signals to PS2 signals
  - ◆ The two interface ports on the FPGA are B17 and C17



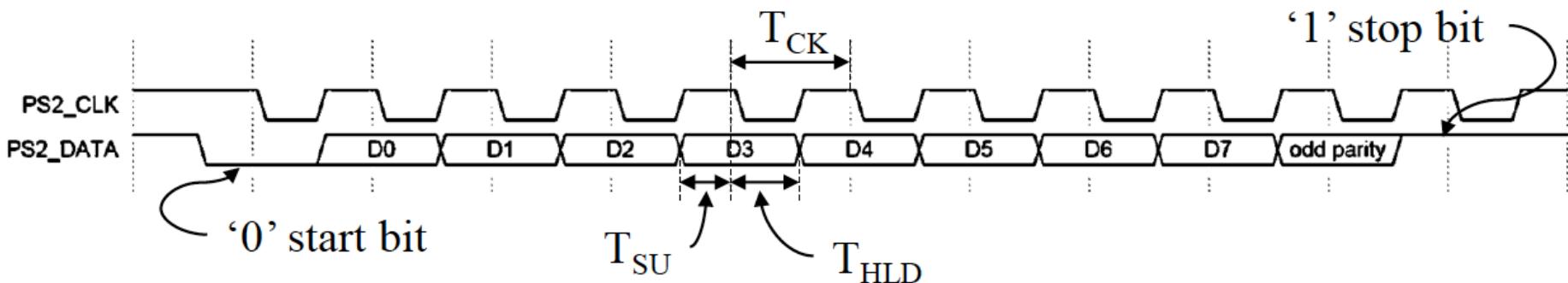
# Microchip PIC24FJ128

- Configuration mode
  - ◆ Download the bitstream to the FPGA.
- Application mode
  - ◆ On Basys 3, this is called USB HID Host mode
  - ◆ Hub support is not currently available
  - ◆ Either a single mouse or a single keyboard can be used
  - ◆ PS2\_CLK and PS2\_DATA are used to implement a standard PS/2 interface



# HID Controller

- 11-bit words are used between keyboard and FPGA
  - Start bit "0" / 8-bit data (LSB first) / a parity bit / stop bit "1"
  - Each bit is read on the falling edge of the clock



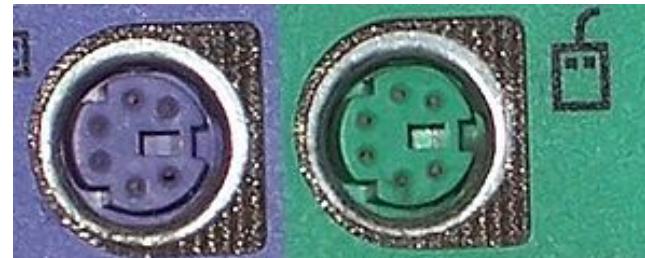
Symbol	Parameter	Min	Max
$T_{\text{CK}}$	Clock time	30 us	50 us
$T_{\text{SU}}$	Data-to-clock setup time	5 us	25 us
$T_{\text{HLD}}$	Clock-to-data hold time	5 us	25 us

# Initialization

- When a keyboard or mouse is connected to the Basys 3, a “self-test passed” command (**0xAA**) is sent to the Basys 3
  - ◆ The command (**0xAA**) is automatically sent by the keyboard to FPGA
  - ◆ This command informs FPGA that the keyboard is ready
  - ◆ Only sent once when the system turns on
- The Basys 3 board may request the keyboard to do several times of self-test by command (**0xFF**)
  - ◆ The keyboard uses **0xFA** to respond to the requests, and may send “self-test passed” again to Basys 3
  - ◆ Keyboard : **0xFA (0xFA 0xAA)**
- The protocol is already implemented
  - ◆ You don’t need to worry about it
  - ◆ But it is still good to know the concept

# PS/2 Port

- PS/2 port was formerly used for keyboards and mice
- Nowadays, they are replaced by USB ports
- Their protocols are still widely used
- Scancodes are used to communicate with the PS/2 port



Example PC compatible (IBM PS/2) scancodes

key	set 1 (IBM PC XT)		set 2 (IBM PC AT)		set 3 (IBM 3270 PC)	
	press	release	press	release	press	release
A (normal letter)	1E	9E	1C	F0 1C	1C	F0 1C
Return / Enter (main keyboard)	1C	9C	5A	F0 5A	5A	F0 5A
Enter (numeric keypad)	E0 1C	E0 9C	E0 5A	E0 F0 5A	79	F0 79
Left Windows key	E0 5B	E0 DB	E0 1F	E0 F0 1F	8B	F0 8B
Right Windows key	E0 5C	E0 DC	E0 27	E0 F0 27	8C	F0 8C

# PS/2 Scancode

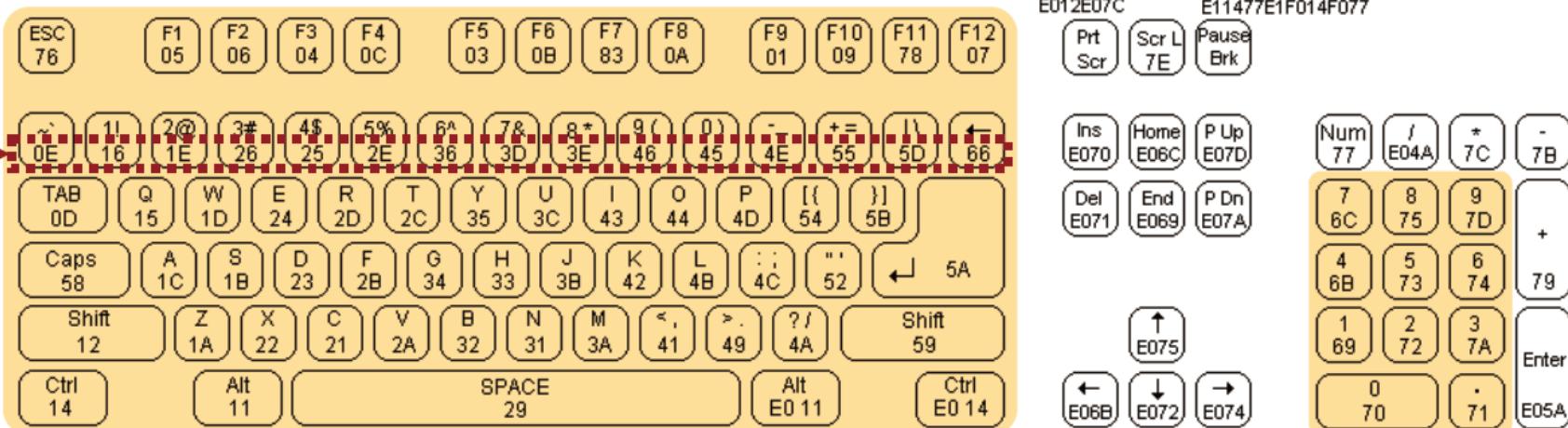
## Three types of scancodes

- ◆ Make codes represent the key values
- ◆ Break code represents the action of “release the key”
- ◆ Extend code represent the duplication of a key

Extend Code	Break Code	Make code
E0	F0	XX

(means “release”)

Make code

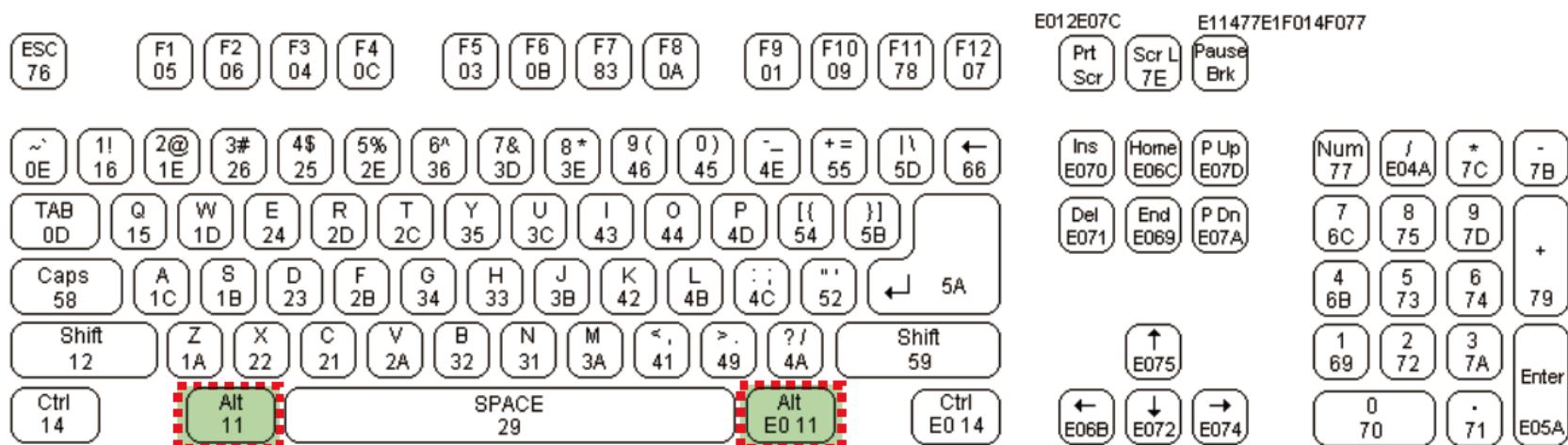


We only use the yellow parts of the keyboard

# PS/2 Scancode (Example)

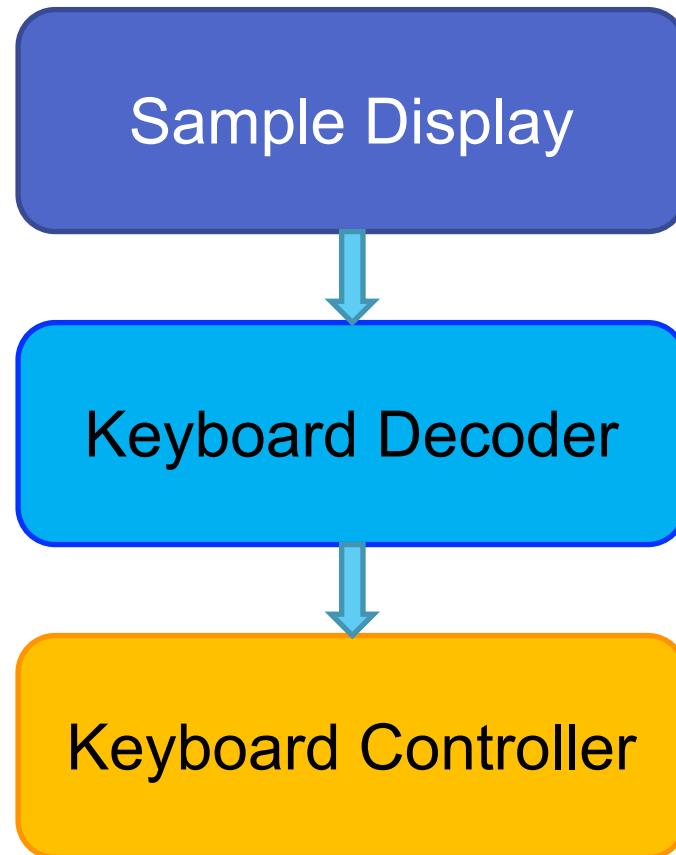
- Two “Alt” key are available on the keyboard
  - Additional keys are encoded as <extend code> + <make code>
  - When releasing a key, a <break code> is inserted

L Alt press			11
L Alt release		F0	11
R Alt press	E0		11
R Alt release	E0	F0	11



# Keyboard Controller

# Program Hierarchy



`SevenSegment.v`

`SampleDisplay.v`

`KeyboardDecoder.v`

`KeyboardCtrl.v`  
`Ps2Interface.v`

`OnePulse.v`

} Vivado IP

# KeyboardCtrl.v

# Verilog Module: KeyboardCtrl (1/2)

- In Keyboard-Controller

- ◆ Ps2Interface.v
- ◆ KeyboardCtrl.v

- I/Os for KeyboardCtrl.v

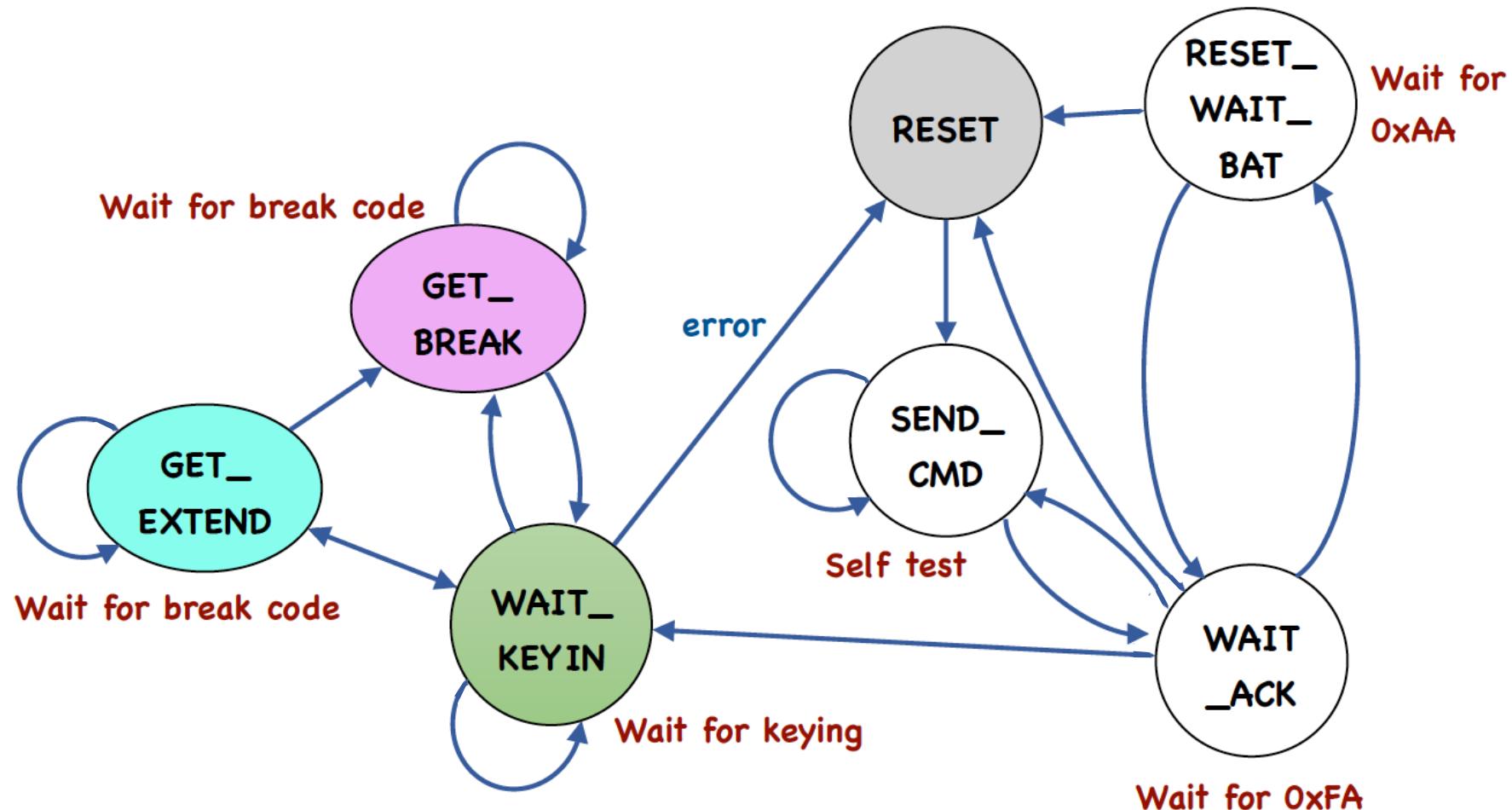
## Output

- key\_in
- is\_extend
- is\_break
- valid
- err

## Input

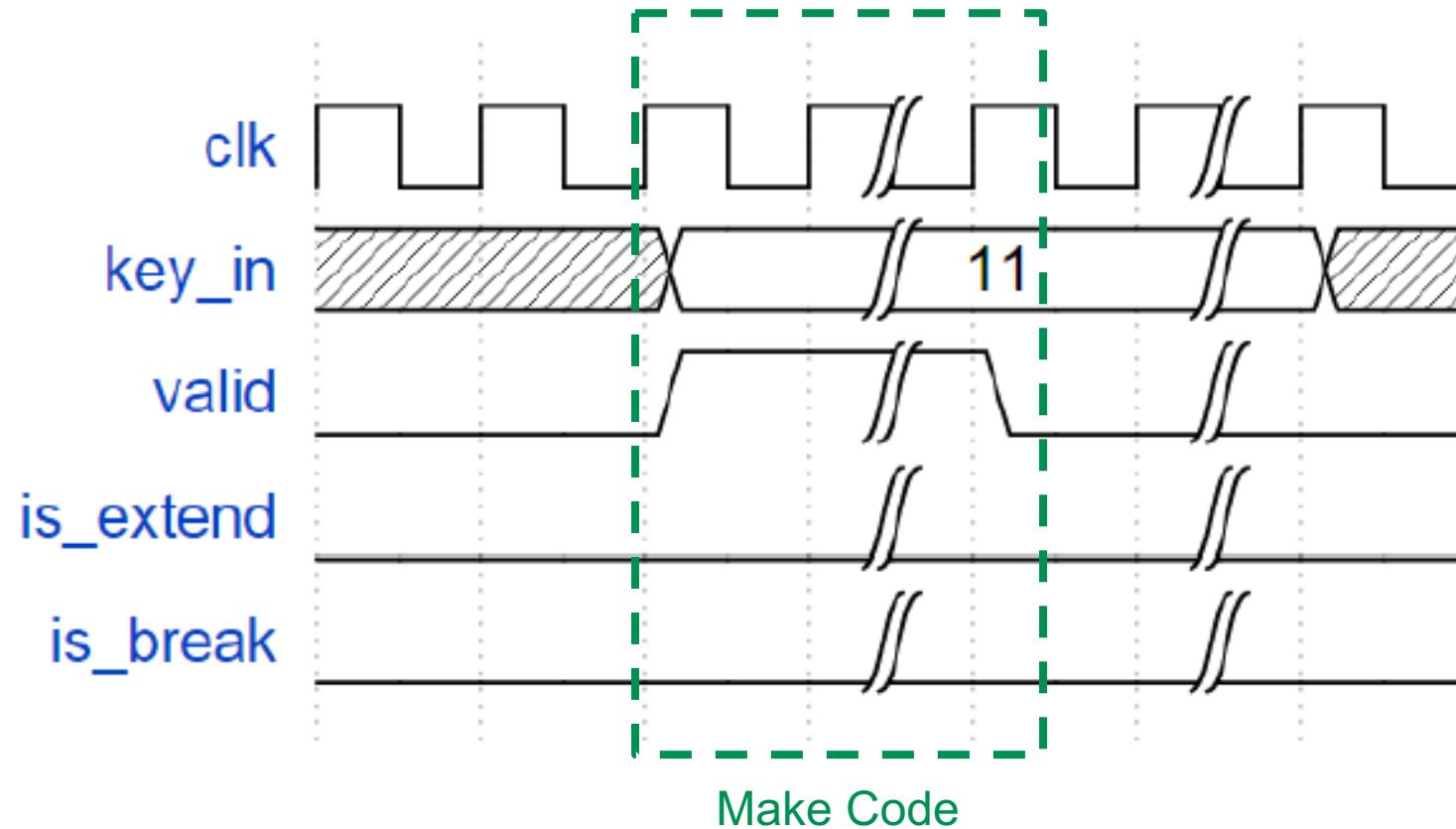
- PS2\_CLK
- PS2\_DATA
- rst
- clk

# Verilog Module: KeyboardCtrl (2/2)



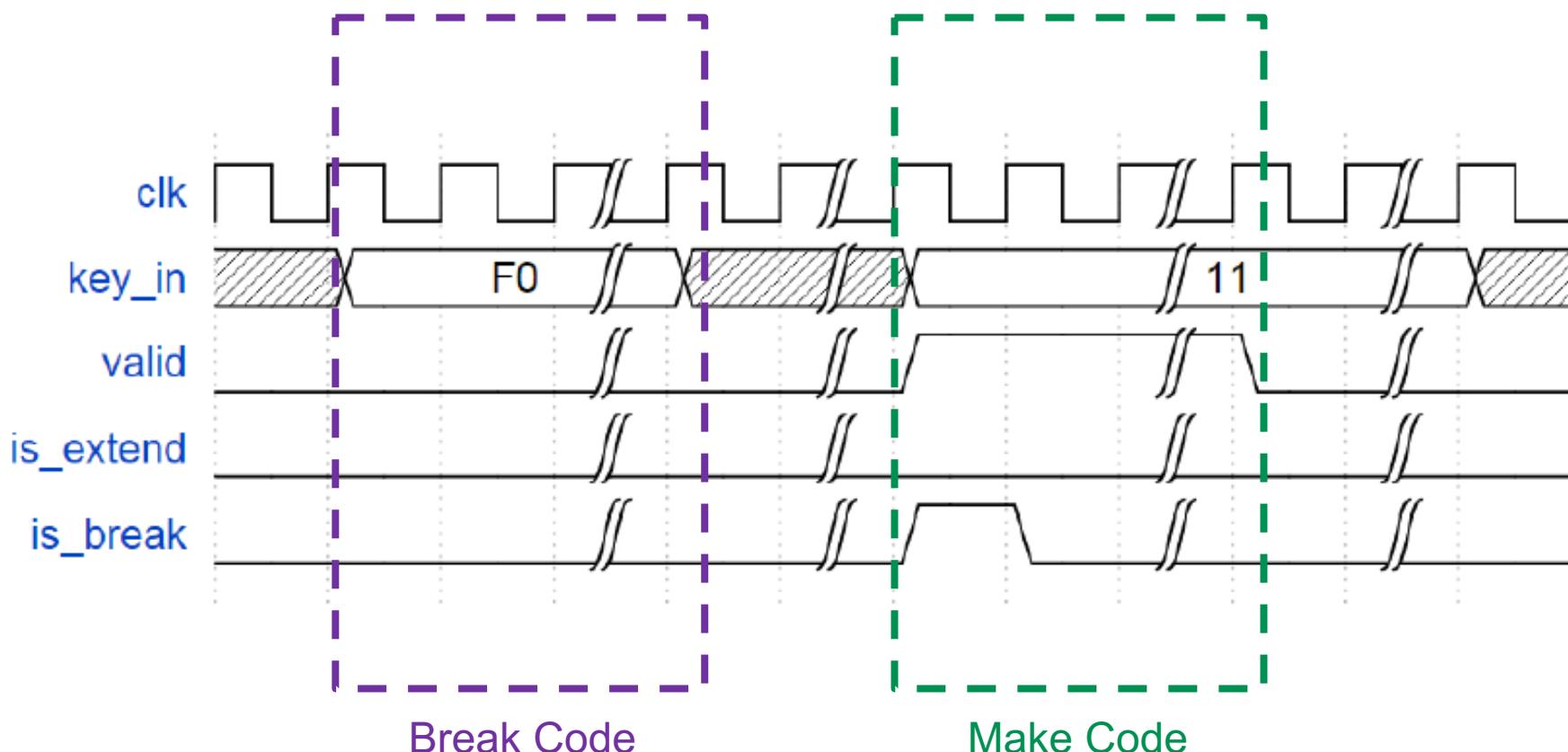
# KeyboardCtrl (Output Example 1)

- L Alt pressed



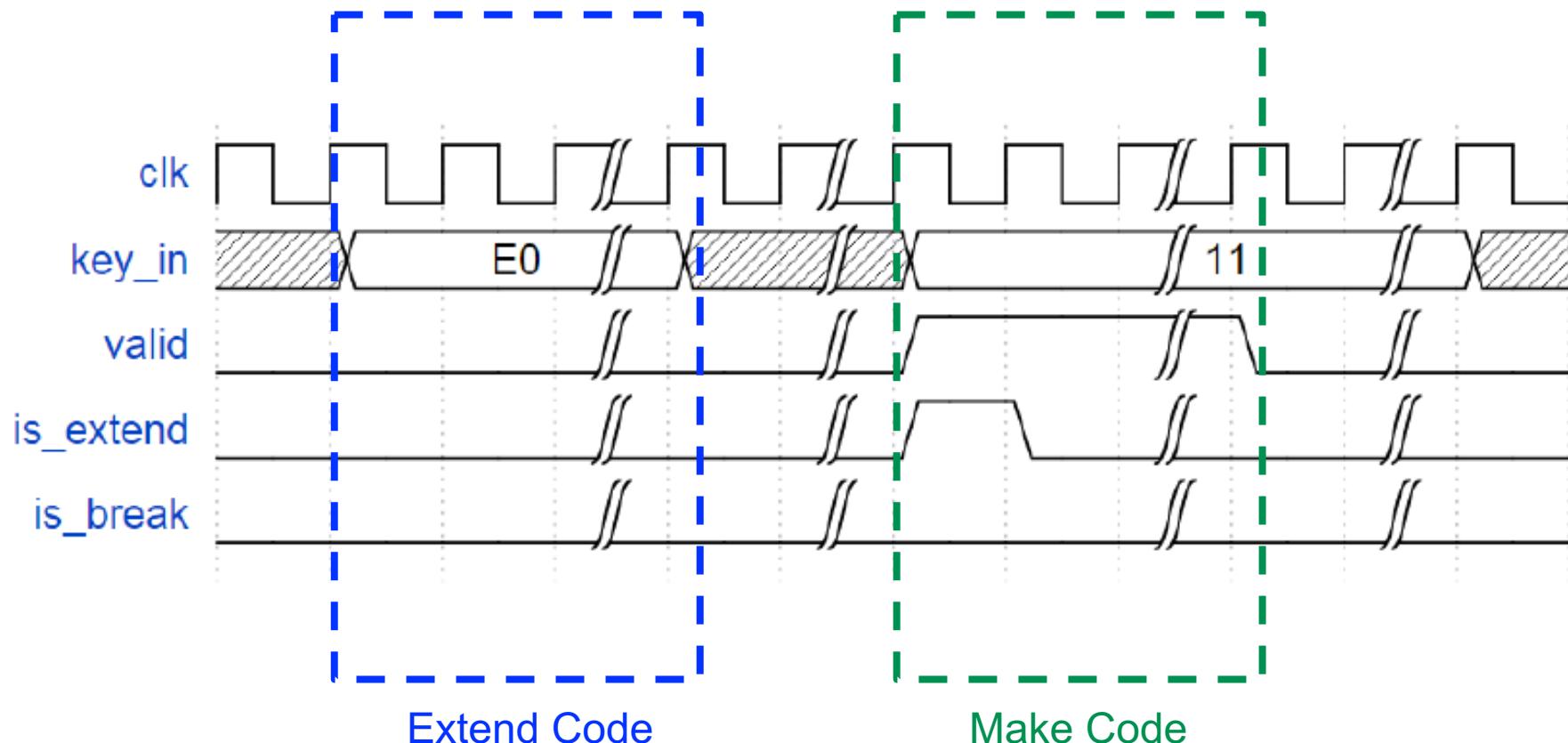
# KeyboardCtrl (Output Example 2)

- L Alt released



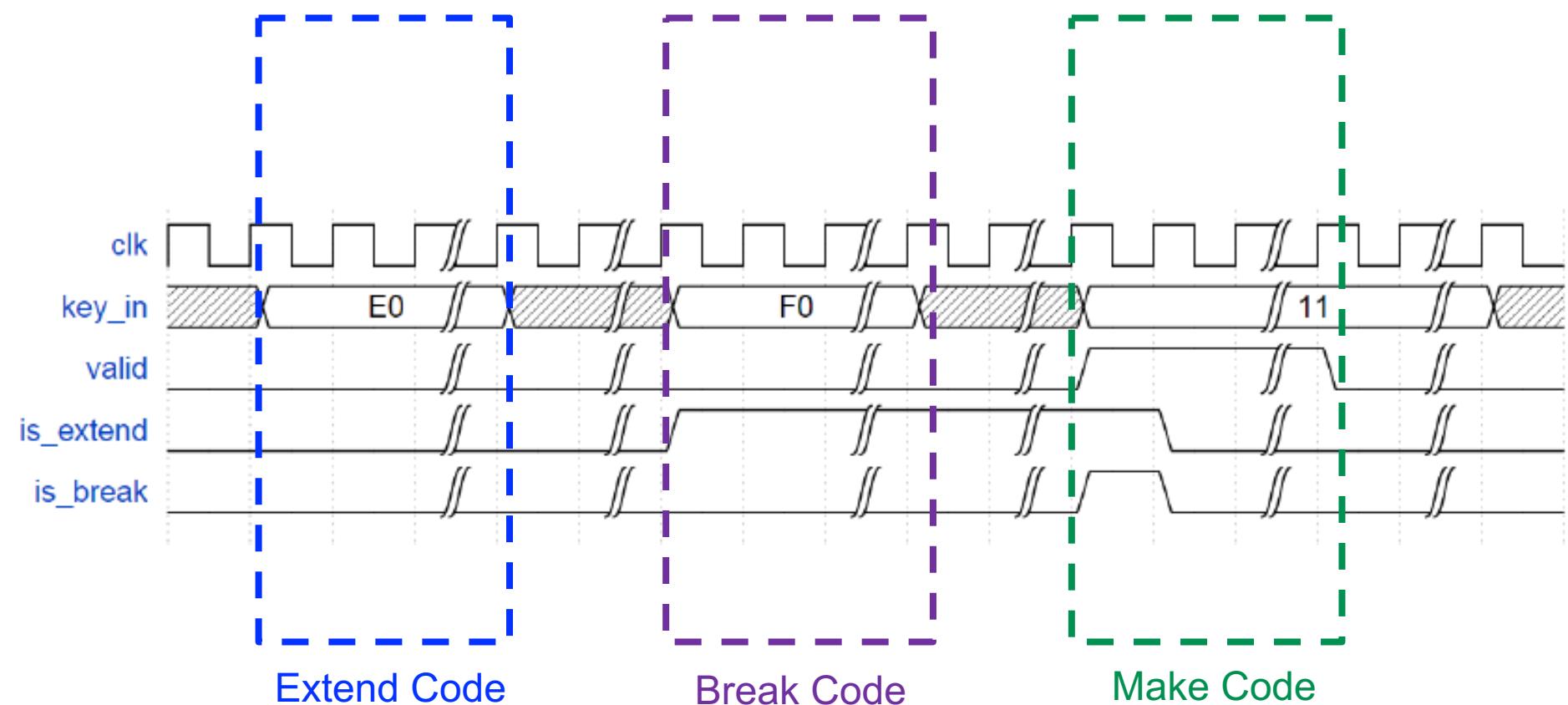
# KeyboardCtrl (Output Example 3)

- R Alt pressed



# KeyboardCtrl (Output Example 4)

- R Alt released



# KeyboardDecoder.v

# Verilog Module: KeyboardDecoder (1/5)

- In Keyboard Sample Code

- ◆ KeyboardDecoder.v
- ◆ SampleDisplay.v

- I/O for KeyboardDecoder

## Output

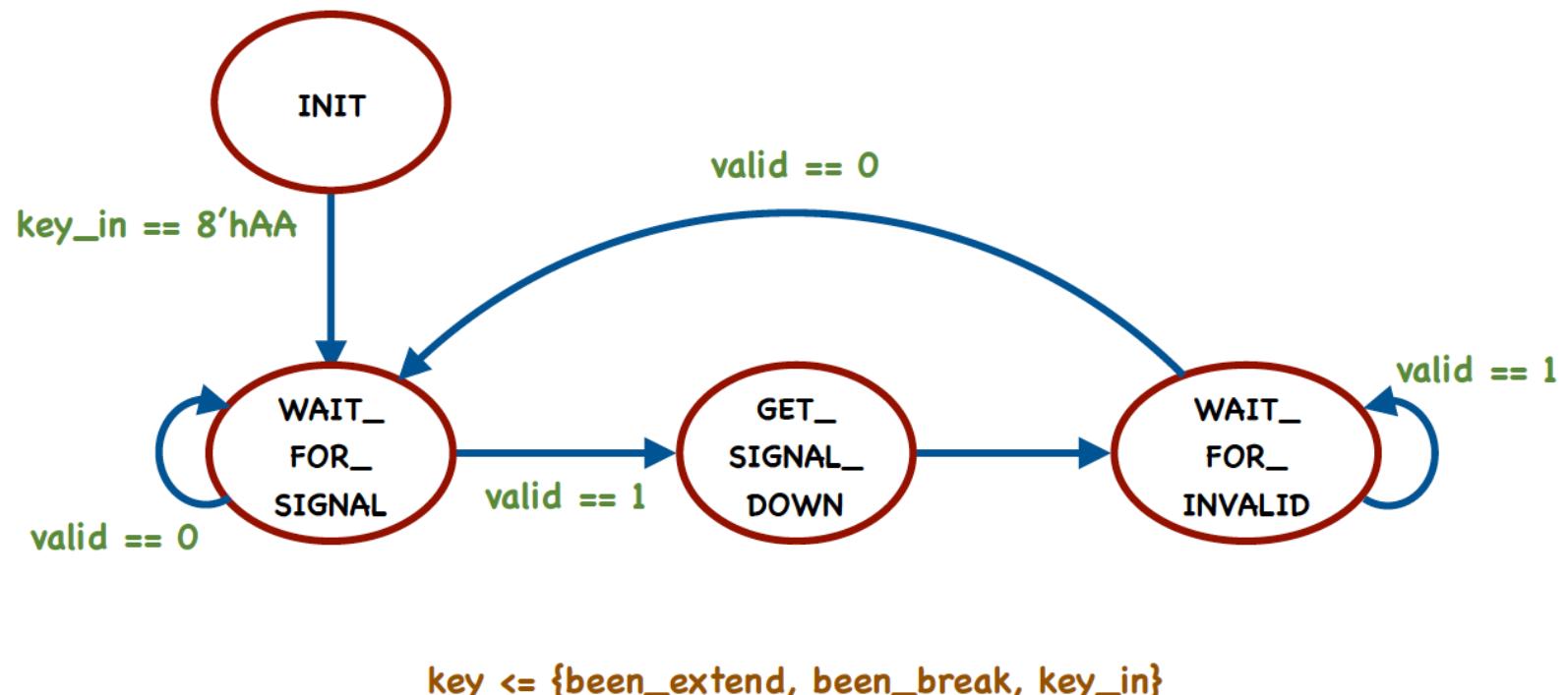
- key\_down
- last\_change
- Key\_valid

## Input

- PS2\_CLK
- PS2\_DATA
- rst
- clk

# Verilog Module: KeyboardDecoder (2/5)

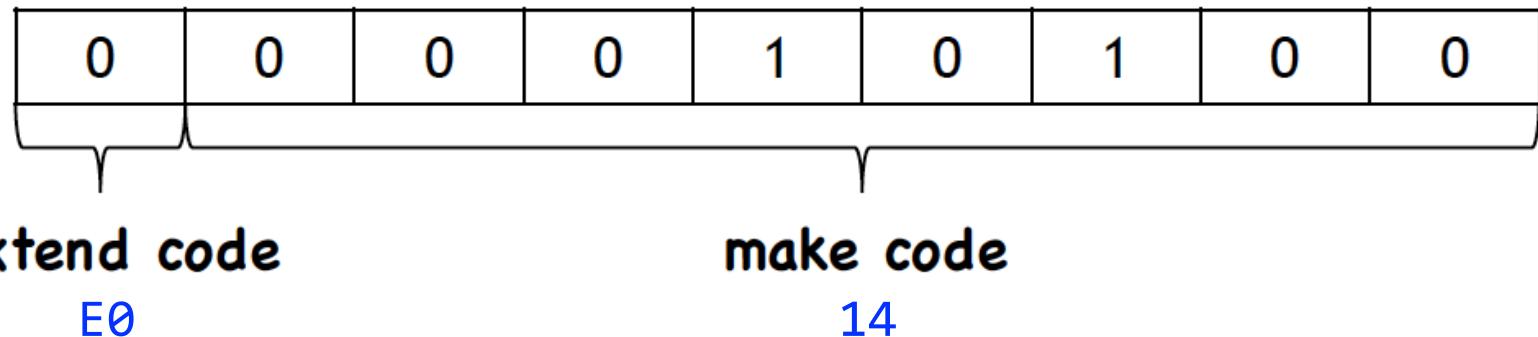
- Retrieving “extend”, “break”, and “key” from KeyboardCtrl
  - Getting value only when  $\text{valid} == 1\text{'b}1$



# Verilog Module: KeyboardDecoder (3/5)

- last\_change: 9 bits

- Represent the key which has been pressed or released

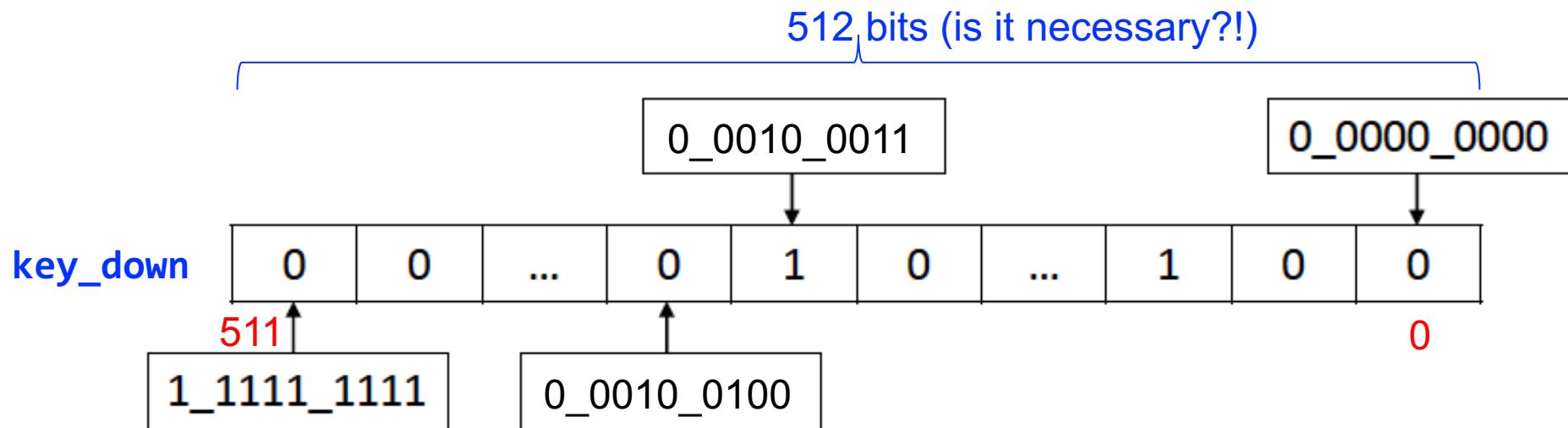


- key\_valid: 1 bit

- Should be active for one clock period (100MHz) when any key is pressed or released

# Verilog Module: KeyboardDecoder (4/5)

- Use an array “`key_down`” of 512 bits to record which key is currently being pressed
  - 1 means “key is pressed”, while 0 means “key is released”
  - Ex: the key indexed by “0\_0010\_0011” (“D”: code 23) is pressed
  - Ex: the key indexed by “0\_0010\_0100” (“E”: code 24) is released
  - Ex: the key indexed by “1\_0101\_1010” is “Right Enter” (Num Pad) (code E0 5A)



- Use another array “`key_decode`” (also 512 bits) to represent the key that was just pressed or released

# Verilog Module: KeyboardDecoder (5/5)

- To represent that a key is pressed
  - ◆ `key_down <= key_down | key_decode;`

<code>key_down</code>		0	1	1	0	1
<code>key_decode</code>	or	0	0	0	1	0
		0	1	1	1	1

- To represent that a key is released
  - ◆ `key_down <= key_down & (~key_decode)`

<code>key_down</code>		0	1	1	0	1
<code>key_decode</code>		0	0	1	0	0
<code>~key_decode</code>	and	1	1	0	1	1
		0	1	0	0	1

# Remark

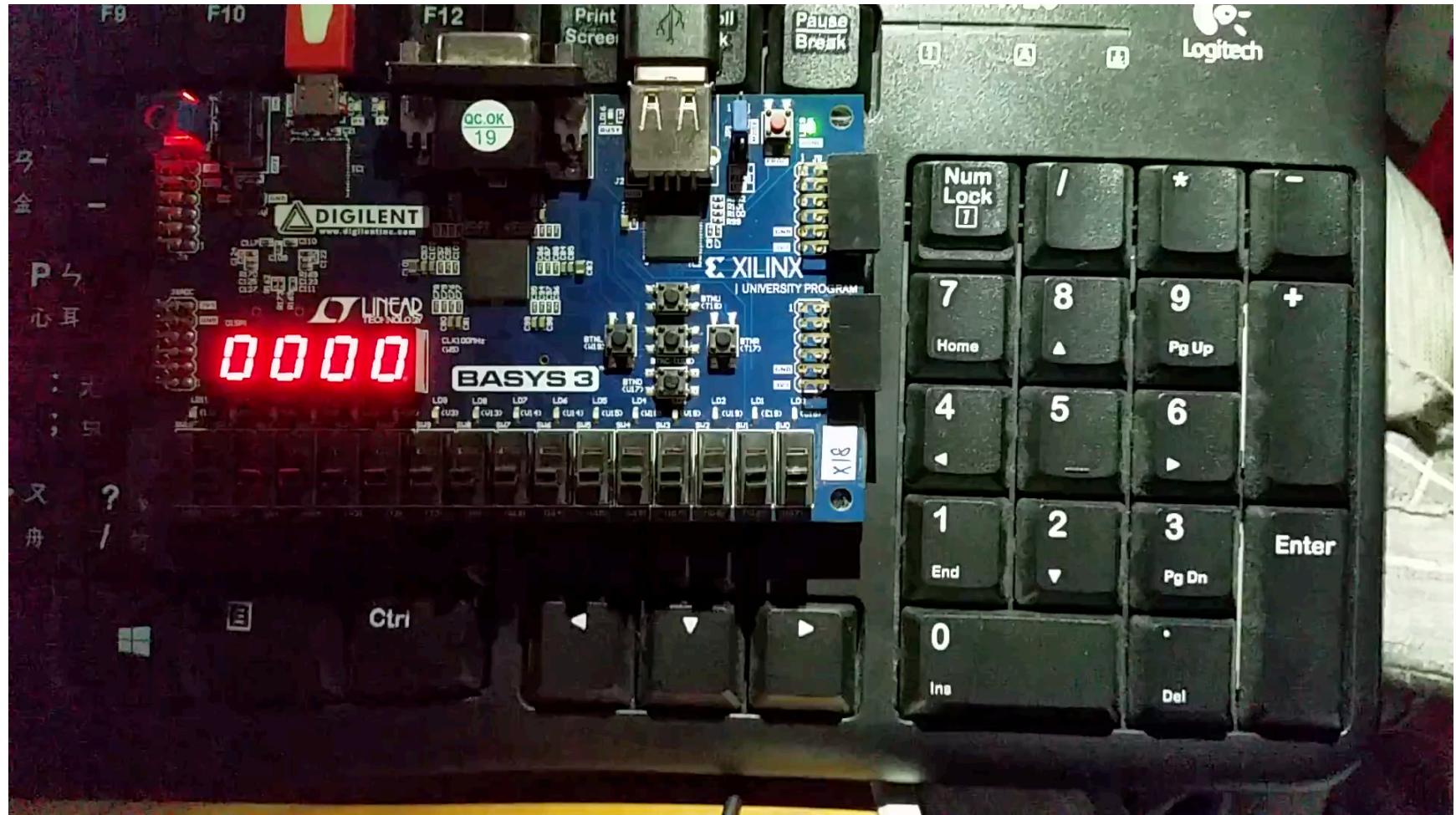
- ◉ You don't really need an array of 512 bits to track the keyboard status
  - ◆ Especially when you only need a few keys in the design
  - ◆  $512 + 512 = 1024$  for key\_down and key\_decode!!
- ◉ You may optimize this scheme to save resource on FPGA

# Verilog Module: SampleDisplay

- In Keyboard Sample Code
  - ◆ KeyboardDecoder.v
  - ◆ SampleDisplay.v
- I/O for KeyboardDecoder

Output	Input
<ul style="list-style-type: none"><li>• display</li><li>• digit</li></ul>	<ul style="list-style-type: none"><li>• PS2_CLK</li><li>• PS2_DATA</li><li>• rst</li><li>• clk</li></ul>

# Sample Display with Keyboard



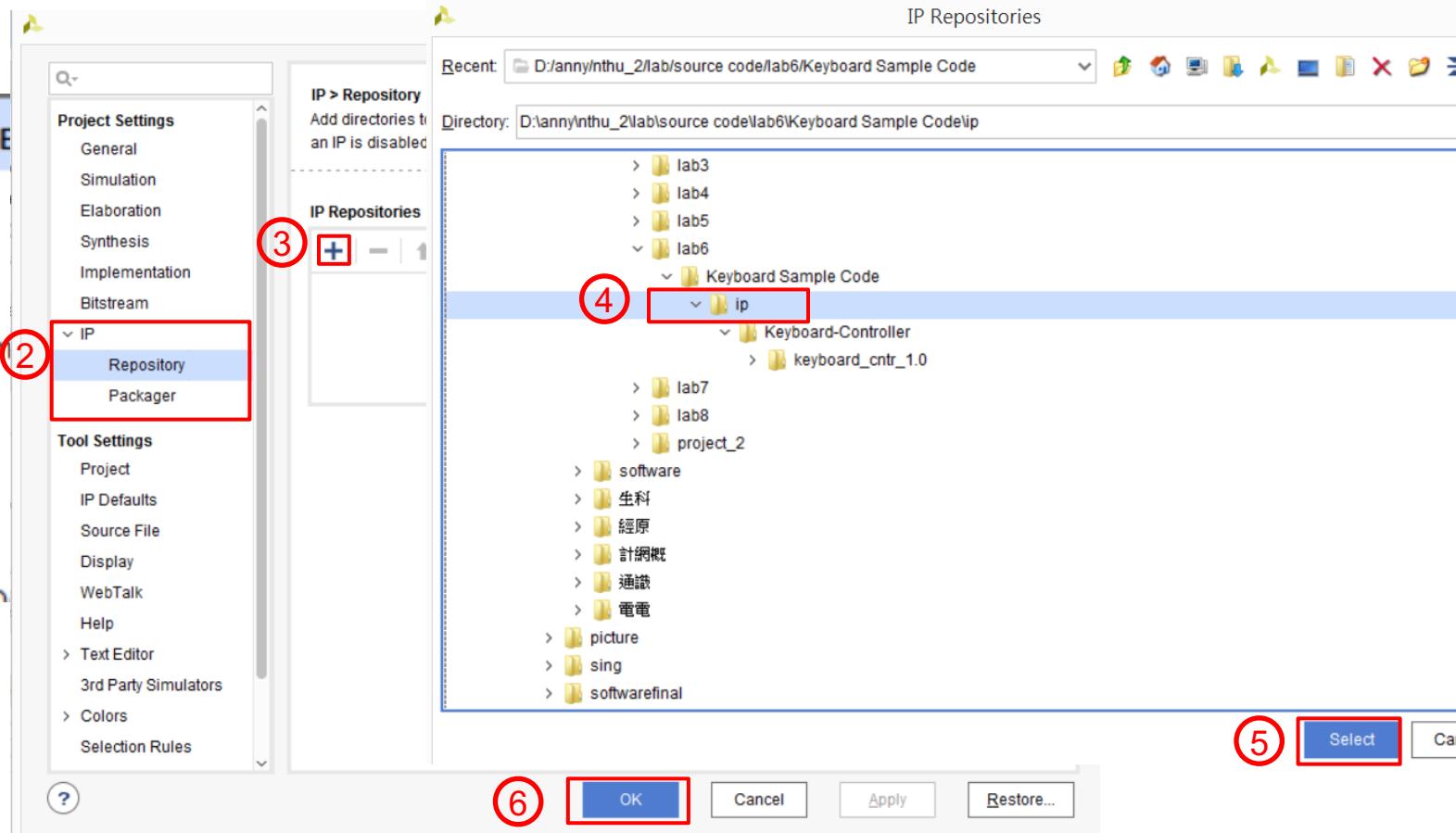
# Keyboard Controller IP from Vivado

# Remarks

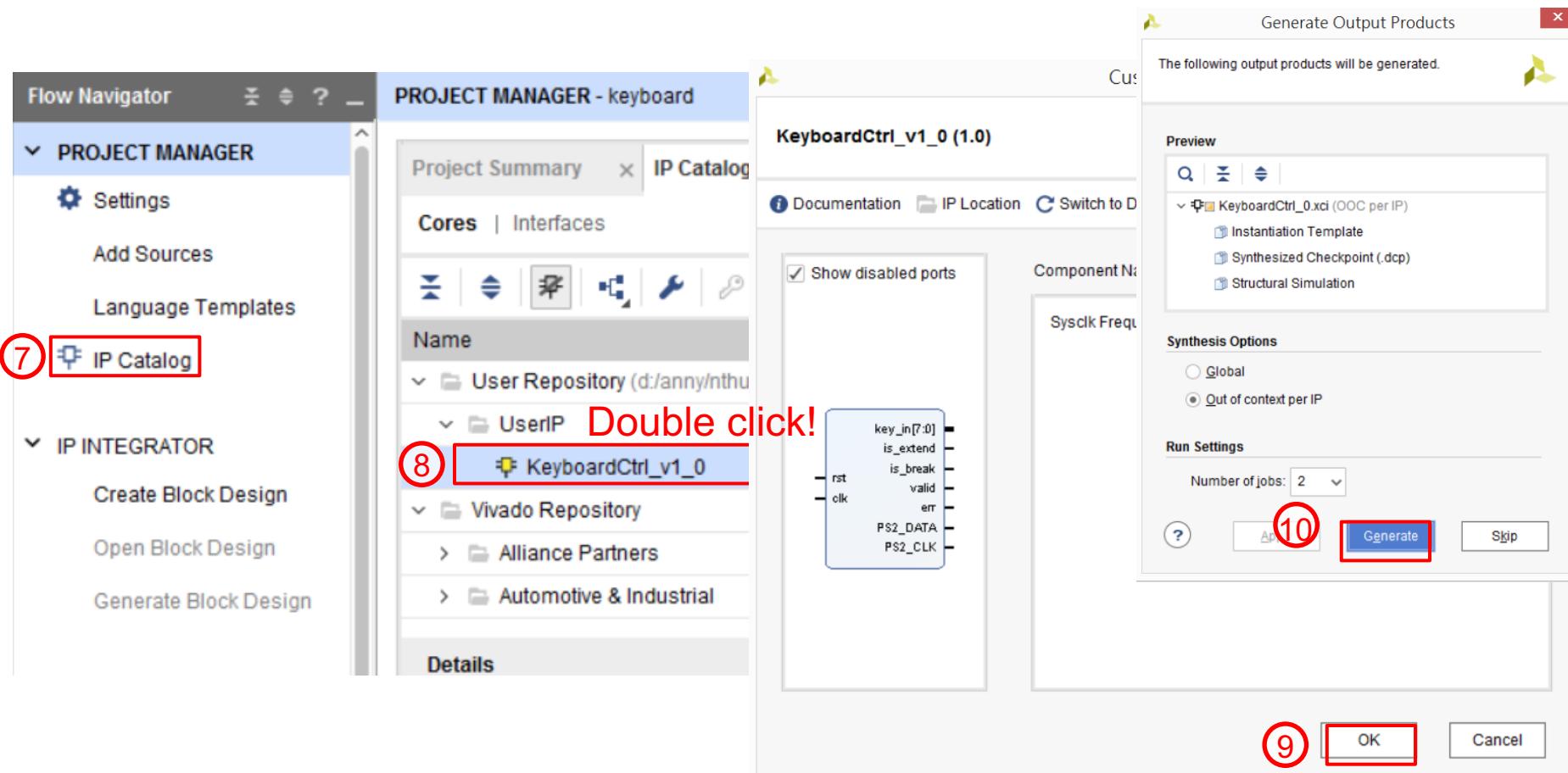
- IP (**Intellectual Property**)
  - ◆ Refer to a software (source / library) or a hardware design (also in its high-level description form)
- Vivado supports a large variety of IP blocks that you can use
  - ◆ Verilog RTL blocks
- You may use the follow steps to add the keyboard controller IP, or
  - ◆ Simply integrate the Verilog source we provide

# How to Use IP (1/3)

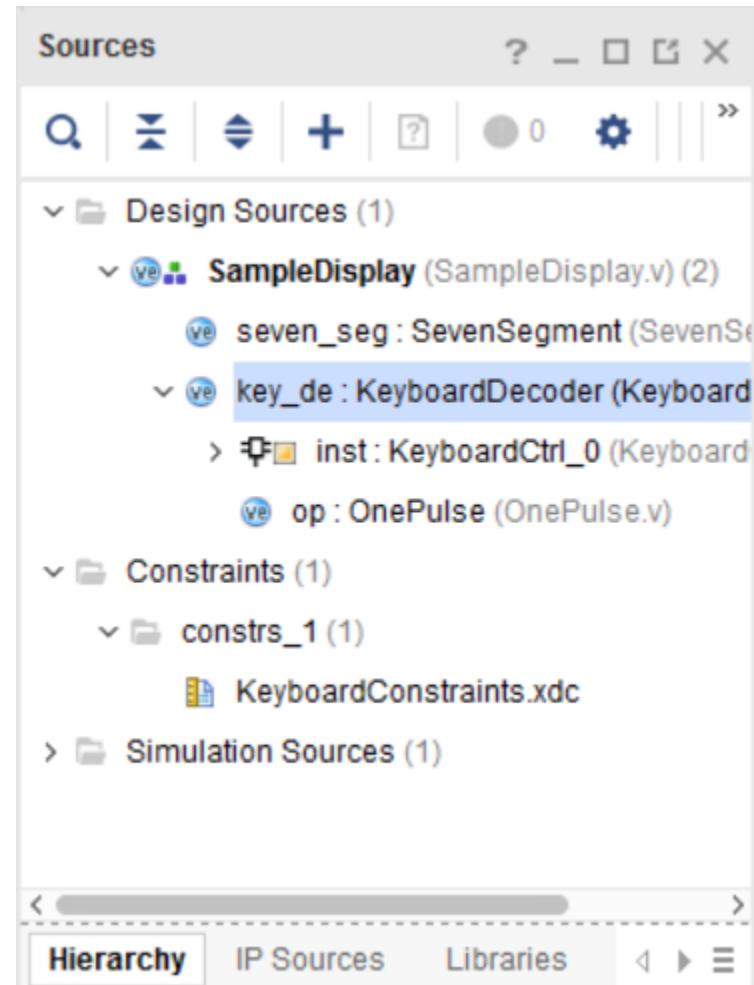
\*The UI may vary slightly in different versions



# How to Use IP (2/3)



# How to Use IP (3/3)



# SystemVerilog Parameter Style

- Vivado will show an error when in Verilog
- But since it supports SystemVerilog too, we still can use this style.
  - ◆ And we do!

```
parameter [8:0] KEY_CODES [0:19] = {  
    9'b0_0100_0101, // 0 => 4F  
    9'b0_0001_0110, // 1 => 1C  
    9'b0_0001_1110, // 2 => 1E  
    9'b0_0010_0110, // 3 => 26  
    9'b0_0010_0101, // 4 => 25  
    9'b0_0010_1110, // 5 => 2E  
    9'b0_0011_0110, // 6 => 36  
    9'b0_0011_1101, // 7 => 3D  
    9'b0_0011_1110, // 8 => 3E
```

Error: parameter with unpacked dimension is only allowed in SystemVerilog  
Error: Cannot assign a packed type to an unpacked type