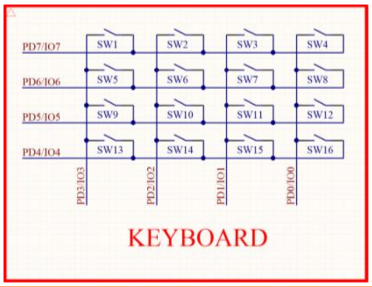
Sensors & Microsystem Electronics: microcontrollers

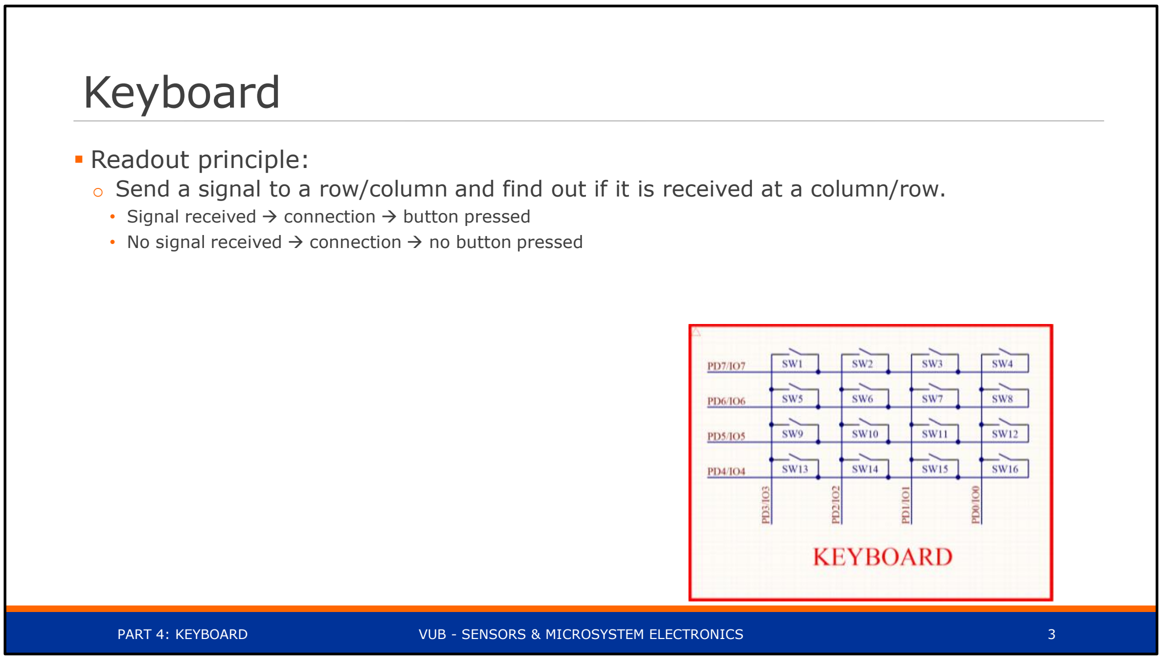
PART 4: KEYBOARD

Keyboard

- 4 x 4 matrix of pushbuttons
- 4 rows and 4 columns connected to I/O port D
- Button pressed → connection between corresponding row and column
  - 1 column can connect to 4 rows
  - 1 row can connect to 4 columns

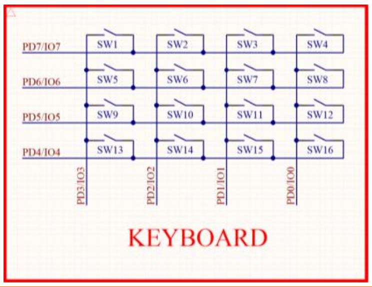


The keyboard is made up of a matrix of 4 by 4 buttons. These are connected in parallel in the rows and columns as shown on the schematic. Each of the rows and columns are connected to a pin of I/O port D. Pressing a button leads to a connection between a certain row and column pin.



Keyboard

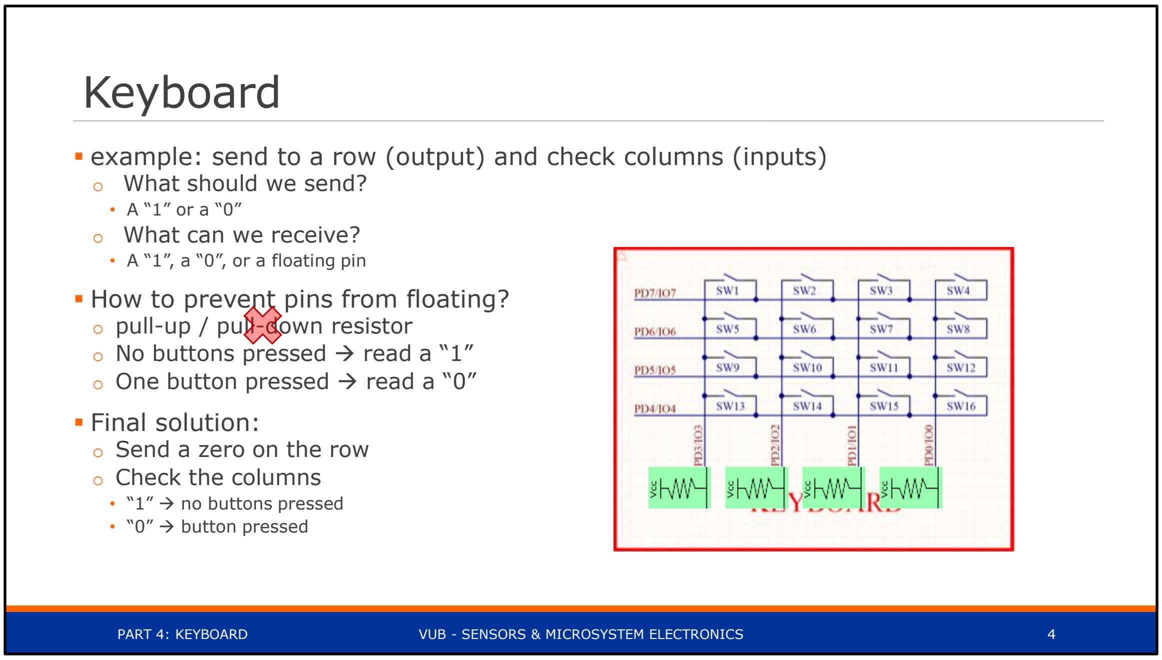
- Readout principle:
  - Send a signal to a row/column and find out if it is received at a column/row.
    - Signal received → connection → button pressed
    - No signal received → connection → no button pressed



PART 4: KEYBOARD VUB - SENSORS & MICROSYSTEM ELECTRONICS 3

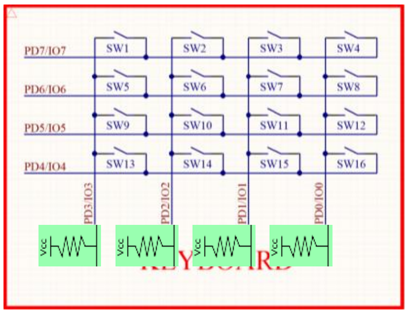
The idea in reading out the keyboard is as follows:

A signal is sent on a row, and by checking if that signal is received on any of the columns we can see whether a button on that column is pressed. If the signal is not received on any of the columns, no button is pressed.



Keyboard

- example: send to a row (output) and check columns (inputs)
  - What should we send?
    - A "1" or a "0"
  - What can we receive?
    - A "1", a "0", or a floating pin
- How to prevent pins from floating?
  - pull-up / ~~pull-down~~ resistor
  - No buttons pressed → read a "1"
  - One button pressed → read a "0"
- Final solution:
  - Send a zero on the row
  - Check the columns
    - "1" → no buttons pressed
    - "0" → button pressed



PART 4: KEYBOARD VUB - SENSORS & MICROSYSTEM ELECTRONICS 4

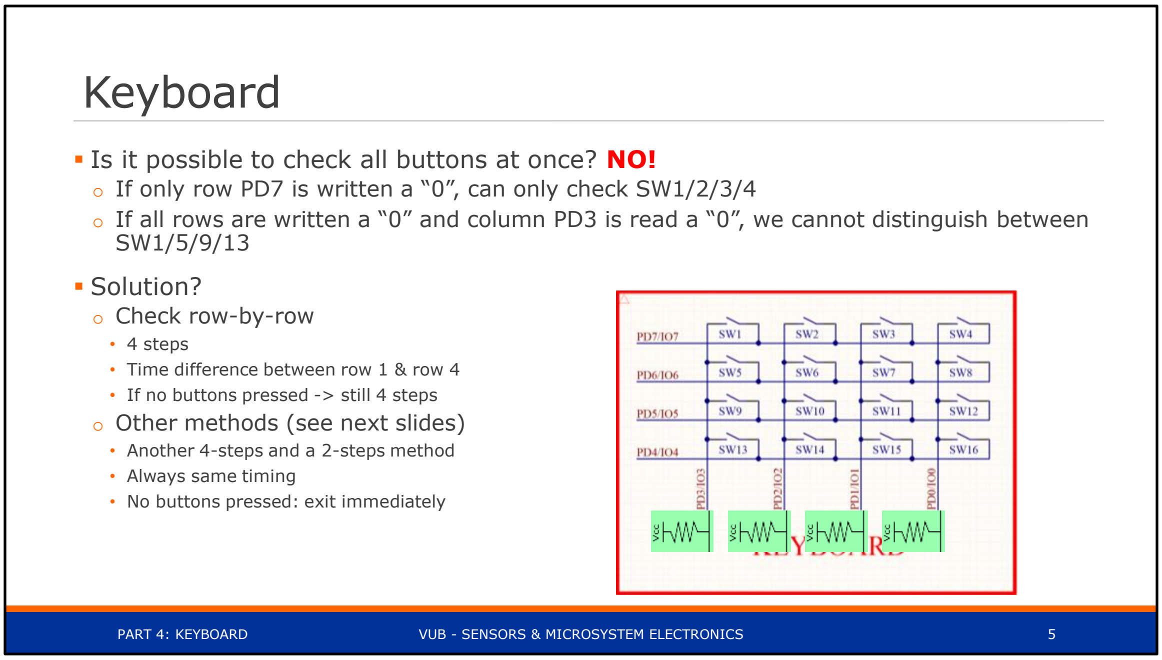
For this example we are going to assume that the rows are outputs and the columns are inputs

What can we send on a digital pin: **1 or 0**
What can we receive: **1 or 0 or floating**

Floating pins is something we want to avoid, so similar to the first presentation, we can avoid floating pins by enabling a pull-up resistor. This is what we are going to do on the input pins.

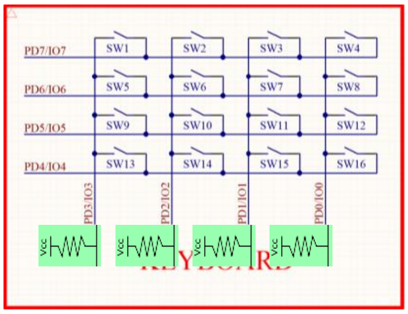
Because when no button is pressed, the columns are pulled to 1 by the pull up resistor, we need a signal that is distinguishable when a row is connected to a column when a button is pressed. The only possibility we have is that we have to send a 0.

So when we read a 1 on the input, no button is pressed, if we read a 0, we know that a button is pressed.



Keyboard

- Is it possible to check all buttons at once? **NO!**
  - If only row PD7 is written a "0", can only check SW1/2/3/4
  - If all rows are written a "0" and column PD3 is read a "0", we cannot distinguish between SW1/5/9/13
- Solution?
  - Check row-by-row
    - 4 steps
    - Time difference between row 1 & row 4
    - If no buttons pressed -> still 4 steps
  - Other methods (see next slides)
    - Another 4-steps and a 2-steps method
    - Always same timing
    - No buttons pressed: exit immediately

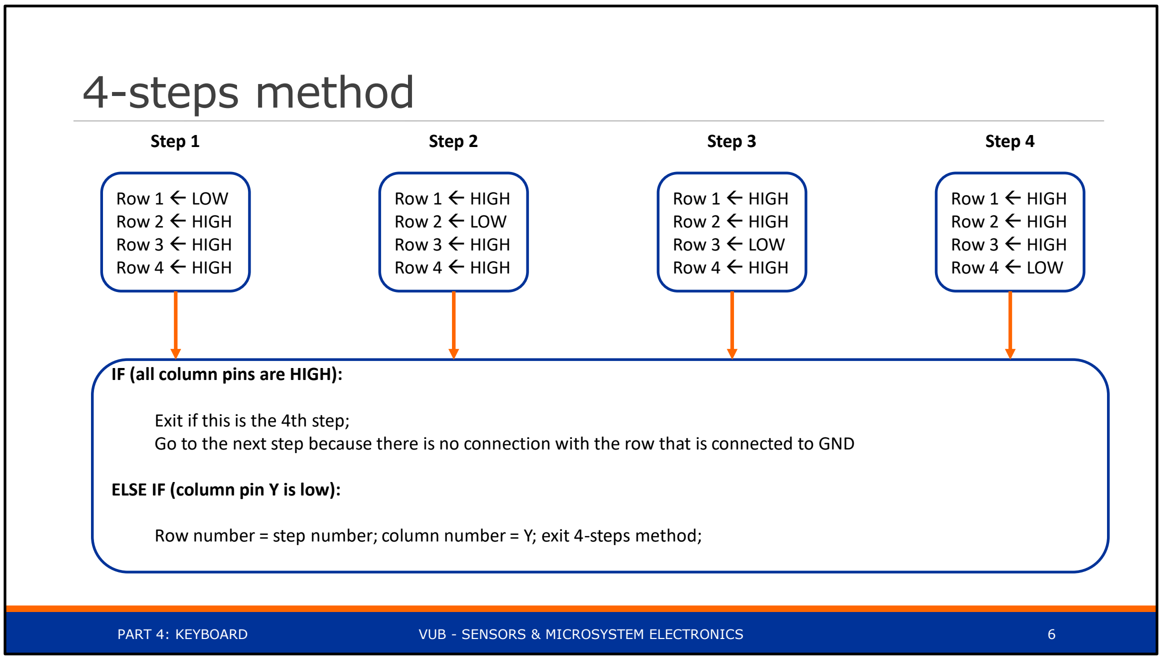


We cannot check all buttons at once, because we cannot distinguish between buttons on the same column at the same time.

The way around this is trough multiplexing. In this technique we are going to read the buttons row by row. By activating a single row, and reading the columns, we can be sure that if a button press is detected that is is on the active row.

By iterating over all 4 rows, we can get the state of all the buttons, but there is always a small time difference between the first and the last row.

There are other methods that have always the same timing regardless of the row, and can also have immidiate exit if no button is pressed.



4-steps method

```
IF (all column pins are HIGH):

    Exit if this is the 4th step;
    Go to the next step because there is no connection with the row that is connected to GND

ELSE IF (column pin Y is low):

    Row number = step number; column number = Y; exit 4-steps method;
```

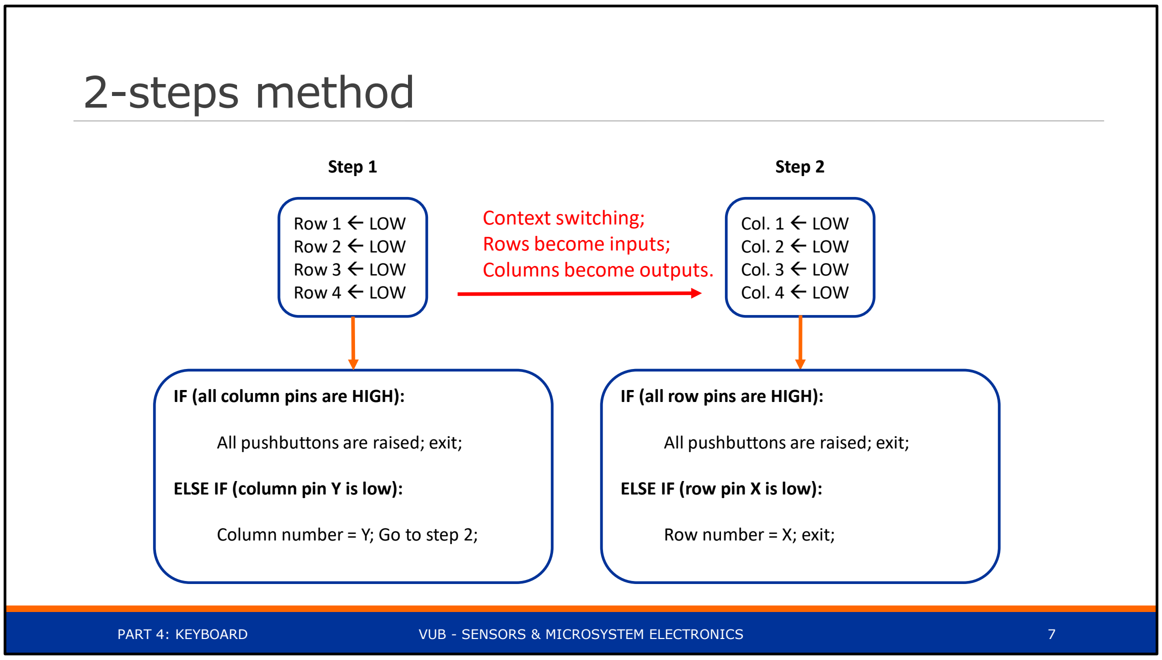
PART 4: KEYBOARD VUB - SENSORS & MICROSYSTEM ELECTRONICS 6

The 4 steps method is the simplest to implement. The example is given here with some pseudo code to explain the principle.

In step 1, row 1 is activated by setting it to a low value, and the others are set high to deactivate them. Now we check the value of all the columns, if one of the columns is low a button press is detected. By checking which pin is low, we know which column the pressed button is on, and we can execute a specific action.

Then we go to the next step and disable row 1,3 and 4 and we enable row 2. Now we do the same thing.

And we repeat this principle untill all rows are checked.



2-steps method

```
IF (all column pins are HIGH):
    All pushbuttons are raised; exit;
ELSE IF (column pin Y is low):
    Column number = Y; Go to step 2;
```

```
IF (all row pins are HIGH):
    All pushbuttons are raised; exit;
ELSE IF (row pin X is low):
    Row number = X; exit;
```

The most time efficient method is the 2 steps method, but it is somewhat harder to implement.

In this method, we first set all rows to low. Now we check in which column a button is pressed, and store this somehow and go to step 2. If no column is low, we simply exit cause no button is pressed.

In step 2, we first switch the function of the row and column: we make the rows the input with a pull-up, and we make the columns the outputs. Now we do the same as in step one, we check which row is low. Now we have the rownumber of the pressed button.

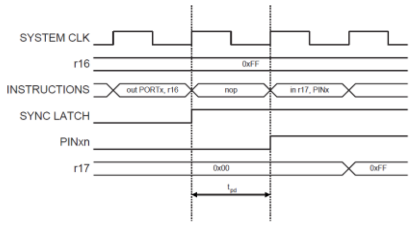
From step 1 we have the index of the column that contains the pressed button, and from step 2 we have the index of the row of the button. Now we can execute the correct function based on this coordinate.

Synchronisation

- Bits in PINx register are always one cycle behind
- Add NOP when changing output connected to an input.

When reading back a software assigned pin value, a nop instruction must be inserted as indicated in Figure 14-4. The out instruction sets the "SYNC LATCH" signal at the positive edge of the clock. In this case, the delay tpd through the synchronizer is 1 system clock period.

Figure 14-4. Synchronization when Reading a Software Assigned Pin Value



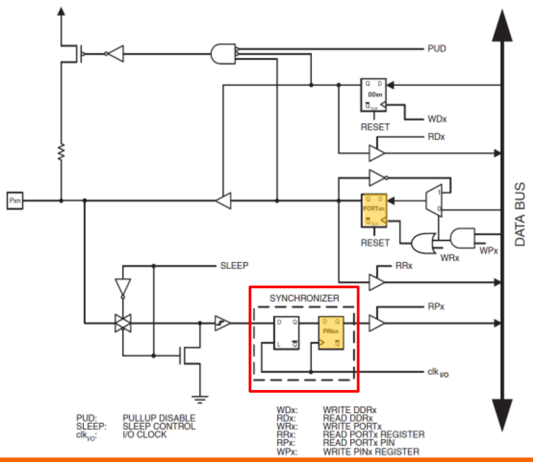
The following code example shows how to set port B pins 0 and 1 high, 2 and 3 low, and define the port pins from 4 to 7 as input with pull-ups assigned to port pins 6 and 7. The resulting pin values are read back again, but as previously discussed, a nop instruction is included to be able to read back the value recently assigned to some of the pins.

The PIN register has a synchronizer that makes the value in the register always 1 clockcycle late. Due to this, if a readback is done of an input that is related to an output that has changed its value in the previous cycle, the bit will not yet be updated in the PIN register.

This issue can be solved by putting a nop before reading the PIN register if the previous instruction was to update the output it is connected to. This way, you are sure that the pin register has the latest value you would assume it to have.

Logical pin interface

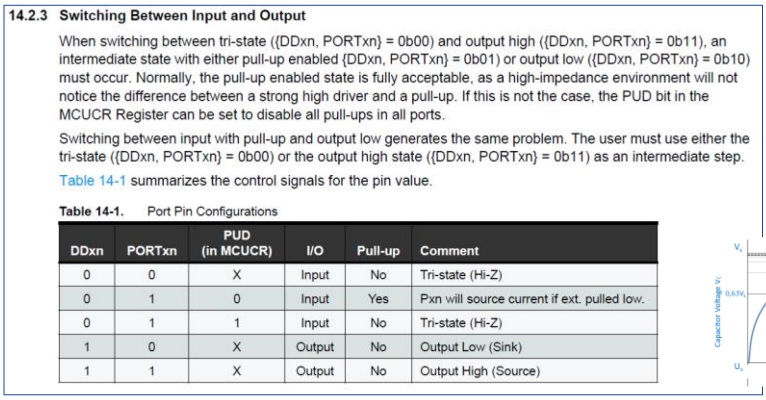
Figure 14-2. General Digital I/O[1]



The image on the right shows the internal structure of a single I/O pin. In red three registers are indicated. We have a PIN, PORT and DDR register.

Keyboard

- Intermediate pin states **Important for 2-steps method**



14.2.3 Switching Between Input and Output

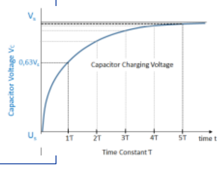
When switching between tri-state ({DDxn, PORTxn} = 0b00) and output high ({DDxn, PORTxn} = 0b11), an intermediate state with either pull-up enabled {DDxn, PORTxn} = 0b01) or output low ({DDxn, PORTxn} = 0b10) must occur. Normally, the pull-up enabled state is fully acceptable, as a high-impedance environment will not notice the difference between a strong high driver and a pull-up. If this is not the case, the PUD bit in the MCUCR Register can be set to disable all pull-ups in all ports.

Switching between input with pull-up and output low generates the same problem. The user must use either the tri-state ({DDxn, PORTxn} = 0b00) or the output high state ({DDxn, PORTxn} = 0b11) as an intermediate step.

Table 14-1 summarizes the control signals for the pin value.

Table 14-1. Port Pin Configurations

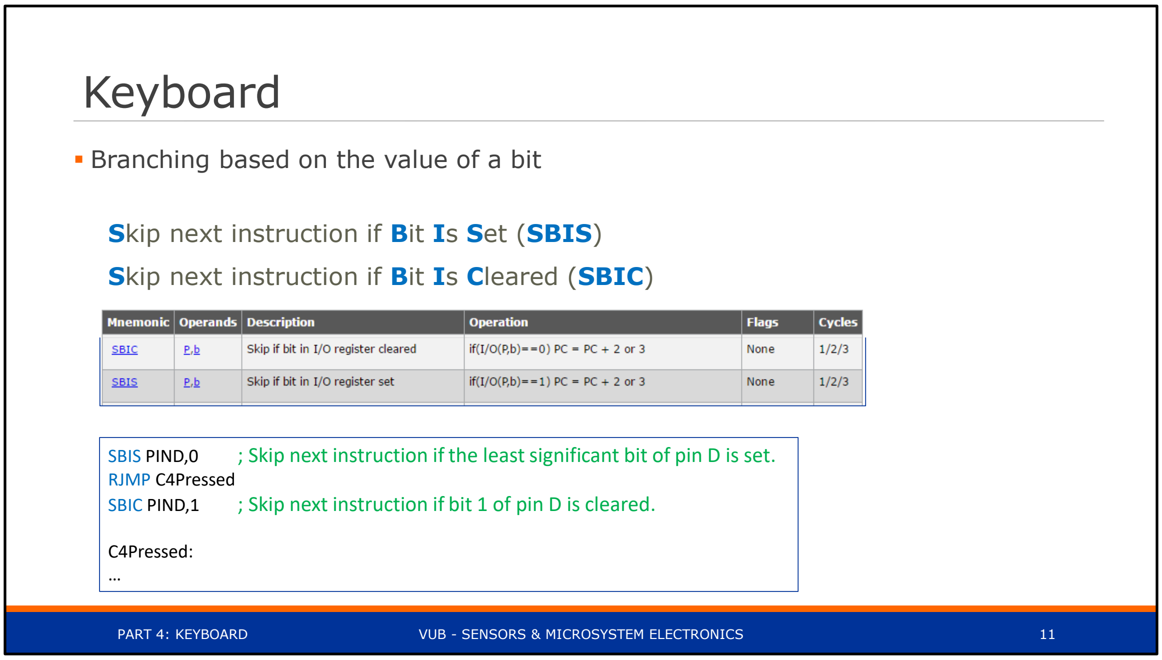
DDxn	PORTxn	PUD (in MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)



In the two steps method, the inputs and outputs are switched. The inputs have a pull up, and the outputs need to be set low during the context switch. This is done in at least two instructions (write to DDRD and write to PORTD). This means the pins have an intermediate state as described above (either input with no pull up, or output that is high.)

If this is done in the wrong order, some of the transistor gates for the input of the microcontrollers are charged slowly trough the pullup resistors. This intermediate state can get unknown states and incorrect/fake button presses can be detected

In case of the 2 steps method the correct way to avoid this is to do the context switch in the following order: A LOW output that needs to go to an INPUT with PULLUP, pass trough the HIGH output state. In other words, first modify PORTD, then modify the DDRD register.



Keyboard

- Branching based on the value of a bit

Skip next instruction if **B**it **I**s **S**et (**SBIS**)

Skip next instruction if **B**it **I**s **C**leared (**SBIC**)

Mnemonic	Operands	Description	Operation	Flags	Cycles
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3

```
SBIS PIND,0       ; Skip next instruction if the least significant bit of pin D is set.
RJMP C4Pressed
SBIC PIND,1       ; Skip next instruction if bit 1 of pin D is cleared.

C4Pressed:
…
```

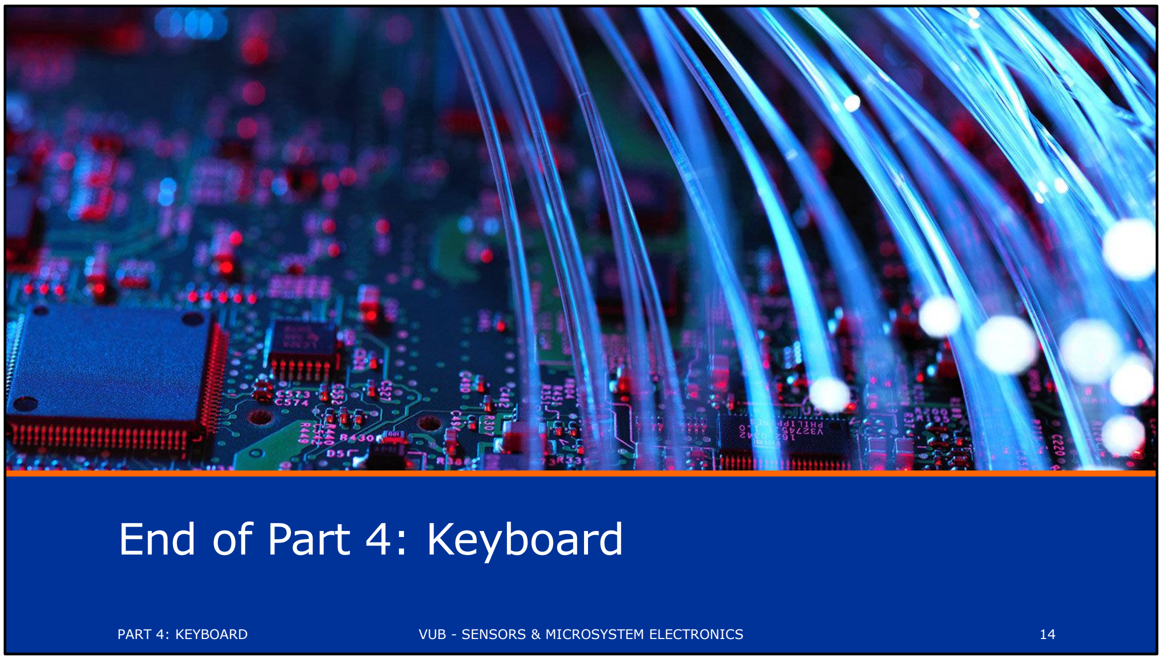
How to branch based on a value of a bit:

Use the SBIS and SBIC instruction. This instruction skips the next instruction based on the value of the bit that is tested. See example.

Task 7

Readout the keyboard

- Button 7: Two leds on
- Button 8: Bottom led on
- Button 4: Top Led on
- All other buttons: Buzzer on
- No buttons pressed: Leds and buzzer off



End of Part 4: Keyboard