Physical Computing

2007-2008

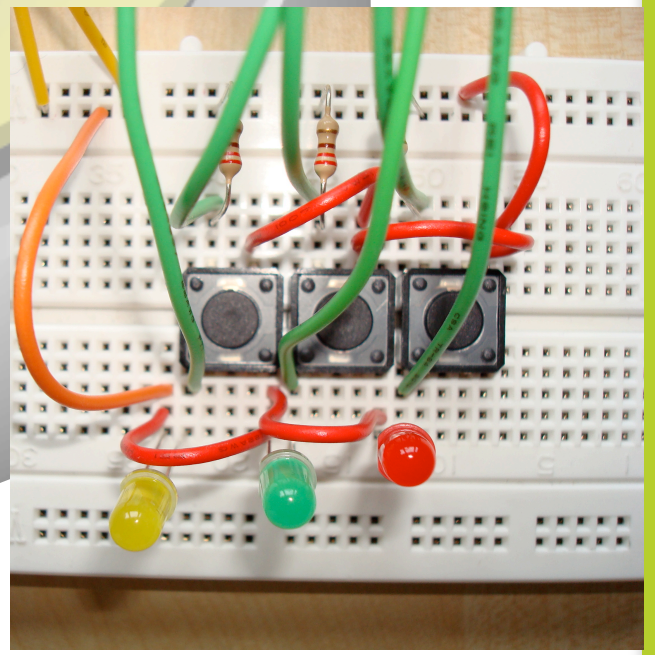
Interpretatie I
Programmeerproject I

Coen De Roover - Christophe Scholliers - Yves Vandriessche

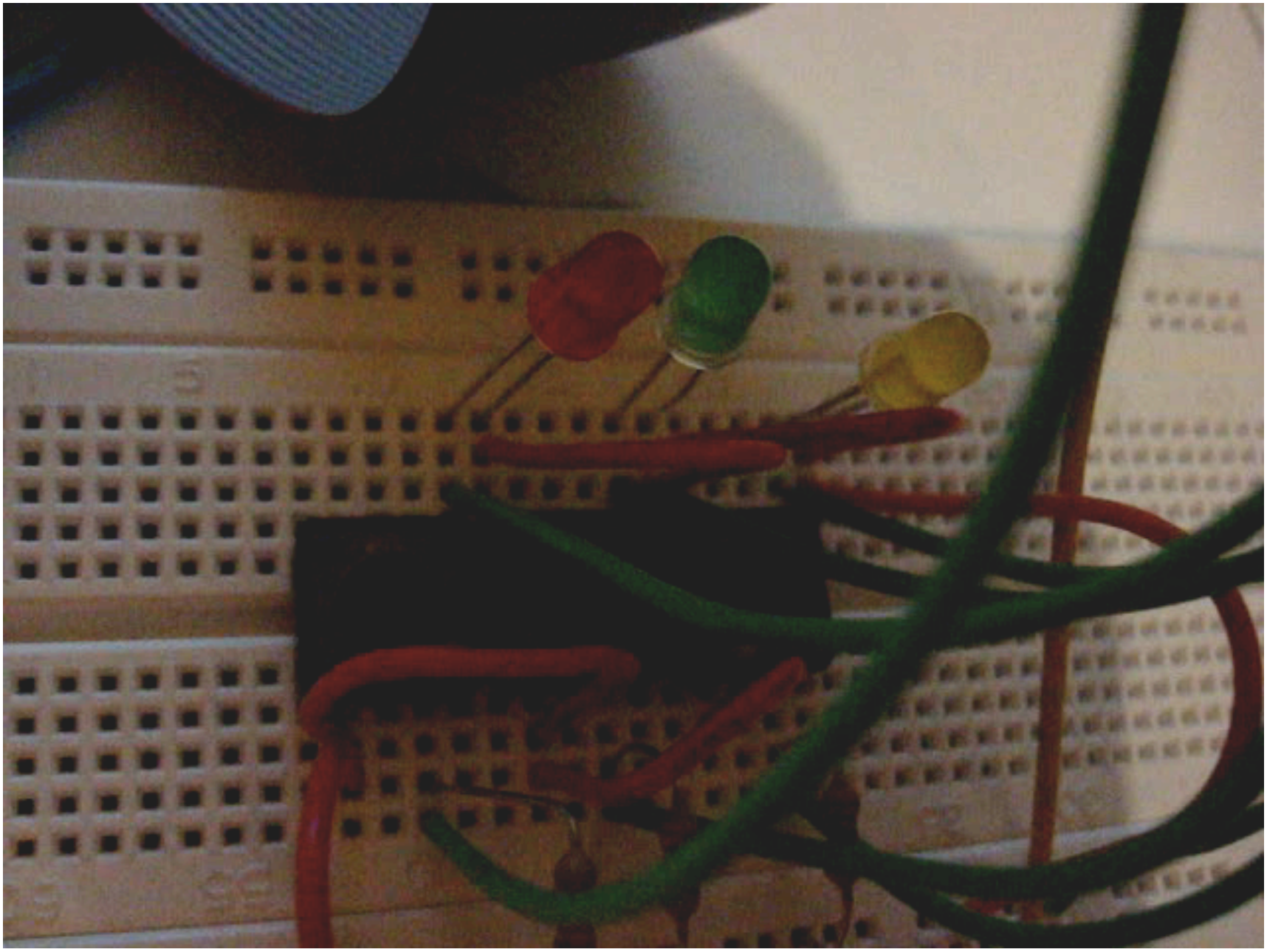
Programming Technology Lab
Vrije Universiteit Brussel
Belgium

Simon

http://www.neave.com/games/simon/game.php



Simon



Essential µC-specific Scheme extensions

binary operations

binary literals

base to format 32-bit number in

```
(number->string (logior #b1100 #b1010) 2) ; -> "00000000000000000000000000001110"
(number->string (logxor #b1100 #b1010) 2) ; -> "00000000000000000000000000000110"
(number->string (logand #b1100 #b1010) 2) ; -> "00000000000000000000000000001000"
(number->string (lognot #b1100) 2)        ; -> "11111111111111111111111111110011"
(number->string (ash #b1100  4) 2)        ; -> "00000000000000000000000011000000"
(number->string (ash #b1100 -2) 2)        ; -> "00000000000000000000000000000011"
```

arithmetic shift

direction

Essential µC-specific Scheme extensions

reading and writing to and from memory

optional arguments

```
(read memory-location offset number-of-bytes)
(write value memory-location offset number-of-bytes)
```

base address — added to base address

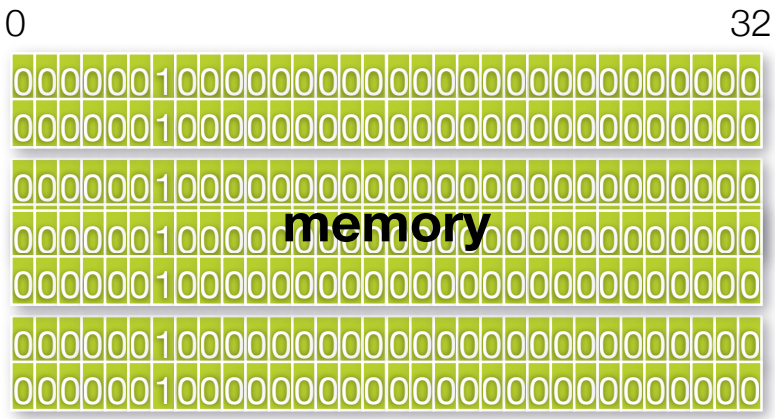
example

(read #xE0028008 0) **=**

(define BASE #xE0028000)
(read BASE #x08)

memory-location → #xE0028000

offset #x08 { #xE0028004, #xE0028008



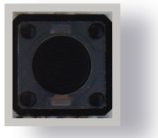
Controlling Pins on Ports

base addresses of port

```
(define GPIO_0 #xE0028000) ; IO0 port
(define GPIO_1 #xE0028010) ; IO1 port
```

pin controlling registers

```
(define IOPIN #x00)  ; IOxPIN register offset
(define IOSET #x04)  ; IOxSET register offset
(define IODIR #x08)  ; IOxDIR register offset
(define IOCLR #x0C)  ; IOxCLR register offset
```



addresses and offsets can be found in the data sheet

(cfr. References on last slide)

Table 68: GPIO Register Map

Generic Name	Description	Access	Reset Value	PORT0 Address & Name
IOPIN	GPIO Port Pin value register. The current state of the GPIO configured port pins can always be read from this register, regardless of pin direction and mode. Activity on non-GPIO configured pins will not be reflected in this register.	Read Only	NA	0xE0028000 IO0PIN
IOSET	GPIO Port Output set register. This register controls the state of output pins in conjunction with the IOCLR register. Writing ones produces highs at the corresponding port pins. Writing zeroes has no effect.	Read/ Write	0x0000 0000	0xE0028004 IO0SET
IODIR	GPIO Port Direction control register. This register individually controls the direction of each port pin.	Read/ Write	0x0000 0000	0xE0028008 IO0DIR
IOCLR	GPIO Port Output clear register. This register controls the state of output pins. Writing ones produces lows at the corresponding port pins and clears the corresponding bits in the IOSET register. Writing zeroes has no effect.	Write Only	0x0000 0000	0xE002800C IO0CLR

Configuring Pin Direction

Table 72: GPIO Direction Register (IO0DIR - 0xE0028008, IO1DIR - 0xE0028018, IO2DIR - 0xE0028028, IO3DIR - 0xE0028038)

IODIR	Description	Value after Reset
31:0	Direction control bits (0 = INPUT, 1 = OUTPUT). Bit 0 in IO0DIR controls P0.0 ... Bit 31 in IO0DIR controls P0.31	0

Setting P0.06 to output:

```
(define pin6 (ash 1 6))
(write pin6 GPIO_0 IODIR)
```

Everything else is input now!!

Setting P0.06 respecting previous settings: logical or

```
(define previous-dir (read port IODIR))
(write (logior previous-dir (ash 1 6)) GPIO_0 IODIR)
```

Let's abstract this away into a function:

```
(define (set-dir-output! port pin)
  (write (logior (read port IODIR) pin)
         port IODIR))
```

Producing High Value On Pin

Table 70: GPIO Output Set Register (IO0SET - 0xE0028004, IO1SET - 0xE0028014, IO2SET - 0xE0028024, IO3SET - 0xE0028034)

IOSET	Description	Value after Reset
31:0	Output value SET bits. Bit 0 in IO0SET corresponds to P0.0 ... Bit 31 in IO0SET corresponds to P0.31	0

Setting P0.06 to high = writing 1 << 6 to

```
(write (ash 1 6) gpio0 IOSET)
```

Does not change other pins !!!

0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0

Let's abstract this away into a function:

```
(define (set-pin! port pin)
  (write pin port IOSET))
```

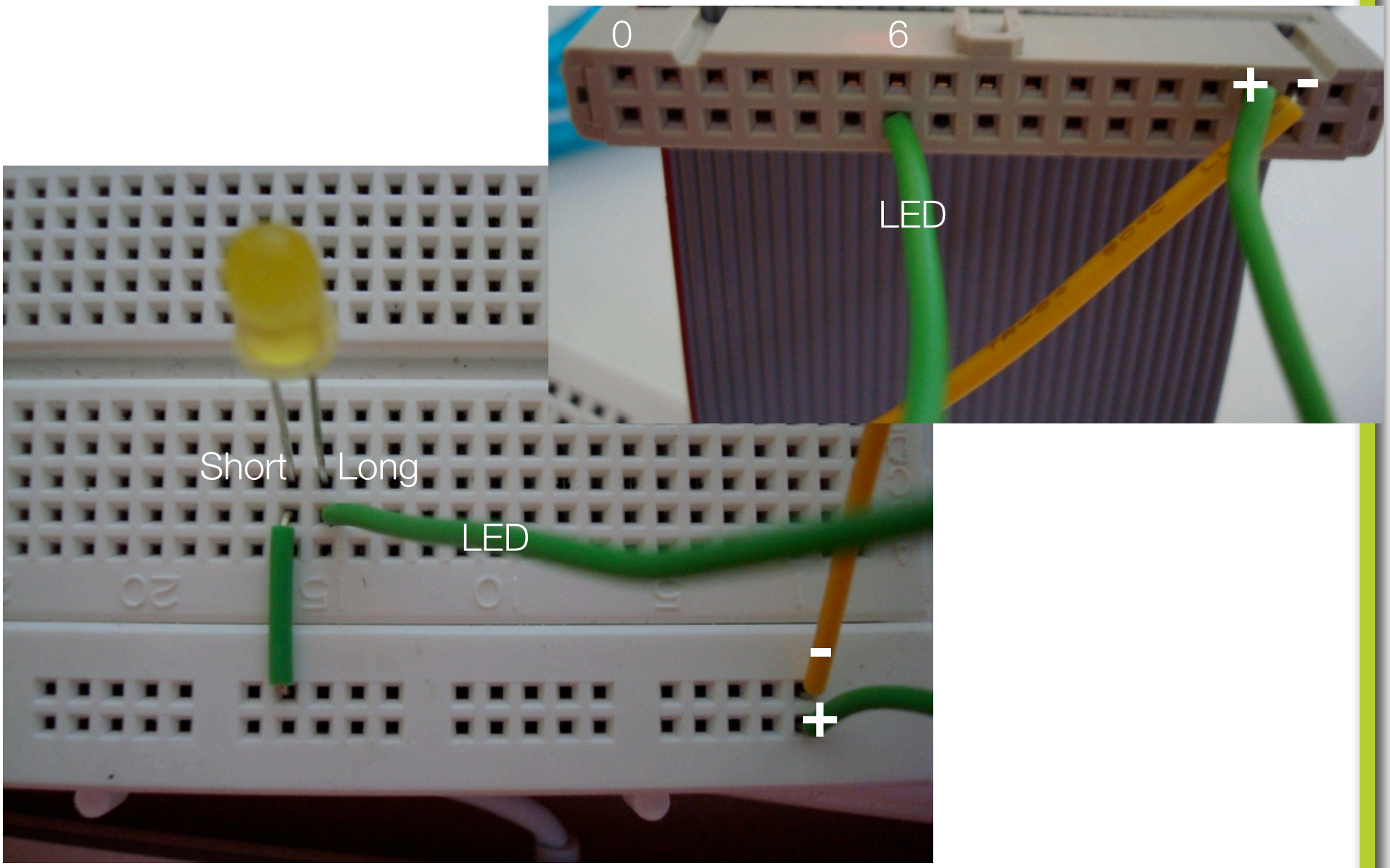
Setting a pin to low is very similar

```
(define (clear-pin! port pin)
  (write pin port IOCLR))

(clear-pin! gpio0(ash 1 6))
```

0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0

Connecting the led



Exercise: Blinking LED

What do we need to do:

1. set target pin direction
2. set pin to high
3. wait for a while
4. set pin to low
5. wait for a while
6. repeat from 2

http://prog.vub.ac.be/~cderoove/project/blinking_led.scm

Exercise: Blinking LED

What do we need to do:

1. set target pin direction
2. set pin to high
3. wait for a while
4. set pin to low
5. wait for a while
6. repeat from 2

```
(define led-pin (ash 1 6))
```

```
(set-dir-output! GPIO_0 led-pin)
```

http://prog.vub.ac.be/~cderoove/project/blinking_led.scm

Exercise: Blinking LED

What do we need to do:

1. set target pin direction
2. set pin to high
3. wait for a while
4. set pin to low
5. wait for a while
6. repeat from 2

```
(define led-pin (ash 1 6))
```

```
(set-dir-output! GPIO_0 led-pin)
```

```
(set-pin! GPIO_0 led-pin)
```

http://prog.vub.ac.be/~cderoove/project/blinking_led.scm

Exercise: Blinking LED

What do we need to do:

1. set target pin direction
2. set pin to high
3. wait for a while
4. set pin to low
5. wait for a while
6. repeat from 2

```
(define led-pin (ash 1 6))
```

```
(set-dir-output! GPIO_0 led-pin)
```

```
(set-pin! GPIO_0 led-pin)
```

```
(clear-pin! GPIO_0 led-pin)
```

http://prog.vub.ac.be/~cderoove/project/blinking_led.scm

Exercise: Blinking LED

What do we need to do:

1. set target pin direction
2. set pin to high
3. wait for a while
4. set pin to low
5. wait for a while
6. repeat from 2

```
(define led-pin (ash 1 6))
```

```
(set-dir-output! GPIO_0 led-pin)
```

```
(set-pin! GPIO_0 led-pin)
```

```
(clear-pin! GPIO_0 led-pin)
```

```
(wait-us 1000000)   ; this will wait for that
                    ; many µsecs, one second
```

http://prog.vub.ac.be/~cderoove/project/blinking_led.scm

Reading an input pin

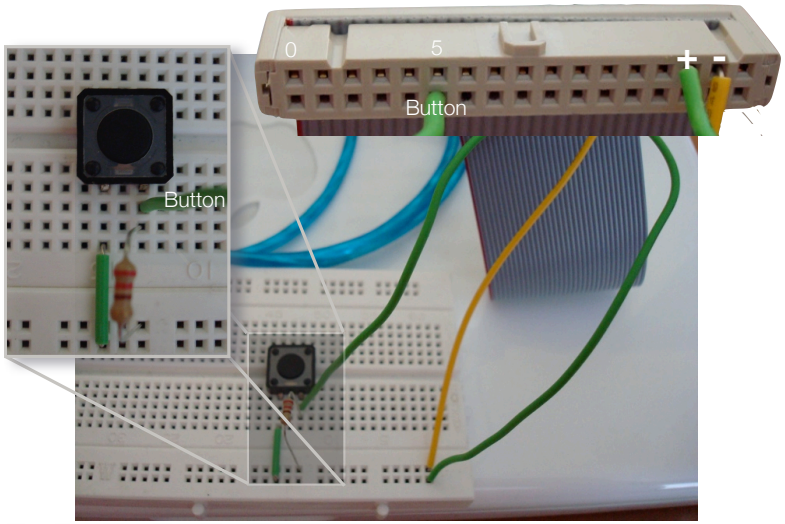
Generic Name	Description	Access	Reset Value	PORT0 Address & Name	PORT1 Address & Name	PORT2 Address & Name	PORT3 Address & Name
IOPIN	GPIO Port Pin value register. The current state of the GPIO configured port pins can always be read from this register, regardless of pin direction and mode. Activity on non-GPIO configured pins will not be reflected in this register.	Read Only	NA	0xE0028000 IO0PIN	0xE0028010 IO1PIN	0xE0028020 IO2PIN	0xE0028030 IO3PIN

Define IOPIN it has offset #x00

```
(define IOPIN #x00)
```

Reading a pin result in 0 or the pin value

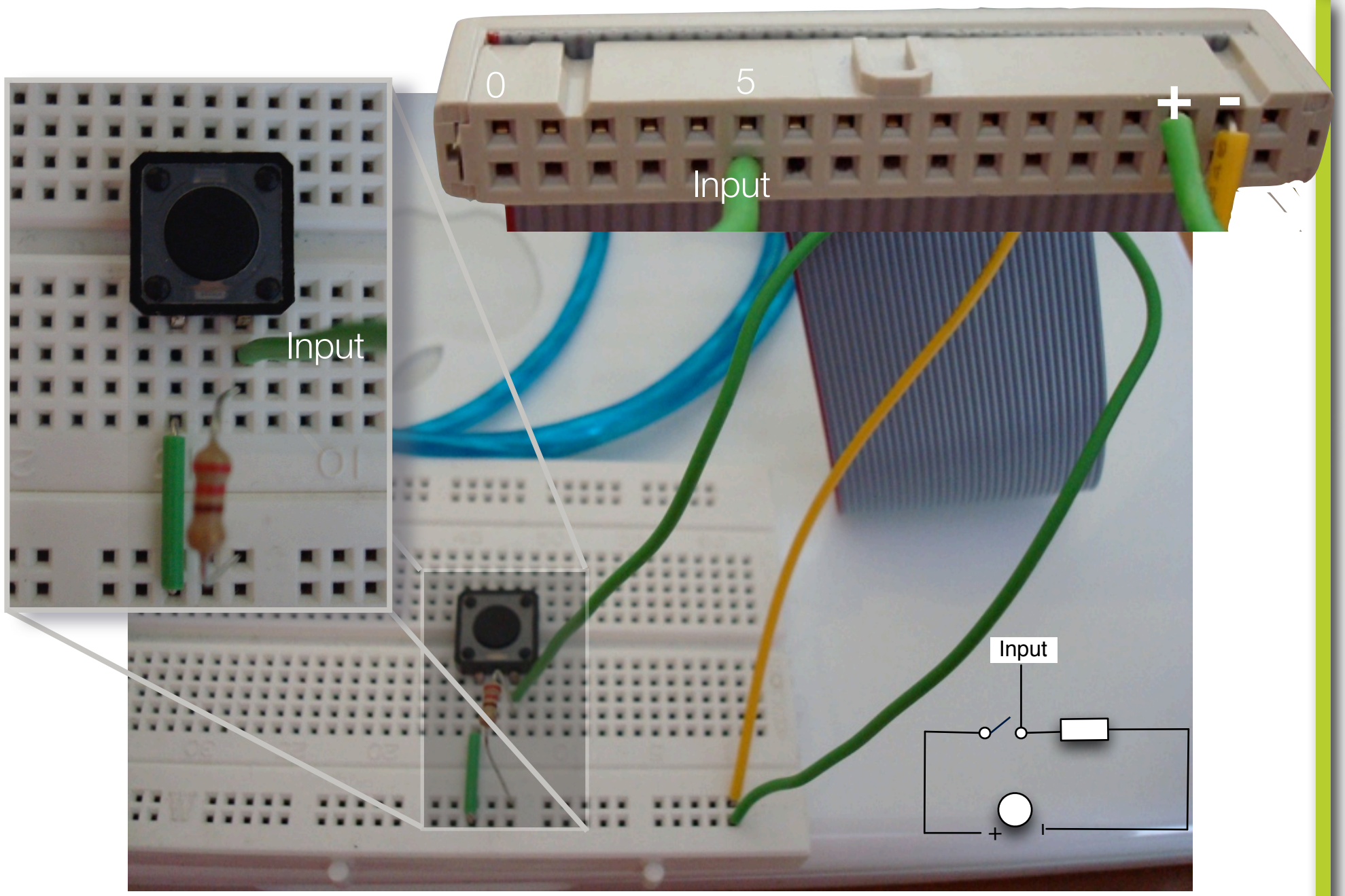
```
(define (read-pin port pin)
  (logand pin (read port IOPIN)))
```



IOPIN ~00001000000~00~0000000000000000

IOPIN ~00000100000~00~0000000000000000

Connecting a push button



Push button test

Register address definitions

```
(define GPIO_0 #xE0028000)
(define IODIR #x08)
(define IOSET #x04)
(define IOCLR #x0C)
(define IOPIN #x00)
```

Abstraction to read a pin

```
(define (read-pin port pin)
  (logand pin (read port IOPIN)))
```

The test function

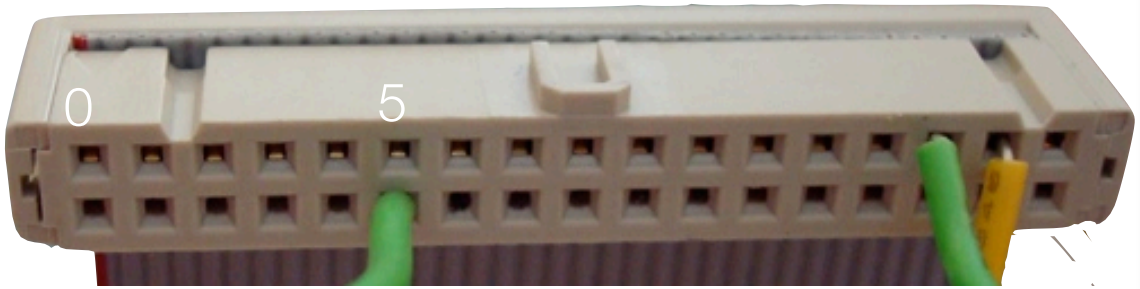
```
(define (test port pin)
  (if (= (read-pin port pin) 0)
      (test port pin)
      (display "Gedrukt")))
```

Calling it for pin 5

```
(test GPIO_0 (ash 1 5))
```

Table 68: GPIO Register Map

Generic Name	Description	Access	Reset Value	PORT0 Address & Name
IOPIN	GPIO Port Pin value register. The current state of the GPIO configured port pins can always be read from this register, regardless of pin direction and mode. Activity on non-GPIO configured pins will not be reflected in this register.	Read Only	NA	0xE0028000 IO0PIN
IOSET	GPIO Port Output set register. This register controls the state of output pins in conjunction with the IOCLR register. Writing ones produces highs at the corresponding port pins. Writing zeroes has no effect.	Read/ Write	0x0000 0000	0xE0028004 IO0SET
IODIR	GPIO Port Direction control register. This register individually controls the direction of each port pin.	Read/ Write	0x0000 0000	0xE0028008 IO0DIR
IOCLR	GPIO Port Output clear register. This register controls the state of output pins. Writing ones produces lows at the corresponding port pins and clears the corresponding bits in the IOSET register. Writing zeroes has no effect.	Write Only	0x0000 0000	0xE002800C IO0CLR



Push button test

Register address definitions

```
(define GPIO_0 #xE0028000)
(define IODIR #x08)
(define IOSET #x04)
(define IOCLR #x0C)
(define IOPIN #x00)
```

Table 68: GPIO Register Map

Generic Name	Description	Access	Reset Value	PORT0 Address & Name
IOPIN	GPIO Port Pin value register. The current state of the GPIO configured port pins can always be read from this register, regardless of pin direction and mode. Activity on non-GPIO configured pins will not be reflected in this register.	Read Only	NA	0xE0028000 IO0PIN
IOSET	GPIO Port Output set register. This register controls the state of output pins in conjunction with the IOCLR register. Writing ones produces highs at the corresponding port pins. Writing zeroes has no effect.	Read/ Write	0x0000 0000	0xE0028004 IO0SET
IODIR	GPIO Port Direction control register. This register individually controls the direction of each port pin.	Read/ Write	0x0000 0000	0xE0028008 IO0DIR
IOCLR	GPIO Port Output clear register. This register controls the state of output pins. Writing ones produces lows at the corresponding port pins and clears the corresponding bits in the IOSET register. Writing zeroes has no effect.	Write Only	0x0000 0000	0xE002800C IO0CLR

Abstraction to read a pin

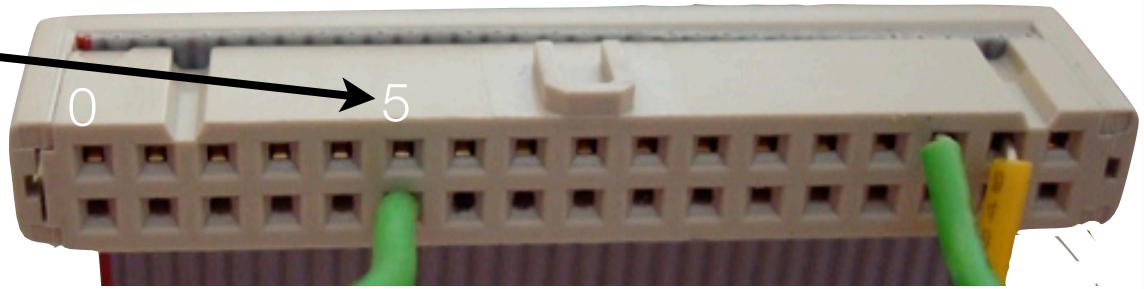
```
(define (read-pin port pin)
  (logand pin (read port IOPIN)))
```

The test function

```
(define (test port pin)
  (if (= (read-pin port pin) 0)
      (test port pin)
      (display "Gedrukt")))
```

Calling it for pin 5

```
(test GPIO_0 (ash 1 5))
```



Implementing Simon

Display the sequence

1. set led pins direction
2. take next of sequence or read (2)
3. set pin to high
4. wait for a while
5. set pin to low
6. wait for a while
7. repeat from 2

Leds 8, 9, 10

Read the sequence

1.set button pins direction
2.take next of sequence or display(2)
3. read until one button pushed
4. check button equals sequence
5. repeat from 2

Buttons 5, 6, 7

Our implementation 94 lines of scheme

This is only a hint

References

Datasheet / User manual ARM Processor

http://www.nxp.com/acrobat_download/usermanuals/UM_LPC2114_2124_2212_2214_2.pdf

Armpit Scheme

http://armpit.sourceforge.net/

AVR Tutorial

http://armpit.sourceforge.net/

Introduction to Electronics

http://library.thinkquest.org/16497/intro/index.html

http://www.hackaday.com/2007/10/26/how-to-introduction-to-soldering/