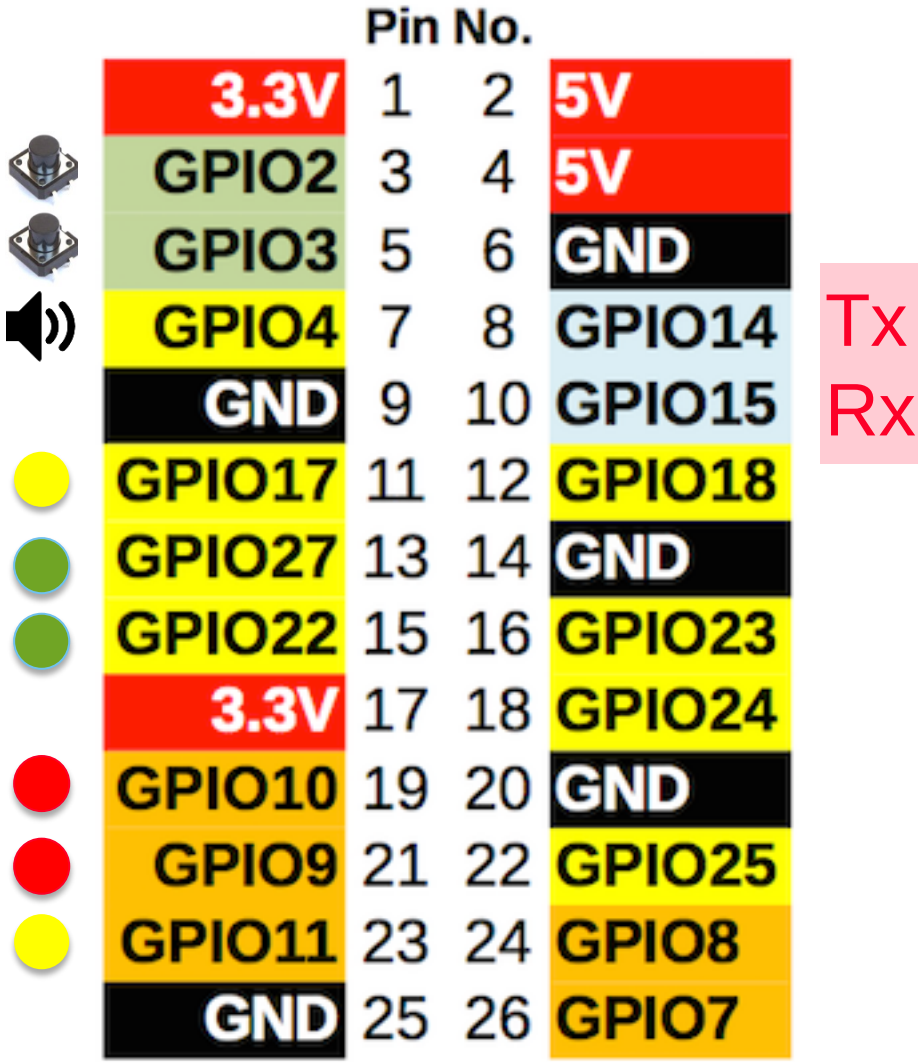
GPIO pins

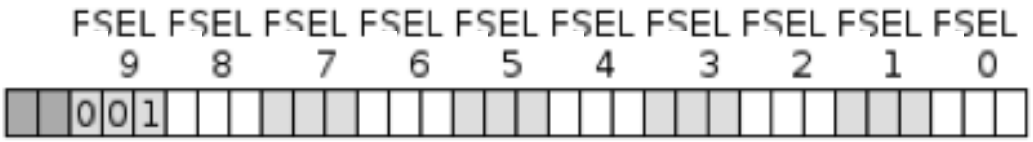
- Raspberry Pi manages up to 54 pins
- Only the showed ones are accessible
- GPIO ports are mapped in memory, starting at 0x3F200000



GPIO memory mapping

Physical address	32 bits
3F200000	GPSEL0
--	
3F200004	GPSEL1
3F200008	GPSEL2
3F20000C	GPSEL3
3F200010	GPSEL4
3F200014	GPSEL5
3F200018	-
3F20001C	GPSET0
3F200020	GPSET1
3F200024	-
3F200028	GPCLR0
3F20002C	GPCLR1

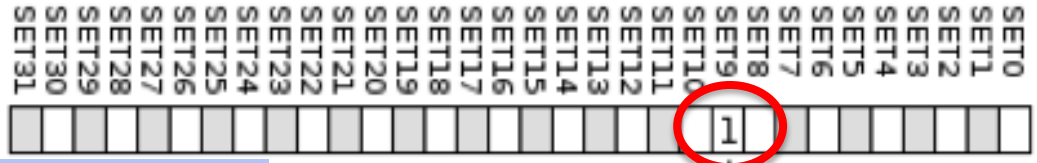
32 bits



Config GPIO 9 as an input

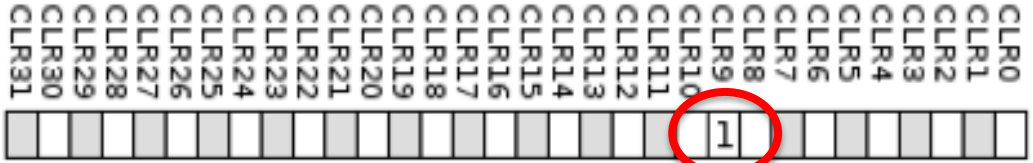
- 000: Input pin
- 001: Output pin
- 010-111: Other modes

GPIO 9 is set as Output



Set GPIO9

LED ON



Clear GPIO9

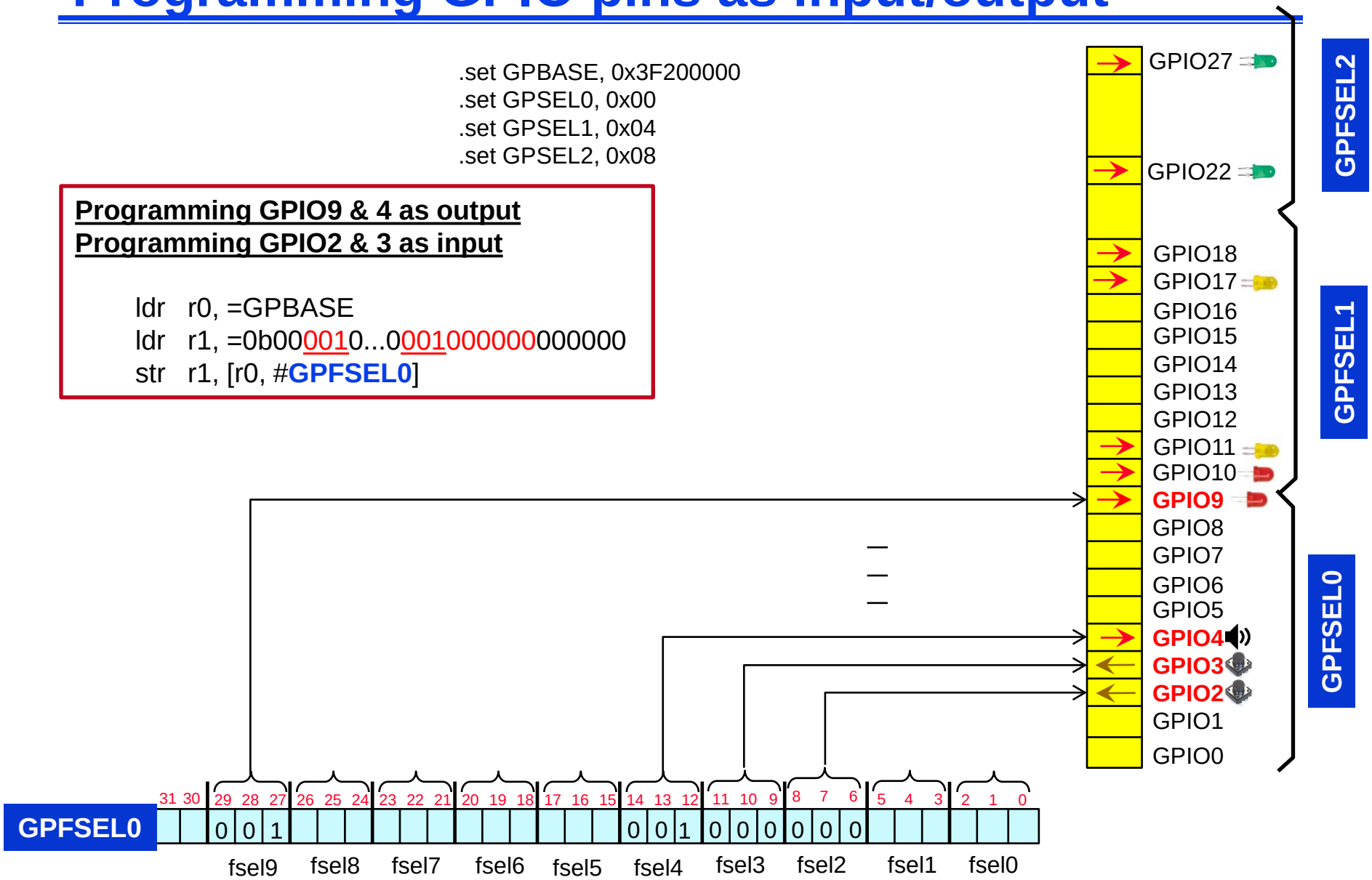
LED OFF

Programming GPIO pins as input/output

```
.set GPBASE, 0x3F200000
.set GPSEL0, 0x00
.set GPSEL1, 0x04
.set GPSEL2, 0x08
```

Programming GPIO9 & 4 as output
Programming GPIO2 & 3 as input

```
ldr   r0, =GPBASE
ldr   r1, =0b00001 0...0001000000000000
str   r1, [r0, #GPFSEL0]
```



Configuration.inc

```
/* Configuration of all the I0 of the expansion board */
      .set    GPBASE,   0x3f200000
      .set    GPFSEL0,       0x00
      .set    GPFSEL1,       0x04
      .set    GPFSEL2,       0x08
.text
    ldr   r0, =GPBASE
                    ldr   r1, [r0,  #GPFSEL0]
                    ldr r4, =0b11001111111111111001000000111111 @ Mask for forcing 0
                    ldr r5, =0b00001000000000000001000000000000 @ Mask for forcing 1
                    and r1,r1,r4
                    orr    r1,r1,r5
      str   r1, [r0, #GPFSEL0]                    @GPIO4&9 as output, GPIO2&3 as input
@ Configure of GPSEL1 (address 0x3F200004) for  GPIO 10,11,17
      ldr   r1, [r0, #GPFSEL1]
                    ldr r4, =0b11111111001111111111111111001001  @ Mask for forcing 0
                    ldr r5, =0b00000000001000000000000000001001  @ Mask for forcing 1
                    and r1,r1,r4
                    orr    r1,r1,r5
      str   r1, [r0, #GPFSEL1]                    @GPIO10&11&17 as output
@ Configure of GPSEL2 (address 0x3F200008) for  GPIO 22,27
      ldr   r1, [r0, #GPFSEL2]
                    ldr r4, =0b11111111001111111111111001111111  @ Mask for forcing 0
                    ldr r5, =0b00000000001000000000000001000000  @ Mask for forcing 1
                    and r1,r1,r4
                    orr    r1,r1,r5
      str   r1, [r0, #GPFSEL2]                    @GPIO22&27 as output
```

inter.inc: using symbolic names for addresses

```
.macro   ADDEXC  vector, dirRTI
 ldr    r1, =(\dirRTI-\vector+0xa7fffffb)
 ror    r1, #2
 str    r1, [r0, #\vector]
.endm
 .set   GPBASE,   0x3F200000
 .set   GPFSEL0,      0x00
 .set   GPFSEL1,      0x04
 .set   GPFSEL2,      0x08
 .set   GPFSEL3,      0x0c
 .set   GPFSEL4,      0x10
 .set   GPFSEL5,      0x14
 .set   GPFSEL6,      0x18
 .set   GPSET0,       0x1c
 .set   GPSET1,       0x20
 .set   GPCLR0,       0x28
 .set   GPCLR1,       0x2c
 .set   GPLEV0,       0x34
 .set   GPLEV1,       0x38
 .set   GPEDS0,       0x40
 .set   GPEDS1,       0x44
 .set   GPFEN0,       0x58
 .set   GPFEN1,       0x5c
 .set   GPPUD,        0x94
 .set   GPPUDCLK0,     0x98
 .set   STBASE,   0x3F003000
 .set   STCS,         0x00
 .set   STCLO,        0x04
 .set   STC1,         0x10
 .set   STC3,         0x18
 .set   INTBASE,  0x3F00b000
 .set   INTFIQCON,    0x20c
 .set   INTENIRQ1,    0x210
 .set   INTENIRQ2,    0x214
```

GPIO

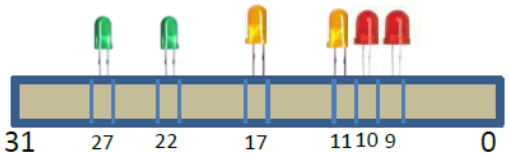
Timer

Interrup.

3F20001C	ON					1					1					1						1	1	1								
3F200028	OFF					1					1					1						1	1	1								
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

3F20001C

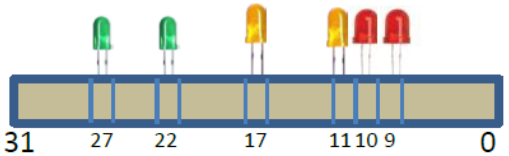
GPSET0



Port *for* ON

3F200028

GPCLR0



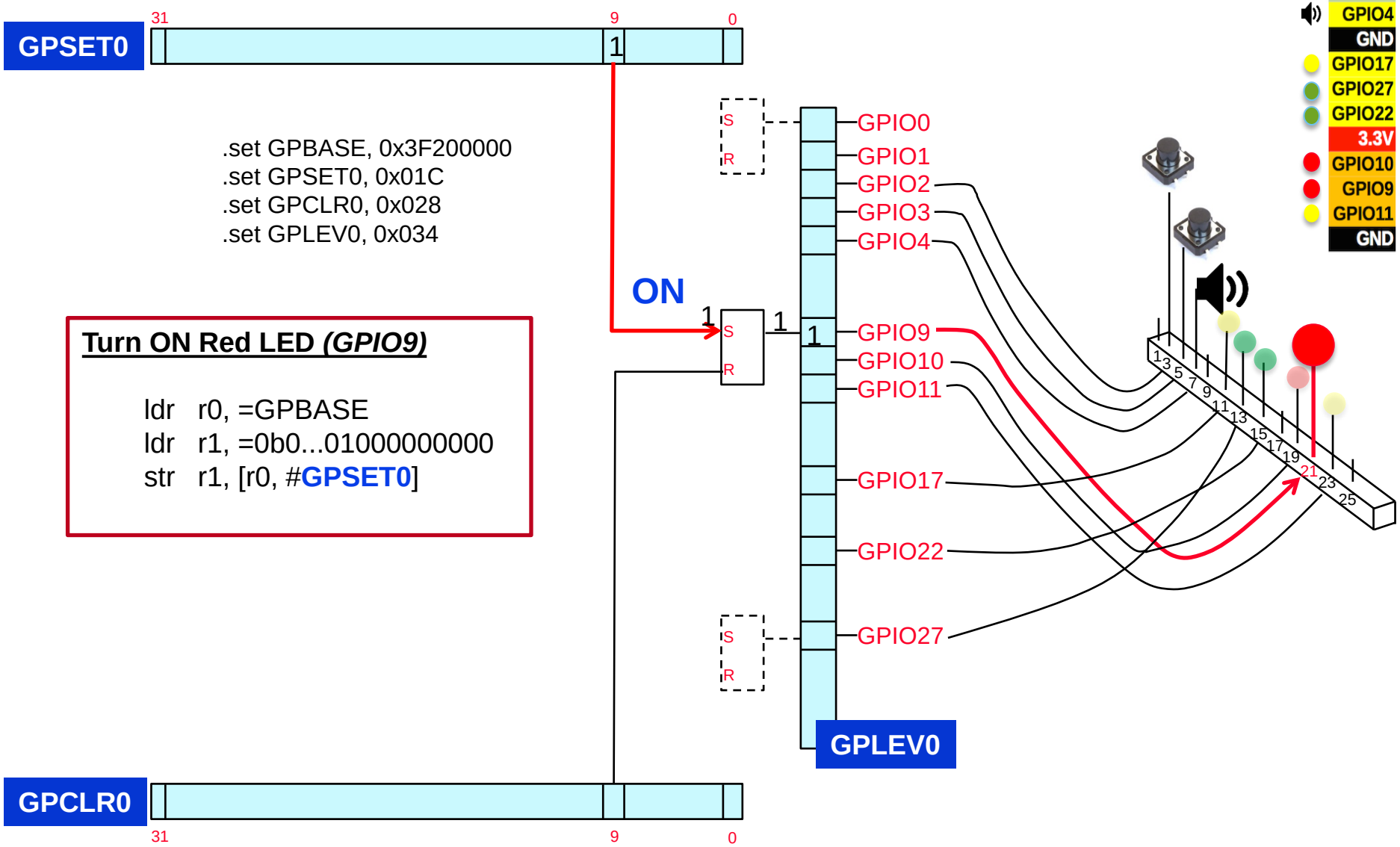
Port *for* OFF

Example 1

- Code for turning on a red LED (GPIO9):

```
        .set    GPBASE,   0x3F200000
        .set    GPFSEL0,  0x00
        .set    GPSET0,   0x1c
.text
        ldr     r0, =GPBASE
/* guia bits          xx9998887776665554443332221110 00*/
        mov     r1, #0b00001000000000000000000000000000
        str     r1, [r0, #GPFSEL0]  @ Configura GPIO 9
/* guia bits          10987654321098765432109876543210*/
        mov     r1, #0b00000000000000000000001000000000
        str     r1, [r0, #GPSET0]   @ Enciende GPIO 9
infi:   b       infi
```

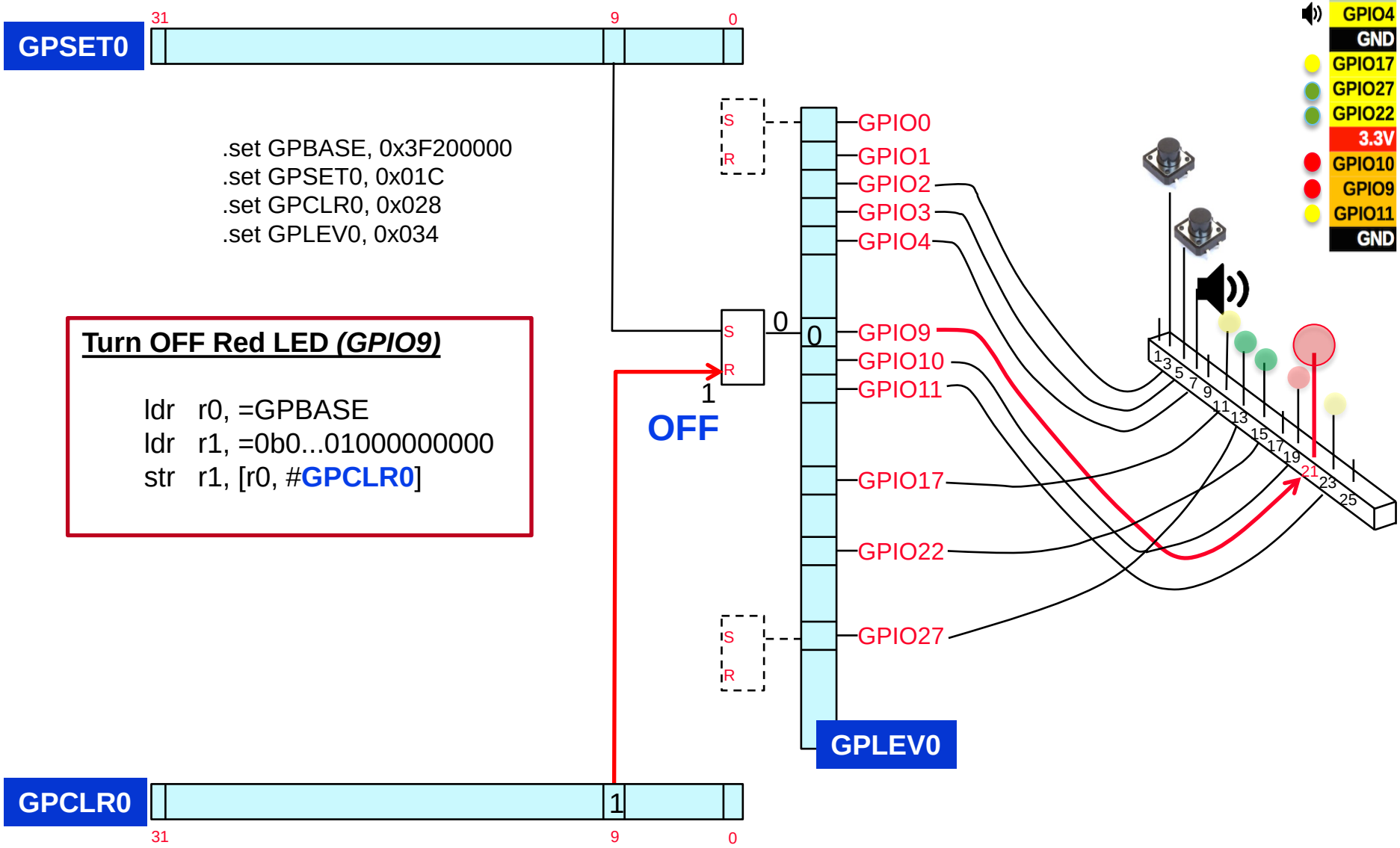
Turing ON the red LED on GPIO9



Turn ON Red LED *(GPIO9)*

```
ldr   r0, =GPBASE
ldr   r1, =0b0...01000000000
str   r1, [r0, #GPSET0]
```

Turing OFF the red LED on GPIO9

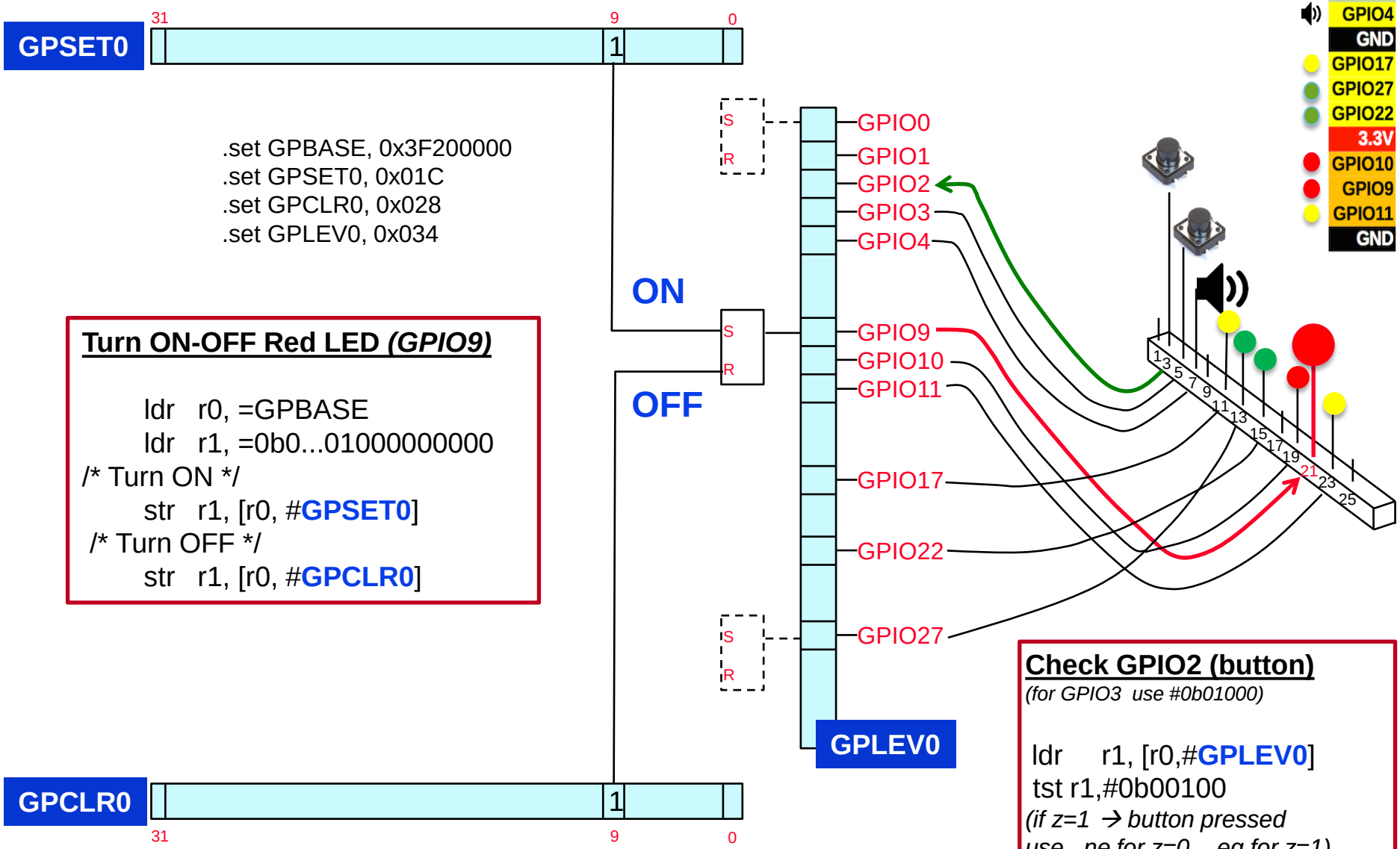



```
.set GPBASE, 0x3F200000
.set GPSET0, 0x01C
.set GPCLR0, 0x028
.set GPLEV0, 0x034
```

Turn OFF Red LED *(GPIO9)*

```
ldr   r0, =GPBASE
ldr   r1, =0b0...01000000000
str   r1, [r0, #GPCLR0]
```

Turing ON-OFF LEDs: summary




```
.set GPBASE, 0x3F200000
.set GPSET0, 0x01C
.set GPCLR0, 0x028
.set GPLEV0, 0x034
```

Turn ON-OFF Red LED *(GPIO9)*

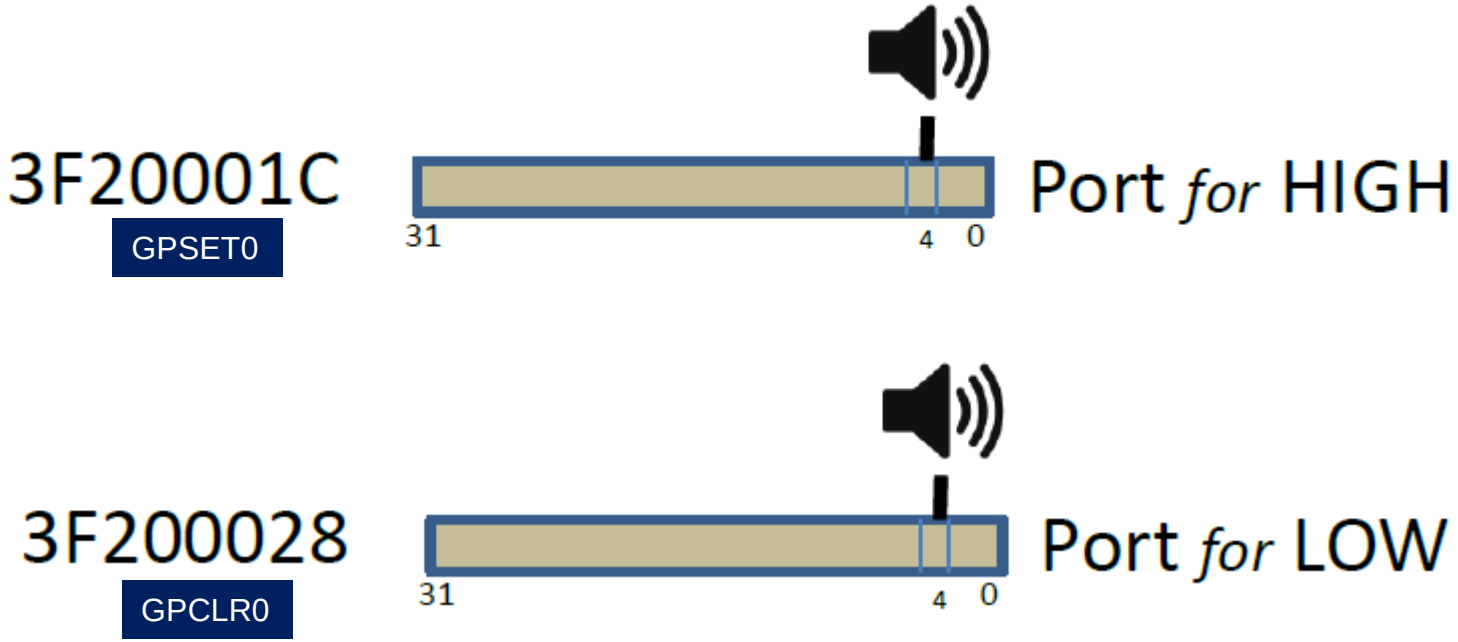
```
        ldr   r0, =GPBASE
        ldr   r1, =0b0...01000000000
/* Turn ON */
        str   r1, [r0, #GPSET0]
 /* Turn OFF */
        str   r1, [r0, #GPCLR0]
```

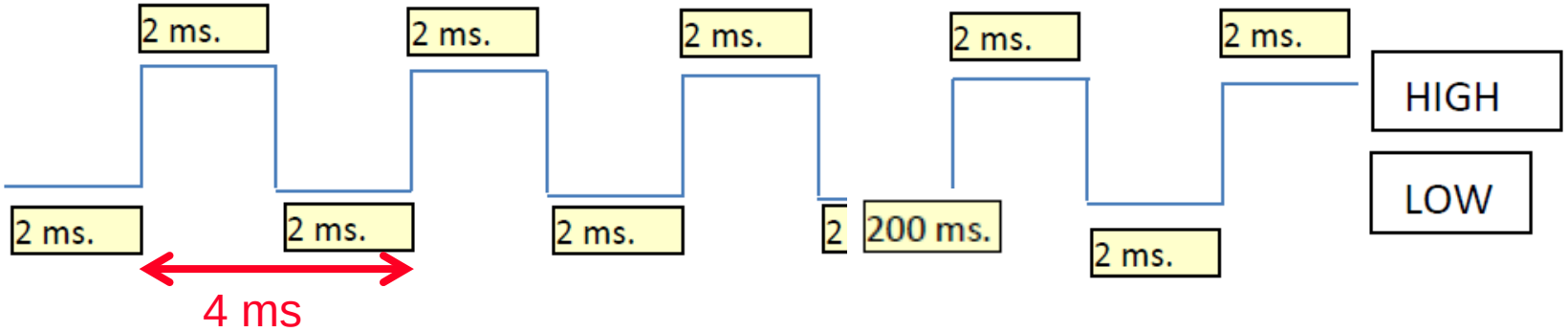
Check GPIO2 (button)

(for GPIO3 use #0b01000)

```
ldr     r1, [r0,#GPLEV0]
tst r1,#0b00100
```

(if z=1 → button pressed
use _ne for z=0, _eq for z=1)





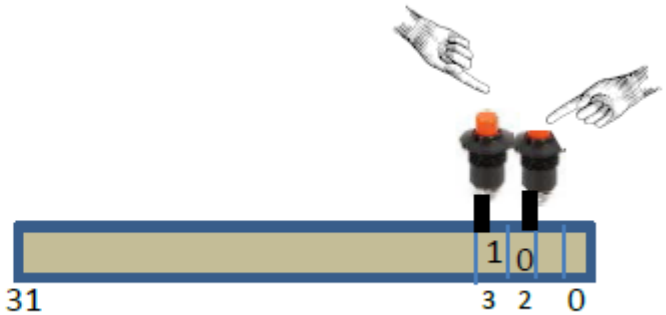
F= 250 Hz → CC= 1/250 s. = 4 ms

3F200034

GPLEV0

3F200034

GPLEV0



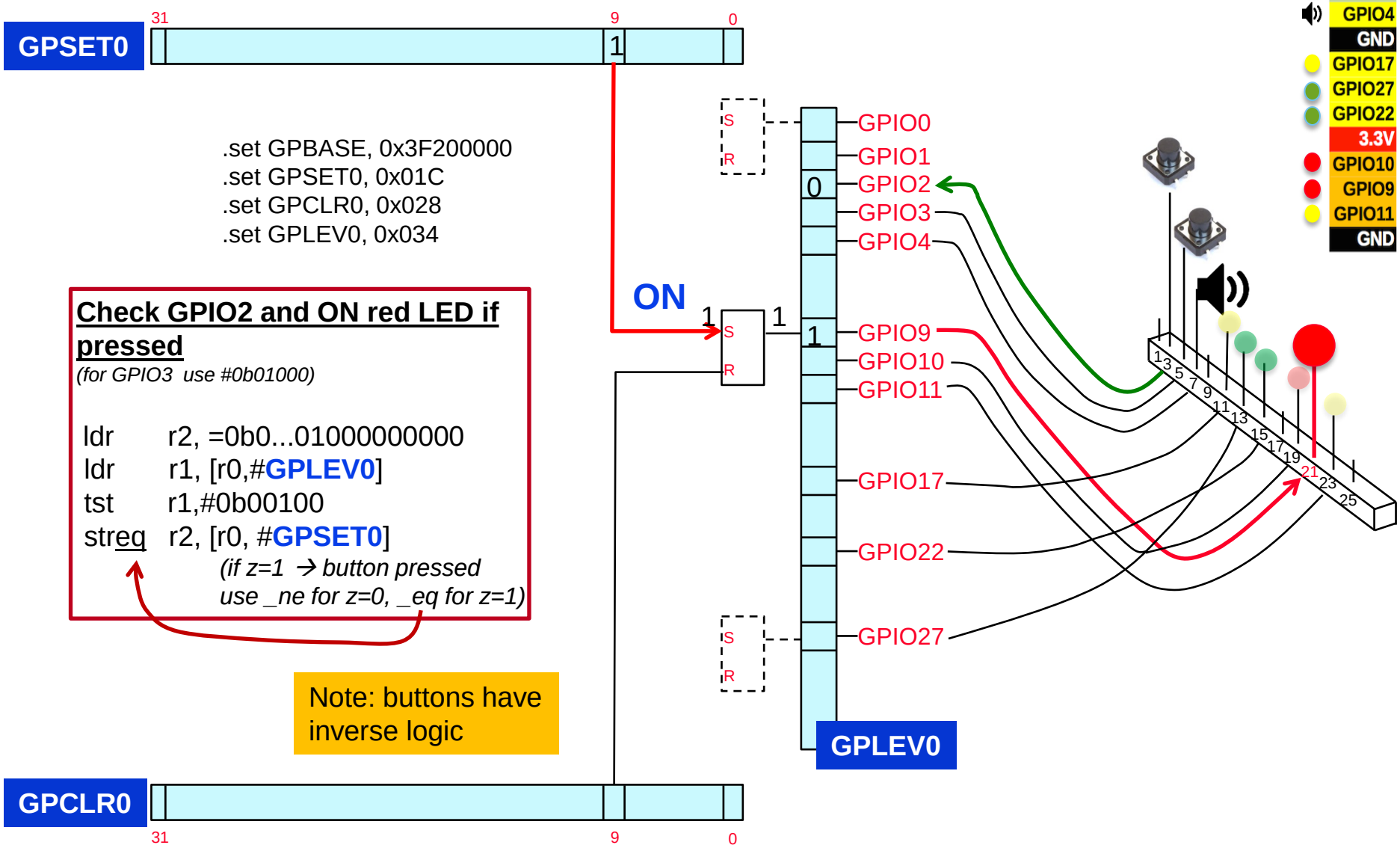
A very basic routine to copy the content of this port in the register r8 (for example) is:

```
ldr  r0,=0x3F200034     /* r0 contents the address of the input port */
ldr  r8,[r0]            /* The content of port 3F200034 is copied into r8 */
                        /* If r8.2=0, the button is pressed (1 for released)*/
                        /* If r8.3=0, the button is pressed (1 for released)*/
```

To check the state of the buttons:

```
tst r8, #0b00100        /* The mask is 00100 for bit 2 */
beq button2pressed  /* if bit2=0 (pressed) → z=1 */
```

Check button (GPIO2) and turn ON a red led




```
.set GPBASE, 0x3F200000
.set GPSET0, 0x01C
.set GPCLR0, 0x028
.set GPLEV0, 0x034
```

Check GPIO2 and ON red LED if pressed

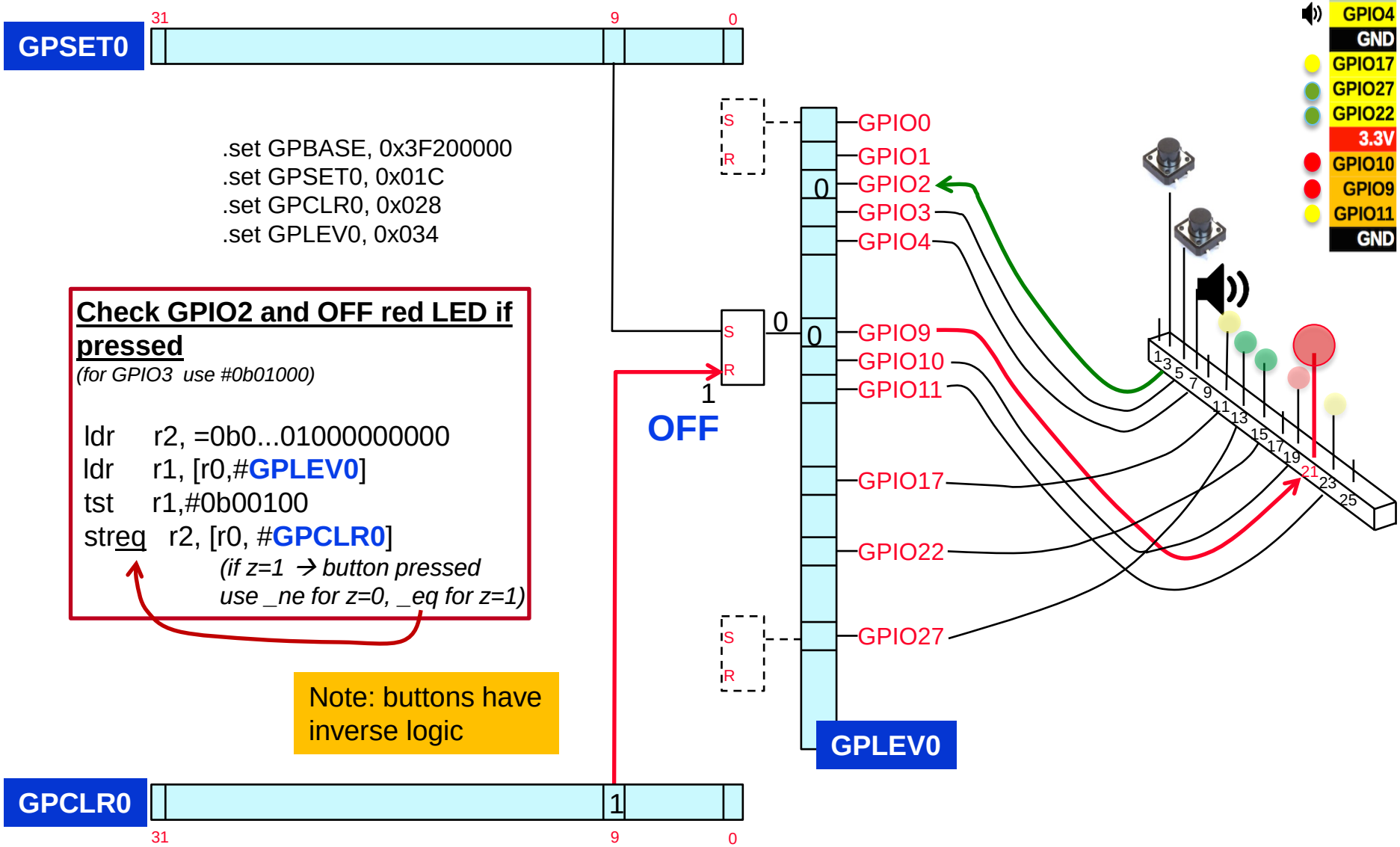
(for GPIO3 use #0b01000)

```
ldr     r2, =0b0...01000000000
ldr     r1, [r0,#GPLEV0]
tst     r1,#0b00100
streq   r2, [r0, #GPSET0]
```

(if z=1 → button pressed
use _ne for z=0, _eq for z=1)

Note: buttons have inverse logic

Check button (GPIO2) and turn OFF a red led




```
.set GPBASE, 0x3F200000
.set GPSET0, 0x01C
.set GPCLR0, 0x028
.set GPLEV0, 0x034
```

Check GPIO2 and OFF red LED if pressed

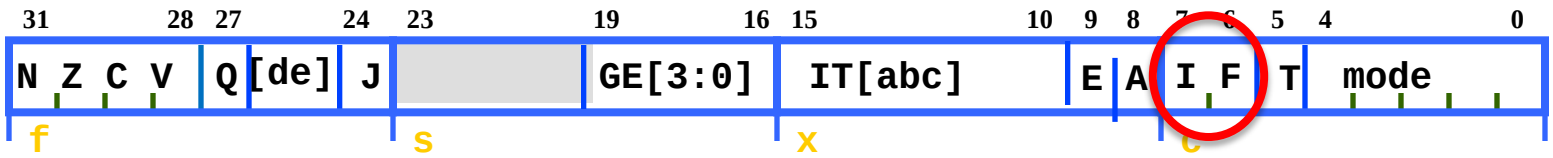
(for GPIO3 use #0b01000)

```
ldr     r2, =0b0...01000000000
ldr     r1, [r0,#GPLEV0]
tst     r1,#0b00100
streq   r2, [r0, #GPCLR0]
```

(if z=1 → button pressed
use _ne for z=0, _eq for z=1)

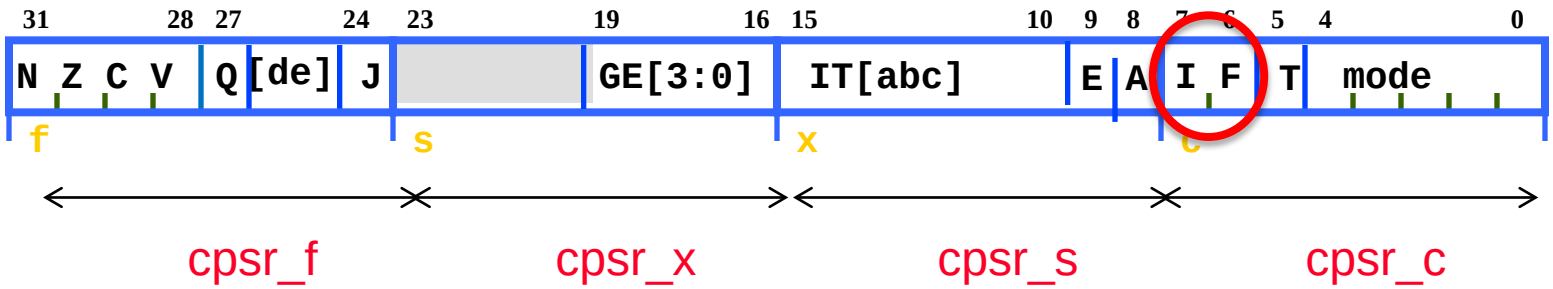
Note: buttons have inverse logic

Status register again: cpsr_fsxc



- **Condition code flags**
  - N = Negative result from ALU
  - Z = Zero result from ALU
  - C = ALU operation Carried out
  - V = ALU operation oVerflowed
- **Sticky Overflow flag - Q flag**
  - Indicates if saturation has occurred
- **SIMD Condition code bits – GE[3:0]**
  - Used by some SIMD instructions
- **IF THEN status bits – IT[abcde]**
  - Controls conditional execution of Thumb instructions
- **T bit**
  - T = 0: Processor in ARM state
  - T = 1: Processor in Thumb state
- **J bit**
  - J = 1: Processor in Jazelle state
- **Mode bits**
  - Specify the processor mode
- **Interrupt Disable bits**
  - I = 1: Disables IRQ
  - F = 1: Disables FIQ
- **E bit**
  - E = 0: Data load/store is little endian
  - E = 1: Data load/store is bigendian
- **A bit**
  - A = 1: Disable imprecise data aborts

Status register again: cpsr_fsxc



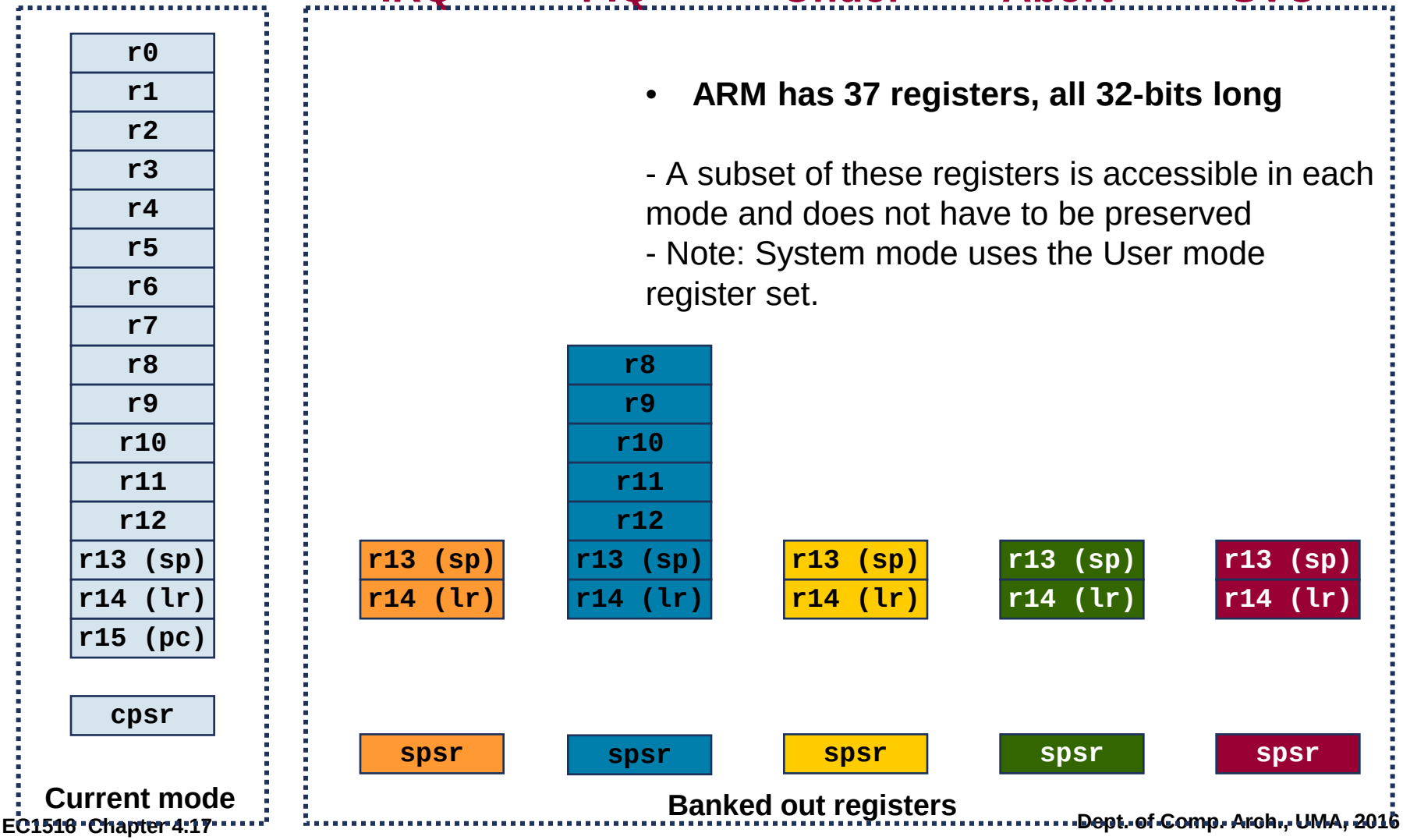
- **We can modify the Current Program Status Register (CPSR) in several ways by mean of the *msr* instruction:**
  - **Every byte can be independently modified by accessing the corresponding byte (cpsr_f, cpsr_x, cpsr_s, cpsr_c):**
    - Example:
      - mov r0,#0bxxxxxxxx
      - msr cpsr_c, r0 → only the 8 LSB are modified
    - Example 2: disable interrupts IQR, FIR for Supervisor (SVC) mode and enter in supervisor mode:
      - mov r0,#0b11010011
      - msr cpsr_c, r0 → only the 8 LSB are modified

Banking of registers

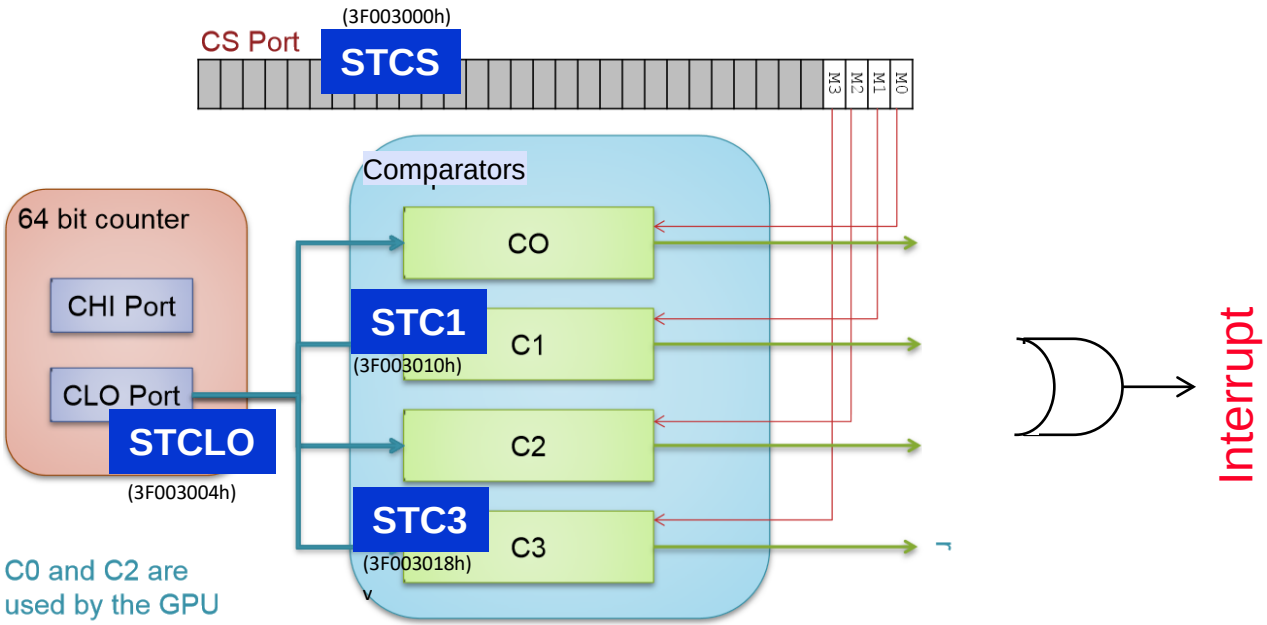
10000	10010	10001	11011	10111	10011
User mode	IRQ	FIQ	Undef	Abort	SVC

- **ARM has 37 registers, all 32-bits long**

- A subset of these registers is accessible in each mode and does not have to be preserved
- Note: System mode uses the User mode register set.



System Timer



Timer (polling example)

STBASE		
STCS	3F20 3000	CS
	3F20 3004	CLO
	3F20 3008	CHI
STCLO	3F20 300C	C0
STC1	3F20 3010	C1
	3F20 3014	C2
	3F20 3018	C3

Ascending counter bytes

Ascending counter bytes

(CHI:CLO increments each microsecond)

Compare registers: if any one of them is equal to CLO, then corresponding bit Mx in CS is set and interrupt is provoked (if it is enabled). C0 and C3 are used by the GPU

Delay loop (polling)

```
          ldr r0, =STBASE      @ r0 is an input parameter (ST base address)
          ldr r1, =500000      @ r1 is an input parameter (waiting time in microseconds)

          push {r4, r5}        @ Save r4 and r5 in the stack
          ldr r4, [r0, #STCLO] @ Load CLO timer
          add r4, r1           @ Add waiting time -> this is our ending time
ret1:     ldr r5, [r0, #STCLO] @ Enter waiting loop: load current CLO timer
          cmp r5, r4           @ Compare current time with ending time
          blo ret1             @ If lower, go back to read timer again
          pop {r4, r5}                   @ Restore r4 and r5
          bx lr                @ Return from routine
```

Example 8: turn on a RED led after 4 sec.

```
1          .include  "inter.inc"
2  .text
3          mov     r0, #0          @apunto tabla excepciones
4          ADDEXC  0x18, irq_handler
           ldr     r0, =GPBASE
           ldr     r1, =0b00001000000000000000000000000000
           str     r1, [r0, #GPFSEL0]
           ldr     r0, =STBASE
           ldr     r1, [r0, #STCLO]
           add     r1, #0x400000   @4,19 segundos
           str     r1, [r0, #STC1]
           ldr     r0, =INTBASE
           mov     r1, #0b0010
           str     r1, [r0, #INTENIRQ1]
           mov     r0, #0b01010011     @modo SVC, IRQ activo
           msr     cpsr_c, r0
17 bucle:  b       bucle
18
19 irq_handler:
20         push    {r0, r1}
           ldr     r0, =GPBASE
           mov     r1, #0b00000000000000000000001000000000
           str     r1, [r0, #GPSET0]
24         pop     {r0, r1}
           subs    pc, lr, #4
```

Set GPIO9 as Output

Load CLO, add 4sec and store result in C1

Enable C1 interruption

Enable I flag

Turn on RED led (GPIO9)

PC← LR - 4

INTERRUPTS

Exception handler

- Basic structure of a exception handler
  - Interruption: the return is done by lr-4
  - Internal exception (as data abort): the return is done by lr-8

```
irq_handler:
        push    {lista registros}
        ...
        pop     {lista registros}
        subs    pc, lr, #4
```

  - User must manage A, I and F flags to disable/enable nesting of new exceptions and interruptions.
    - Initially the interruptions are disabled (I=F=1).

1- Initialize Vector Table

- To write in the Vector Table:
  - Example for IRQ

```
mov     r0, #0          @Vector table Base = 0
ADDEXC  0x18, irq_handler
```

- ADDEXC is a macro that computes the offset of the exception handler and writes the Vector in the Vector Table.

Mem. address	Contents
0x00000000	b reset_handler_routine
0x00000004	b Unexisingcode_hadeler_routine
0x00000008	b SVC_handler_routine
0x0000000C	b Abort1_handler_routine
0x00000010	b Abort2_handler_rountie
0x00000014	-
0x00000018	***b irq_handler***
0x0000001C	b FIQ_hadler_routine

2- Disable Interruptions and initialize the stack

- Each mode has its stack pointer (sp)
  - Change the mode (via cpsr_c)
  - Instructions msr (sr <- reg) y mrs (reg <-sr).
  - Initialize the corresponding sp register
- Initial state in BareMetal is SVC
  - sp_fiq=0x4000, sp_irq=0x8000, sp_svc=0x8000000:

Entering in FIQ mode

Entering in IRQ mode

Entering in SVC mode

```
mov     r0, #0          @apunto tabla excepciones
ADDEXC  0x18, irq_handler
ADDEXC  0x1c, fiq_handler
mov     r0, #0b11010001   @modo FIQ, FIQ&IRQ desact
msr     cpsr_c, r0
mov     sp, #0x4000
mov     r0, #0b11010010   @modo IRQ, FIQ&IRQ desact
msr     cpsr_c, r0
mov     sp, #0x8000
mov     r0, #0b11010011   @modo SVC, FIQ&IRQ desact
msr     cpsr_c, r0
mov     sp, #0x8000000
```

4- Enable sources of interruption cont.

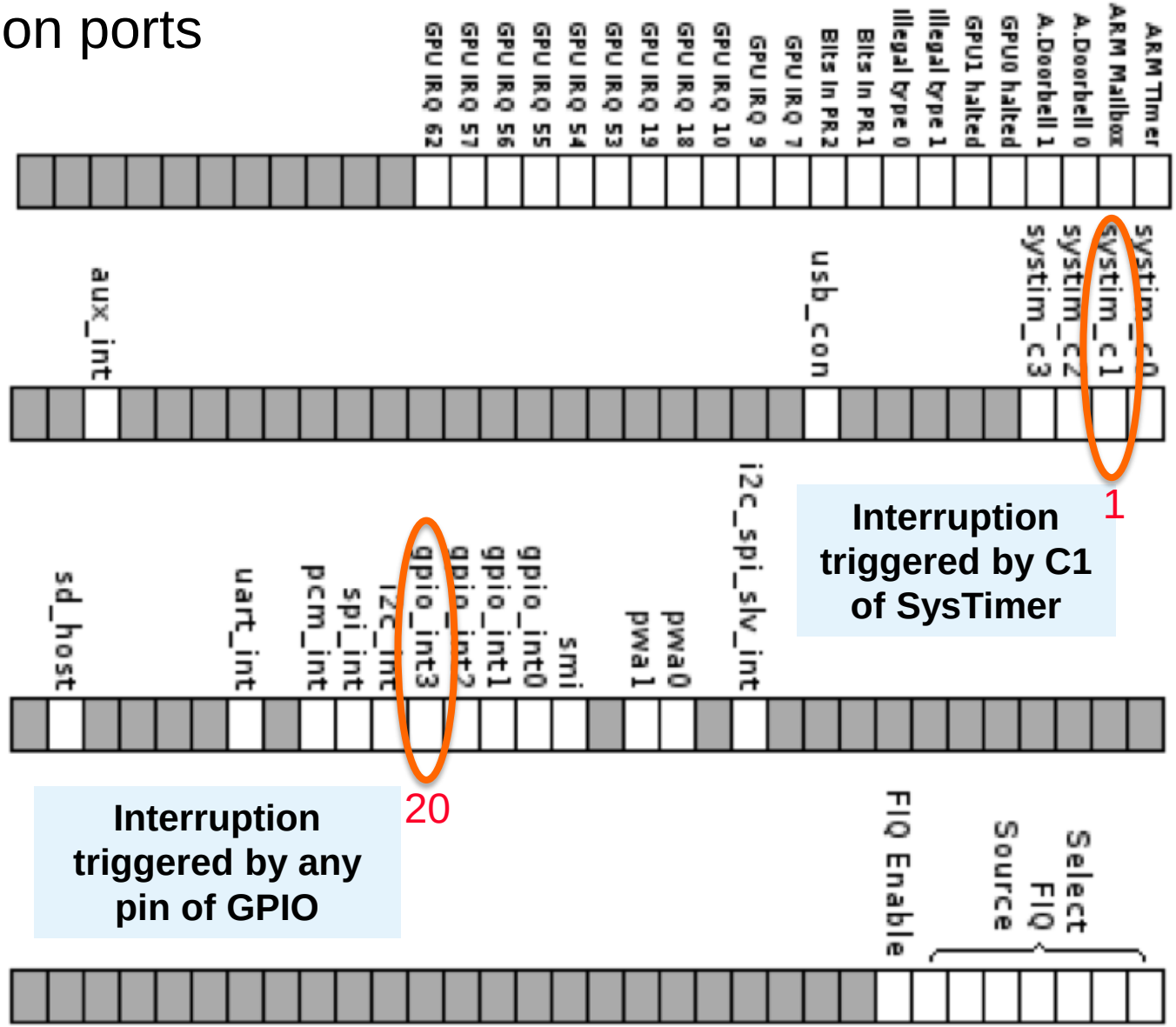
- Interruption ports

IRQ Basic

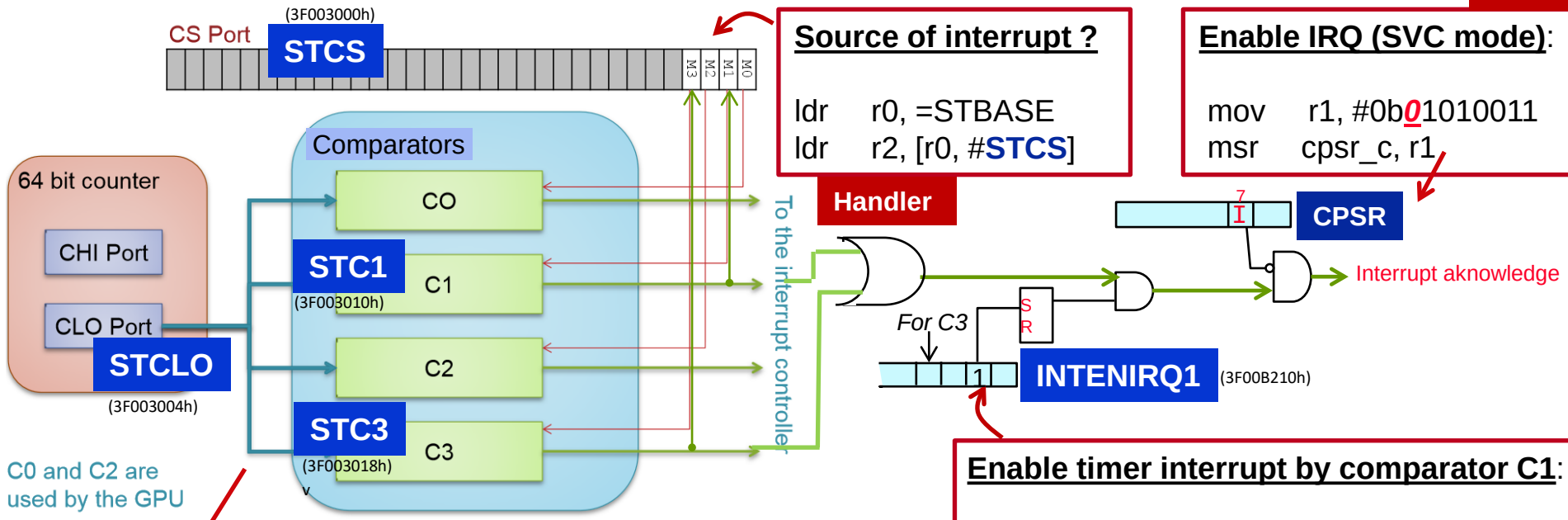
IRQ pending 1

IRQ pending 2

FIQ control



Regular Interrupts by timer *(using comparator C1)*



C0 and C2 are used by the GPU

Main

Source of interrupt ?

```
ldr     r0, =STBASE
ldr     r2, [r0, #STCS]
```

Handler

Enable IRQ (SVC mode):

```
mov     r1, #0b01010011
msr     cpsr_c, r1
```

Enable timer interrupt by comparator C1:

```
ldr     r0,=INTBASE
mov     r1, #0b0010 @(#0b1000 for C3)
str     r1,[r0,#INTENIRQ1]
```

Main

Preparing C1 to send an interrupt after "*y*" microseconds

```
ldr     r0, =STBASE
ldr     r1, [r0, #STCLO]
add     r1, #y      @y microseconds
str     r1, [r0, #STC1] @#STC3 for C3
```

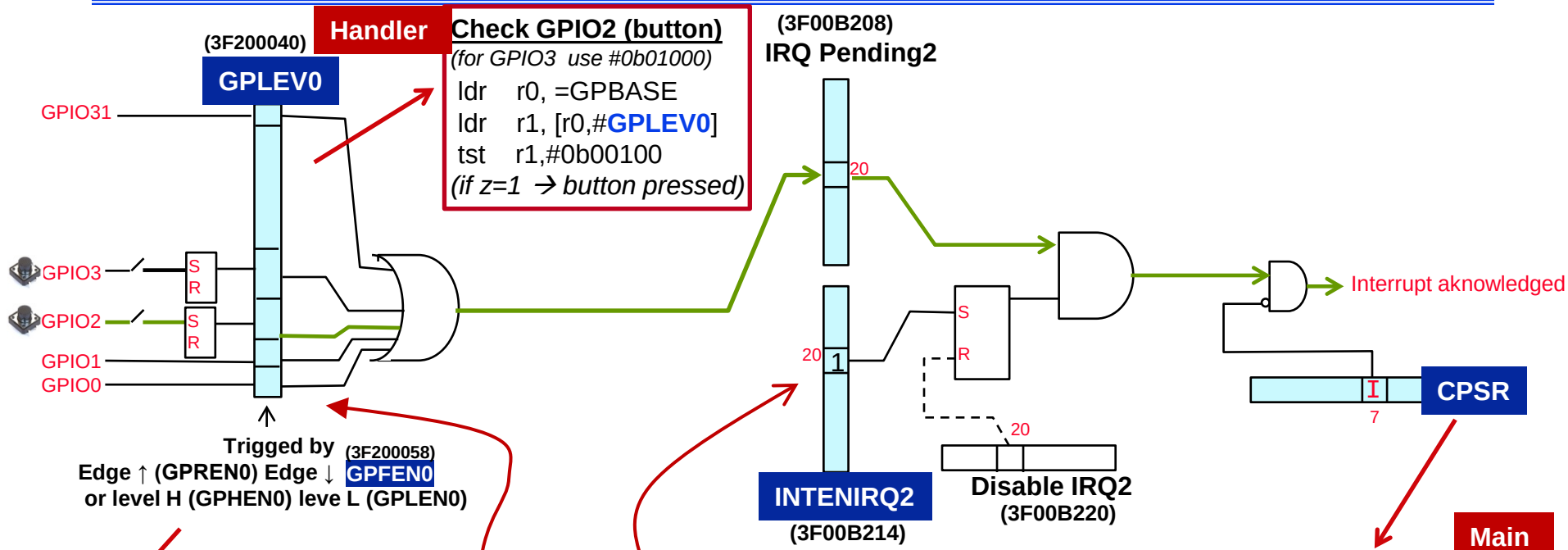
Main and/or handler

Reset (end) of the timer interrupt by comparator C1 *(to allow a new interrupt)*:

```
ldr     r0, =STBASE
mov     r1, #0b0010   @(#0b1000 for C3)
str     r1,[r0,#STCS]
    (insert just here for rearming the timer)
subs    pc, lr, #4
```

Handler

Generating a regular interruption (IRQ) by push buttons



Handler

Check GPIO2 (button)

(for GPIO3 use #0b01000)

```
ldr   r0, =GPBASE
ldr   r1, [r0,#GPLEV0]
tst   r1,#0b00100
```

(if z=1 → button pressed)

GPED0 Trigged by falling edge

(for both GPIO2 & 3)

```
ldr   r0, =GPBASE
mov   r1, #0b011100
str   r1,[r0,#GPFEN0]
```

Main

Enable local interrupt IRQ

```
ldr   r0, =INTBASE
mov   r1, #0b00...010...0
str   r1,[r0,#INTENIRQ2]
```

Main

Main

Enable global interrupt IRQ

(for SVC mode)

```
mov   r1, #0b01010011
msr   cpsr_c, r1
```

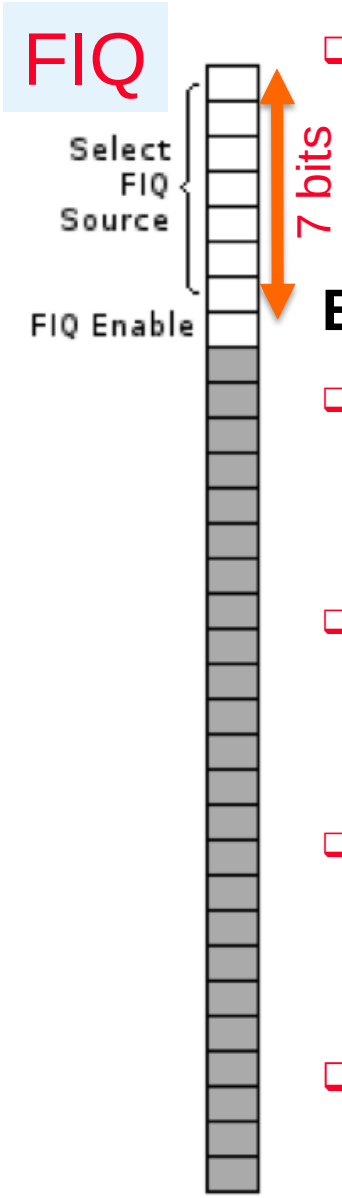
Reset (end) of interrupt IRQ2 (GPIO2&3)

(to allow a new interrupt via GPIO2&3):

```
ldr   r0, =GPBASE
mov   r1, #0b011100
str   r1,[r0,#GPEDS0]
subs pc, lr, #4
```

Handler

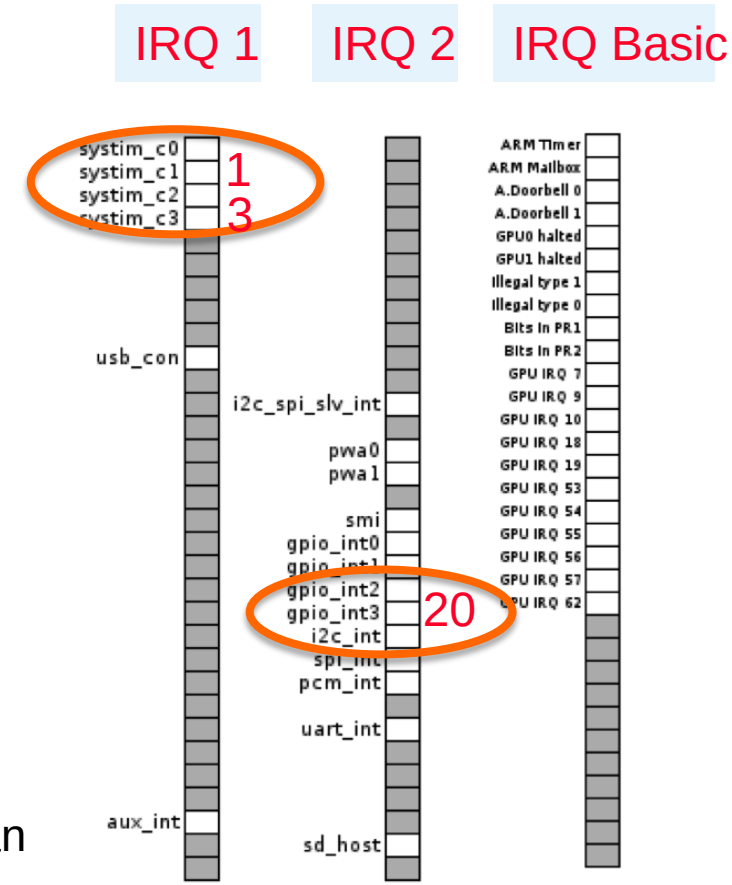
Using FIQ



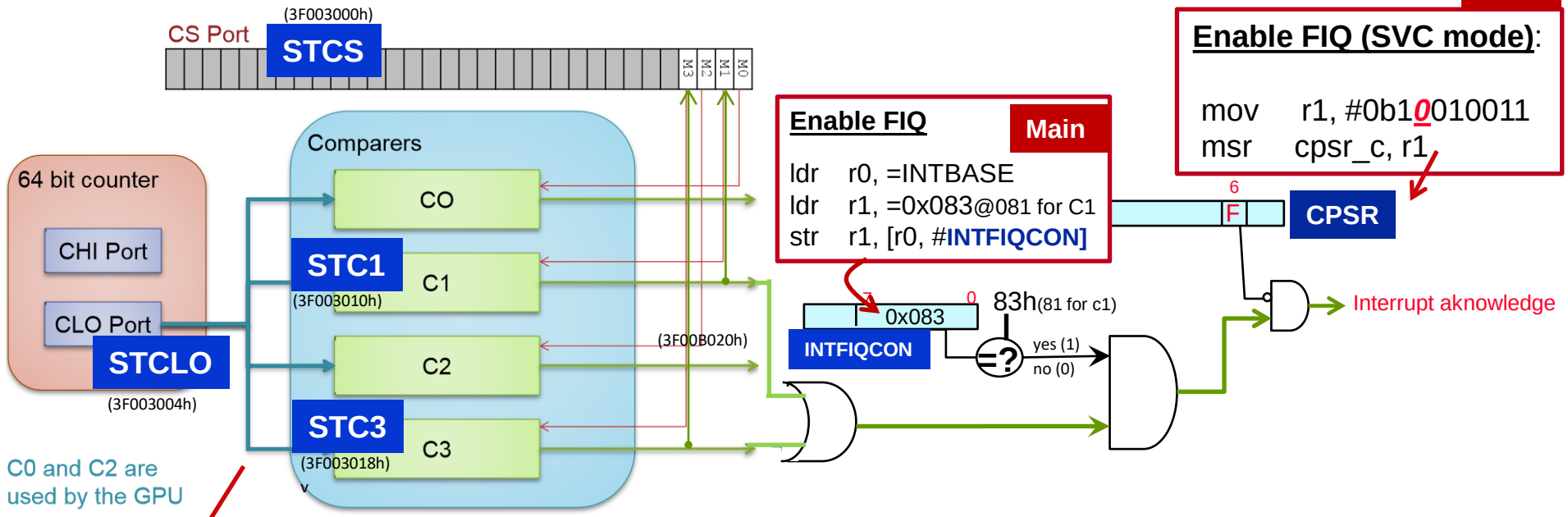
- Select FIQ Source = 7 bits → 128 sources
  - 0-31 represent 32 interruption sources of IRQ 1
  - 32-63 represent 32 interruption sources of IRQ 2
  - 64-95 represent 32 interruption sources of IRQ basic

Examples:

- Enable FIQ for C1 of SysTimer
  - Bit 1 of IRQ1 → Code 1
  - Also 1 in FIQ Enable
  - Result: 0b10000001 → 0x81
- Enable FIQ for C3 of SysTimer
  - Bit 3 of IRQ1 → Code 3
  - Also 1 in FIQ Enable
  - Result: 0b10000011 → 0x83
- Enable FIQ for GPIO_int3
  - Bit 20 of IRQ2 → Code 20+32
  - Also 1 in FIQ Enable
  - Result: 0b10110100 → 0xB4
- Disadventage:
  - Just one source interruption can be enabled!



Fast Interrupts by timer *(using comparator C3)*



Preparing C3 to send an interrupt after "*y*" microseconds

```
ldr   r0, =STBASE
ldr   r1, [r0, #STCLO]
add   r1, #y      @y microseconds
str   r1, [r0, #STC3] @#STC1 for C1
```

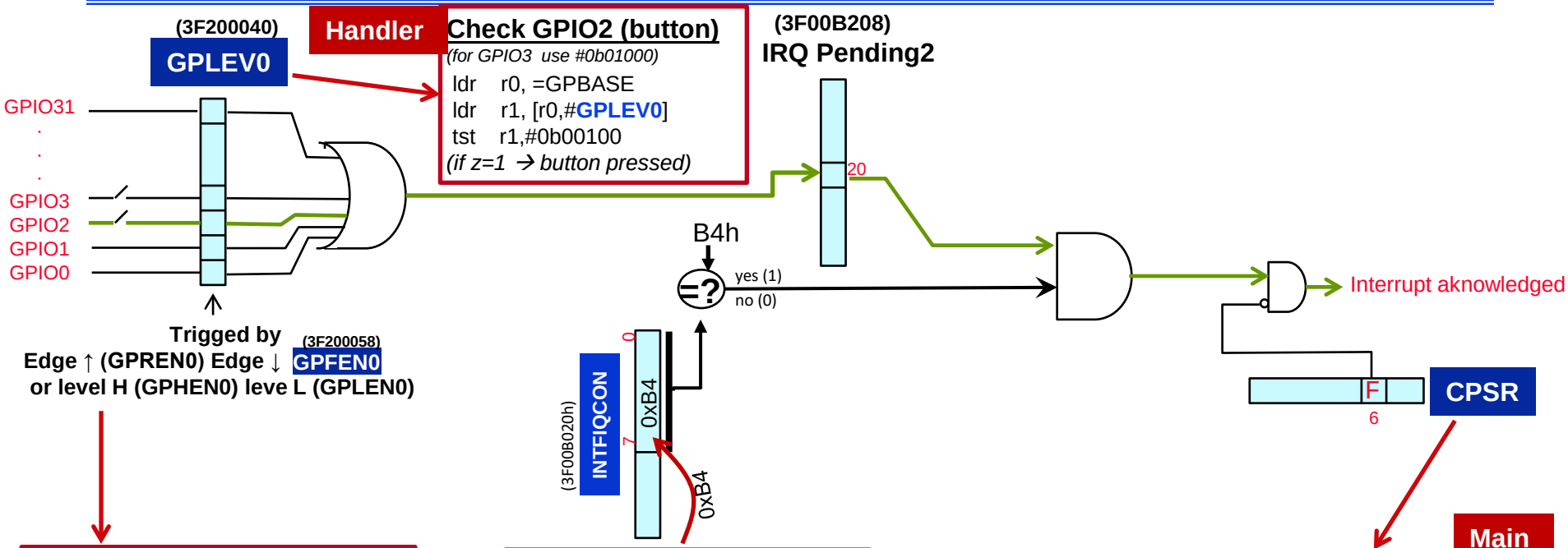
Main and/or handler

Reset (end) of the timer interrupt by comparator C3
(to allow a new interrupt):

```
ldr   r0, =STBASE
mov   r1, #0b01000 @ #0b0010 for C1
str   r1,[r0,#STCS]
    (insert just here for rearming the timer)
subs  pc, lr, #4
```

Handler

Fast interrupts by push buttons



Handler

Check GPIO2 (button)

(for GPIO3 use #0b01000)

```
ldr   r0, =GPBASE
ldr   r1, [r0,#GPLEV0]
tst   r1,#0b00100
```

(if z=1 → button pressed)

Trigged by falling edge

(for both GPIO2 & 3)

```
ldr   r0, =GPBASE
mov   r1, #0b01100
str   r1,[r0,#GPFEN0]
```

Main

Enable local FIQ

Main

```
ldr   r0, =INTBASE
ldr   r8, =0x0B4
str   r8, [r0, #INTFIQCON]
```

Main

Enable global interrupt FIQ

(for SVC mode)

```
mov   r1, #0b10010011
msr   cpsr_c, r1
```

Reset (end) of interrupt IRQ2 (GPIO2&3)

(to allow a new interrupt):

```
ldr   r0, =GPBASE
mov   r1, #0b01100
str   r1,[r0,#GPEDS0]
subs pc, lr, #4
```

Handler