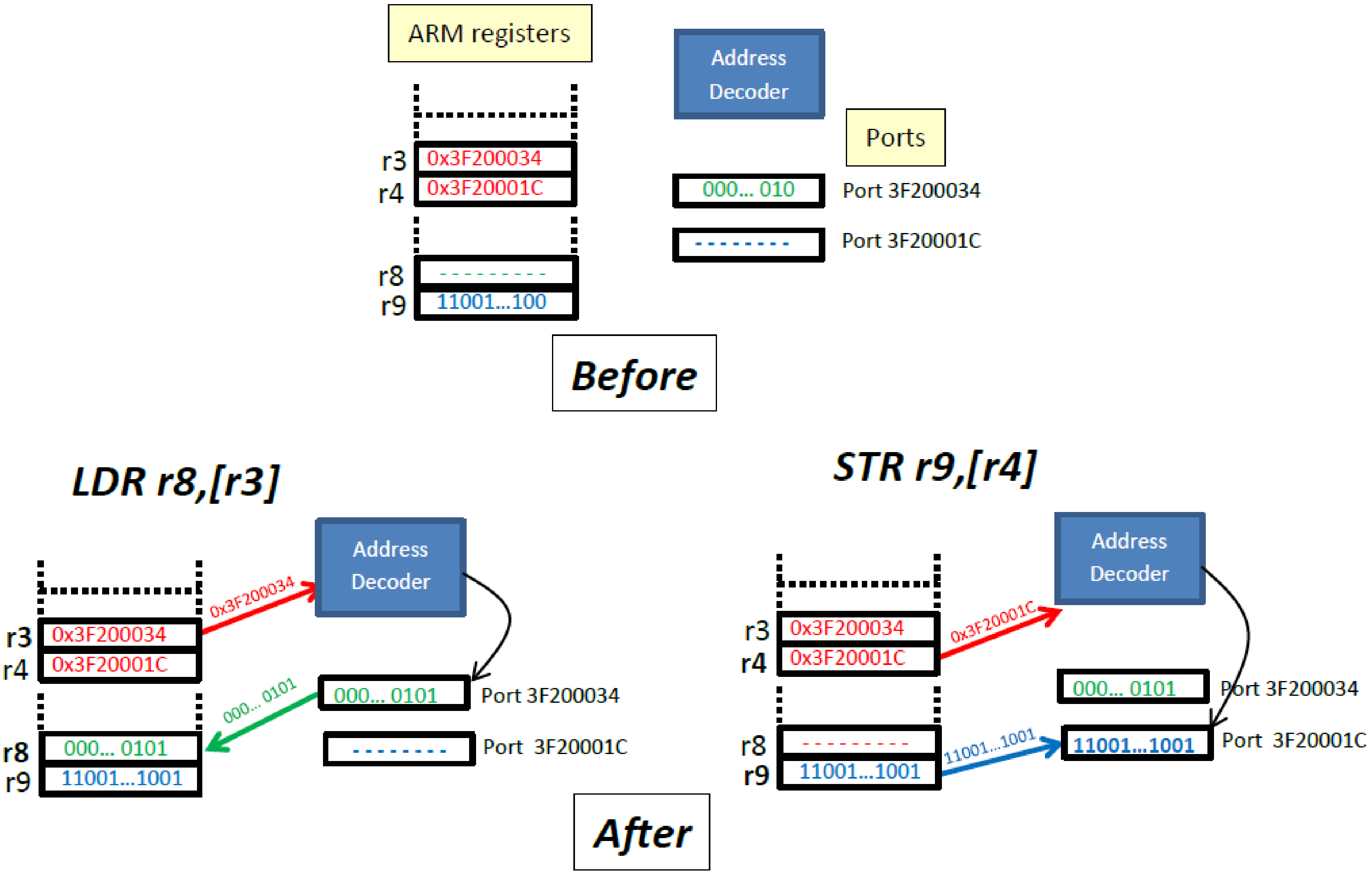
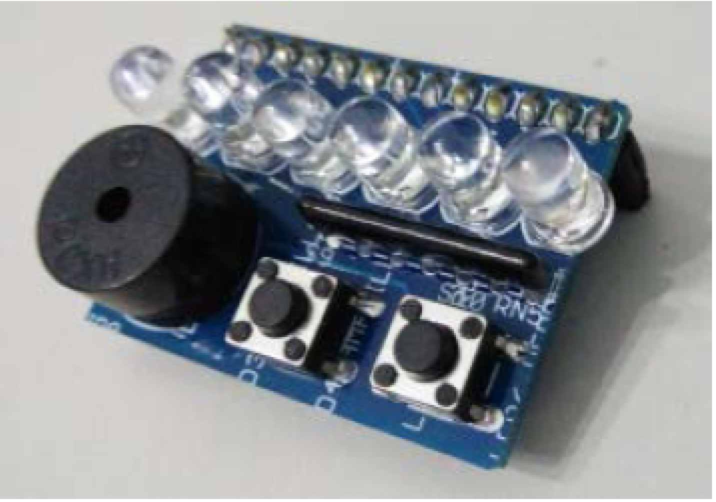
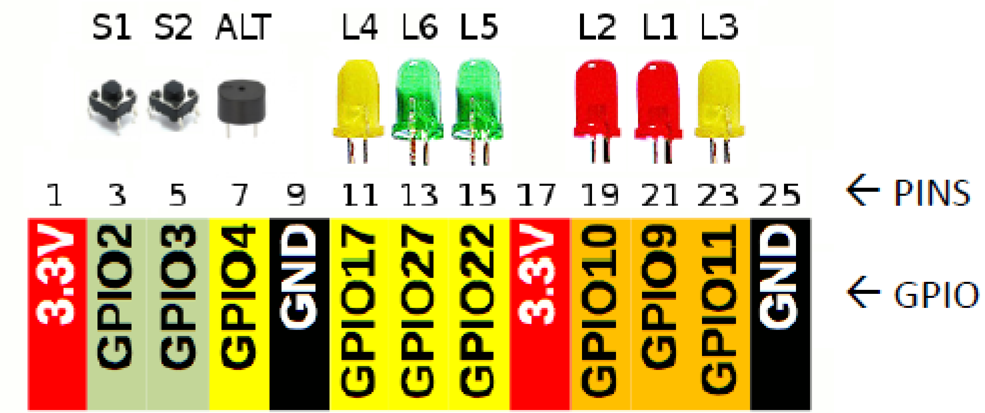
Practice Raspberry Pi2

PART I:

INPUT/OUTPUT PORTS


ARM registers
Address Decoder
Ports
r3
0x3F200034
r4
0x3F20001C
r8
- - - - - - - - -
r9
11001…100
000… 010
Port 3F200034
- - - - - - - -
Port 3F20001C
Before
LDR r8,[r3]
r3
0x3F200034
r4
0x3F20001C
0x3F200034
Address Decoder
000… 0101
Port 3F200034
- - - - - - - -
Port 3F20001C
000… 0101
r8
000… 0101
r9
11001…1001
STR r9,[r4]
r3
0x3F200034
r4
0x3F20001C
0x3F20001C
Address Decoder
000… 0101
Port 3F200034
r8
- - - - - - - - -
r9
11001…1001
11001…1001
11001…1001
Port 3F20001C
After





The 9 GPIO signals of the expansion board that we use in our practice are following:

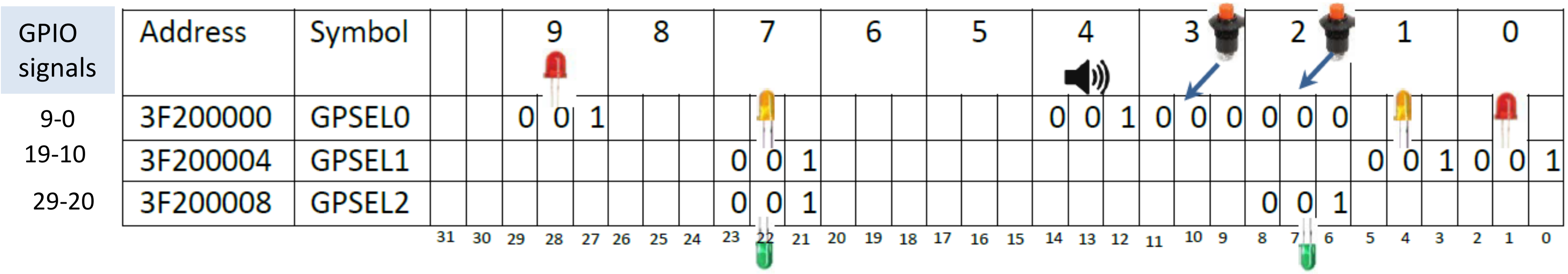
Device	GPIO	Board Pins	Input/output
LED1 (red)	9	21	Output
LED2 (red)	10	19	Output
LED3 (yellow)	11	23	Output
LED4 (yellow)	17	11	Output
LED5 (green)	22	15	Output
LED6 (green)	27	13	Output
PUSH_BUTTON1	2	3	Input
PUSH_BUTTON2	3	5	Input
SPEAKER	4	7	Output

GPFSEL0· (3F20 0000)

1. Configure: GPIO9 as Output

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

X		FSEL9			FSEL8			FSEL7			FSEL6			FSEL5			FSEL4			FSEL3			FSEL2			FSEL1			FSEL0		
		0	0	1																											



GPIO signals	Address	Symbol			9			8			7			6			5			4			3			2			1			0		
9-0	3F200000	GPSEL0			0	0	1													0	0	1	0	0	0	0	0	0						
19-10	3F200004	GPSEL1									0	0	1																0	0	1	0	0	1
29-20	3F200008	GPSEL2									0	0	1													0	0	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

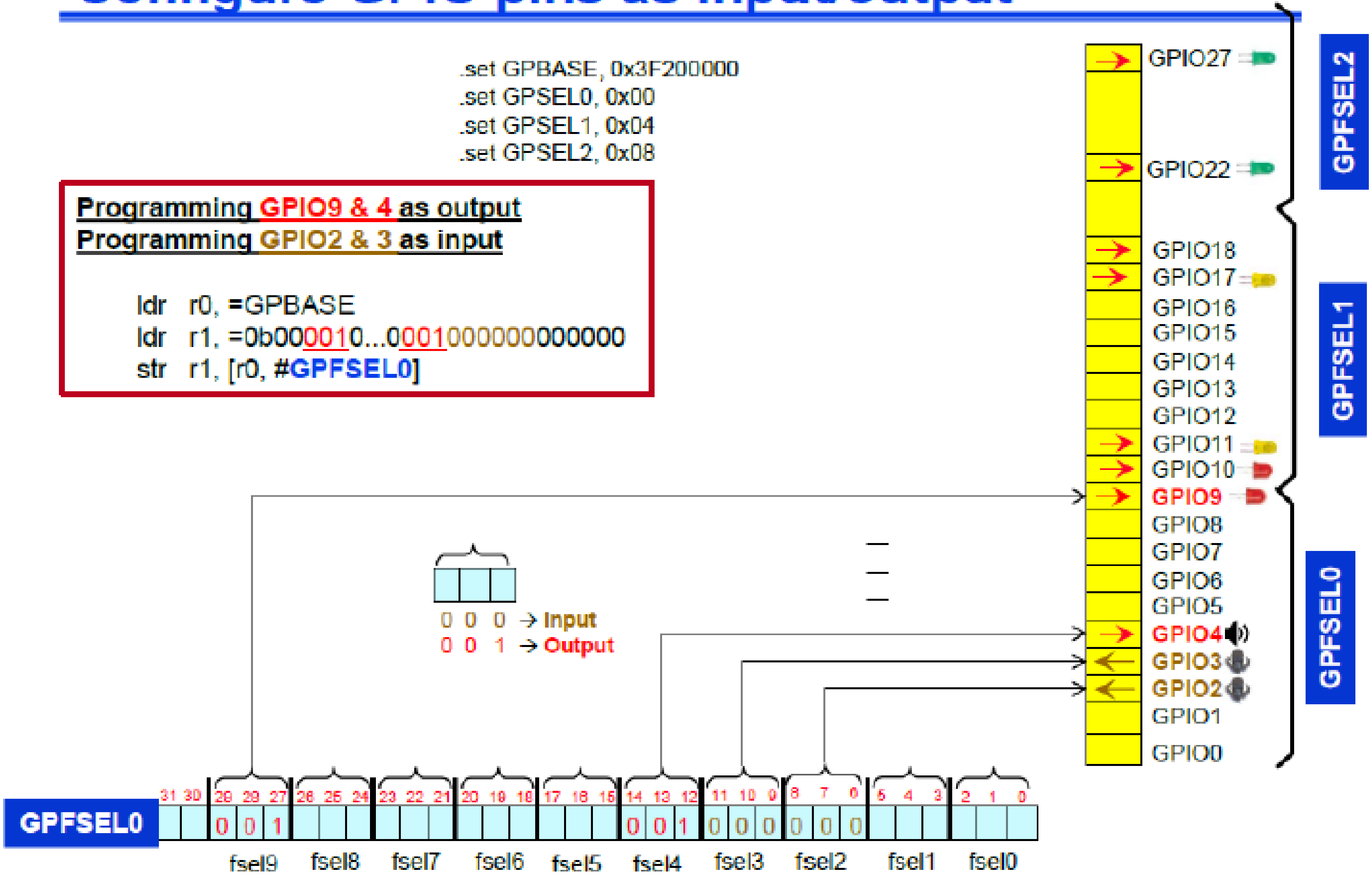
Configure GPIO pins as input/output

```
.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, =0b000010...0001000000000000
str   r1, [r0, #GPFSEL0]
```



GPFSEL0		fsel9	fsel8	fsel7	fsel6	fsel5	fsel4	fsel3	fsel2	fsel1	fsel0
Bits	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
Value		0 0 1					0 0 1	0 0 0	0 0 0		

configuration.inc

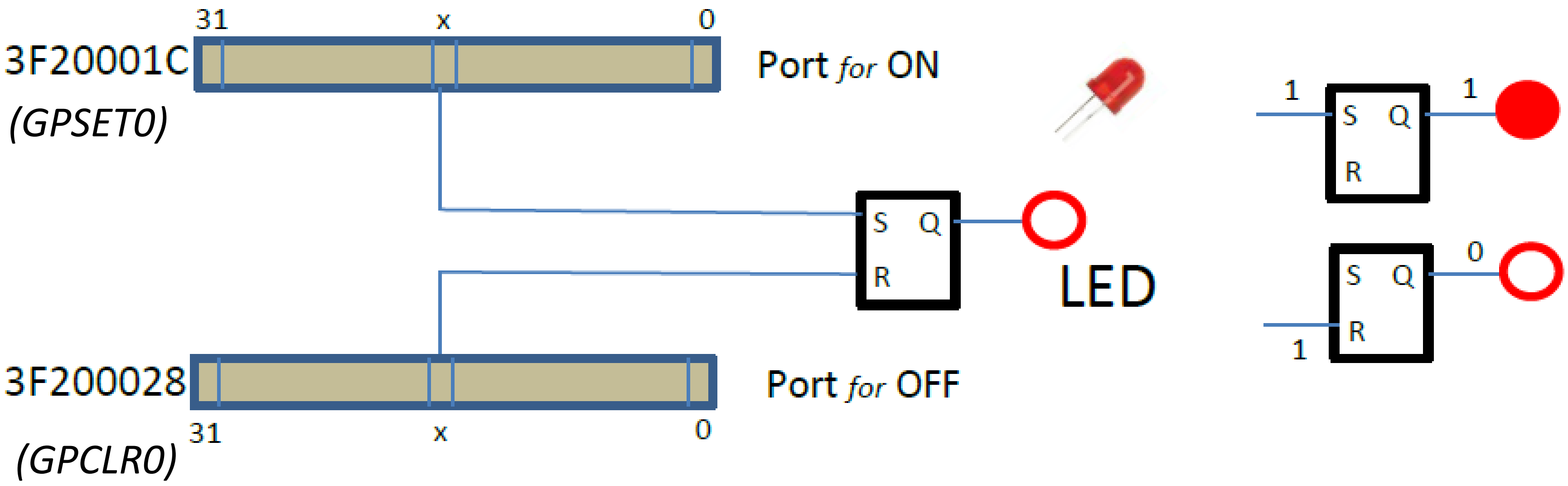
```
/* Configuration of all the I0 of the expansion board */
     .set   GPBASE,   0x3f200000
     .set   GPFSEL0,      0x00
     .set   GPFSEL1,      0x04
     .set   GPFSEL2,      0x08
.text
   ldr   r0, =GPBASE
         ldr   r1, [r0,  #GPFSEL0]
         ldr r4, =0b11001111111111111001000000111111 @ 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
```

Symbolic names

```
.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
```

```
.include "configuration.inc"
.include “symbolic.inc"
```



3F20001C

(GPSET0)

Port *for* ON

3F200028

(GPCLR0)

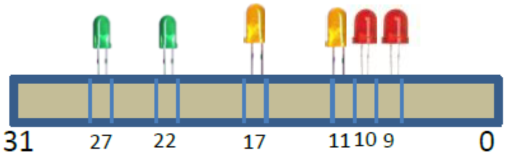
Port *for* OFF

LED

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

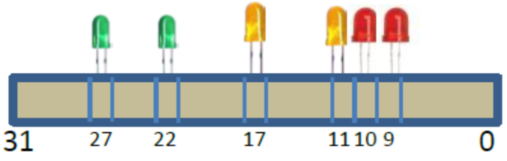
(GPSET0)



Port *for* ON

3F200028

(GPCLR0)



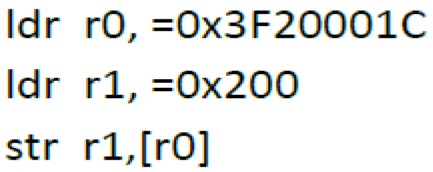
Port *for* OFF

A very basic routine to turn ON the red LED of GPIO9 is

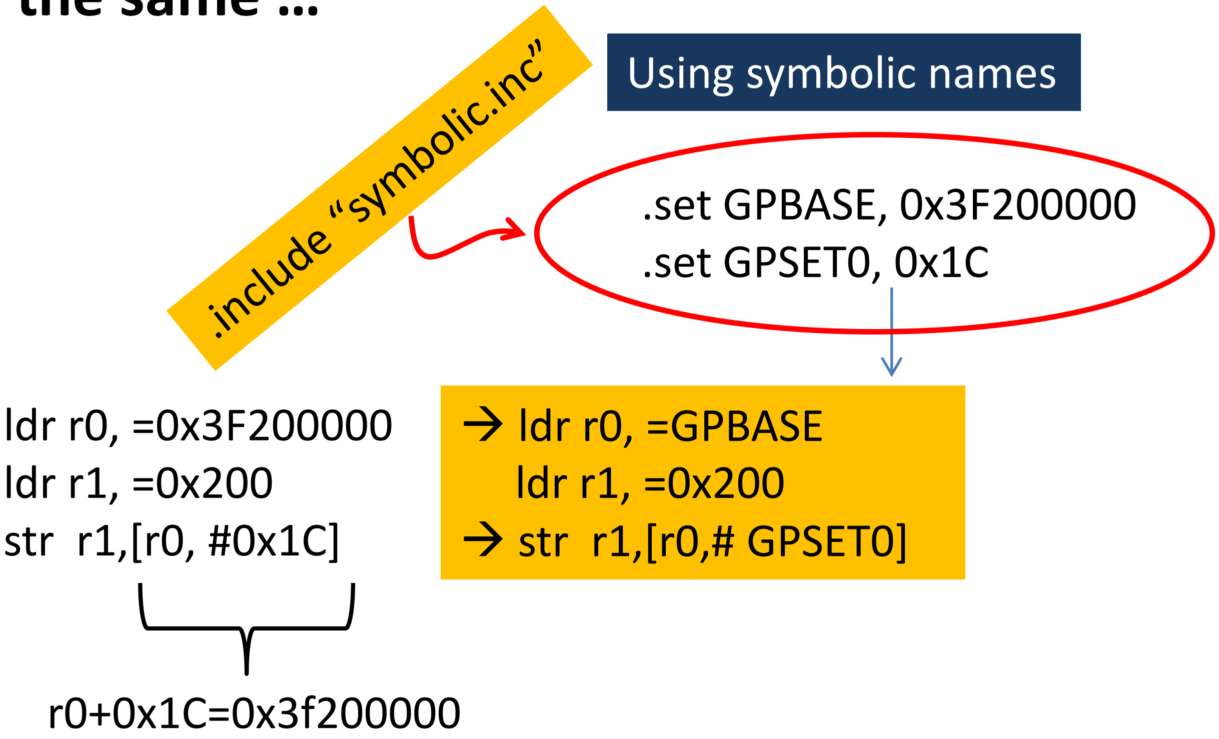
```
ldr  r0, =0x3F20001C
ldr  r1, =0x200
str  r1,[r0]
```

```
ldr r0, =GPBASE
ldr r1, =0x200
str r1 ,[r0,#GPSET0]
```

Another way to do the same …



```
ldr  r0, =0x3F20001C
ldr  r1, =0x200
str  r1,[r0]
```



Using symbolic names

.include "symbolic.inc"

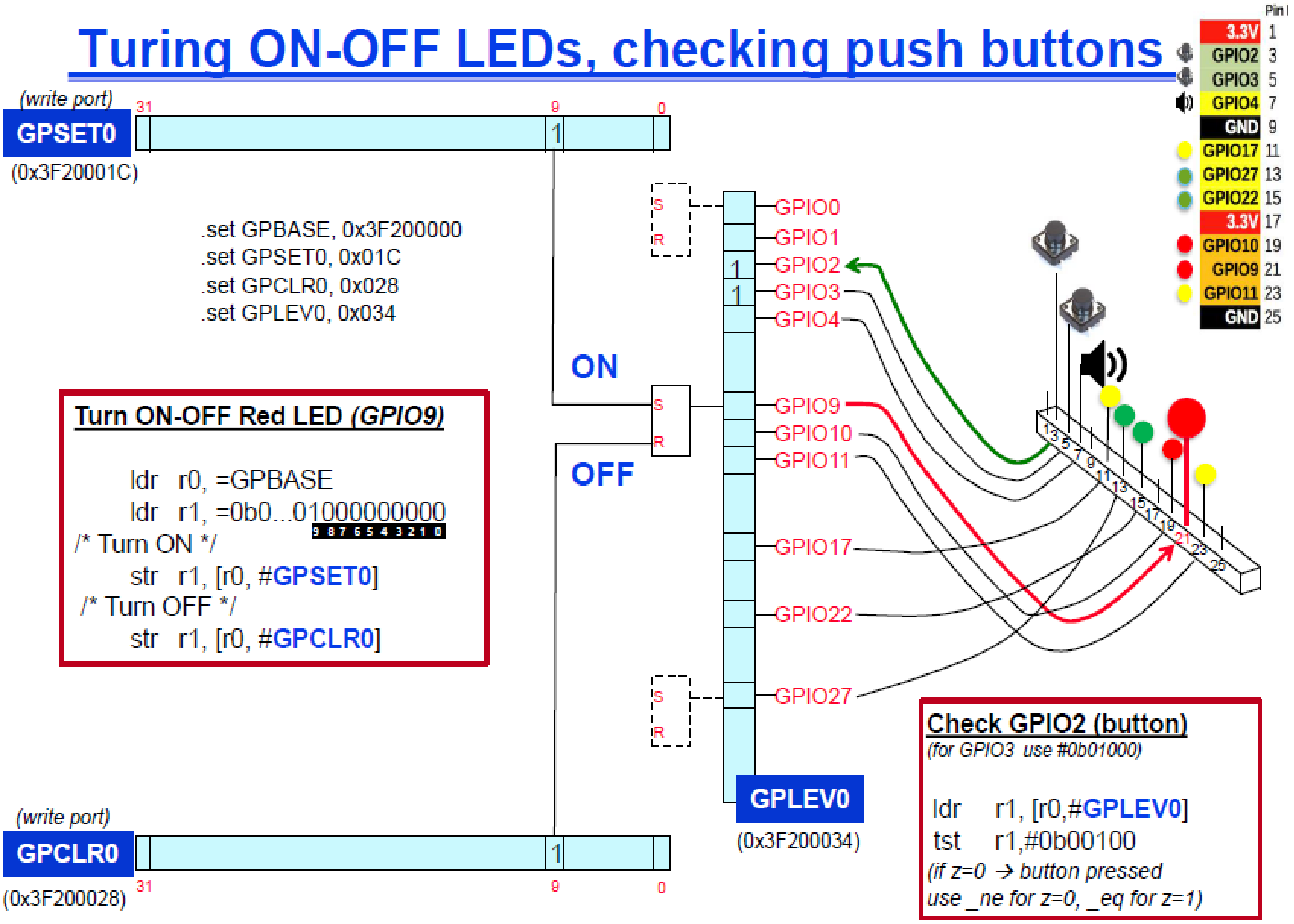
```
.set GPBASE, 0x3F200000
.set GPSET0, 0x1C
```

```
ldr r0, =0x3F200000
ldr r1, =0x200
str  r1,[r0, #0x1C]
```

r0+0x1C=0x3f200000

```
→ ldr r0, =GPBASE
  ldr r1, =0x200
→ str  r1,[r0,# GPSET0]
```

Turing ON-OFF LEDs, checking push buttons



3.3V	1
GPIO2	3
GPIO3	5
GPIO4	7
GND	9
GPIO17	11
GPIO27	13
GPIO22	15
3.3V	17
GPIO10	19
GPIO9	21
GPIO11	23
GND	25

```
.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
                   9 8 7 6 5 4 3 2 1 0
/* Turn ON */
      str   r1, [r0, #GPSET0]
 /* Turn OFF */
      str   r1, [r0, #GPCLR0]
```

Check GPIO2 (button)

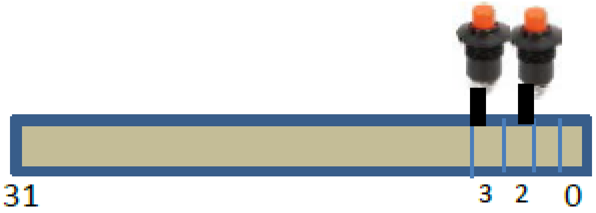
(for GPIO3 use #0b01000)

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

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

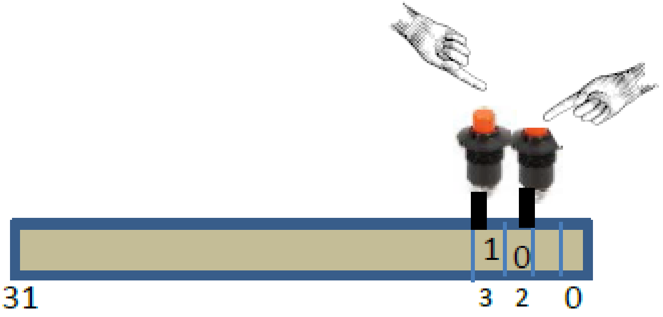
3F200034

(GPLEV0)



3F200034

(GPLEV0)



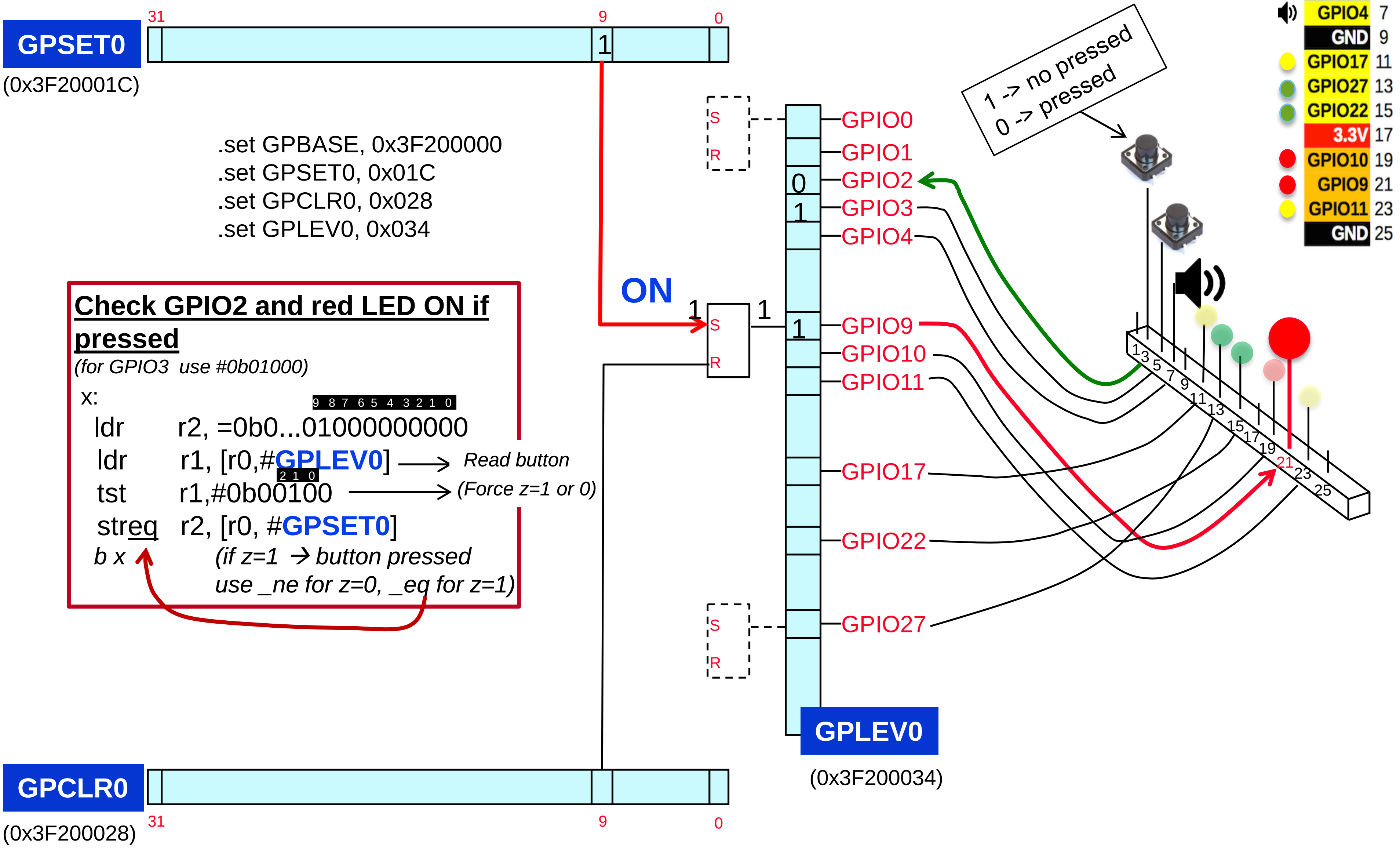
A very basic routine to copy the content of this port in the register r8 (for example) is:

```
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 red led



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

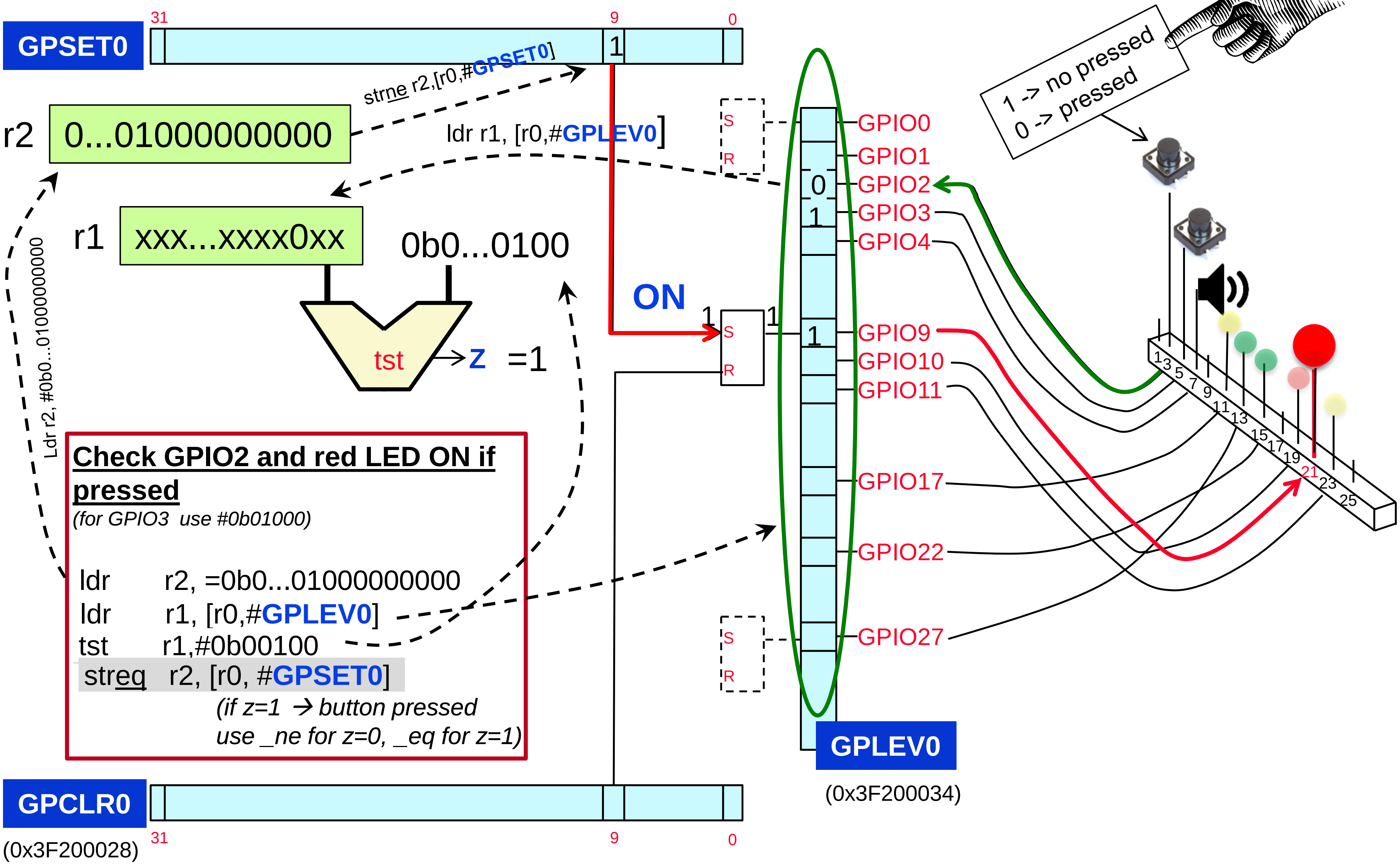
Check GPIO2 and red LED ON if pressed

(for GPIO3 use #0b01000)

```
x:
                               9 8 7 6 5 4 3 2 1 0
  ldr    r2, =0b0...01000000000
  ldr    r1, [r0,#GPLEV0]  ---> Read button
                 2 1 0
  tst    r1,#0b00100      ---> (Force z=1 or 0)
  streq  r2, [r0, #GPSET0]
  b x          (if z=1 → button pressed
               use _ne for z=0, _eq for z=1)
```

	Pin
3.3V	1
GPIO2	3
GPIO3	5
GPIO4	7
GND	9
GPIO17	11
GPIO27	13
GPIO22	15
3.3V	17
GPIO10	19
GPIO9	21
GPIO11	23
GND	25

Check button (GPIO2) and turn on red led



Check GPIO2 and red LED ON if pressed

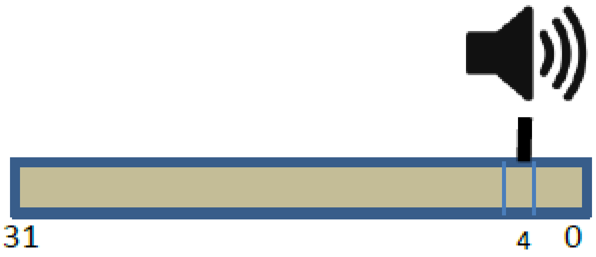
(for GPIO3 use #0b01000)

```
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)

3F20001C

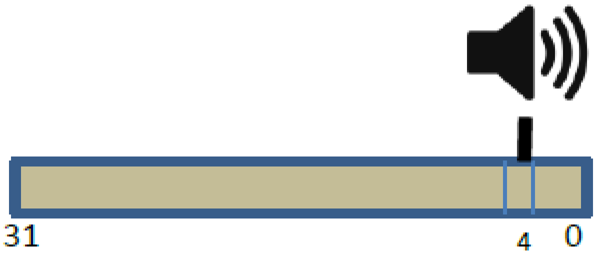
(GPSET0)



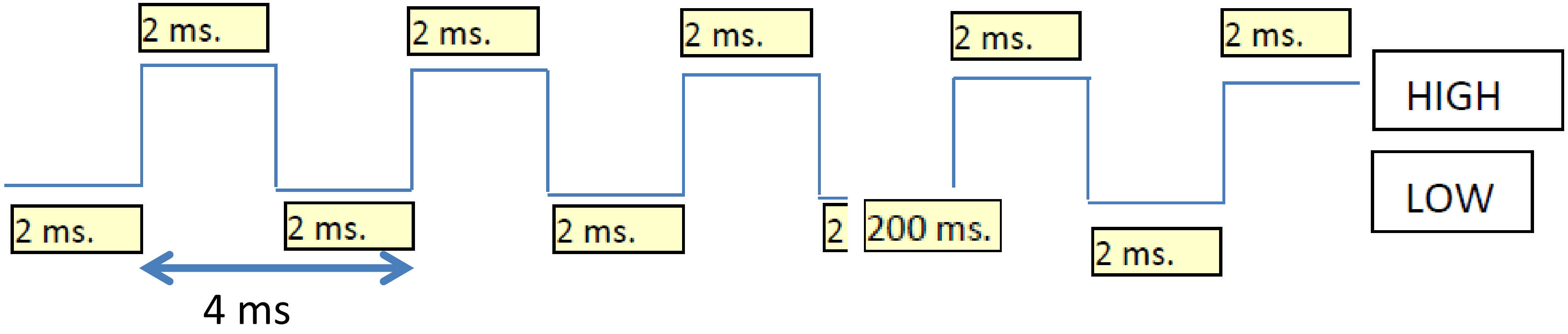
Port *for* HIGH

3F200028

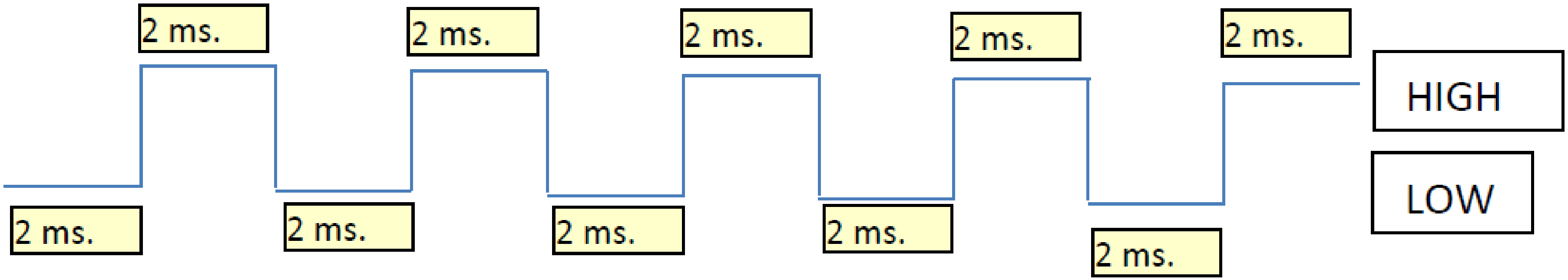
(GPCLR0)



Port *for* LOW



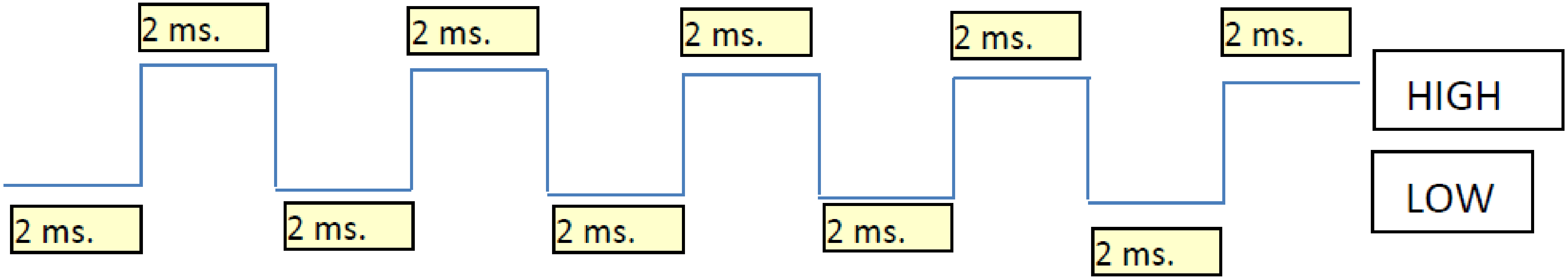
F= 250 Hz → CC= 1/250 s. = 4 ms



A very basic routine to produce a sound is:

```
        ldr  r0, =0x3F20001C     /*r0 contents the port address for HIGH */
        ldr  r2, =0x3F200028     /*r2 contents the port address for LOW */
        ldr  r1, =0x010          /* r1= 0x20= 00… 0001 0000   → bit4=1*/

loop:   str  r1,[r0]             /* HIGH */
        BL   wait                /* Routine for waiting 200 ms. */
        str  r1,[r2]             /* LOW */
        BL   wait                /* Routine for waiting 200 ms. */
        B    loop
```



A very basic routine to produce a sound is:

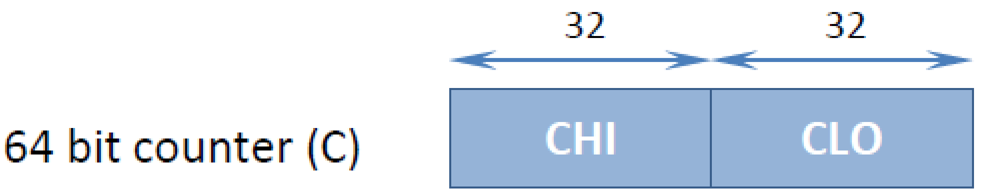
```
        ldr  r0, =0x3F20001C
        ldr  r2, =0x3F200028
        ldr  r1, =0x010

loop:   str  r1,[r0]
        BL   wait
        str  r1,[r2]
        BL   wait
        B    loop
```

```
        ldr  r0, =GPBASE
        ldr  r1, =0x010


loop:   str  r1,[r0,#GPSET0]         /* HIGH */
        BL   wait
        str  r1,[r0,#GPCLR0]         /* LOW */
        BL   wait
        B    loop
```

Clock rate: 1 Mhz
Period: 1 µs

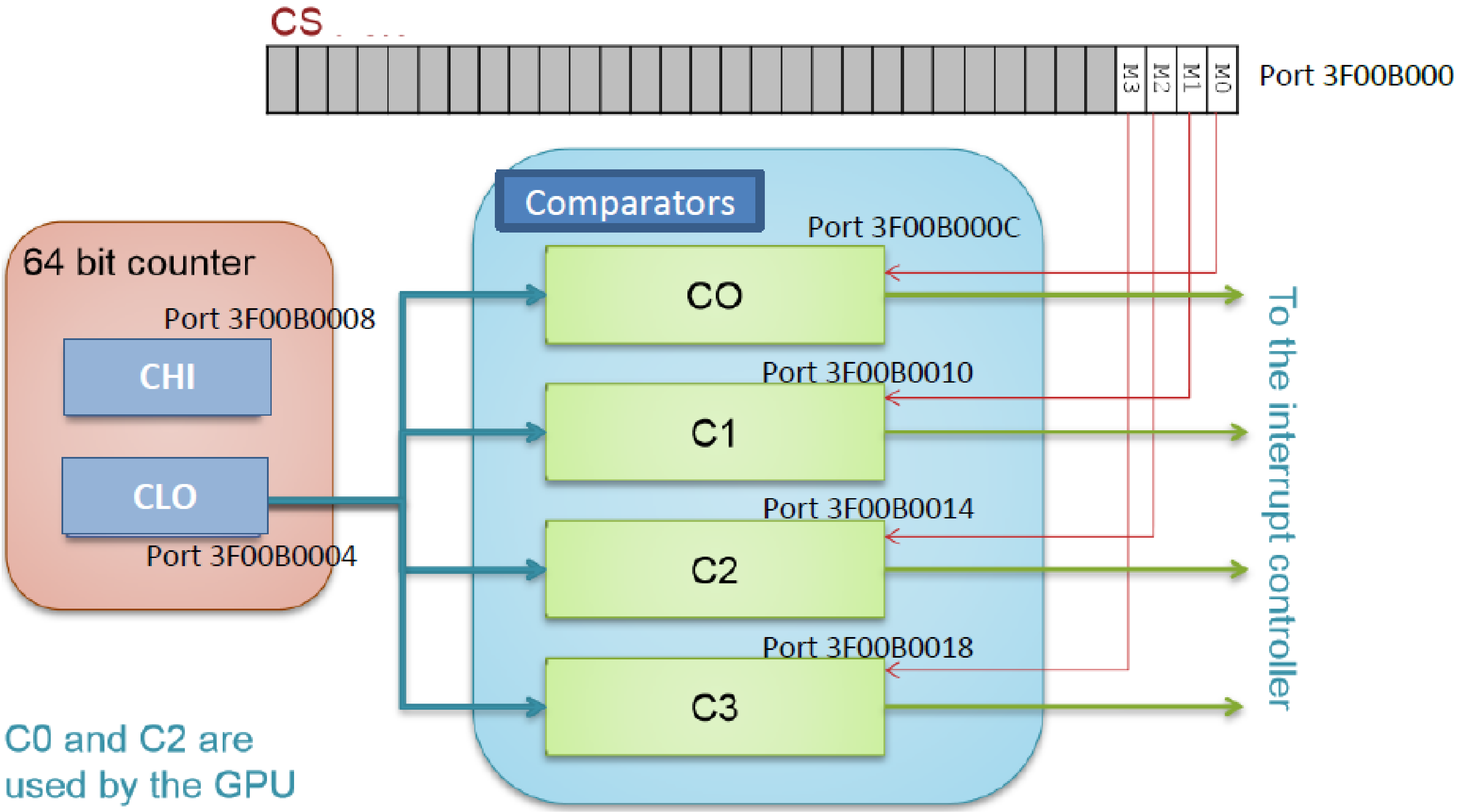

64 bit counter (C)

32

32

CHI

CLO


CS

M3

M2

M1

M0

Port 3F00B000

Comparators

Port 3F00B000C

C0

Port 3F00B0010

C1

Port 3F00B0014

C2

Port 3F00B0018

C3

To the interrupt controller

64 bit counter

Port 3F00B0008

CHI

CLO

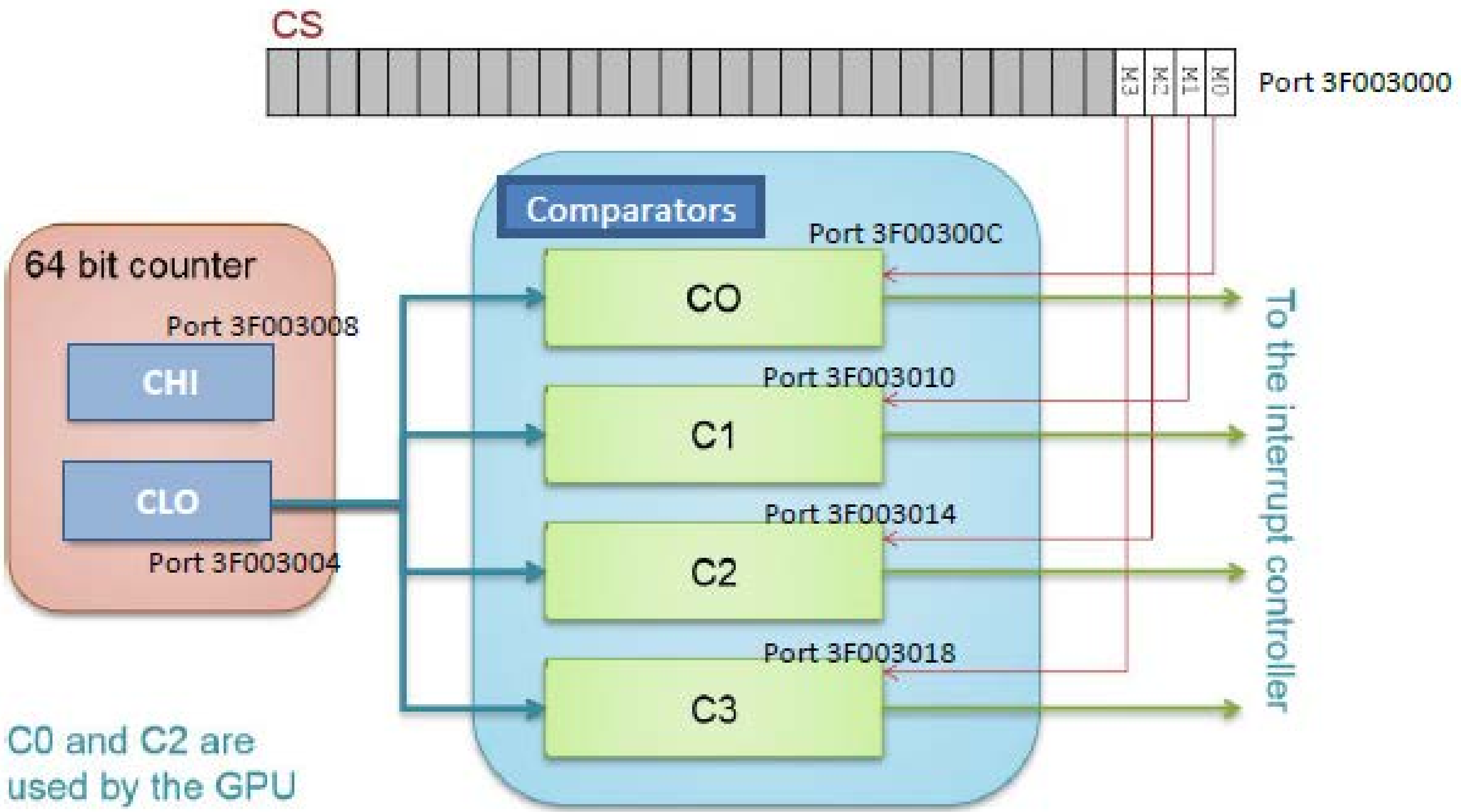
Port 3F00B0004

C0 and C2 are used by the GPU

wait 2 ms = 2000 µs needed to generate a sound of 250 Hz

```
       ldr  r0, =GPBASE
       ldr  r1, =0x010
loop:  str  r1,[r0,#GPSET0]  /* HIGH */
       BL   wait
       str  r1,[r0,#GPCLR0] /* LOW */
       BL   wait
       B    loop
```

```
wait: ldr r7, =0x3F003004 /* r7=0x3F003004 (address of counter CLO) , ldr r7,=STBASE*/
      ldr r3,[r7]          /* Read the value of the counter,  ldr r3,[r0,#STCLO */
      ldr r4, =2000 /* r4= 2 000 µs */
      add r4, r3, r4  /* Adding 2 000 µs to the current count to get the final count*/
ret1: ldr r3,[r7]      /* Read the current count, ldr r3,[r0,#STCLO */
      cmp  r3,r4      /* Comparing current count with the final count */
      blt ret1
      bx   lr   /* returning to the main program after 2 000 µs*/
```



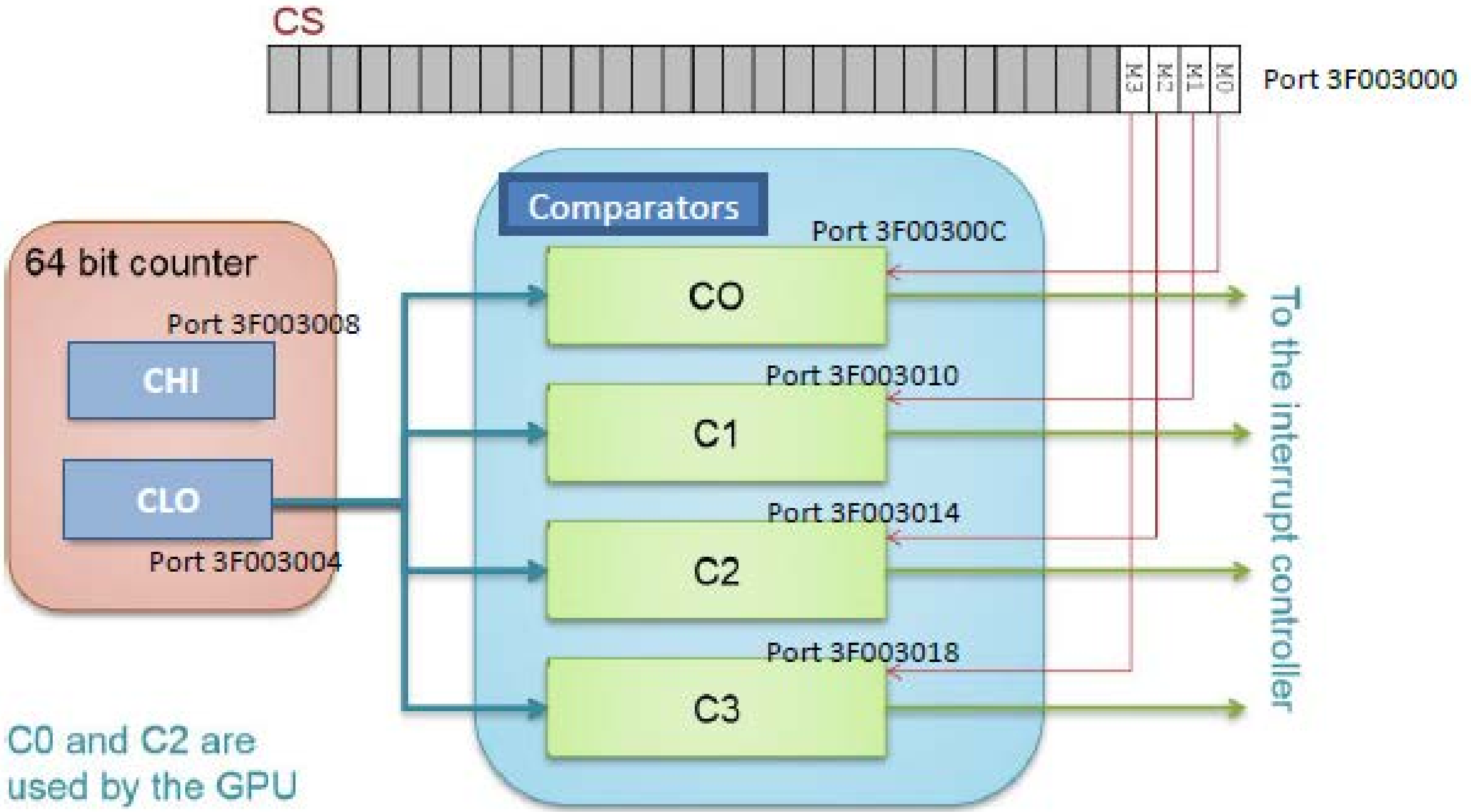
wait 2 ms = 2000 µs needed to generate a sound of 250 Hz

```
       ldr  r0, =GPBASE
       ldr  r1, =0x010
loop:  str  r1,[r0,#GPSET0]  /* HIGH */
       BL   wait
       str  r1,[r0,#GPCLR0] /* LOW */
       BL   wait
       B    loop
```

```
wait: ldr r7, =0x3F003004 /
      ldr r3,[r7]           /* I
      ldr r4, =2000 /* r4= 2
      add r4, r3, r4  /* Addi
ret1: ldr r3,[r7]       /* Read
      cmp  r3,r4      /* Comp
      blt ret1
      bx   lr   /* returning to
```

```
wait:     ldr r7,=STBASE
          ldr r3,[r7,#STCLO]
          ldr r4, =2000
          add r4, r3, r4
ret1:     ldr r3,[r7,#STCLO]
          cmp  r3,r4
          blt ret1
          bx   lr
```

```
BASE*/
*/
```



Using subroutines → use PUSH and POP instructions → Initialize Stack

```
/* Stack init for SVC mode  */
            mov     r0, #0b11010011
            msr     cpsr_c, r0
            mov     sp, #0x8000000
```

Using subroutines → use PUSH and POP instructions → Initialize Stack

```
/* Basic skeleton for programs using ports (without  interrupts) */
.include "configuration.inc"
.include "symbolic.inc"
/* Stack init for SVC mode                    */
        mov     r0, #0b11010011
        msr     cpsr_c, r0
        mov     sp, #0x8000000
/* Continue my program here */


end:   b end
```

Basic skeleton from now on

```
wait:       PUSH {r3,r4,r7}
            ldr r7,=STBASE
            ldr r3,[r7,#STCLO]
            ldr r4, =2000
            add r4, r3, r4
ret1:       ldr r3,[r7,#STCLO]
            cmp  r3,r4
            blt ret1
            POP {r3,r4,r7}
            bx   lr
```

New Subroutine *wait*