Linnéuniversitetet
Kalmar Växjö

Linnaeus University

1DT301 - Computer Technology 1
Assignment 4

Group number: Group I
Group member: Jiaxin Wang, Zejian Wang
Student ID: jw223jc@student.lnu.se
zw222bb@student.lnu.se

Linnéuniversitetet
Kalmar Växjö
Faculty of Te
Department

Task1

```
@
@ Assembler program to flash one LED controlled by two buttons
@ connected to the Raspberry Pi GPIO writing to the registers
@ directly.
@

#include "hardware/regs/addressmap.h"
#include "hardware/regs/sio.h"
#include "hardware/regs/io_bank0.h"
#include "hardware/regs/pads_bank0.h"

            .EQU LED_PIN, 0
            .EQU BUTTON_1, 1
            .EQU BUTTON_2, 2

.thumb_func
.global main                    @ Provide program starting address

            .align  4   @ necessary alignment
main:

MOV         R0,         #LED_PIN        @ Initialize the LED and set it to output
            BL          gpioinit
            MOV         R0, #BUTTON_1   @ Initialize BUTTON_1 and set it to input
            BL          gpioinit
            MOV         R0, #BUTTON_2   @ Initialize BUTTON_2 and set it to input
            BL          gpioinit

button_1:
            MOV R0, #BUTTON_1
            BL gpio_get
            CMP R0, #1      @ compare the value in R0 with 1
            BNE led_off     @ if not equal to 1, branch to led_off
            B button_2      @ if equal to 1, branch to button_2
button_2:
            MOV R0, #BUTTON_2
            BL gpio_get
            CMP R0, #1      @ compare the value in R0 with 1
            BNE led_on      @ if not equal to 1, branch to led_on
            B button_1      @ if equal to 1, branch to button_1


led_on:
            MOV         R0, #LED_PIN            @ turn on the LED
            BL          gpio_on
            B           button_2
            BX          lr

led_off:
            MOV         R0, #LED_PIN            @ turn off the LED
            BL          gpio_off
            B           button_1
            BX          lr

@ Initialize the GPIO to SIO. r0 = pin to init.

gpioinit:
@ Initialize the GPIO
            MOV         R3, #1
            LSL         R3, R0                  @ shift over to pin position
            LDR         R2, gpiobase             @ address we want
            STR         R3, [R2, #SIO_GPIO_OE_SET_OFFSET]
```

```
            STR         R3, [R2, #SIO_GPIO_OUT_CLR_OFFSET]

@ Enable input and output for the pin
            LDR         R2, padsbank0
            LSL         R3, R0, #2              @ pin * 4 for register address
            ADD         R2, R3                  @ Actual set of registers for pin
            MOV         R1, #PADS_BANK0_GPIO0_IE_BITS
            LDR         R4, setoffset
            ORR         R2, R4
            STR         R1, [R2, #PADS_BANK0_GPIO0_OFFSET]

@ Set the function number to SIO.
            LSL         R0, #3                  @ each GPIO has 8 bytes of registers
            LDR         R2, iobank0 @ address we want
            ADD         R2, R0                  @ add the offset for the pin number
            MOV         R1, #IO_BANK0_GPIO3_CTRL_FUNCSEL_VALUE_SIO_3
            STR         R1, [R2, #IO_BANK0_GPIO0_CTRL_OFFSET]
            BX          LR

@ Turn on a GPIO pin.
gpio_on:
            MOV         R3, #1
            LSL         R3, R0      @ shift over to pin position
            LDR         R2, gpiobase            @ address we want
            STR         R3, [R2, #SIO_GPIO_OUT_SET_OFFSET]
            BX          LR

@ Turn off a GPIO pin.
gpio_off:
            MOV         R3, #1
            LSL         R3, R0      @ shift over to pin position
            LDR         R2, gpiobase            @ address we want
            STR         R3, [R2, #SIO_GPIO_OUT_CLR_OFFSET]
            BX          LR

@ Get the state of a push button
gpio_get:
            MOV R3, #1
            LSL R3, R0  @ shift over to pin position
            LDR R2, gpiobase        @ address we want
            @ load the address we want in R1
            LDR R1, [R2, #SIO_GPIO_IN_OFFSET]
            @ compare R3 and R1 by using AND, store it in R3
            AND R3, R3, R1
            @ shift right and return one bit digit, storing it in R0
            LSR R3, R3, R0
            MOV R0, R3
            BX          LR


                        .align  4   @ necessary alignment


gpiobase:   .word       SIO_BASE     @ base of the GPIO registers
iobank0:    .word       IO_BANK0_BASE @ base of io config registers
padsbank0:  .word       PADS_BANK0_BASE
setoffset:  .word       REG_ALIAS_SET_BITS
```

Task2

```
@
@ Assembler program to flash four LEDs connnected to the
@ Raspberry Pi GPIO using timer interrupts to show 0000 to
@ 1111 in hexadecimal mode.
@

#include "hardware/regs/addressmap.h"
#include "hardware/regs/sio.h"
#include "hardware/regs/timer.h"
#include "hardware/regs/io_bank0.h"
#include "hardware/regs/pads_bank0.h"
#include "hardware/regs/m0plus.h"

            .EQU RESET_BUTTON, 0
            .EQU LED_PIN1, 1
            .EQU LED_PIN2, 2
            .EQU LED_PIN3, 3
            .EQU LED_PIN4, 4

            .EQU alarm0_isr_offset, 0x40

.thumb_func             @ Needed since SDK uses BX to call us
.global main                    @ Provide program starting address

            .align  4   @ necessary alignment
main:
            BL          stdio_init_all          @ initialize uart or usb

@ Init the reset button and four pins for the four LEDs as output
            MOV         R0, #RESET_BUTTON
            BL          gpioinit
            MOV         R0, #LED_PIN1
            BL          gpioinit
            MOV         R0, #LED_PIN2
            BL          gpioinit
            MOV         R0, #LED_PIN3
            BL          gpioinit
            MOV         R0, #LED_PIN4
            BL          gpioinit

            BL          set_alarm0_isr          @ set the interrupt handler
            LDR         R0, alarmtime           @ load the time to sleep
            BL          set_alarm0  @ set the first alarm

            MOV         R7, #0                  @ counter
loop:
            LDR         R0, =printstr           @ string to print
            MOV         R1, R7                  @ counter
            BL          printf                  @ print counter
            MOV         R0, #1                  @ add 1
            ADD         R7, R0                  @   to counter

    B       loop        @ loop forever

set_alarm0:
            @ Set the next alarm on alarm 0
            @ R0 is the length of the alarm

            @ Enable timer 0 interrupt
            LDR         R2, timerbase
            MOV         R1, #1                  @ for alarm 0
            STR         R1, [R2, #TIMER_INTE_OFFSET]
```

```
            @ Set alarm
            LDR         R1, [R2, #TIMER_TIMELR_OFFSET]
            ADD         R1, R0
            STR         R1, [R2, #TIMER_ALARM0_OFFSET]

            BX          LR

.thumb_func @ necessary for interrupt handlers
@ Alarm 0 interrupt handler and state machine.
alarm_isr:
            PUSH        {LR}        @ calls other routines
            @ Clear the interrupt
            LDR         R2, timerbase
            MOV         R1, #1                  @ for alarm 0
            STR         R1, [R2, #TIMER_INTR_OFFSET]

            @ Disable/enable LEDs based on state
            LDR         R2, =state @ load address of state
            LDR         R3, [R2]   @ load value of state
            MOV         R0, #1
            ADD         R3, R0         @ increment state
            STR         R3, [R2]   @ save state
step1:      MOV         R1, #1         @ case state == 1
            CMP         R3, R1
            BNE         step2          @ not == 1 check next
            MOV         R0, #LED_PIN1
            BL          gpio_off
            MOV         R0, #LED_PIN2
            BL          gpio_off
            MOV         R0, #LED_PIN3
            BL          gpio_off
            MOV         R0, #LED_PIN4
            BL          gpio_off
            B           finish
step2:      MOV         R1, #2      @ case state == 2
            CMP         R3, R1
            BNE         step3       @ not == 2 then case next
            MOV         R0, #LED_PIN1
            BL          gpio_on
            MOV         R0, #LED_PIN2
            BL          gpio_off
            MOV         R0, #LED_PIN3
            BL          gpio_off
            MOV         R0, #LED_PIN4
            BL          gpio_off
            B           finish
step3:      MOV         R1, #3      @ case state == 3
            CMP         R3, R1
            BNE         step4       @ not == 3 then case next
            MOV         R0, #LED_PIN1
            BL          gpio_off
            MOV         R0, #LED_PIN2
            BL          gpio_on
            MOV         R0, #LED_PIN3
            BL          gpio_off
            MOV         R0, #LED_PIN4
            BL          gpio_off
            B           finish
step4:      MOV         R1, #4      @ case state == 4
            CMP         R3, R1
            BNE         step5       @ not == 4 then case next
            MOV         R0, #LED_PIN1
            BL          gpio_on
            MOV         R0, #LED_PIN2
```

```
            BL          gpio_on
            MOV         R0, #LED_PIN3
            BL          gpio_off
            MOV         R0, #LED_PIN4
            BL          gpio_off
            B           finish
step5:      MOV         R1, #5      @ case state == 5
            CMP         R3, R1
            BNE         step6       @ not == 5 then case next
            MOV         R0, #LED_PIN1
            BL          gpio_off
            MOV         R0, #LED_PIN2
            BL          gpio_off
            MOV         R0, #LED_PIN3
            BL          gpio_on
            MOV         R0, #LED_PIN4
            BL          gpio_off
            B           finish
step6:      MOV         R1, #6      @ case state == 6
            CMP         R3, R1
            BNE         step7       @ not == 6 then case next
            MOV         R0, #LED_PIN1
            BL          gpio_on
            MOV         R0, #LED_PIN2
            BL          gpio_off
            MOV         R0, #LED_PIN3
            BL          gpio_on
            MOV         R0, #LED_PIN4
            BL          gpio_off
            B           finish
step7:      MOV         R1, #7      @ case state == 7
            CMP         R3, R1
            BNE         step8       @ not == 7 then case next
            MOV         R0, #LED_PIN1
            BL          gpio_off
            MOV         R0, #LED_PIN2
            BL          gpio_on
            MOV         R0, #LED_PIN3
            BL          gpio_on
            MOV         R0, #LED_PIN4
            BL          gpio_off
            B           finish
step8:      MOV         R1, #8      @ case state == 8
            CMP         R3, R1
            BNE         step9       @ not == 8 then case next
            MOV         R0, #LED_PIN1
            BL          gpio_on
            MOV         R0, #LED_PIN2
            BL          gpio_on
            MOV         R0, #LED_PIN3
            BL          gpio_on
            MOV         R0, #LED_PIN4
            BL          gpio_off
            B           finish
step9:      MOV         R1, #9      @ case state == 9
            CMP         R3, R1
            BNE         step10      @ not == 9 then case next
            MOV         R0, #LED_PIN1
            BL          gpio_off
            MOV         R0, #LED_PIN2
            BL          gpio_off
            MOV         R0, #LED_PIN3
            BL          gpio_off
            MOV         R0, #LED_PIN4
            BL          gpio_on
```

```
            B           finish
step10:     MOV         R1, #10     @ case state == 10
            CMP         R3, R1
            BNE         step11      @ not == 10 then case next
            MOV         R0, #LED_PIN1
            BL          gpio_on
            MOV         R0, #LED_PIN2
            BL          gpio_off
            MOV         R0, #LED_PIN3
            BL          gpio_off
            MOV         R0, #LED_PIN4
            BL          gpio_on
            B           finish
step11:     MOV         R1, #11     @ case state == 11
            CMP         R3, R1
            BNE         step12      @ not == 11 then case next
            MOV         R0, #LED_PIN1
            BL          gpio_off
            MOV         R0, #LED_PIN2
            BL          gpio_on
            MOV         R0, #LED_PIN3
            BL          gpio_off
            MOV         R0, #LED_PIN4
            BL          gpio_on
            B           finish
step12:     MOV         R1, #12     @ case state == 12
            CMP         R3, R1
            BNE         step13      @ not == 12 then case next
            MOV         R0, #LED_PIN1
            BL          gpio_on
            MOV         R0, #LED_PIN2
            BL          gpio_on
            MOV         R0, #LED_PIN3
            BL          gpio_off
            MOV         R0, #LED_PIN4
            BL          gpio_on
            B           finish
step13:     MOV         R1, #13     @ case state == 13
            CMP         R3, R1
            BNE         step14      @ not == 13 then case next
            MOV         R0, #LED_PIN1
            BL          gpio_off
            MOV         R0, #LED_PIN2
            BL          gpio_off
            MOV         R0, #LED_PIN3
            BL          gpio_on
            MOV         R0, #LED_PIN4
            BL          gpio_on
            B           finish
step14:     MOV         R1, #14     @ case state == 14
            CMP         R3, R1
            BNE         step15      @ not == 14 then case next
            MOV         R0, #LED_PIN1
            BL          gpio_on
            MOV         R0, #LED_PIN2
            BL          gpio_off
            MOV         R0, #LED_PIN3
            BL          gpio_on
            MOV         R0, #LED_PIN4
            BL          gpio_on
            B           finish
step15:     MOV         R1, #15     @ case state == 15
            CMP         R3, R1
            BNE         step16      @ not == 15 then case else
            MOV         R0, #LED_PIN1
```

```
            BL          gpio_off
            MOV         R0, #LED_PIN2
            BL          gpio_on
            MOV         R0, #LED_PIN3
            BL          gpio_on
            MOV         R0, #LED_PIN4
            BL          gpio_on
            B           finish
step16:     MOV         R0, #LED_PIN1           @ case else
            BL          gpio_on
            MOV         R0, #LED_PIN2
            BL          gpio_on
            MOV         R0, #LED_PIN3
            BL          gpio_on
            MOV         R0, #LED_PIN4
            BL          gpio_on
            MOV         R3, #15         @ set state back to 15
            LDR         R2, =state @ load address of state
            STR         R3, [R2]    @ save state == 15

finish:

            MOV R0, #RESET_BUTTON
            BL gpio_get                 @ get the state of the reset button
            CMP R0, #1                  @ compare it with value 0
            BEQ reset   @ if equal to 1, means the button is pressed, branch to reset

            LDR         R0, alarmtime           @ sleep time
            BL          set_alarm0  @ set next alarm
            POP         {PC}                    @ return from interrupt

reset:
            MOV         R3, #0          @ set state back to 0
            LDR         R2, =state @ load address of state
            STR         R3, [R2]    @ save state
            B step1


set_alarm0_isr:
            @ Set IRQ Handler to our routine
            LDR         R2, ppbbase
            LDR         R1, vtoroffset
            ADD         R2, R1
            LDR         R1, [R2]
            MOV         R2, #alarm0_isr_offset @ slot for alarm 0
            ADD         R2, R1
            LDR         R0, =alarm_isr
            STR         R0, [R2]

            @ Enable alarm 0 IRQ (clear then set)
            MOV         R0, #1      @ alarm 0 is IRQ0
            LDR         R2, ppbbase
            LDR         R1, clearint
            ADD         R1, R2
            STR         R0, [R1]
            LDR         R1, setint
            ADD         R1, R2
            STR         R0, [R1]

            BX          LR

@ Initialize the GPIO to SIO. r0 = pin to init.
gpioinit:
@ Initialize the GPIO
            MOV         R3, #1
```

```
            LSL         R3, R0      @ shift over to pin position
            LDR         R2, gpiobase            @ address we want
            STR         R3, [R2, #SIO_GPIO_OE_SET_OFFSET]
            STR         R3, [R2, #SIO_GPIO_OUT_CLR_OFFSET]

@ Enable input and output for the pin
            LDR         R2, padsbank0
            LSL         R3, R0, #2  @ pin * 4 for register address
            ADD         R2, R3                  @ Actual set of registers for pin
            MOV         R1, #PADS_BANK0_GPIO0_IE_BITS
            LDR         R4, setoffset
            ORR         R2, R4
            STR         R1, [R2, #PADS_BANK0_GPIO0_OFFSET]

@ Set the function number to SIO.
            LSL         R0, #3                  @ each GPIO has 8 bytes of registers
            LDR         R2, iobank0 @ address we want
            ADD         R2, R0                  @ add the offset for the pin number
            MOV         R1, #IO_BANK0_GPIO3_CTRL_FUNCSEL_VALUE_SIO_3
            STR         R1, [R2, #IO_BANK0_GPIO0_CTRL_OFFSET]
            BX          LR

@ Turn on a GPIO pin.
gpio_on:
            MOV         R3, #1
            LSL         R3, R0      @ shift over to pin position
            LDR         R2, gpiobase            @ address we want
            STR         R3, [R2, #SIO_GPIO_OUT_SET_OFFSET]
            BX          LR

@ Turn off a GPIO pin.
gpio_off:
            MOV         R3, #1
            LSL         R3, R0      @ shift over to pin position
            LDR         R2, gpiobase            @ address we want
            STR         R3, [R2, #SIO_GPIO_OUT_CLR_OFFSET]
            BX          LR

@ Get the state of a push button
gpio_get:
            MOV R3, #1
            LSL R3, R0                          @ shift over to pin position
            LDR R2, gpiobase        @ address we want
            LDR R1, [R2, #SIO_GPIO_IN_OFFSET] @ load the address we want in R1
            AND R3, R3, R1          @ compare R3 and R1 by using AND, store it in R3
            LSR R3, R3, R0          @ shift right and return one bit digit, storing it
in R0
            MOV R0, R3
            BX          LR

                        .align  4   @ necessary alignment
gpiobase:   .word       SIO_BASE     @ base of the GPIO registers
iobank0:    .word       IO_BANK0_BASE @ base of io config registers
padsbank0:  .word       PADS_BANK0_BASE
setoffset:  .word       REG_ALIAS_SET_BITS
timerbase:  .word       TIMER_BASE
ppbbase:    .word       PPB_BASE
vtoroffset: .word       M0PLUS_VTOR_OFFSET
clearint:   .word       M0PLUS_NVIC_ICPR_OFFSET
setint:                 .word       M0PLUS_NVIC_ISER_OFFSET
alarmtime:  .word       2000000
printstr:   .asciz      "Couting %d\n"

.data
state:                  .word       0
```