ECE 352 Lab 4: Prelab

Neo Lou and Ethan Fong

1. Read the following documents to learn how to interface with both the Lego controller and the timer:
    a. Lego Controller Device Documentation
    b. Timer Documentation
2. Since we will be using the stack in this lab, the stack pointer must be initialized (In Lab 3, the C Runtime Library did this for you). What is a good value to initialize it to, and why? (The computer's memory map may be useful.)

0x17fff80, since the stack grows to lower addresses and there is free memory here and we don't use many of the addresses before it. According to the memory map it's in the region of 64 MB SDRAM.

3. Answer the following questions about bit masking:
    a. Write assembly instruction(s) to set bit 11 **(assume 0 indexed)** of r2 to 1, without changing other bits

```
movia r3, 0x800 #1000 0000 0000

or r2, r2, r3
```

    b. Write assembly instruction(s) to set bit 8 **(assume 0 indexed)** of r2 to 0, without changing other bits

```
movia r3, 0xFFFFFEFF # … F 1110 F F

and r2, r2, r3
```

4. Answer the following questions, assuming the Lego controller is plugged into JP1:
    a. To properly configure the Lego controller I have to write the value 0x07f557ff (motors and sensors) to the direction register, which is located at address 0xFF200064 (DIR register) .

Bits 26 downto 21 and 18,16,14,12,10 downto 0 must be set to 1's **outputs**

Bits 31 downto 27,19,17,15,13,11 must be set to 0's **inputs**.

    b. To turn on motor 2 I have to set bits 4/5 (fwd/on) at memory location 0xFF200060

       to the value 00.

    c. Give an example of a 32-bit value that will enable sensor 1 for reading when written to the GPIO port: 0xffffefff. This 32-bit word will implicitly turn motor 2 off (on/off/unchanged/undefined/?).
    d. To check if sensor 1 is giving a valid data I have to read bit 13 from address 0xFF200060 and test if the bit has the value 0.

e. To read the current value of sensor 1, I have to read bits 27 to 30 from address 0xFF200060.

5. Answer the following questions about the Timer:
   a. Does the timer count up or down? Count down
   b. How do you initialize the timer's period? Set the periodl (base +8) and periodh (base+12) to a certain value. The timer then counts down from the period on a 100MHz clock

base+8	R/W	Periodl - lower 16 bits of Timeout period
base+12	R/W	Periodh - upper 16 bits of Timeout period

   c. How can you check if the timer has completed its period?

Continuously poll the timeout bit (base+0) (1 if timer has timed out - write 0 to this address to clear)

   d. Suppose the processor executes one instruction every clock cycle, and you start the timer at cycle 7, as shown in the table below. At what time does the next instruction execute?

Time (cycles)	Instruction
1	movia r3, 0xff202000
3	movi r2, 10000
4	stwio r2, 8(r3)
5	stwio r0, 12(r3) # 10000 cycle period
6	movi r2, 4
7	stwio r2, 4(r3) # Start timer
???	ldwio r2, 0(r3)

Could be cycle 8? [Or Cycle 10007, which corresponds to around 0.1 ms at 100Mhz]

6. Write the timer delay subroutine to be used in Part 2. This must be a proper subroutine that follows the NIOS II ABI

```
# DELAY Subroutine

#     counts down a period of N on a 100MHz clock speed. Returns when
done.
```

```
#     Takes in arguments: r4 = N

DELAY:                              # lets use caller saved registers since
its only a callee. r8-r15

movia r8, 0xFF202000    # r7 contains the base address for the timer

stwio r0, 0(r8)              # clear timer

stwio r4, 8(r8)          # set the period to be N clock cycles lower half

srli r4, r4, 16

stwio r4, 12(r8)        # set the period to be N clock cycles upper half

movui r11, 4

stwio r11, 4(r8)         # start the timer without continuing or interrupts

POLL:

ldwio r10, 0(r8)        # load potential time out bit

andi r10, r10, 0x1           # isolate bit 1

movia r12, 0x1

beq r12, r10, TIMEOUT   # is timer == 1? if it is then we have timed out

br POLL

TIMEOUT:

Ret
```

7. Write a simple test program that uses your timer subroutine to blink an LED with a period of 1 second. (Recall: Red LEDs are at location ff200000)

```
.global _start

_start:

movia sp, 0x17fff80          # Initialise stack pointer

movia r8, 0x0                # Initialise LED value at r8

movia r7, 0xff200000   # Initialise red LED address at r7

LOOP:
```

```
movia r4, 100000000    # Counts down a period of 1 second at 100mhz
PRE:
addi sp, sp, -4        # size of stack frame
stw ra, 0(sp)               # store ra
stw r8, 4(sp)               # store r8
call DELAY
POST:
ldw ra, 0(sp)               # recover regs
ldw r8, 4(sp)
addi sp, sp, 4              # remove stack frame
movia r9, 0x1               # xor mask
xor r8, r8, r9              # not operation on bit 1 of r8
stwio r8, 0(r7)             # write to led on -> off, off -> on
br LOOP                          # go again



# DELAY Subroutine
#     counts down a period of N on a 100MHz clock speed. Returns when
done.
#     Takes in arguments: r4 = N
DELAY:                           # lets use caller saved registers since
its only a callee. r8-r15
movia r8, 0xFF202000    # r7 contains the base address for the timer
stwio r0, 0(r8)             # clear timer
stwio r4, 8(r8)         # set the period to be N clock cycles lower half
srli r4, r4, 16
stwio r4, 12(r8))       # set the period to be N clock cycles upper half
```

```
movui r11, 4

stwio r11, 4(r8)         # start the timer without continuing or interrupts

POLL:

ldwio r10, 0(r8)        # load potential time out bit

andi r10, r10, 0x1            # isolate bit 1

movia r12, 0x1

beq r12, r10, TIMEOUT    # is timer == 1? if it is then we have timed out

br POLL

TIMEOUT:

ret
```

8. Write the assembly language programs for part 1 and part 2 described above. Your code must be well commented. Compile your code using the Altera Monitor program and fix any compilation errors.

Part 1 program:

In 1 loop:

Read both sensors, calculate tilt, turn motor accordingly

[s1 motor0 s2]

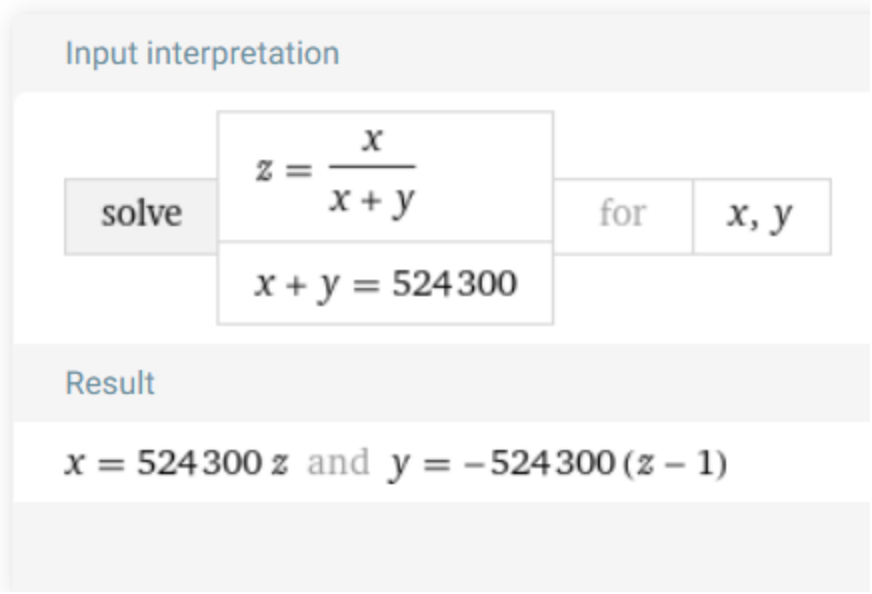
if s1 > s2, turn motor ccw
if s1 < s2, turn motor cw

Duty cycle = on/(on+off)

On+off = 524300

Solve for on and off:

$$z = \frac{x}{(x+y)}, \; x+y = 524300, \text{ solve for x and y in terms of z}$$



On = 524300*dutycycle%, off = 524300*(1-dutycycle%)

Part 2 program:

```
.equ ADDR_JP1, 0xFF200060   # Address GPIO JP1

.equ STACK_BEGIN, 0x17fff80  # Address of initial stack

.equ THRESHOLD, 1 # minimum difference between the sensors for the motor
to turn on

.equ NEG_THRESHOLD, -1  # minimum difference between the sensors for the
motor to turn on

.equ DUTY_CYCLE_TOT, 5243    # total number of duty cycles/100, on+off at
100MHz, 524300 in example

.equ DUTY_CYCLE, 50 # % out of 100

.equ DUTY_CYCLE_C, 50 # 1-DUTY_CYCLE%

.global _start

_start:

/*    Important register usage:

      R4 passing argument N into delay subroutine

      R8 address of GPIO, unchanged throughout

      R9 varying use, sensor 1 value, then direction of motor rotation
```

```
      R10 varying use, sensor 2 value

      R11 varying use, written to DR register to turn on motor and set
direction

      R12 ON duty cycles number

      R13 OFF duty cycles number

      R14 varying use, for calculating duty cycles, then stores negative
threshold

      R15 stores threshold

*/

movia sp, STACK_BEGIN        # Initialise stack pointer

LOOP:

movia r8, ADDR_JP1           # Initialise GPIO address at r8

movia r9, 0x07f557ff        # set direction registers to motor output
sensor input

stwio  r9, 4(r8)             # put in DIR

movia r14,  DUTY_CYCLE_TOT   # total duty cycle

mul r12, r14, DUTY_CYCLE     # number of cycles on

mul r13, r14, DUTY_CYCLE_C   # number of cycles off

movia r14,  NEG_THRESHOLD    # -threshold

movia r15,  THRESHOLD   # threshold

# LOOP:

SENSOR1:    # read sensor 1 and put in r9

movia  r9, 0xffffefff     # enable sensor 1, disable all motors

stwio  r9, 0(r8)

ldwio  r9,  0(r8)          # checking for valid data sensor 1

srli   r10,  r9, 13          # is valid if bit 13 == 0 for sensor 1

andi   r10,  r10, 0x1
```

```
bne    r0,  r10, SENSOR1        # wait for valid bit to be low: sensor 3
needs to be valid

srli   r9, r9, 27          # shift to the right by 27 bits so that 4-bit
sensor value is in lower 4 bits

andi   r9, r9, 0x0f          # mask

SENSOR2:    # read sensor 2 and put in r10

movia  r10, 0xffffbfff     # enable sensor 2, disable all motors

stwio  r10, 0(r8)

ldwio  r10,  0(r8)          # checking for valid data sensor 2

srli   r11,  r10, 15          # bit 15 is valid bit for sensor 2

andi   r11,  r11,0x1

bne    r0,  r11, SENSOR2        # wait for valid bit to be low: sensor 2
needs to be valid

srli   r10, r10, 27          # shift to the right by 27 bits so that 4-bit
sensor value is in lower 4 bits

andi   r10, r10, 0x0f        # mask

CALCULATE: # Calculate direction of motor rotation, store '10' in r9 if
cw, '00' if ccw.

sub r9, r9, r10 #r9-r10

bgt r9 , r15, ABOVETHRESHOLD

blt r9, r14, ABOVETHRESHOLD # difference < -threshold

BELOWTHRESHOLD:

Movia r9, 0x01    # motor off

br MOTORON

ABOVETHRESHOLD:

Blt r9, r0, RIGHT # r9-r10<0, r10>r9

LEFT:

Movia r9, 0x10    r9<=r10

br MOTORON
```

```
RIGHT:      # r9<r10

Movia r9, 0x00


MOTORON:

movia  r11, 0xfffffffc       # motor0 enabled (bit0=0), direction set to
forward (bit1=0)

Add r11, r11, r9 # make use of calculation

stwio  r11, 0(r8) # turn motor on

add r4, r0, r12  # ON duty cycles into parameter N

PRE:                         # duty cycles need to be saved in pre and
recovered in post

addi sp, sp, -4

stw ra, 0(sp)                # store ra

stw r13, 4(sp)               # store number of off duty cycles

call DELAY

POST:

ldw ra, 0(sp)                # recover regs

ldw r13, 4(sp)               # store number of off duty cycles

addi sp, sp, 4

MOTOROFF:

movia r4, r13    # number of off duty cycles

PRE2:

stw ra, 0(sp)                # store ra

call DELAY

POST2:

ldw ra, 0(sp)                # recover regs

movia  r12, 0xffffffff       # motor0 disabled (bit0=1)
```

```
stwio  r12, 0(r8) # turn motor off
br LOOP                      # go again


# DELAY Subroutine
#     counts down a period of N on a 100MHz clock speed. Returns when
done.
#     Takes in arguments: r4 = N
DELAY:                              # lets use caller saved registers since
its only a callee. r8-r15
movia r8, 0xFF202000    # r7 contains the base address for the timer
stwio r0, 0(r8)              # clear timer
stwio r4, 8(r8)          # set the period to be N clock cycles lower half
srli r4, r4, 16
stwio r4, 12(r8))        # set the period to be N clock cycles upper half
movui r11, 4
stwio r11, 4(r8)         # start the timer without continuing or interrupts
POLL:
ldwio r10, 0(r8)        # load potential time out bit
andi r10, r10, 0x1           # isolate bit 1
movia r12, 0x1
beq r12, r10, TIMEOUT   # is timer == 1? if it is then we have timed out
br POLL
TIMEOUT:
ret
```