Homework 3 Solution

1. **DSP Processor Concepts.** In a few sentences each, briefly describe the following and how/when it’s used in our lab.

   (a) Analog interface circuit
   Chip that does A/D and D/A to allow analog input and receive analog output. We initialize the AIC via various registers and access it by using trcv and tdxr and the appropriate interrupts.

   (b) Anti-aliasing (input) filter
   This low pass filter guarantees that the input signal (before sampling) is band limited. Ideally, the anti-aliasing filter is set less than half the sampling frequency to avoid aliasing. It is directly in the input path of the A/D converter, and is always used whenever we use analog input.

   (c) Smoothing (output) filter
   This low pass filter is on the output path of the D/A conversion. Sampling causes periodic copies of a signal’s spectrum to be repeated. A low pass is the equivalent of sinc reconstruction, necessary to recreate the analog signal.

   (d) Circular addressing
   Used with the +% suffix and the BK register, this allows a register to be accessed in a ‘loop’. We have used it for delaying an output signal in experiment B.

2. **Setting Sampling Rate.** Recall the following expressions,

\[
F_S = \frac{10 \times 10^6}{2AB} \quad (1)
\]
\[
F_{LP} = \frac{10 \times 10^6}{80A} \quad (2)
\]

modify the section of initialization code for the 'AC01 chip show below

```
...  
REG1 .set 112h ; Solution: change to 11Fh
REG2 .set 212h ; Solution: change to 210h
REG3 .set 300h
REG4 .set 405h
...  
```

to implement, as close as possible,

- sampling frequency, \( F_S \), of 10kHz and
- a low-pass filter cut-off frequency, \( F_{LP} \), of 4kHz

and because you may have rounded A and B, what is the resulting \( F_S \) and \( F_{LP} \)?
Solution:

\[ A = \frac{10 \times 10^6}{80 \times F_{LP}} = 31.1 \Rightarrow \text{round to 31 or 0x1F} \quad (3) \]

\[ B = \frac{10 \times 10^6}{2 \times 31 \times F_S} = 16.1 \Rightarrow \text{round to 16 or 0x10} \quad (4) \]

\[ F_S = \frac{10 \times 10^6}{2 \times 31 \times 16} = 10080.65 \text{ Hz} \quad (5) \]

\[ F_{LP} = \frac{10 \times 10^6}{80 \times 31} = 4032.23 \text{ Hz} \quad (6) \]

3. Sampling Rate.

(a) Many wireless LAN products today are based on the IEEE 802.11g standard. An IEEE 802.11g compliant transmission is about 20MHz wide. Sampling theorem tells us that we should sample this incoming signal at 40MHz or higher. Suppose that our DSP board can receive and transmit wireless signals, can we program the AIC to sample at 40MHz? Why or why not?

**One solution:**
According to the TI’s datasheet for our analog interface circuit, TLC320AIC01, it could not sample at such a high frequency.

**Another solution:**
Because A and B register must be integers, the highest \( F_S \) is achieved if both A and B are equal to 1. In such case, \( F_S = \frac{10 \times 10^6}{2 \times 1 \times 1} = 5 \times 10^6 \text{Hz} = 5 \text{MHz} \), which does not satisfy our requirement of 40MHz.

(b) Suppose that our AIC can sample at 40MHz. Recall that our DSP, the C542, runs at 40MHz and has 40MIPS. Suppose that we were to write a program to process the wireless signal, how many instruction cycles are available to process the current sample before the next sample arrives?

**Solution:**
DSP board sampling at 40MHz means that the Analog Interface Circuit is providing \( 40 \times 10^6 \) samples per second to the DSP. 40MIPS means that the DSP operates at \( 40 \times 10^6 \) instruction cycles per sec. Thus there is a sample ready for processing at each DSP cycle. Therefore, there is 1 instruction cycle available for our program.

4. Programming Exercise. We will modify Experiment B’s sample.asm to implement a half wave rectifier. In other words, whenever an input sample from \( \text{trcv} \) is negative, output 0; if positive, output that sample.

**Example:**

- Input sequence: 0 2 4 2 0 -2 -4 -2 0 2 4 2 0 ...
- Output sequence: 0 2 4 2 0 0 0 0 0 2 4 2 0 ...

```assembly
.mmxregs
.setsect "text", 0x500,0
.setsect "data", 0x800,1
.setsect "vectors", 0x180,0
.sect "vectors"
.copy "vectors.asm"

.data
```
temp .word 0 ; temp storage for positive numbers

.text

start: intm = 1 ; globally disable interrupts
call AC01INIT
pmst = #01a0h ; set up iptr
sp = #0ffah ; init stack pointer.
a = #0 ; initialize a to zero
imr = #280h

; ***** initialize pointers here if you need any
AR0 = #temp ; point to temp storage

intm = 0 ; ready to rcv int’s

wait  nop
goto wait

; --------------- Receive Interrupt Routine ---------------
XINT:  b = trcv ; load acc b with input

; ***** your code starts here
nop
; because b = trcv doesn’t sign extend
; cmps would always be comparing input samples to zero.
cmps(b,*AR0) ; compare input to zero
b = *AR0 ; read the positive number
; ***** your code ends here

b = #0FFFCh & b
tdxr = b ; transmit the data.
return_enable

; ----------------------- end ISR -----------------------------
.copy "ac01init.asm"
.end