Jonathan W. Valvano May 14, 2009, 2-5pm

(6) Question 1. Consider how the ACK bit is used in a CAN network.

Part a) The transmitter outputs a recessive 1 and if a receiver gets an incorrect LRC, it drives a dominate 0.

Part b) The ACK bit high means all receivers got a correct LRC and the message is valid.

Part c) The ACK bit low means at least one receiver got an incorrect LRC, the message is invalid, and the transmitter will resend the message.

(4) Question 2. A **Power Budget** is a quick first order calculation that gives you a ballpark figure of the "total average current" supported by your power source. From the system specifications we are given how long the system must operate without replacing batteries, $\mathbf{t_{life}}$ in hours. From the battery datasheet we determine the storage capacity of the battery, **E** in mA-hours. Be aware, however, that for many batteries the storage capacity depends also on current.

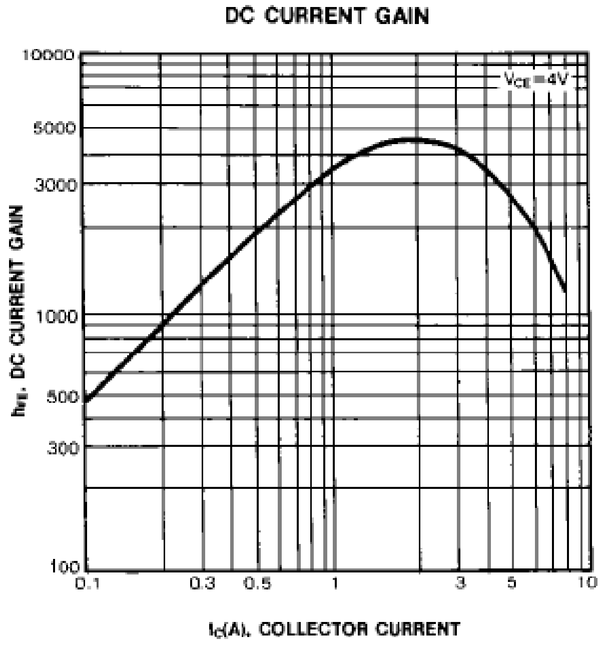
Average Current must be less than **E**/ $\mathbf{t_{life}}$

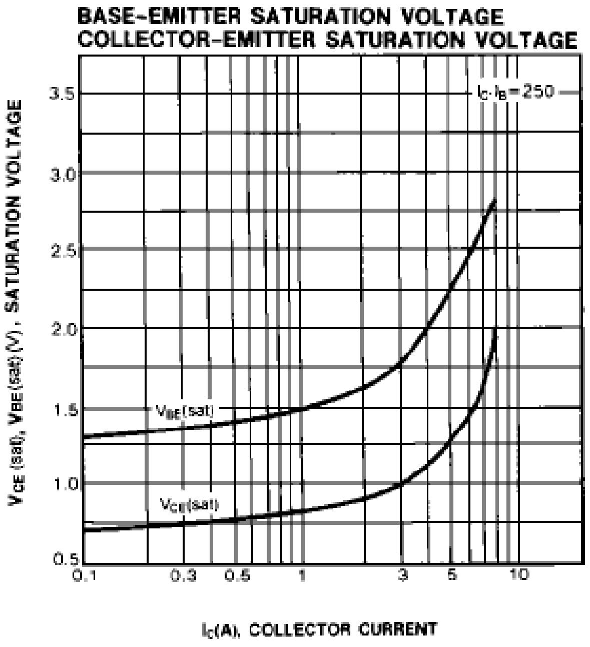
(15) Question 3.

Part a) I'd use a TIP120 because the maximum I_{CE} (3A) is larger than the 200 mA needed. (We could also have used the TIP29, TIP31 or TIP41). A MOSFET would be ok too.

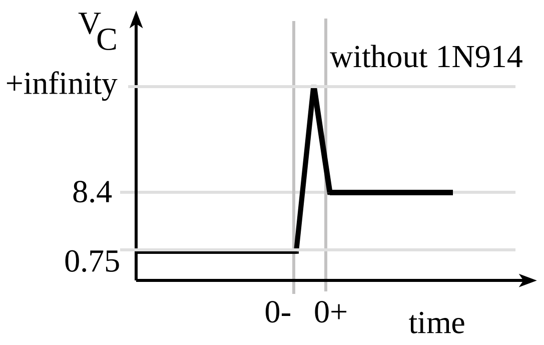
Part b) We use the data sheet of the TIP120 to find h_{fe}=3000, V_{be}=1.3V, V_{ce}=0.75V at 0.2A.

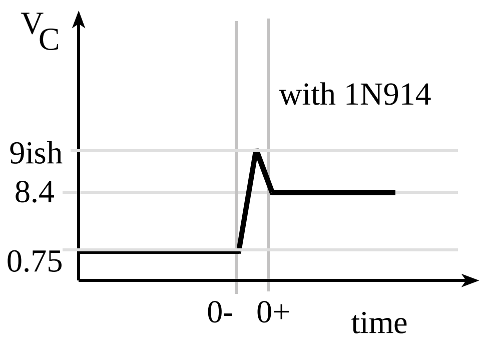
$I_b = I_c/3000 = (V_{OH}-V_{BE})/R_b$, $0.2A/3000 = (4.44-1.3)/ R_b$, $R_b = 15000(4.44-1.3) = 47k\Omega$. I will choose a value of 4.7 kΩ, then test it, because the 3000 is only approximate.



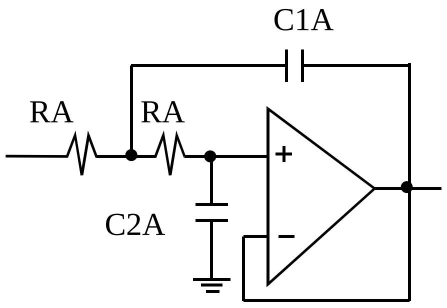


Part c) The current goes from 200 mA to 0 instantaneously. This dI/dt is negative infinity. Without the diode, this causes the voltage across the inductor to be negative infinity (V=Ldi/dt). The voltage at V_C becomes positive infinity. With the diode, 4ns after V_C goes above 8.4V, the 1N914 becomes forward biased and will short any current to the 8.4V battery clamping voltage to 8.4V (or a little bit above 8.4V). The 1N914 is called a snubber diode and is chosen because it is fast (4ns).





(10) Question 4. Design a two-pole analog low-pass filter with a cutoff frequency of 10 Hz. Notice that C1 is two 0.22µF capacitors in parallel, making a 0.44 µF.



initial starting point		
fc (Hz)	1	
R (kohm)	10	
C1 (µF)	141.4	
C2 (µF)	70.7	
first design step is to select the cutoff		
fc (Hz)	10	fill this in
RA (kohm)	10	same as initial R
C1A (µF)	2.2505	is 141.4/(2•π•fc)
C2A (µF)	1.1252	is 70.7/(2•π•fc) or 0.5•C1A
second design step is to choose convenient Capacitor values		
fc (Hz)	10	same as previous fc
RB (kohm)	51.147	new value to match exact fc
C1B (µF)	0.44	fill this in
C2B (µF)	0.22	is 0.5•C1B
third design step is to choose a convenient resistor value		
fc (Hz)	10.029	cutoff based on these convenient values
RC (kohm)	51.000	fill this value in
C1C (µF)	0.44	same as C1B
C2C (µF)	0.22	same as C2B

(25) Question 5. First, add an entry into the TCB defining the function to call

```
struct TCB{
  struct TCB *Next;            // Link to Next TCB
  unsigned char *StackPt;      // Stack Pointer
  void (*TheFunction)(void);   // user program
  unsigned char MoreStack[101]; // additional stack
  unsigned char InitialCCR;
  unsigned char InitialRegB;
  unsigned char InitialRegA;
  unsigned short InitialRegX;
  unsigned short InitialRegY;
  void (*InitialPC)(void);
};
```

Next we define a private system function that calls the user program over and over.

```
void UserMain(void){
  for(;;){
    RunPt->TheFunction(); // call user program
  }
}
```

During initialization we set the `TheFunction` parameter in the TCB and attach the `UserMain`.

```
short OS_AddThread(void(*fp)(void)){
  if(NumThread >= MAX_THREADS){
    return 0;  // structure is full
  }
  if(NumThread){
    SystemTCB[NumThread-1].Next = &SystemTCB[NumThread];
  }
  SystemTCB[NumThread].StackPt = &SystemTCB[NumThread].InitialCCR;
  SystemTCB[NumThread].TheFunction = fp;        // user code
  SystemTCB[NumThread].InitialCCR = 0x40;       // I bit clear
  SystemTCB[NumThread].InitialPC = &UserMain;   // Initial PC
  SystemTCB[NumThread].Next = &SystemTCB[0];    // link
  NumThread++;
  return 1;
}
```

(16) Question 6. Four design choices one must make when implementing a spectrum analyzer.

1) ADC range. This defines the smallest and largest voltage that can be measured.

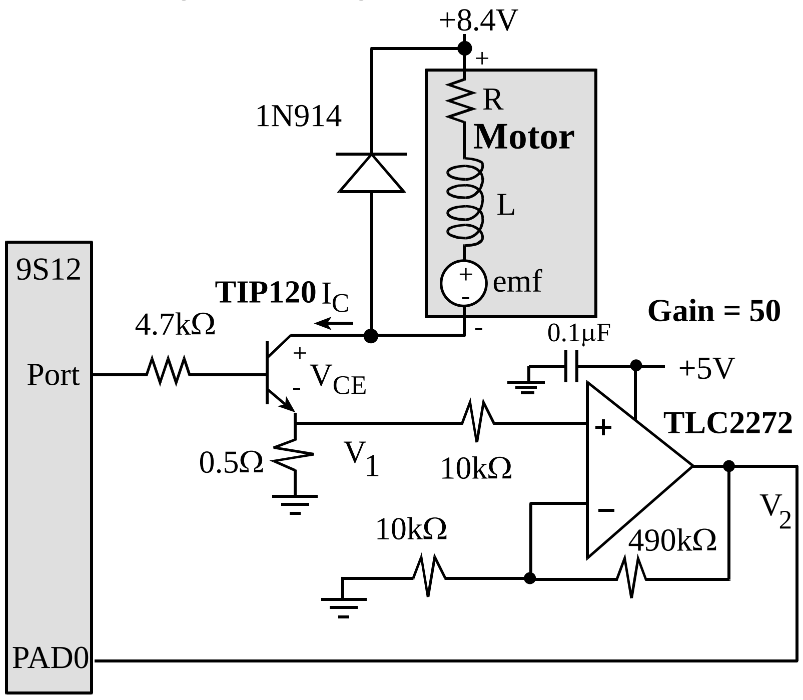
2) ADC precision. Range and precision define the voltage resolution of the system. Resolution is Range/Precision; where precision is given in alternatives

3) ADC sampling rate, f_s. The sampling rate defines the range of frequencies that can be measured: 0 to ½ f_s.

4) The buffer size n. The buffer size and the sampling rate together define the frequency resolution of the spectrum analyzer, $\Delta f = f_s/n$.

Other choices one could have made are FFT windowing, data format (fixed point, floating point), and zero padding the data.

(24) Question 7. The overall goal is to design a feedback motor controller based on I_C.



(12) Part a) Because the maximum current is 0.2A and the maximum voltage loss is 0.1V, the largest resistor we can use is 0.5Ω. We can place the 0.5Ω resistor anywhere in series with the motor current. The simplest place is between the emitter and ground. This way, a simple voltage amplifier with a gain of 5/0.1=50 will convert the 0 to 0.2A current into 0 to +5V on PAD0. Any rail-to-rail op amp is OK. The current I_C will be

$I_C = V_1/0.5\Omega$,

or $I_C = 2*V_1$.

Because of the gain 50 amp, $V_1 = V_2/50$. If we let n be the 10-bit sample, then $V_2 = 5*n/1024$. Thus,

$I_C = V_2/25$,

or $I_C = 5*n/1024/25$,

or $I_C = n/5120$.

Converting to 0.001A fixed-point, we have

result = 1000*n/5120,

or result = 25*n/128,

or result = (25*n+64)/128.

At n=1023, we get result equal to 199, which is close to the I_C of 0.2A defined in Problem 3. If we add 64 to 25*n before dividing by 128, it will round to the closed integer. Notice also that 25*1023+64 is 25639, therefore overflow can not occur.

(12) Part b) Write one C function that samples the ADC channel 0 and calculates I_C.

```
unsigned short Motor_Ic(void){ unsigned short n,Ic;
  ATDCTL5 = 0x80;                    // start channel 0
  while((ATDSTAT1&0x01)==0){};       // wait for CCF0
  n = ATD0DR0;                       // 0 to 1023
  Ic = (25*n+64)/128;                // 0 to 200
  return Ic;
}
```