Jonathan W. Valvano May 10, 2008, 9am-12n

(5) Question 1. The CRC 15-bit checksum will be wrong. Since no CAN controller correctly receives this message, no acknowledgement bit will be sent. The transmitter checks for the presence of this bit, and if no acknowledge is received, the message is retransmitted.

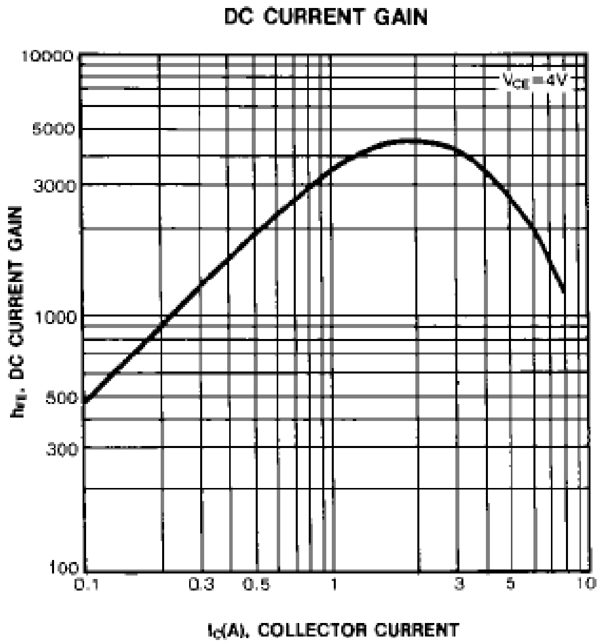
(5) Question 2. D (hardware trigger mode) has NO jitter!! (at least down to the stability of the crystal).

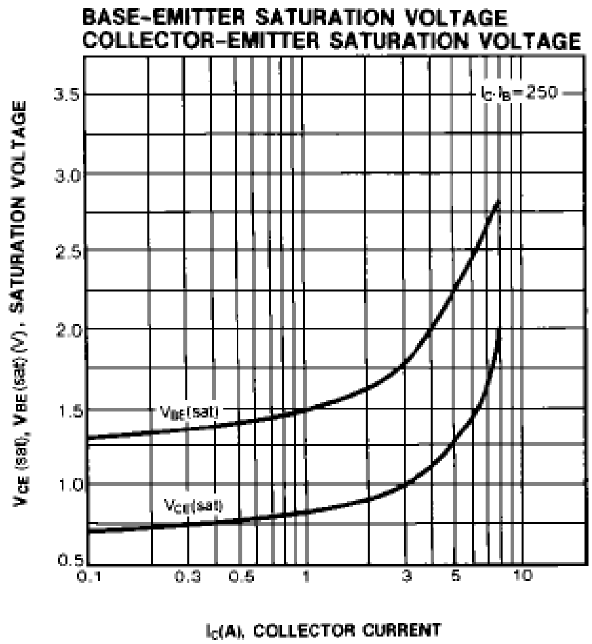
(15) Question 3.

Part a) Since I need 1A, I will use the TIP120, that can handle 3A.

Part b) We use the data sheet of the TIP120 to find h_{fe}=3000, V_{be}=1.5V, V_{ce}=0.8V at 1A.

$I_b = I_c/3000 = (V_{OH}-V_{BE})/R_b$?) 1A/3000= (4.44-1.5)/ R_b , R_b= 3000(4.44-1.5) = 8.8kΩ. I will choose a value from 1 to 5 kΩ, then test it, because the 3000 is only approximate.





Part c) Derive an equation for the motor (I_c) current as a function of time.

The voltage across both LC together is 8.4-V_{ce} = 7.6V at time = 0^+.

At time = 0^+, the inductor is an open circuit.

At time = ∞, the inductor is a short circuit.

The current through LC is 0 at time = 0^+

The current through LC is (8.4-0.8V)/50Ω= 152mA at time = ∞

$$7.6V = I_c * R + L*d\,I_c/dt$$

General solution

$$I_c = I_0 + I_1 e^{-t/\tau} \qquad dI_c/dt = -(I_1/\tau)e^{-t/\tau}$$

plug in

$$7.6V = (I_0 + I_1 e^{-t/\tau})*R - L*(I_1/\tau)e^{-t/\tau}$$

Solve in general

$$\tau = L/R = 2\ \mu sec$$

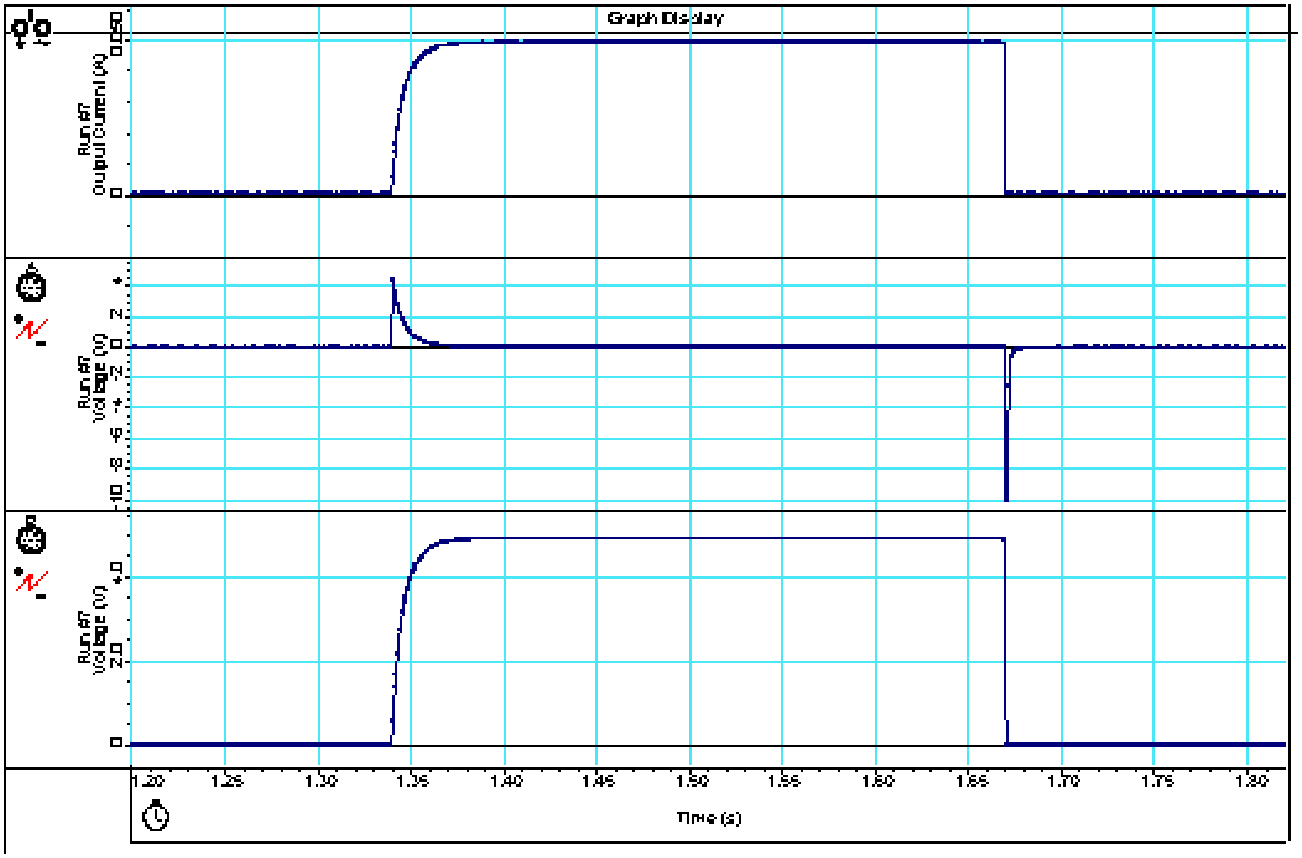
using initial conditions

$$I_0 = 7.6V/50\Omega = 152mA$$

$$I_1 = -7.6V/50\Omega = -152mA$$

$I_c = 152mA*(1- e^{-t/2\mu s})$

http://www.greenandwhite.net/~chbut/new_page_28.htm



(20) Question 4. Consider a robot powered by two DC motors on the rear wheels.
Part a) Write an initialization routine and two input capture ISRs

```
unsigned char NumRight, NumLeft;
void Sensor_Init(void){
asm sei             // make atomic
  TIOS &= ~0x03;   // PT1,PT0 input capture
  DDRT &= ~0x03;   // PT1,PT0 are input
  TSCR1 = 0x80;    // enable TCNT, 667ns
  TCTL4 = (TCTL4&0xF0)|0x0A;    // falling edges IC1,IC0
  TIE |= 0x03;      // Arm IC1,IC0
  NumRight = NumLeft = 0;
asm cli
}
void interrupt 0 IC0Han(void){   // left wheel moved
  NumLeft++;
  TFLG1 = 0x01;      // clear C0F
}
void interrupt 1 IC1Han(void){   // right wheel moved
  NumRight++;
  TFLG1 = 0x02;      // clear C1F
}
```

Part b) The left wheel motor is connected to PT2 and the right wheel motor is connected to PT3.

```
void Motor_Init(void){
  DDRT |= 0x0C;       // PT2 and PT3 are output to motor
  MODRR |= 0x08;      // PT3 associated with PWM
  PWME |= 0x08;       // enable channel 3
  PWMPOL |= 0x08;     // high at beginning, then low
  PWMCLK |= 0x08;     // clock SB for channel 3
  PWMPRCLK &= ~0x70; // B is bus clock/1= 4MHz (0.25us)
// Eperiod*2*PWMSCLB*255*(2**0) about 10ms
// 0.25us*2*PWMSCLB*255*(2**0) about 10,000us
// PWMSCLB = 78,  0.25us*2*78*255*(2**0) = 9,945us
  PWMCAE &= ~0x08;    // left aligned mode
  PWMCTL &= ~0x20;    // no Concatenate 2+3
  PWMSCLB = 78;       // SB prescaled B by 156 = 25.641kHz
  // Clock SB = Clock B / (2 * PWMSCLB)
  PWMPER3 = 255;      // period3
  PWMDTY3 = 0;        // duty cycle3, off
}
void Motor_OutRight(unsigned char U){
  PWMDTY3 = U;
}
```

Part c) Run PID controller every 50 ms (10 times faster than the time constant of the motor)

error will be signed short so it can have the full -255 to +255 range

error = **NumRight** – **NumLeft**; // unsigned to signed conversion

Up Ui Ud Last are all signed short variables

```
Up  = (K1*error)/1000           // proportional term (K1 is negative)
Ui  = Ui + (K2*error)/1000      // integral term (K2 is negative)
Ud  = (K3*(error-last) )/1000   // derivative term (K3 is negative)
last = error
U = Up+Ui+Ud
if(U<0) U=0;
if(U>255) U=255;
Motor_OutRight(U);
```

(15) Question 5. Spinlock semaphore used with a round robin preemptive scheduler.

Part a) There is no critical section in `OS_Signal`, because the read/modify/write to the global is atomic

Part b) This instruction will use the standard methods for establishing the effective address. E.g.,

```
wait $3800  ; 8-bit semaphore at memory location $3800
wait 0,x    ; 8-bit semaphore pointed to by Reg X
```

To make **wait** more general (to be used with blocking semaphores too) we will have the opcode test the 8-bit value in memory. If the value is greater than or equal to 1, then the value is decremented and the Zero bit (Z) is not set. If the value less than or equal to 0, then the value is not changed and the Zero bit (Z) is set.

```
void OS_Wait(char *semaPt){
  asm       tfr  D,X      // Register X points to the semaphore
  asm loop: wait 0,X      // Z=1 if failed
  asm       beq  loop
}                         // enabled
```

(10) Question 6. In order to measure noise, the sensor on a data acquisition system is removed
Part a) All signals (except DC) are less than the ADC resolution (there is no noise here), so what looks like sampling error is simply the finite resolution caused by the 10-bit ADC.
Part b) Increase the number of bits in the ADC (use a 12-bit or 16-bit ADC).
(10) Question 7. Consider a producer/consumer problem linked by a FIFO queue.
Part a) Every 1 second, we receive 8*3=24 bytes. Bandwidth is 24 bytes/sec.
Part b) The maximum SCI bandwidth is 10000bits/sec*(1frame/10bits)*(1byte of data/frame)=1000 bytes/sec
Part c) Assuming no hardware buffering in the SCI transmit channel, we need place for 22 bytes

Time	Number of bytes in FIFO	Interrupt action
0	8	CAN interrupt puts 8 bytes
1	8+8-1 = 15	CAN interrupt puts 8 bytes, SCI gets one
2	15+8-1 = 22	CAN interrupt puts 8 bytes, SCI gets one
3	22-1 = 21	SCI gets one

Assuming two bytes will be buffered in the SCI transmit channel, we need place for 21 bytes

Time	Number of bytes in FIFO	Interrupt action
0	8	CAN interrupt puts 8 bytes
0+	8-2 = 6	SCI interrupts will remove two bytes
1	6+8 = 14	CAN interrupt puts 8 bytes
1+	14-1 =13	SCI interrupt will remove one byte
2	13+8 = 21	CAN interrupt puts 8 bytes
2+	21-1 =20	SCI interrupt will remove one byte
3+	20-1 = 19	SCI interrupt will remove one byte

(20) Question 8. Like the PC, SP and other registers, the **PPAGE** must be switched.

```
struct TCB{
  struct TCB *Next;              // Link to Next TCB
  unsigned char *StackPt;        // Stack Pointer
  unsigned char ThePPAGE;
  unsigned char TheStack[97];    // stack
};
typedef struct TCB TCBType;
typedef TCBType * TCBPtr;
TCBPtr RunPt;  // Pointer to thread currently running
interrupt 11 void threadSwitchISR(void){
asm ldx RunPt
asm sts 2,x
  RunPt->ThePPAGE = PPAGE;  // save PPAGE
  RunPt = RunPt->Next;
  PPAGE = RunPt->ThePPAGE;  // restore PPAGE
  TC3 = TCNT+1000;  // Thread runs for a unit of time
  TFLG1 = 0x08;     // acknowledge by clearing TC3F
asm ldx RunPt
asm lds 2,x
}
```