Jonathan W. Valvano

**(4) Question 1.** PWM

**B) Provides high precision** basically the precision is the PWM reload value, which is PWM period divided by the PWM base clock period. In lab, we used a PWM base clock of 40 MHz (25ns), a reload value of 40000, and a PWM period of 1ms. The precision is 40,000 (15 bits), because there are 40,000 different duty cycles possible.

**(6) Question 2.** Consider the lab 10 system used to control motor speed.

**(3) Part a)** PI vs I

**B) Reduces controller response time**, P makes it respond faster

**(3) Part b)** PI vs P

**A) Reduces controller error,** I reduces error

**(2) Question 3. Friction**, holding torque required to get it to start

**(4) Question 4.** Active filter

D) Lower input impedance (it could have higher input impedance)

**E) Lower output impedance**

**(4) Question 5.** Resistance bridge; **all apply**.

**A) Converts resistance to voltage**

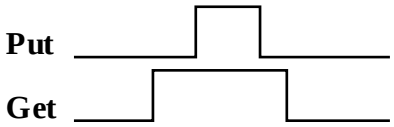
**B) Handles the offset needed because the resistance varied from $R_{min}$ to $R_{max}$**

**C) Nonlinear correction, so the output is more linear with temperature than resistance alone**

**D) Allows the use of a voltage reference, so the signals are low noise**

**(5) Question 6.** You are debugging a software FIFO queue

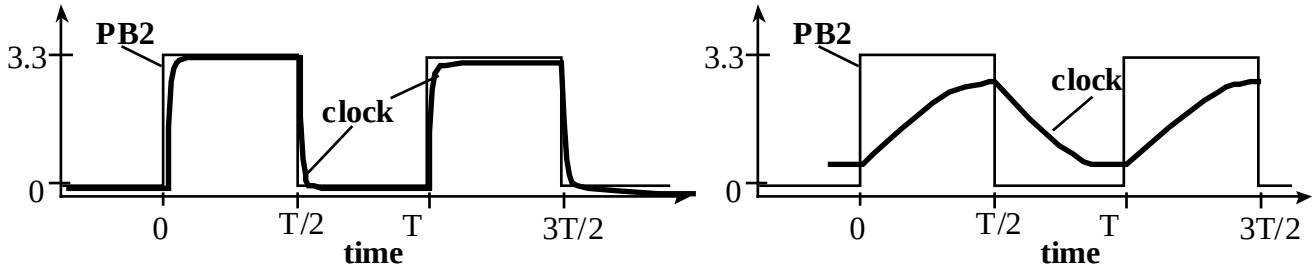
**Part a)** Set an output bit high at the start of **Put**, clear it at end. Set an output bit high at the start of **Get**, clear it at end.




**Part b)** Show the logic analyzer signals

**Part c)** Set trigger on combination (AND) of Get high and rising edge of Put. Alternatively, trigger on combination (AND) of Get=high and Put=high.

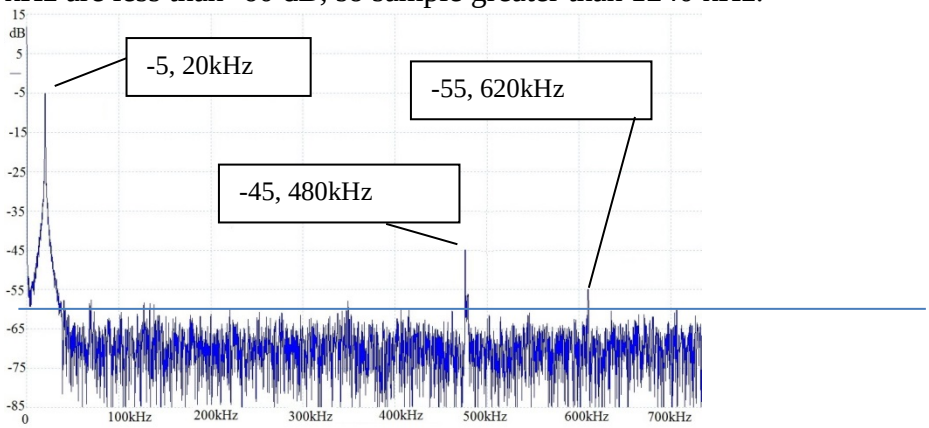
**(5) Question 7.** Clock signal transmitted across a long wire. Calculate time constant $\tau$=100-Ω *10-nF = $10^{2}$*10*$10^{-9}$ = $10^{-6}$sec = 1 µs. Fastest clock speed is around ($f$=1/1µs =1 MHz). Any answer from 100 kHz to 1 MHz will receive credit if students calculate time constant. Personally, I'd choose the period to be 4$\tau$ or 5$\tau$, 200-250 kHz. At 4-µs period, this gives you 2$\tau$ for rise and 2$\tau$ for fall. 1-$e^{-2}$ = 0.86. This means 0.86*3.3V = 2.85V, which means the clock will oscillate between 0.22V and 3.08V




Slower than 250kHz, f=125 kHz, the signal looks ok. Each pulse is 4$\tau$, 1-$e^{-4}$ = 0.98. This means 0.98*3.3V = 3.24V, which means the clock will oscillate between 0.03V and 3.27V

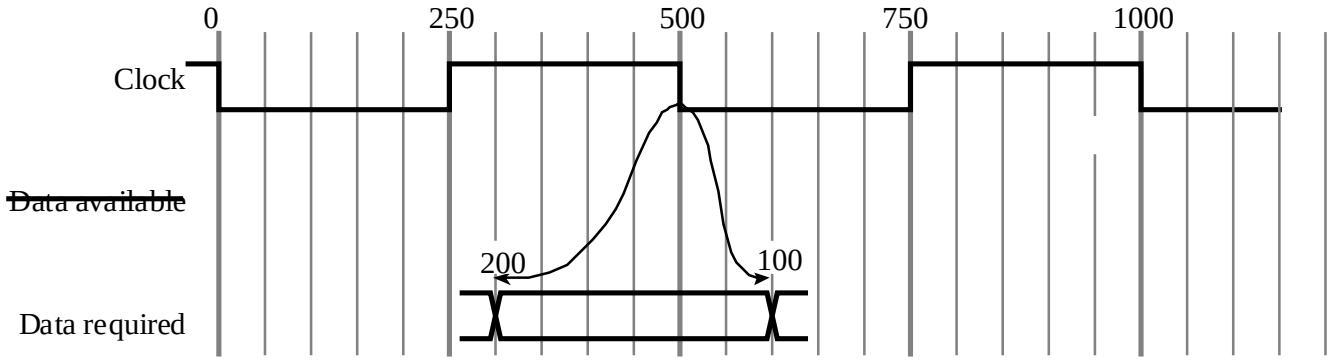
Faster than 250kHz, f=500 kHz, the signal probably does not work Each pulse is 1$\tau$, 1-$e^{-1}$ = 0.63. This means 0.63*3.3V = 2.09V, which means the clock will oscillate between 0.61V and 2.69V, which means it may not go below $V_{IL}$ or above $V_{IH}$.

**(5) Question 8**. ADC resolution = $20\log_{10}(1/1024)$ = -60.2 $dB_{FS}$ The signals higher than frequency 620 kHz are less than -60 dB, so sample greater than 1240 kHz.




**(5) Question 9.** This FIFO queue implementation with shared globals **`Size`**, **`GetI`**, **`PutI`**
**(3) Part a)** "no critical sections", removes sharing
**(2) Part b)** "yes, adds a bug", FIFO does not work, because there are two Size variables now

**(5) Question 10.** We specify *data required* because it is an input




**(10) Question 11.** Consider a battery with voltage $V$ (in volts). The battery has $S$ storage (in mA-hr). The regulator has a power efficiency of $E$, creating a 3.3V supply for the system. To save power, the system runs $x$% of the time at $I_{run}$ (in mA) and sleeps (100-$x$)% at $I_{sleep}$ (in mA). The units of $x$ are percent (0 to 100). Derive an equation for how long will this battery run the system? Show your work.
Calculate average current in system $I_{sys} = ( x*I_{run} +(100-x)*I_{sleep} )/100$
Consider regulator $V*I_{bat}*E = 3.3V * I_{sys}$
Solve for $I_{bat} = (3.3V * I_{sys})/(V*E)$
Consider storage $S = I_{bat}*T$
Solve $T = S / I_{bat} = S / ((3.3V * I_{sys})/(V*E)) = S / ((3.3V *(( x*I_{run} +(100-x)*I_{sleep} )/100))/(V*E))$

**(5) Question 12**. I'd answer in the negative. Being unethical will cost you your freedom (jail). Being unethical will cost you money (lawsuits). Being unethical will cost you your reputation (lost clients). Being unethical will cost you your soul (leads to unhappiness).

**(15) Problem 13.** You are given an input analog signal connected to the microcontroller.
**(8) Part a)** Show the global variables, and initialization function.

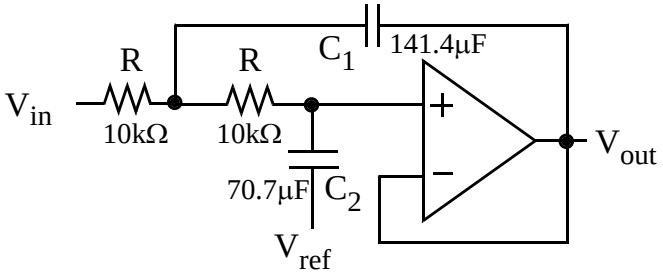
```
int32_t Voltage;    // current input in mV
int32_t Previous;   // input 1ms ago in mV
int32_t Derivative; // slope 1V/s = 1mV/1ms
void SysTick_Init(void){ // sample at 1000 Hz
  NVIC_ST_RELOAD_R = 79999; // any value faster than 200 Hz is ok
  NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF); // priority
  NVIC_ST_CTRL_R = 0x07;
  ADC_Init();
  Voltage = Previous = (3300*ADC_In())/4096;
  Enable_Interrupts();
}
```

**(8) Part b)** Show the interrupt service routine that runs in the background, sampling the ADC, calculating the derivative, and passing the results to the foreground. You do not write **`ADC_In()`**.

```
void SysTick_Handler(void){         // every 1ms
  Voltage = (3300*ADC_In())/4096;   // mV
  Derivative = Voltage-Previous;    // mV/mS
  Previous = Voltage;               // for next time
}
```

**(10) Question 14.** Design an analog amplifier
ADC range is 2V, input range is 0.05V, so need a gain of 2/0.05 = 40
Consider offset $-1V < 40(V_1 - V_2) < 1V$
Need an offset of 1.5V $0.5V < 40(V_1 - V_2)+1.5 < 2.5V$
Use an instrumentation amp, because it is a differential amplifier needing a large input impedance
INA122 gain is 5+200k/Rg, so Rg = 200k/(40-5) = 5.71k
To get offset, connect 1.5V into pin 5 of the INA122, connect output of INA123 to ADC

**(5) Question 15.** Design an anti-aliasing filter. Pick a cutoff between 100 and 500 Hz (slower than signals of interest, and faster than ½ fs. Start at 250 Hz.




$$\left|\frac{V_{out}}{V_{in}}\right| = \sqrt{\frac{1}{1+\left(f / f_c\right)^4}}$$

first design step is to select the cutoff

| | | |
|---|---|---|
| fc (Hz) | 250 | fill this in |
| RA (kohm) | 10 | same as initial R |
| C1A (μF) | 0.09002 | is 141.4/(2•π•fc) |
| C2A (μF) | 0.04501 | is 70.7/(2•π•fc) or 0.5•C1A |

second design step is to choose convenient Capacitor values

| | | |
|---|---|---|
| fc (Hz) | 250 | same as previous fc |
| RB (kohm) | 20.459 | new value to match exact fc |
| C1B (µF) | 0.044 | fill this in |
| C2B (µF) | 0.022 | is 0.5•C1B |

third design step is to choose a convenient resistor value

| | | |
|---|---|---|
| fc (Hz) | 255.733 | new cutoff based on these convenient values |
| RC (kohm) | 20.000 | fill this value in |
| C1C (µF) | 0.044 | same as C1B |
| C2C (µF) | 0.022 | same as C2B |

**(10) Question 16.** The goal of the problem is to design a system to measure the phase

**Part a)** One line modification to **`PWMeasure2_Init`** (Program 6.5) in the book.

```
uint32_t PW;                // 24 bits, 12.5 ns units
#define PERIOD 800000       // 10ms, 12.5 ns units
void PWMeasure2_Init(void){ // TM4C123 code
  SYSCTL_RCGCTIMER_R |= 0x01;      // activate timer0
  SYSCTL_RCGCGPIO_R |= 0x02;       // activate port B
  Done = 0;                        // allow time to finish activating
  GPIO_PORTB_DIR_R &= ~0xC0;       // make PB6, PB7 inputs
  GPIO_PORTB_DEN_R |= 0xC0;        // enable digital PB6, PB7
  GPIO_PORTB_AFSEL_R |= 0xC0;      // enable alt funct on PB6, PB7
  GPIO_PORTB_PCTL_R = (GPIO_PORTB_PCTL_R&0x00FFFFFF)+0x77000000;
  TIMER0_CTL_R &= ~0x00000101;     // disable timers 0A and 0B
  TIMER0_CFG_R = 0x00000004;       // configure for 16-bit timer mode
  TIMER0_TAMR_R = 0x00000007;
  TIMER0_CTL_R = (TIMER0_CTL_R&(~0x0C))+0x00; // rising edge
  TIMER0_TAILR_R = 0x0000FFFF;     // start value
  TIMER0_TAPR_R = 0xFF;            // activate prescale, creating 24-bit
  TIMER0_IMR_R |= 0x00000004;      // enable capture match interrupt
  TIMER0_ICR_R = 0x00000004;       // clear timer0A capture match flag
  TIMER0_TBMR_R = 0x00000007;
  TIMER0_CTL_R = (TIMER0_CTL_R&(~0x0C00))+0x00; // rising edge
  TIMER0_TBILR_R = 0x0000FFFF;     // start value
  TIMER0_TBPR_R = 0xFF;            // activate prescale, creating 24-bit
  TIMER0_IMR_R &= ~0x0700;         // disable all interrupts for timer0B
  TIMER0_CTL_R |= 0x00000101;      // enable timers 0A and 0B
  NVIC_PRI4_R = (NVIC_PRI4_R&0x00FFFFFF)|0x40000000; // Timer0=priority 2
  NVIC_EN0_R = 1<<19;              // enable interrupt 19 in NVIC
  EnableInterrupts();
}
```

**Part b)** Interrupt service routine that measures phase. Signal the semaphore when new data is available

```
void Timer0A_Handler(void){
  TIMER0_ICR_R = 0x00000004;  // acknowledge timer0A capture flag
  PW = (TIMER0_TBR_R-TIMER0_TAR_R)&0x00FFFFFF;// from rise to rise
  Phase = (3600*PW)/PERIOD;
  Done = 1;
}
```