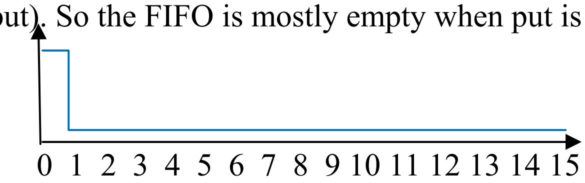
Jonathan W. Valvano October 10, 2014, 10:00am-10:50am.

(15) Question 1. This debugging instrument measures the FIFO probability density function (pdf)

Part a) **`TxPDF[1]`** is the number of times the FIFO contained one element at the time when put was called.

Part b) *Highly intrusive* because it takes 19 cycles to execute and is called on average every 160 cycles, which represents over 10% of the available bus cycles.

Part c) CPU bound means the output hardware (thread that calls get) runs faster than the software (main program that calls put). So the FIFO is mostly empty when put is called.



Bonus) The carry bit will set when the PDF counts from 0xFFFFFFFF rolling back over to 0, meaning the data in the PDF is no longer valid.

(5) Question 2. Consider the debugging instrument shown in Question 1 in other context, where

```
TxPDF[(uint32_t)(TxPutI-TxGetI)]++;
```

is invoked from multiple ISRs with different priorities. The three necessary conditions are:

Access to shared global,	which is the **`TxPDF`**
Multi-step access, with at least one write,	the **`++`** causes a read modify write to global
Nonatomic sequence,	run in ISR with different priorities

The critical section is between LDR and ADDS or between ADDS and STR

```
0x0000076A 6801      LDR      r1,[r0,#0x00]                 [2]
0x0000076C 1C49      ADDS     r1,r1,#1         ; increment PDF [1]
0x0000076E 6001      STR      r1,[r0,#0x00]                 [2]
```

(10) Question 3. Can we interface a 9S12DP512 to the TM4C12xx?

Part a) Yes, a 9S12DP512 GPIO output can be connected to a TM4C12xx GPIO input.

9S12DP512 $I_{OH} \geq$ TM4C12xx I_{IH}	10mA $\geq$ 2μA
9S12DP512 $I_{OL} \geq$ TM4C12xx I_{IL}	10mA $\geq$ 2μA
9S12DP512 $V_{OH} \geq$ TM4C12xx V_{IH}	4.2V $\geq$ 2.0V (GPIO pins are 5V-tolerant)
9S12DP512 $V_{OL} \leq$ TM4C12xx V_{IL}	0.8V $\geq$ 2.0V

Part b) No, a TM4C12xx GPIO output cannot be connected to a 9S12DP512 GPIO input.

TM4C12xx $V_{OH} \ngeq$ 9S12DP512 V_{IH}	2.4V $\ngeq$ 3.25V

(25) Question 4. **`Task1()`** should be executed on the rising edge of PB1

Part a) Show the ritual to initialize this system.

```
GPIO_PORTB_DIR_R &= ~0x06;     // make PB2,PB1 in
GPIO_PORTB_DEN_R |= 0x06;      // enable digital I/O on PB2,PB1
GPIO_PORTB_IS_R &= ~0x06;      // PB2,PB1 is edge-sensitive
GPIO_PORTB_IBE_R &= ~0x06;     // PB2,PB1 is not both edges
GPIO_PORTB_IEV_R |= 0x06;      // PB2,PB1 rising edge events
GPIO_PORTB_IM_R |= 0x06;       // arm interrupt on PB2,PB1
NVIC_PRI0_R = (NVIC_PRI0_R&0xFFFF00FF)|0x00000000; // priority 0
```

Part b) Check and run Task2 first because it is shorter.

```
void GPIOPortB_Handler(void){
  if(GPIO_PORTB_RIS_R&0x04){ // check PB2 first because it is faster
    Task2();
    GPIO_PORTB_ICR_R = 0x04; // acknowledge flag2
  }
  if(GPIO_PORTB_RIS_R&0x02){ // check PB1 second because it is slower
    Task1();
    GPIO_PORTB_ICR_R = 0x02; // acknowledge flag1
  }
}
```

Acceptable but a little bit more latency.

```
void GPIOPortB_Handler(void){
  if(GPIO_PORTB_RIS_R&0x04){ // check PB2 first because it is faster
    GPIO_PORTB_ICR_R = 0x04; // acknowledge flag2
    Task2();
  }
  if(GPIO_PORTB_RIS_R&0x02){ // check PB1 second because it is slower
    GPIO_PORTB_ICR_R = 0x02; // acknowledge flag1
    Task1();
  }
}
```

Doesn't work because RIS register cleared before it is checked.

```
void GPIOPortB_Handler(void){
  GPIO_PORTB_ICR_R = 0x06; // acknowledge both flag2 flag1
  if(GPIO_PORTB_RIS_R&0x04){ // check PB2 first because it is faster
    Task2();
  }
  if(GPIO_PORTB_RIS_R&0x02){ // check PB1 second because it is slower
    Task1();
  }
}
```

---A **race condition** occurs when the relative timing between two independent events cause weird and undesirable effects. The classic race is one ritual executes `DIR = 4; // make bit 2 output` interacts with an independent ritual executing `DIR = 2; // make bit 1 output` ---

Doesn't work because it is miss very short pulses. It also has a race condition: if PB2 rises, execute ICR=6, PB1 rises, execute Task2, execute Task1, another interrupt occurs, executes Task1 incorrectly a second time

```
void GPIOPortB_Handler(void){
  GPIO_PORTB_ICR_R = 0x06; // acknowledge both flag2 flag1
  if(GPIO_PORTB_DATA_R&0x04){
    Task2();
  }
  if(GPIO_PORTB_DATA_R&0x02){ // check PB1 second because it is slower
    Task1();
  }
```

```
}
```

Doesn't work because it has a race condition. If PB2 rises, execute Task2, skip Task1, PB1 rises, ICR=6, it never executed Task1 even though PB1 rises

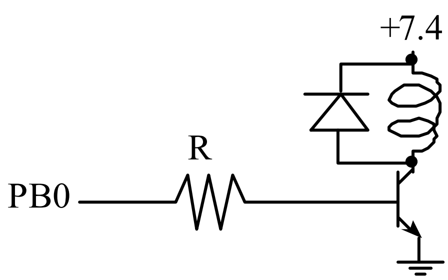
```
void GPIOPortB_Handler(void){
  GPIO_PORTB_ICR_R = 0x06; // acknowledge both flag2 flag1
  if(GPIO_PORTB_DATA_R&0x04){
    Task2();
  }
  if(GPIO_PORTB_DATA_R&0x02){ // check PB1 second because it is slower
    Task1();
  }
}
```

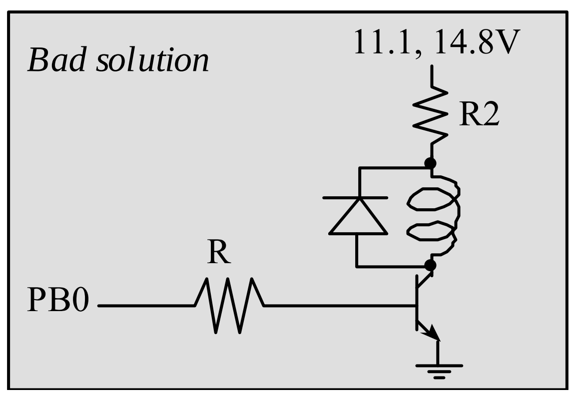
(10) Question 5. $I = C*dV/dt$**.** In the Laplace domain Z=1/(sC) so V/I = 1/(sC). Remember it this way:

> **If the voltage is constant across a capacitor, no current flows. If the current is constant across an inductor, no voltage is generated.**

(10) Question 6. The period of the PWM will be 1ms, which is 16000 bus cycles. The duty cycle can be adjusted from 2 to 15998, which is 15,997 alternatives, which is about **14 bits.**

(25) Question 7. Interface an electromagnetic relay to the microcontroller.
Use **PN2222** because its collector current I_C is large enough to handle the relay. 150 mA > 50 mA.
Choose 7.4V supply because it will apply (supply-V_{CE})=(7.4V-0.3V) = 7.1V to the relay coil
The needed collector current is I_C =50 mA. Since h_{fe} is 100, we will need 0.5mA of base current I_B
The V_{BE} will be 0.6V at saturation.
The base current will be I_B = (V_{OH} - V_{BE})/R = (2.4V-0.6V)/R = 1.8V/R
Choose resistance to be small enough to get the needed I_B Thus, R < 1.8V/0.5mA = 3.6kΩ.
If we repeat at the largest V_{OH}, R < (3.3V-0.6V)/0.5mA = 2.7V/ 0.5mA= 5.4kΩ.
I would make it a little smaller, just in case h_{fe} less than 100, R = 2kΩ.





The **bad solution** places a resistor in series with the inductive load, which is a bad idea because we are uncertain of the current.