Question 1

a. Pins 1.1 RxD P1SEL1 |= BIT1;
1.2 TxD P1SEL2 |= BIT2;

~~P 4.4~~
4.5

$$\boxed{\begin{array}{l} \text{P1SEL} \mathrel{|}= (\text{BIT1} \mid \text{BIT2}); \\ \text{P1SEL2} \mathrel{|}= (\text{BIT1} \mid \text{BIT2}); \end{array}} (1,1)$$

b. Oversampling is during a transmission rate for UART, the current baudrate would have a secondary signal that would read 16x more than the baud rate as long as both systems can handle the frequency, Rx faster than TX.

Ex. baud rate = 9600 hz
oversamp = 16·9600 = 153600hz

c. yes where Smclk = 1048000hz (1.048Mhz)
if 9600·16 ≤ 1.048mhz then oversampling is possible
9600·16 = 153600hz < 1,048,000 hz ✓
Oversampling Possible

d. No where ACLK = 32768hz (32Khz)
if 9600·16 < 32Khz then oversampling is possible
153600 hz > 32768hz X
Oversampling not possible

e. Dividers and modulators are found by Clock frequency over our selected baud rate or Oversampling.

ex SMCLK @ ~~32khz~~ 1.048mhz w/ 9600 baud w/ oversampling

$$\frac{1.048\text{Mhz}}{(16 \cdot 9600)} = \underset{\downarrow}{6}.\underset{\searrow \text{ modulator,}}{82}$$

divide

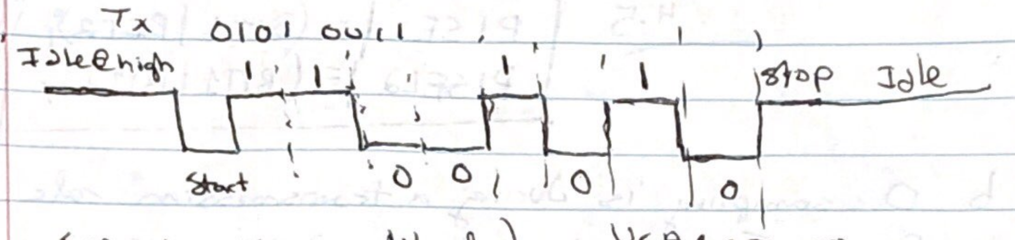
We look @ the family users guide to determine the values for the registers UCBR = Divider, UCBRS = ~~Divider~~ Modulator

Question 2.

A). 9600 baud, 8bit data, no parity, LSB first, one stop bit, no flow control

UCACTL = 0

B,



C. SMCLK 1Mhz calibrated

		UCA1CTLW0
Baud	38,400 baud	UCPEN
Datasize	7 bit	UCPAR
First bit	MSB	UCMSB
Parity	even	UC7BIT
Stop b	2bit	UCSPB
Flow	none	UCOS16
		UCSSEL_3

UCA1BRW = 1
UCA1MCTLW
UCBRF=10
UCOS16
UCBRS = 0x0

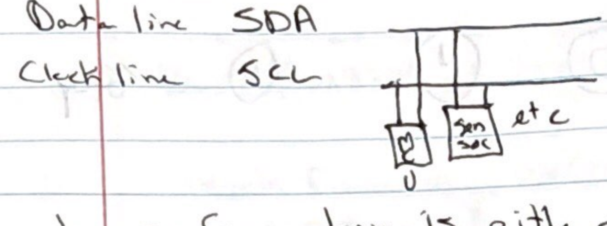
Initialize UART(){

```
UCA1CTLW0 |= UCPAR|UCPEN|UCMSB|UC7BIT|
             UCSPB | ~~WCAD~~ UCSSEL-3;
UCA1MCTLW &= ~UCBRS3; //UCBRS = 0x0
UCA1BRW= 1;
UCA1MCTLW= UCBRF3|UCBRF1|UCOS16;
~~UCA1MCTLW=~~ }
```

D. If the configuration doesn't match, the the data will appear as garbage.

Question 3

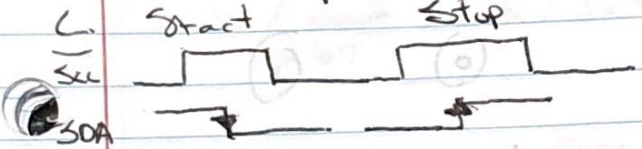
a. allows multiple devices on a single bus with a data and clock wire. It is easily expandable. Each periferal is assigned an address.



half duplex (Bidirectional)

b. configuration is either single or multimaster, Has bidirectional lines reading between 7-10 bits from the device. By default they read on high (pull up). Reads a zero if pulled down.

c.

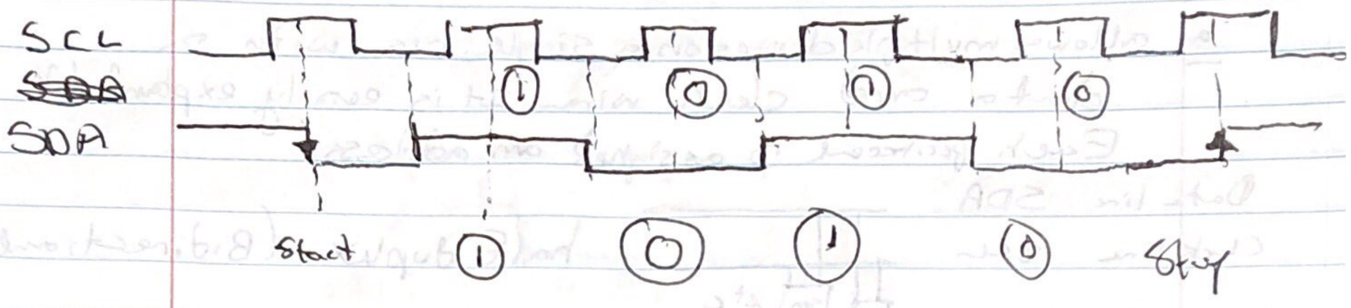


As the SCL is positive, @ 50%, the SDA is pulled down to zero
As the scl is positive, @ 50%, the SDA is pulled up to 1
These signals are unique because @ Stop the Master clk send a NACK (Not Acknowledge) to signify end transmission. @ Start, it sends ACK (Acknoledge) to signify transmission start.

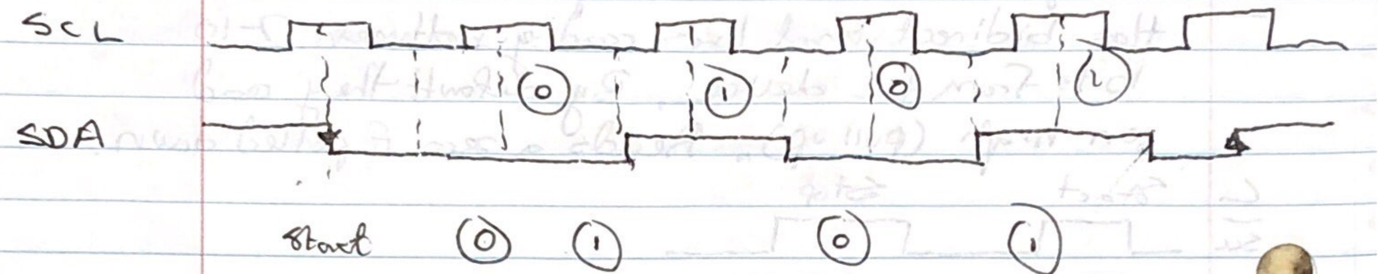
D. ~~A~~ Yes. It requires clock synchronization so it collides harmoniously to create a single signal. (~~Rsu~~ P Resultand clock)
And Arbitration to decide which data can proceed through ~~too~~ logic.

*flipped part e/F

Part F. ~~parts~~ 1010 ~~start~~ + start or stop



part E. 0101 + start stop



part g. ~~The~~ A light sensor is configured by first directing the pins to the correct I2C selection by looking at the users guide. After configuring, ~~[illegible]~~ if we want to read a value on a sensor, we first must access the address of the device located on the schematic, then read from the register @ the devices address. For example, each device will have an address, and within that address are a multitude of addresses within it containing registers. After determining the register from device address, we can either write or read the address by ~~at~~ referencing data to or from it respectively.

Question 4, I2C $\overset{\text{reads}}{\text{from}}$ 0x22 data = 0x1234 (MSB)

a. $\overrightarrow{\text{start}}$ / $\overrightarrow{\text{0x22, R}}$ / $\overleftarrow{\text{ack}}$ / $\overleftarrow{\text{0x12}}$ / $\overrightarrow{\text{ack}}$ / $\overleftarrow{\text{0xAB}}$ / $\overrightarrow{\text{nack}}$ / $\overrightarrow{\text{stop}}$

b. Writes 0x5678 to I2C 0x33

$\overrightarrow{\text{start}}$ / $\overrightarrow{\text{0x33}\overset{w}{\text{R}}}$ / $\overleftarrow{\text{ack}}$ / $\overrightarrow{\text{0x56}}$ / $\overleftarrow{\text{ack}}$ / $\overrightarrow{\text{0x78}}$ / $\overleftarrow{\text{ack}}$ / $\overrightarrow{\text{stop}}$

c. reads from reg 0xCD on I2C 0xEF
0x6789 read.

$\overrightarrow{\text{start}}$ / $\overrightarrow{\text{0xEF, w}}$ / $\overleftarrow{\text{ack}}$ / $\underset{\text{reg}}{\overrightarrow{\text{0xCD}}}$ / $\overleftarrow{\text{ack}}$ / $\overrightarrow{\text{start}}$ / $\overrightarrow{\text{0xCD, R}}$ / . . .

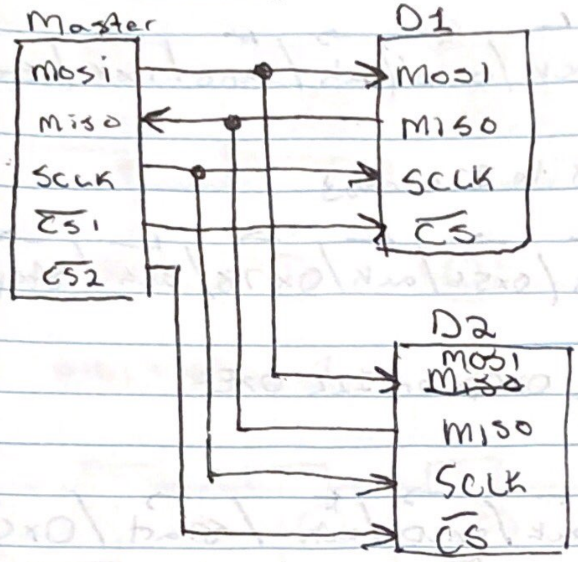
. . . / $\overleftarrow{\text{ack}}$ / $\underset{\text{0x67}}{\overleftarrow{\cancel{\text{data}}}}$ / $\overrightarrow{\text{ack}}$ / $\overleftarrow{\text{0x89}}$ / $\overrightarrow{\text{Nack}}$ / $\overrightarrow{\text{stop}}$

d. writes to 0x12 @ I2C 0x34 Data 0xABCD

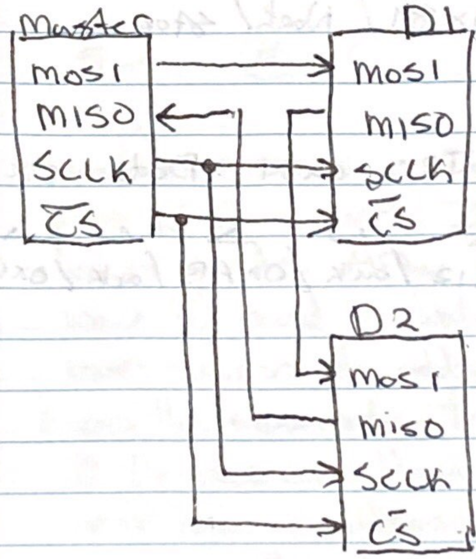
$\overrightarrow{\text{start}}$ / $\overrightarrow{\text{0x34, W}}$ / $\overleftarrow{\text{ack}}$ / $\overrightarrow{\text{0x12}}$ / $\overleftarrow{\text{ack}}$ / $\overrightarrow{\text{0xAB}}$ / $\overleftarrow{\text{ack}}$ / $\overrightarrow{\text{0xCD}}$ / $\overleftarrow{\text{ack}}$ / $\overrightarrow{\text{stop}}$

Qestion 5

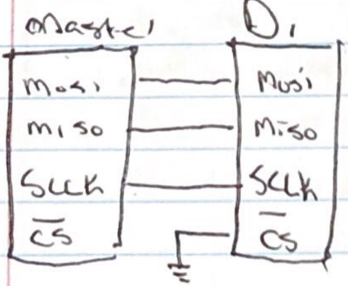
SPI a)
2 devices
individual



b)



c.



3 wire is different because it uses a bidirectional wire to transfer data.

$\overline{CS}$ to ground is 3-Pin

Question 6

ADC CLK [10-20]hz

SHT = 3 seconds

10, 40, 80, 140 . . . .

Part a) 10-20hz using highest frequency for analysis

3 seconds $\cdot$ 20 = 60 ~~ms~~ cycles

or 3 seconds $\cdot$ 10 = 30 cycles

Depending on implementation we could us 80 cycles for a faster reading @ 20hz W/ 60 cycles or 40 cycles for less power consuption @ 10hz w/30 cycles

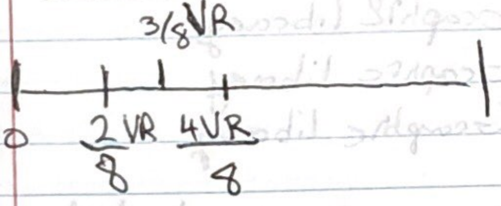
$\boxed{80}$ cycles

Part B.) Total time = SHT + Conversion / f

@ 20hz T_{min} = $\frac{80 + 20}{20hz}$ = 5 seconds

@ 10hz T_{max} = $\frac{40+20}{10hz}$ = 6 seconds

Question 7 (2 ways)



$V_{in} = \frac{3.5}{8} VR = \frac{.875}{2} VR$

$= \frac{1.75}{4} VR$

Bits 0 1 1

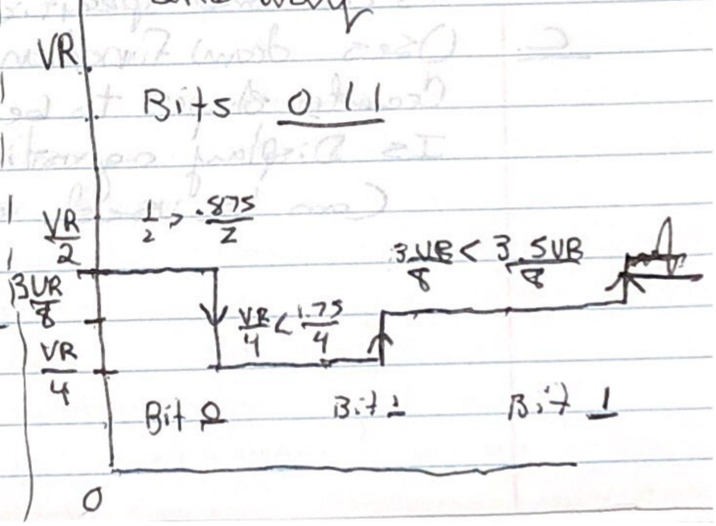
$\frac{3.5}{8} < \frac{4}{8}$ therefore 0

$\frac{3.5}{8} > \frac{2}{8}$ therefore 1

$\frac{3.5}{8} > \frac{3}{8} VR$ therefore 1

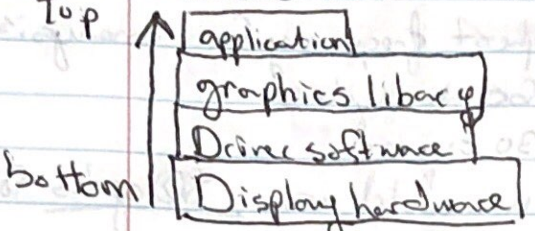
2nd way

Bits 0 1 1



Question 8

a. The software stack provides services to the layer above and utilizes the services from below.
There are 4 layers to the graphic stack



From the bottom, the display provides info to the driver for the basic parameters for images which are size, color capacity, and features.

The driver uses this info to pass through to the graphic library to create the images to be displayed, Finally the application uses this to produce the required images.

b. Driver

lowelevel interaction ——> Display

pulse on reset to start/shut display $\xrightarrow{from}$ Display

Translates color $\xrightarrow{from}$ Display

Sends command data $\xrightarrow{from}$ Display (mcu specific)

Pixel resolution $\xleftarrow{to}$ Graphic library

Touch position $\xleftarrow{to}$ Graphic library

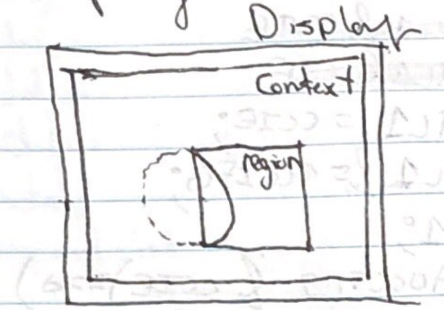
Provides Draw pixel function $\xrightarrow{to}$ Graphic library

Resolution, color, features and command data is command specific

c. Uses draw function from driver.
Creates shapes to be used in application
Is Display agnostic.
Can be used w/ any driver/display.

d. A graphics Context is an area on a display where different graphic objects can be implemented. It can contain many objects of graphics. The logical display.

e. The clipping Region is an area within the graphic context where images can be displayed. For example, you can define an image with the context but if it is not in the Clipping region, it won't be displayed.



Question 9.

a. ~~Operating System level~~
Register level

b. int get_available_timer_channel();

```
int get_available_timer_channel() {
    If ((TA0CCTL0 & CCIE) ==0) {
        TA0CCTL0 |= CCIE;
        TA0CCTL0 &= ~CCIFG; return 0;
    } else if ((TA0CCTL1 & CCIE)==0) {
        ~~(TA0CCTL1 & CCIE~~
        ~~TA0CCTL1 & CCIE~~
        TA0CCTL1 |= CCIE;
        TA0CCTL1 &= ~CCIFG;
        return 1;
    } else if ((TA0CCTL2 & CCIE)==0) {
        TA0CCTL2 |= CCIE;
        TA0CCTL2 &= ~CCIFG;
        return 2;
    } else
        return -1;
}
```

Question 10

address : 100

Data:

_ _ _ _ _ _ _ _
7 6 5 4 3 2 1 0

2009 2000

part a) read bit 4 of data into temp

```
#define bit0 0x2000
#define bit1 0x2001
#define bit2 0x2002
#define bit3 0x2003
#define bit4 0x2004
#define bit5 0x2005
#define bit6 0x2006
#define bit7 0x2007
~~#defi~~
 unsigned int temp=0;
      *ptr[] ={&bit0, &bit1 &bit2, &bit3, &bit4, &bit5,
              &bit6, &bit7}
    temp=*ptr[4];    // part A
```

part b)

```
    ~~*ptr[5] &~~
     *ptr[5] = newData;
```

part c)

```
      int n=0;
      int count=0;
    while(n<7){
       ~~if(*ptr[5]~~ if(*ptr[n] != 0){
                  count++;
               }
               n++;
          }
```