1)

Master clock = 4.4 MHz

Data throughput rate = 10 KB/s

Total bits = 1(start) + 1(stop) + 1(parity) + 8(data) = 11

Baud rate = $\frac{11}{1\ byte}$ × Data throughput rate = 110 Kb/s

$$Baud\ rate = \frac{Master\ Clock}{16 \times Clock\ Divider} = 110 \rightarrow \text{Clock Divider} = 2.4414$$

$\rightarrow floor(\text{Clock Divider}) = 2$

2)

Non-atomic shared data problem: اگر هنگام خواندن یک داده یک وقفه رخ بدهد و داده تغییر کند، داده‌ی خوانده‌شده دیگر درست نیست. اگر داده‌ها atomic باشند، تا اتمام یک عملیات روی آن‌ها عملیات دیگری قابل اجرا نخواهد بود و این مشکل دیگر اتفاق نمی‌افتد.

غیرفعال کردن وقفه‌ها: اگر قبل از ناحیه بحرانی وقفه‌ها را غیر فعال و بعد اتمام دستورات ناحیه بحرانی دوباره وقفه ها را فعال کنیم، می‌توانیم مشکل مذکور را حل کنیم. قبل از غیر فعال کردن، وقفه‌های آمده را ذخیره می‌کنیم و بعد از انجام شدن عملیات مورد نظر آن را به رسیدگی به آن‌ها می‌پردازیم.

مثال از اسلاید ۱۴ – Interrupt(2):

```
void GetDateTime(DateTimeType * DT){
        uint32_t m;
        m = __get_PRIMASK();
         __disable_irq();

         DT->day = TimerVal.day;
         DT->hour = TimerVal.hour;
         DT->minute = TimerVal.minute;
         DT->second = TimerVal.second;
         __set_PRIMASK(m);
}
```

در کد بالا مشکل Non-atomic shared data در هنگام خواندن TimerVal پیش می آید که این مشکل را همانطور که در بالا گفته شد با ذخیره و غیرفعال کردن وقفه ها قبل از ناحیه بحرانی و فعال کردن مجدد آن بعد از اتمام کار حل می کنیم.

3) $N_{ADC} = 4095 \times \frac{V_{in}-V_{r+}}{V_{r+}-V_{r-}} \rightarrow V_{in} = N_{ADC} \times \frac{V_{r+}-V_{r-}}{4095} + V_{r-} =$
$2048 \times \frac{5-(-5)}{4095} = 5.001221 \rightarrow -5 \rightarrow 1.221 \times 10^{-3}\ V$

4)

الف) ورودی با بیت ۰(start – 1bit, high to low) شروع می‌شود؛ سپس ۸ بیت داده می‌آید و در آخر با بیت ۱(stop - 2bit) تمام می‌شود. پس در شکل داده‌شده از ۱۰۰ تا ۵۰۰ میکروثانیه را به عنوان داده در نظر می‌گیریم.

ب)

$$Baud\ rate = \frac{1b}{50\ \mu s} = 20000\frac{b}{s}$$

ج)

$$Bandwidth = \frac{Data}{n} \times Baud\ rate = \frac{8}{10} \times 20000\frac{b}{s} = 16000\frac{b}{s}$$

```
5)      MOV     R12, #0x4000000   ;memory address register
        MOV     R0, #0x0
        MOV     R1, #0x1000000  ;amount of delay = 1000000 cycles
TOGGLE  STR     R0, [R12]
        ORN     R0, R0, R0
        MOV     R1, #0x1000000
        AGAIN   SUBS    R1, R1, #1
        BNE     AGAIN   ;repeat until loop counter(R1) is zero
        B       TOGGLE
```