King Fahd University of Petroleum and Minerals
College of Computer Science and Engineering
Computer Engineering Department

COE 306: INTRODUCTION TO EMBEDDED SYSTEMS

Term 171 (Fall 2017-2018)
Major Exam II
Wednesday Nov. 29, 2017

Time: 150 minutes, Total Pages: 13

Name:__KEY_____ ID:_____ Section: _____

Notes:

- Do not open the exam book until instructed
- Answer all questions
- All steps must be shown
- Any assumptions made must be clearly stated

Question	Max Points	Score
Q1	27	
Q2	11	
Q3	10	
Q4	14	
Q5	12	
Total	74	

Dr. Aiman El-Maleh

[27 Points]

(Q1) Fill in the blank in each of the following questions:

(1) A HAL (Hardware Abstraction Layer) defines a set of routines, protocols and tools for interacting with the hardware and has the advantage of allowing changing hardware without changing application.

(2) An API (Application Programming Interface) defines a set of routines, protocols and tools for creating an application.

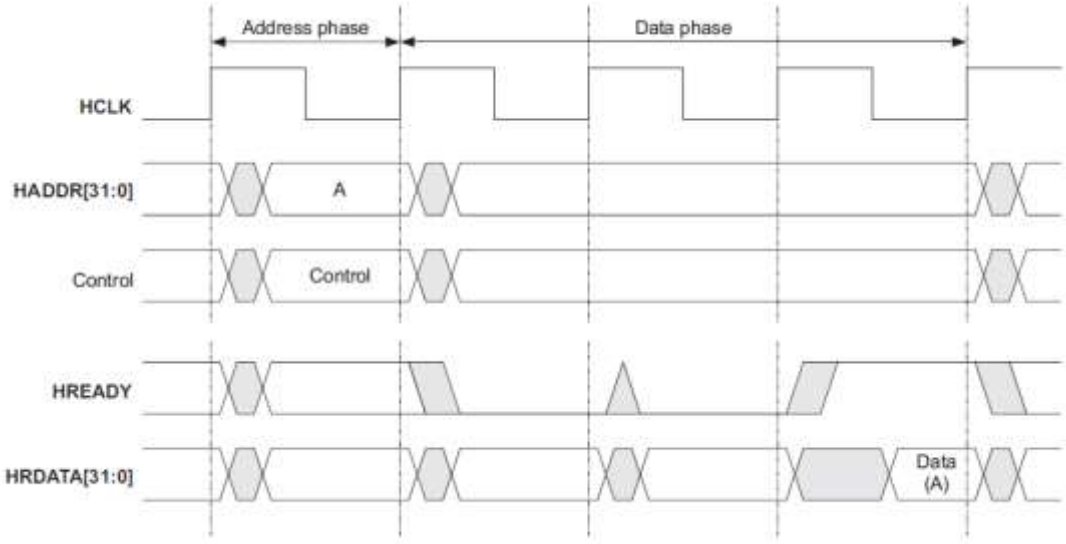
(3) To start a transfer using Direct Memory Access (DMA), the CPU sets the 3 registers Starting address: where the transfer begins, Length: number of words to be transferred and Status: to operate the DMA controller and once the transfer is complete, the DMA interrupts the CPU.

(4) Reasons for using multiple buses in a microcontroller system are Higher-speed buses may use wider data connections, Higher-speed buses require more expensive circuits and connectors, Lower-speed devices can use lower-speed circuits and connectors, lowering their prices, and bridges connecting two buses may allow them to operate independently.

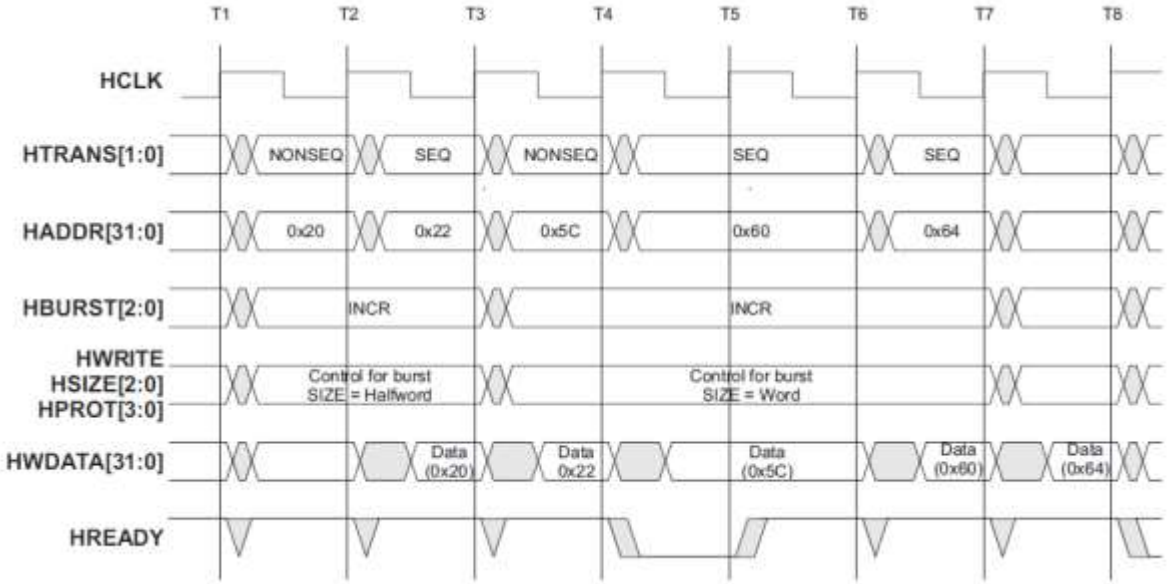
(5) In a microcontroller system, two devices connected on the high speed bus are CPU, DMA, Ethernet, RAM, ROM and two devices connected on the low-speed bus are SPI, I2C, UART, ADC. DAC, PWM.

(6) In AMBA AHB bus, during a read operation the HRDATA is selected from the slave being read based on decoding the slave address.

(7) Complete the given timing diagram for an AMBA AHB read transfer:

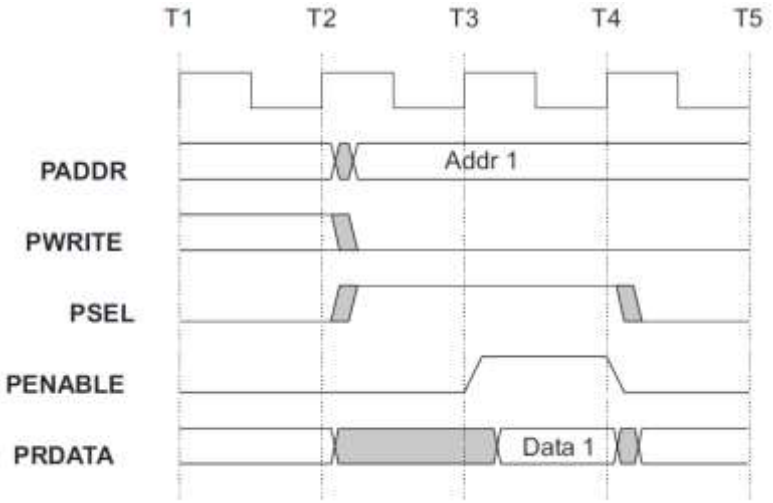


(8) Complete the given timing diagram for AMBA AHB burst write transfers:



(9) During an AMBA AHB transfer, whenever a slave sends a transfer response as ERROR, RETRY or SPLIT, a <u>two-cycle</u> response is required.

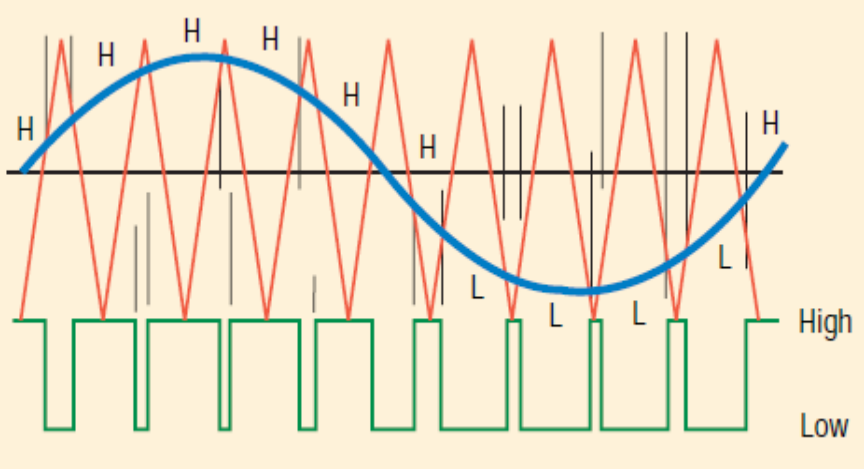
(10) Complete the given timing diagram for an AMBA APB read transfer:



(11) In AMBA AHP bus, a split transfer works as follows: <u>The SPLIT response provides a mechanism for slaves to release the bus when they are unable to supply data for a transfer. Once the slave is ready he will request that the bus to be given to the master who made the original request so that the transfer is completed.</u>

(12) Given a pulse width modulated signal with duty cycle of 40%, V_L=0v and V_H=5v, the average value of the signal is 0.4*5=2v.

(13) Complete the given diagram for a pulse width modulated signal. The modulation type used is Pulse Center Two Edge Modulation.

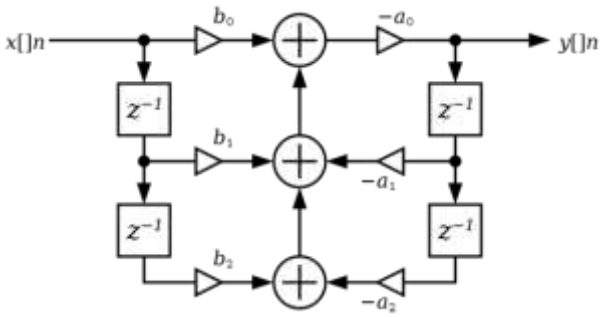


(14) Three example applications of pulse width modulation include modulating different analog values when no DAC exists, to control servo motors, controlling brightness of led, voltage regulator and to transmit data in telecommunication.

(15) Given that MR0=100, PWMEN2, PWMSEL2, to configure PWM2 channel to be leading edge modulated with 80% duty cycle, we need to set MR1 to be equal to 20 and MR2 to be equal to 100.

[11 Points]

(Q2) It is required to implement the following IIR filter using circular buffers:



The C code for the function init(buf, n, &pos) for initializing buffer buf with size n and position pos is given below:

```
void init(int buf[], int n, int *pos) {
        for (int i = 0; i < n; i++)
                buf[i] = 0;
        *pos = n - 1;
}
```

(i) Define the needed buffers, their sizes, their needed position variables and the code for initializing the buffers.

```
#define SIZE 2
int buf1[SIZE], buf2[SIZE];
int pos1, pos2;
init(buf1, SIZE, &pos1);
init(buf2, SIZE, &pos2);
```

(ii) Show the C code for the function put(buf, n, &pos, val) for adding a new value, val, to buffer buf with size n and position pos.

```
void put(int buf[], int n, int *pos, int value) {
        *pos = (*pos + 1) % n;
        buf[*pos] = value;
}
```

(iii) Show the C code for the function get(buf, n, &pos, i) for getting the ith value earlier from buffer buf of size n and position pos, with zero being the latest value put in the buffer.

```
int get(int buf[], int n, int *pos, int i) {
        int index = (*pos - i) % n;
        if (index >=0)
                return buf[index];
        else return buf[n+index];
}
```

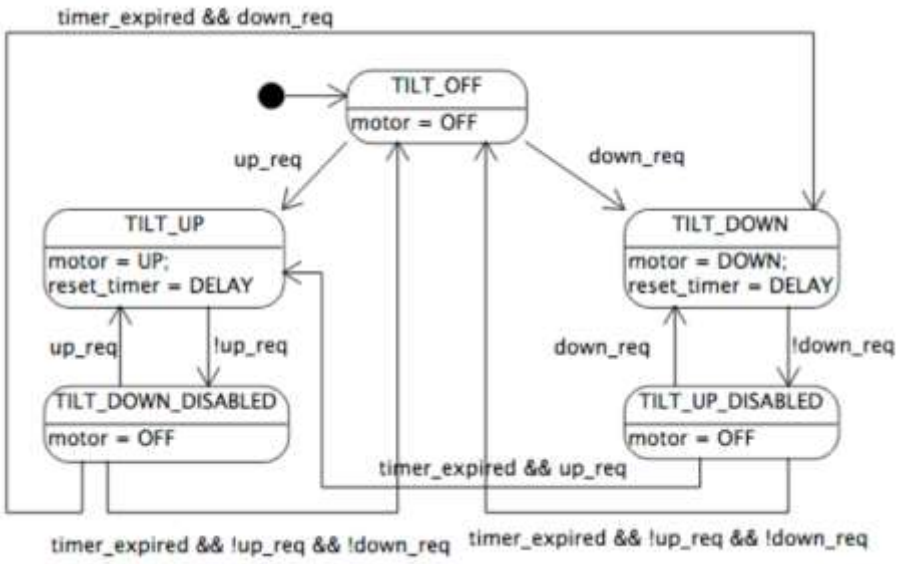
(iv) Show the C code for the function iir(x) that receives a new value x and returns a computed value y. Assume that the coefficients a and b are stored in two integer arrays as given below.

```
int a[3] = {a0, a1, a2};
int b[3] = {b0, b1, b2};



int iir(int x) {
  int i, y=0;
  for (i = 0, y = 0; i < SIZE; i++)
    y += b[i+1] * get(buf1, SIZE, &pos1, i)- a[i+1] * get(buf2, SIZE, &pos2, i);
  y += x*b[0];
  y = -a[0]*y;
  put(buf1, SIZE, &pos1, x);
  put(buf2, SIZE, &pos2, y);

return y;
}
```

[10 Points]

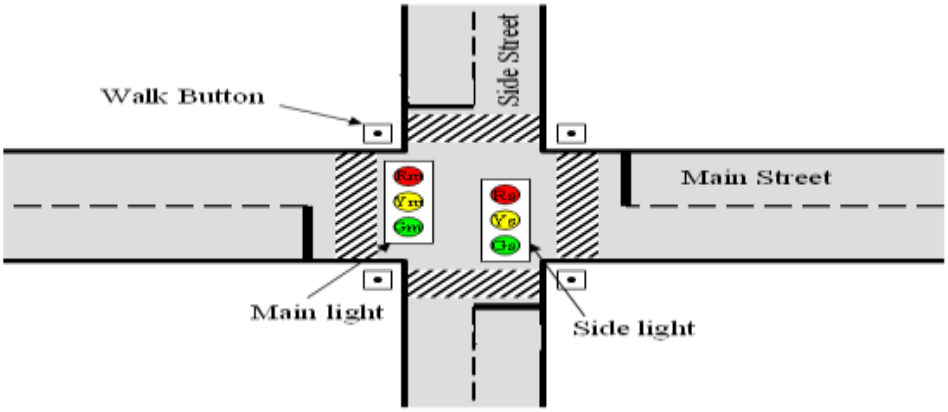
(Q3) Suppose that you are hired with a company that designs electronics for high-end furniture. You were asked to work on a motorized couch, with drink holders and a built-in drink cooler and reclining seats that tilt up and down. You are asked to write an embedded program that controls the tilt motor using a microcontroller. There's an up button and a down button. When the person switches from down to up, or up to down, you have to wait 100 milliseconds so that the motor doesn't burn out (i.e., move the motor from up/down to off and then to down/up positions).

Draw the state machine diagram for the motorized couch controller using a Moore model.

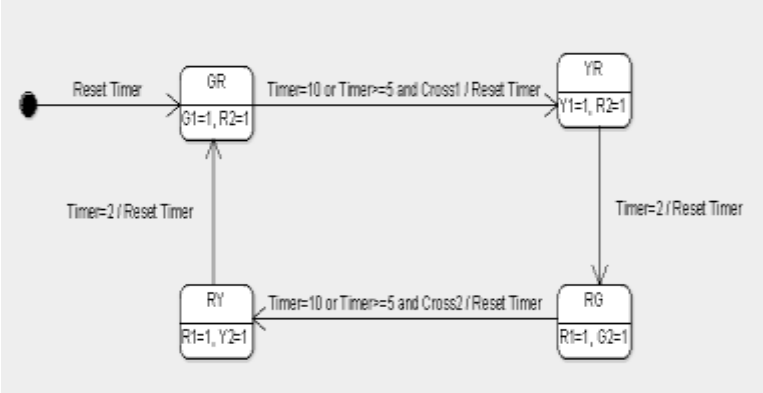


[14 Points]

(Q4) It is required to design an embedded system that controls the traffic lights at an intersection of main and side streets. It receives inputs from all four corners indicating pedestrians that want to cross. In absence of crossing requests, it should allow each direction 10 seconds of green light, followed by 2 seconds of yellow light while the other traffic light will be red light (i.e., for 12 seconds). In presence of crossing requests at or after 5 seconds, immediately proceed with yellow. Two buttons, Cross1 and Cross2, are used to indicate request for crossing across the main and side streets respectively. A pair of Red, Yellow and Green leds will be used for the two traffic lights.



A Mealy state machine diagram for the traffic light controller unit is given below:



It is required to write a C program that implements this state machine of the traffic light controller. Assume that any change in the inputs **Cross1** and **Cross2** will cause interrupts to update their values. Timer0 and Timer1 will be used to implement the required timing requirements. The two functions SetTimer0 and SetTimer1 are used to set Timer0 and Timer1 to call their respective timer handler for a given delay in secs.

Complete the given C code for implementing the state machine of the traffic light controller.

```
#ifdef __USE_CMSIS
#include "LPC17xx.h"
#endif

#include <cr_section_macros.h>


#define GR 0
#define YR 1
#define RG 2
#define RY 3
int state=0;
int Cross1=0, Cross2=0;
int timer0=0; timer1=0;

void TIMER0_IRQHandler() {
   timer0=1;
   LPC_TIM0->IR |= 1;
}

void TIMER1_IRQHandler() {
   timer1=1;
   LPC_TIM1->IR |= 1;
}

void EINT0_IRQHandler()
{
      if (state==GR) Cross1=1;
      for (int j=0; j<1000000; j++); // to avoid effect of bouncing
      LPC_SC->EXTINT |= 1;
}

void EINT1_IRQHandler()
{
      if (state==RG) Cross2=1;
      for (int j=0; j<1000000; j++); // to avoid effect of bouncing
      LPC_SC->EXTINT |= 2;
}

void SetTimer0(uint32_t delayInSec) {
      LPC_TIM0->TCR = 0x02; /* reset timer */
      LPC_TIM0->MR0 = delayInSec*2000 * (12500000 / 1000 - 1);
      LPC_TIM0->MCR = 0x05; /* stop timer on match and enable interrupt*/
      LPC_TIM0->TCR = 0x01; /* start timer */
}

void SetTimer1(uint32_t delayInSec) {
      LPC_TIM1->TCR = 0x02; /* reset timer */
      LPC_TIM1->MR0 = delayInSec*2000 * (12500000 / 1000 - 1);
      LPC_TIM1->MCR = 0x05; /* stop timer on match and enable interrupt*/
      LPC_TIM1->TCR = 0x01; /* start timer */
}


int main(void) {


      LPC_GPIO0->FIODIR |= 7<<7;  // set pins 0.7, 0.8, 0.9 for GYR for
Main Street TL
      LPC_GPIO0->FIODIR |= 7<<23; // set pins 0.23, 0.24, 0.25 for GYR for
Side Street TL
```

```
LPC_PINCON->PINSEL4 |= (1<<20); // using pin p2.10 for Cross1
LPC_PINCON->PINSEL4 |= (1<<22); // using pin p2.11 for Cross2

LPC_SC->EXTMODE  |= 3;
LPC_SC->EXTPOLAR |= 3;

NVIC_EnableIRQ(EINT0_IRQn);
NVIC_EnableIRQ(EINT1_IRQn);
NVIC_EnableIRQ(TIMER0_IRQn);  // Enable interrupt for timer 0
NVIC_EnableIRQ(TIMER1_IRQn);  // Enable interrupt for timer 1

Cross1=0; Cross2=0;

LPC_TIM0->PR = 0x00; /* set prescaler to zero */
SetTimer0(10);
timer0=0;

LPC_TIM1->PR = 0x00; /* set prescaler to zero */
SetTimer1(5);
timer1=0;

while(1) {

      switch(state){

      case GR:
            printf("state GR\n");
            LPC_GPIO0->FIOSET |= (1<<7); LPC_GPIO0->FIOCLR |= (1<<8);
            LPC_GPIO0->FIOCLR |= (1<<9);
            LPC_GPIO0->FIOCLR |= (1<<23); LPC_GPIO0->FIOCLR |=
      (1<<24);
            LPC_GPIO0->FIOSET |= (1<<25);

            if (timer0 || (timer1 && Cross1)){
                  state = YR; Cross1=0;
                  SetTimer0(2);
                  timer0=0;
            }
            else
                  state = GR;
            break;
      case YR:
            printf("state YR\n");
            LPC_GPIO0->FIOCLR |= (1<<7); LPC_GPIO0->FIOSET |= (1<<8);
            LPC_GPIO0->FIOCLR |= (1<<9);
            LPC_GPIO0->FIOCLR |= (1<<23); LPC_GPIO0->FIOCLR |=
(1<<24);
            LPC_GPIO0->FIOSET |= (1<<25);
            if (timer0){
                  state = RG;
                  SetTimer0(10);
                  timer0=0;
                  SetTimer1(5);
                  timer1=0;
            }
            else
                  state = YR;
            break;
```

```
            case RG:
                  printf("state RG\n");

                  LPC_GPIO0->FIOCLR |= (1<<7); LPC_GPIO0->FIOCLR |= (1<<8);
                  LPC_GPIO0->FIOSET |= (1<<9);
                  LPC_GPIO0->FIOSET |= (1<<23); LPC_GPIO0->FIOCLR |=
      (1<<24);
                  LPC_GPIO0->FIOCLR |= (1<<25);

                  if (timer0 || (timer1 && Cross2)){
                        state = RY; Cross2 = 0;
                        SetTimer0(2);
                        timer0=0;
                  }
                  else
                        state = RG;
                  break;
            case RY:
                  printf("state RY\n");
                  LPC_GPIO0->FIOCLR |= (1<<7); LPC_GPIO0->FIOCLR |= (1<<8);
                  LPC_GPIO0->FIOSET |= (1<<9);
                  LPC_GPIO0->FIOCLR |= (1<<23); LPC_GPIO0->FIOSET |=
                  (1<<24); LPC_GPIO0->FIOCLR |= (1<<25);
                  if (timer0){
                        state = GR;
                        SetTimer0(10);
                        timer0=0;
                        SetTimer1(5);
                        timer1=0;
                  }
                  else
                        state = RY;
                  break;
            }
          }


    return 0;
}
```

[12 Points]

(Q5)

(i) (3 Points) Assume an A/D converter is supplying samples at 44.1 kHz.

(a) How much time is available per sample for CPU operations?

Time per sample = 1 / 44.1 k = 0.0227 ms

(b) If the interrupt handler executes 100 instructions obtaining the sample and passing it to the application routine, what is the CPU utilization of a 20 MHz RISC processor that executes 1 instruction per cycle?

CPU can execute (20 * 1000000) instructions in one second.

Each sample needs 100 instructions.

In one second, 44100 * 100 = 4410000 instructions need to be executed for the samples.

CPU utilization = 4410000/(20000000) = 22.05%.

(ii) (9 Points) A real-time system receives data through an I/O device, the CPU processes the data, then the results of the processing are transferred to system memory. The I/O device, the CPU, and the memory controller are all on the same system bus, which runs at 1MHz. The CPU runs at 10 MHz. Each bus transaction (transfer) between any two devices on the bus takes 5 bus cycles, 1 of which is used to transfer data, and the remaining cycles are used by the bus protocol. The bus has 32 data lines, transferring 32 bits per data-transfer cycle.

The I/O device receives 512 bytes at a time. While processing the received data, for each received byte, the CPU generates 4 bytes. Only generated data is transferred from the CPU to system memory.

If the I/O device receives new data at a rate of 200 times per second (512 bytes each), how many CPU cycles can be spent processing each byte without violating the real-time requirements? Assume that the memory is fast enough to handle any requests received by the memory controller.

$N_{I/O}$ = 512 B x 200 = 102400 B

$T_{I/O}(N)$ = (D +O) N/W = 5 x 102400/4= 128000 cycles

N_{mem} = 512 B x 200 x 4 = 409600 B

$T_{mem}(N)$ = 5 x 409600/4 = 512000 cycles

T_{bus} = 128000+ 512000= 640000 cycles

$t_{bus} = T_{bus} P = 640000 \times 10^{-6} = 0.64$ s

$t_{CPU} = 1 - 0.64 = 0.36$ s

$T_{CPU} = t_{CPU} * f_{CPU} = 0.36 \times 10 \times 10^6 = 3600000$ cycles

Number of CPU cycles that can be spent processing each byte without violating the real-time requirements = 3600000 / 102400 = 35.156 => 35 cycles per Byte.