**Swati Kadivar**

**Homework3**

**Q1**

1. Plated or Non plated holes if no specifications provided holes will be plated)

2. Non plated or plated slots/edges and cutout's

3. Various laminates including high temp (tg) FR4, Rogers, Polimide, and Aluminum Clad

•4. Full range of finished board thicknesses

5. Different solder mask colors (default green)

6. Different silkscreen (legend) colors (default white)

7. Trace/Space down to 4/4 mils (8/8 typical)

8. UL Markings (Also 94V O upon request)

9. Lead Free logo markings on boards (upon request)

10. Controlled Dielectric and/or Controlled Impedance

11. Inner layer Cu. Wt. ½ oz. 3 oz.

12. Countersinks/Counterbores

13. Fiducials and/or Tooling Holes (no additional charge if requested)

14. Hole Tolerance +/ 0.003” (if requested)

2.

Graphical user interface, application, table, Excel

Description automatically generated

1. **Processor: TI : AM5K2E04**

Graphical user interface, text, application, email

Description automatically generated

Text, timeline

Description automatically generated

Graphical user interface, table

Description automatically generated

This processor has almost all the features as required, but price is ~70$+ for all of AM5K2E04 family processors as they have multiple PCIe ports, higher end Ethernet switch subsystem and DDR3 support too.

1. Under $50 - **NO**

2. DDR3 Memory Interface **- YES**

3. NAND Memory Interface **- YES**

4. 3 PCIe ports **- YES**

5. 3 Gigabit Ethernet ports **- YES**

6. 2 USB or ULPI ports **- YES**

7. 4 UARTs **– 2 UARTs**

8. SPI and I2C port **- YES**

9. SD Card or MMC port **- YES**

10. Wake on LAN capability **– NO**

**So if we can effort little higher price for given requirements then this processor is best fit. But if price is something we can not give up on then below processor is better choice at little less features matching with given requirements.**

**2. TI: TMS320DM8127**

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application

Description automatically generated

**This processor is available at lower price but it has just 1 PCIe Gen2 available. So if we increase PCIe ports then price will go above 50$.**

Text

Description automatically generatedText, chat or text message

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generatedText

Description automatically generated

Graphical user interface, text, application

Description automatically generated

1. Under $50 **- YES**

2. DDR3 Memory Interface **-YES**

3. NAND Memory Interface **- YES**

4. 3 PCIe ports – **1 PCIe port**

5. 3 Gigabit Ethernet ports **- YES**

6. 2 USB or ULPI ports **- YES**

7. 4 UARTs – **6 UARTs**

8. SPI and I2C port - **YES**

9. SD Card or MMC port - **YES**

10. Wake on LAN capability - **NO**

**3.**

**A. FreeRTOS : Its available at no cost, available as open source and x86 is supported**

A picture containing application

Description automatically generated

B. **Windows 10 IoT** : Core is free; Core services are $90 + $0.30/device/month and It has storage of 2GB which should be there for a security camera to store the video recordings

**C.** [MicroC/OS-III](https://en.wikipedia.org/wiki/Micro-Controller_Operating_Systems)

Graphical user interface, application, Word

Description automatically generated

<https://en.wikipedia.org/wiki/Comparison_of_real-time_operating_systems>

Text

Description automatically generated with low confidence

Automotive application : <https://www.researchgate.net/publication/251931744_The_design_and_realization_of_vehicle_real-time_operating_system_based_on_UCOS-II>

D.**MQX**– It is free, and opensource, and supports Freescale MCUs

A picture containing chart

Description automatically generated

**E. RIOT:**

This OS is opensource, free(Low cost) and supports basic MCUs which are perfect for IOT applications like MSP430(TI), ARM + FREESCALE, supports Atmel.

Also it has RISC-V arch support with Microsemi SoCs

Also this OS is developer friendly which makes it easy to develop various iot applications and bug fixing, upgrade also easier.

<https://www.riot-os.org/>

**F. Android :**

Android is available at no cost, opensource, easy to upgrade and bug fix which makes it perfect for layman users. Also it supports many interfaces and so easy to port to new system/sale.

**G. Windows 10 / MacOS –** this OS are having high speed and multithreading environment support which allows us to switch between multiple tasks and work on them at the same time. Also it has huge memory access support so can be connected to external memory(hard drives).

**4.**

Let Y = Buy, D = Build

YC = Buy Costs, DC = Build Costs

YI = Buy Income, DI = Build Income

YPR = Buy Profit Rate,

DPR = Build Profit Rate

UC = unit cost

GP/U = Gross Profit/Unit

FC = Fixed Cost Rate,

T = time in months since start.

TP = Time in months since production

V = Volume in U/month 200per/month

**In this case,**

**YC = FC \* T +FDA\*T=** $500k\*T + $200K\*T for 6 months; then FC\*T.

**YPR = GP/U \* V** = 10000 \* 200/month **= 2000k/month;**

**YI = YPR\*YTP – YC = 2000k\*(T-6) – 500k \* T - 200k\*6; for T>6; @ T=24 YI = YI = 22.8M**

**Cost savings/month = (YUC – DUC)\*V = (2500-400)\*200 = 4200/month**

**Breakeven for Build: Find T so that**

**DI >= YI DI = DPR\*DTP – FC\*T –NRE -FDA = V\*(GP/U + (YUC-DUC))\*(T-24) –FC\*T -NRE -FDA**

**2000k\*(T-6) – 500k \* T - 200k\*6 = 200\*(10000 +2100)\*(T-24) –500k\*T –2400k**

**2000k\*T – 12000k -500k \* T – 1200k = 2420k \*T – 58080k – 500k\*T – 2400k**

**1500k\*T – 13200k = 1920k\*T – 60480k**

**420k\*T = 47280k**

**T = 112.5 months**

**a. The profit made in the first 2 years, assuming the COTS solution. \_\_$22.8M\_\_\_\_\_\_\_\_**

**b. The costs savings/month for the design and build solution versus the COTS solution. \_\_\_\_\_$4200/month\_\_\_\_\_\_\_\_\_\_\_\_**

**c. The number of months until breakeven for the build solution**

**\_\_112.5~113months\_\_or 24+89\_\_**

**d. Circle what your company should do BUY BUILD**

**if build time is more than 2 years then it is better to buy a machine. Here in our case, build time is 24+89 > way more than 2 years. So I choose BUY option.**

5.

**// missing header files – std libraries + math library for PI**

Int a, b, c, d; // **a user should minimize the use of global variable as much as possible in embedded software design**

int main()

{

char x='Y';

while(x='Y') // **should check with ‘==’**

{

//...

cout<<"Continue? (Y/N)"; **// std libraries are not added so cout, cin will throw error**

cin>>x;

}

menu(x, a); // **menu function is defined after it’s use and not declared previously!**

a=b; /\* assignment // missing closing comment which comments next line

c=d; /\* of both pairs \*/

if( 0 < a < 5) c=b;

} // **return type of main is int but the statement is missing**

\_\_interrupt double compute\_area(double radius) //  **interrupt service routines never take either argument nor returns anything – this ISR definition won’t work**

{

double area = PI \* radius \* radius; // **PI is not defined anywhere – user has to define own such constants at top of the code – or include math library**

printf("\nArea = °/.•f", area);

return area;

}

void menu() // **Function definition + declaration should be present before it’s use**

**// no arguments here but calling the function with argument**

{

//...

}

6.

We have 22 bit address line and 16x data bus(2 byes).

Total : 2 \* 2^22 = 2 \* 4,194,304 byes ~ 8Mb of memory can be accessed with 22-bit address line

**S29AL008J55BFIR20**

|  |  |  |  |
| --- | --- | --- | --- |
| **Manufacturer** | Cypress Semiconductor Corp | |  |
| **Manufacturer Product Number** | S29AL008J55BFIR20 | |  |
| **Digi-Key Part Number** | | 428-4185-ND | |

Graphical user interface

Description automatically generated

<https://www.digikey.com/en/products/detail/cypress-semiconductor-corp/S29AL008J55BFIR20/3862778?utm_adgroup=Integrated%20Circuits%20%28ICs%29&utm_source=google&utm_medium=cpc&utm_campaign=Shopping_Supplier_Cypress%20Semiconductor%20Corp_0428_Co-op&utm_term=&utm_content=Integrated%20Circuits%20%28ICs%29&gclid=CjwKCAjw_L6LBhBbEiwA4c46uiI_4XKB5mWGBwhLhf0H_kHs202hzqQA79CNoFmoUS6nfw3H62mRXhoCB-gQAvD_BwE>

3.6

A sequential circuit, or more formally, a *finite-state machine*, is how we ultimately transform the behavior expressed in the state diagram into a hardware and/or software implementation.

Such circuits form the basis for the sophisticated computation and control algorithms that one finds at the core of most modern digital systems. Finite State Machines can be used to model problems in many fields, including mathematics, artificial intelligence, games or linguistics. A Finite State Machine, or FSM, is a computation model that can be used to simulate sequential logic, or, in other words, to represent and control execution flow.

3.30

Four fundamental parameters should be considered when designing or selecting a clock system or time base. For the basic clock, these parameters are:

**• Frequency and frequency range:**

**• Rise times and fall times:**

**• Precision and Stability :**

Because ripple counters are asynchronous, one should never decode any of the state variable combinations to generate a specific frequency. Building a higher frequency from a lower one is done using a phase locked loop (PLL). Crystal-based sources are generally the best solution for stable and accurate timing signals. When using such devices, one can start either with the basic crystal and then design the analog electronics necessary to implement the desired oscillator or buy a prepackaged oscillator. If the application demands greater accuracy and stability than are available with standard devices.

3.18

**Timer design:**

Below is the code for Arduino PWM pin programmed for motor:

int motorPin = 9;

int button1 =2;

int button2= 3;

int val\_motor= 0;

void setup(){

pinMode(motorPin, OUTPUT);

pinMode(button1, INPUT);

pinMode(button2, INPUT);

}

void loop() {

if(button1 && val\_motor!=255)

{

val\_motor +=5;

analogWrite(motorPin, val\_motor);

delay(30);

}

else if (button2 && val\_motor!=0)

{

val\_motor -=5;

analogWrite(motorPin, val\_motor);

delay(30);

}

}

**Frequency selection:**

Taking **8 – bit timer1** This timer counts from 0 - 255

So it will roll after 255 timer ticks

This pin9 of Arduino has frequency of **490Hz:**

Here, I designed a system in which **duty cycle is controlled by button inputs**. On each button1 press, duty cycle will be increased by 5% and button2 press will decrease duty cycle by 5% as shown in above loop example.

**11.15**

Child processes and, consequently, their threads share the same firmware memory area. As a result, two different threads can be executing the same function at the same time. Functions using *only* local variables are inherently *reentrant*. That is, they can be simultaneously called and executed in two or more contexts.

Local variables are copied to the stack and each invocation will get new copies. On the other hand, functions that use global variables, variables local to the process, variables passed by reference, or shared resources are not reentrant. One must be particularly careful to ensure that all accesses to any common resources are coordinated. When designing the application, one must make certain that one thread cannot corrupt the values of the variables in a second. Any shared functions must be designed to be reentrant.

**11.23**

Below are the characteristics which are different between real time and non real time systems:

1. **Time Constraints:**

Time constraints related with real-time systems simply is time interval allotted for the response of the ongoing program. The task should be completed within this time interval. Real-time system is responsible for the completion of all tasks within their time intervals. Non real time systems do not have any deadlines or any time constraints associated with them.

1. **Correctness:**

Real-time systems are required to produce correct result within the given time interval. If the result is not obtained within the given time interval then also result is not considered correct. Non real time systems do not have any such hard requirement for correct result within given time.

1. **Embedded:**

In embedded systems, combination of hardware and software designed for a specific purpose. Real-time systems collect the data from the environment and passes to other components of the system for processing. Non real time systems can be embedded or non embedded systems.

1. **Safety:**

Safety is necessary for any system but real-time systems are expected to provide critical safety. Real-time systems are supposed to perform for a long time without failures. It also recovers very soon when failure occurs int he system and it does not cause any harm to the data and information. For non real time systems, safety is not a critical factor to consider, for example display of a toy or street light bulb.

1. **Concurrency:**

Real-time systems are concurrent that means it can respond to a several number of processes at a time. There are several different tasks going on within the system and it responds accordingly to every task in short intervals. This makes the real-time systems concurrent systems.

1. **Distributed:**

In various real-time systems, all the components of the systems are connected in a distributed way. The real-time systems are connected in such a way that different components are at different geographical locations. Thus all the operations of real-time systems are operated in distributed ways. Non real time systems do not have as such requirements on connections.

1. **Stability:**

Even when the load is very heavy, real-time systems respond in the time constraint i.e. real-time systems does not delay the result of tasks even when there are several task going on a same time. Stability in real-time systems is critical factor.

Ref: <https://www.geeksforgeeks.org/characteristics-of-real-time-systems/>

11.31

Below are the steps followed when interrupt has occurred:

1. The value of flag register is pushed into the stack. It means that first the value of SP (Stack Pointer) is decremented by 2 then the value of flag register is pushed to the memory address of stack segment.
2. The value of starting memory address of CS (Code Segment) is pushed into the stack.
3. The value of IP (Instruction Pointer) is pushed into the stack.
4. IP is loaded from word location (Interrupt type) \* 04.
5. CS is loaded from the next word location.
6. Interrupt and Trap flag are reset to 0.

11.10

The systems in which the foreground tasks are those initiated by interrupt or by a real-time constraint that must be met, will be assigned the higher priority levels in the system – are best suited for foreground model.

In contrast, system with tasks which are noninterrupt driven and are assigned the lower priorities, Once started, the task will typically run to completion; however, it can be interrupted or preempted by any foreground task at any time – this type of systems can be modeled with background model.

Often separate ready queues will be maintained for the two types of tasks.

So to conclude, foreground modelling is good for real time systems while background modelling is good for nonreal time systems. The background tasks should include all those that do not have tight time constraints.

11.4

Diagram

Description automatically generated

I have scheduled given tasks with **complete fair scheduler-Round Robin** as we don’t know the priorities of the tasks. So to minimize the average waiting time, it will be better to give each task one time slice in sequence. If I schedule them with some other policy then when P(25) is executing, at that time other services may behave like they are never getting a CPU(starving for CPU time). So complete fair scheduler – Round robin policy will be perfect to schedule given tasks.

12.7

In a real-time context, a task that can be determined to always meet its timeliness constraints is said to be *schedulable*. A task that can be guaranteed to always meet all deadlines is said to be *deterministically schedulable*. Such a situation occurs when an event’s worst case response time is less than or equal to the task’s deadline. When all tasks can be scheduled, the overall system can be scheduled.

12.8

In addition to satisfying time constraints, a goal in formulating a task schedule is to keep the CPU as busy as possible, ideally close to 100%, but with some margin for additional tasks. Such a metric is referred to as *CPU utilization*.

Actual CPU utilization varies depending on the amount and type of managed computing tasks. Certain tasks require heavy CPU time, while others require less because of non-CPU resource requirements. CPU utilization may be used to gauge system performance. For example, a heavy load with only a few running programs may indicate insufficient CPU power support, or running programs hidden by the system monitor - a high indicator of viruses and/or malware.

12.11

Embedded system has the following tasks (exec. times, periods): P1(4,16), P2(3,8), P3(2,7).

1. CPU Utilization = (4/16 + 3/8 + 2/7)\*100 = (0.625)\*100 = **62.5%**
2. n = 3 ; 3(2^(1/3) – 1) = 0.77975 => **77.97%**

here, CPU Utilization = 62.5% < 77.97%

so **YES, set of tasks can be scheduled using a rate-monotonic schedule**

1. **UML sequence diagram:**

Diagram

Description automatically generated