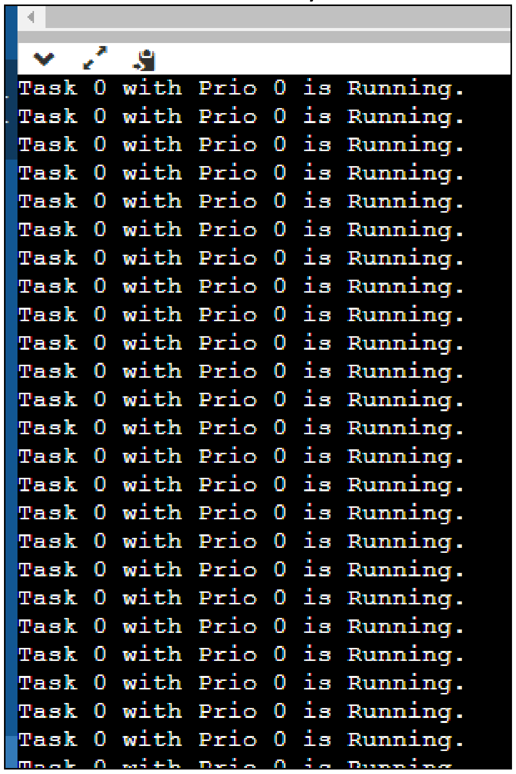
HOMEWORK 5
SWATI KADIVAR

1.a Please find the code in attached zip.

1.b

For now, Just one task 0 with highest priority is running and based on application and requirement, we can enable more tasks. They will run based on priorities assigned.



```
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
Task 0 with Prio 0 is Running.
```

1.c Could you create this state machine in hardware? Would you – why or why not?

- Designing and implementing correct state machines in hardware is generally (much) more complex than doing it in software. I would not create this state machine in hardware because of below reasons.

  - It is physically impossible for the clock signal to arrive at all chip locations at the same time (this is called clock skew), so some flip-flops will be activated before others, a concern that simply does not exist in software.
  - A naive design in hardware might lead to the inference of latches, which impair the time response. It might also lead to the other extreme, which consists of reregistering one or more signals, causing unwanted latency.
  - In hardware, signals might be subject to glitches, another concern that does not occur in software.
  - Contrary to software, hardware allows no abstraction. For example, if a state machine must produce in the next state the same output value produced in the current state, in software we can simply omit the corresponding expression; or if it requires an incrementor, we can simply write x = x + 1. In

either case, an explicit expression would be required in hardware (x = x or x = x + 1), which can only be evaluated if the value of x is available, so the machine itself must provide a means for storing (and properly retrieving) x.

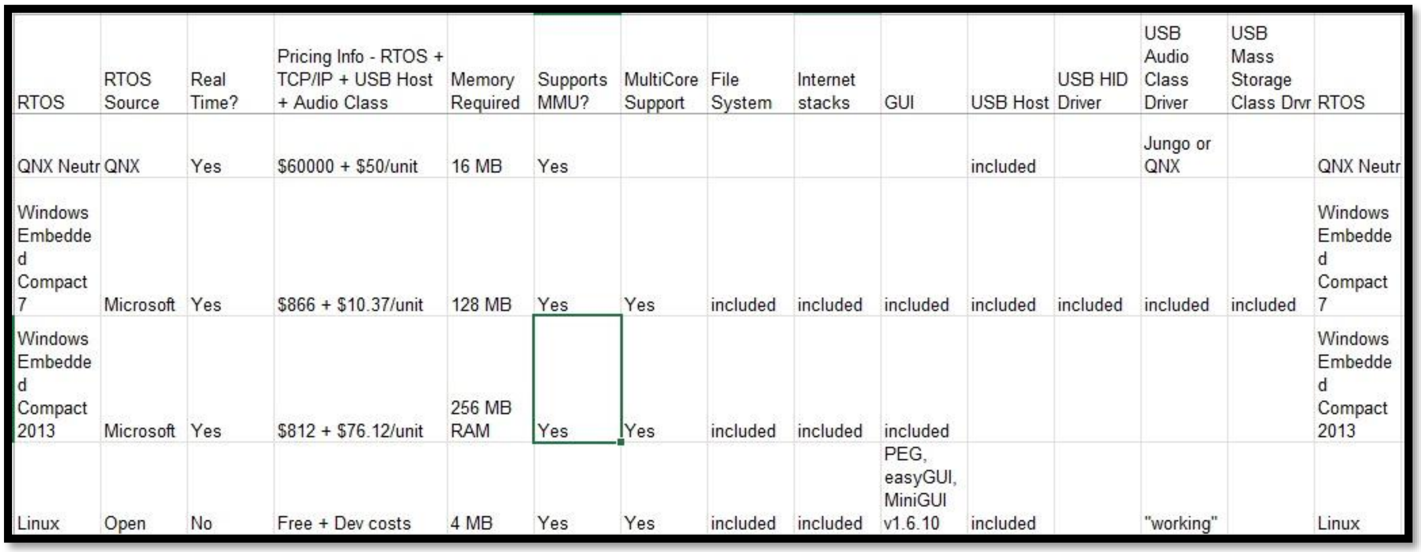
- In hardware, signals represent physical wires, so we cannot assign a signal source to an interconnection now and simply assign another later.
- Many machines have asynchronous inputs, so the use of synchronizers (to avoid flip-flop metastability) must be considered, another concern that does not occur in software-implemented FSMs.
- Some circuits need a special clock, obtained by "gating" the main clock. Depending on how it is done, the resulting clock might be subject to glitches, another issue that simply does not exist in software.
- Several other concerns, such as "reset generation", "capturing the first bit", "keeping the final result stable" and "stretching the decision pulse" are also not a problem in software-implemented machines.

Reference: https://electrovolt.ir/wp-content/uploads/2017/07/Finite-State-Machines-in-Hardware-Volnei-A.-Pedroni-ElectroVolt.ir_.pdf

---

2.a

I would choose Windows Embedded Compact.



RTOS	RTOS Source	Real Time?	Pricing Info - RTOS + TCP/IP + USB Host + Audio Class	Memory Required	Supports MMU?	MultiCore Support	File System	Internet stacks	GUI	USB Host	USB HID Driver	USB Audio Class Driver	USB Mass Storage Class Drvr	RTOS
QNX Neutr	QNX	Yes	$60000 + $50/unit	16 MB	Yes					included		Jungo or QNX		QNX Neutr
Windows Embedded Compact 7	Microsoft	Yes	$866 + $10.37/unit	128 MB	Yes	Yes	included	included	included	included	included	included	included	Windows Embedded Compact 7
Windows Embedded Compact 2013	Microsoft	Yes	$812 + $76.12/unit	256 MB RAM	Yes	Yes	included	included	included					Windows Embedded Compact 2013
Linux	Open	No	Free + Dev costs	4 MB	Yes	Yes	included	included	PEG, easyGUI, MiniGUI v1.6.10	included		"working"		Linux

Linux Operating system:

Pros:

- It's an open-source system so it is free of cost.
- Requires very less memory and resources compared to Windows.
- It has USB audio class driver and USB drivers already included it is used worldwide and has excellent customer support as it is open source.
- It can be run on multiple cores and supports MMU.

- More stable and secure.
- Compatible with wide range of controllers, processors, and embedded PCs.

Cons:

- The major drawback of Linux is it does not support real time process execution.
- For the given application unfortunately, Linux would not be a good choice as the system needs to support Hard real time and soft real time requirements.
- Not very user friendly as drivers are difficult to find as per hardware requirements.

<u>QNX Operating system:</u>

Pros:

- It can perform real time (hard + soft deadlines) requirements.
- The microkernel architecture is reliable that means single component failure doesn't bring the system or kernel down.
- It provides better security which is very important for military purposes.
- It supports multicore architecture.
- It is compatible with wide range of processors and embedded PCs.

Cons :

- Not open source and it is very expensive.
- Doesn’t support low-cost micro controllers from NXP, Silicon Labs etc.

<u>Windows Embedded Compact</u>:

pros:

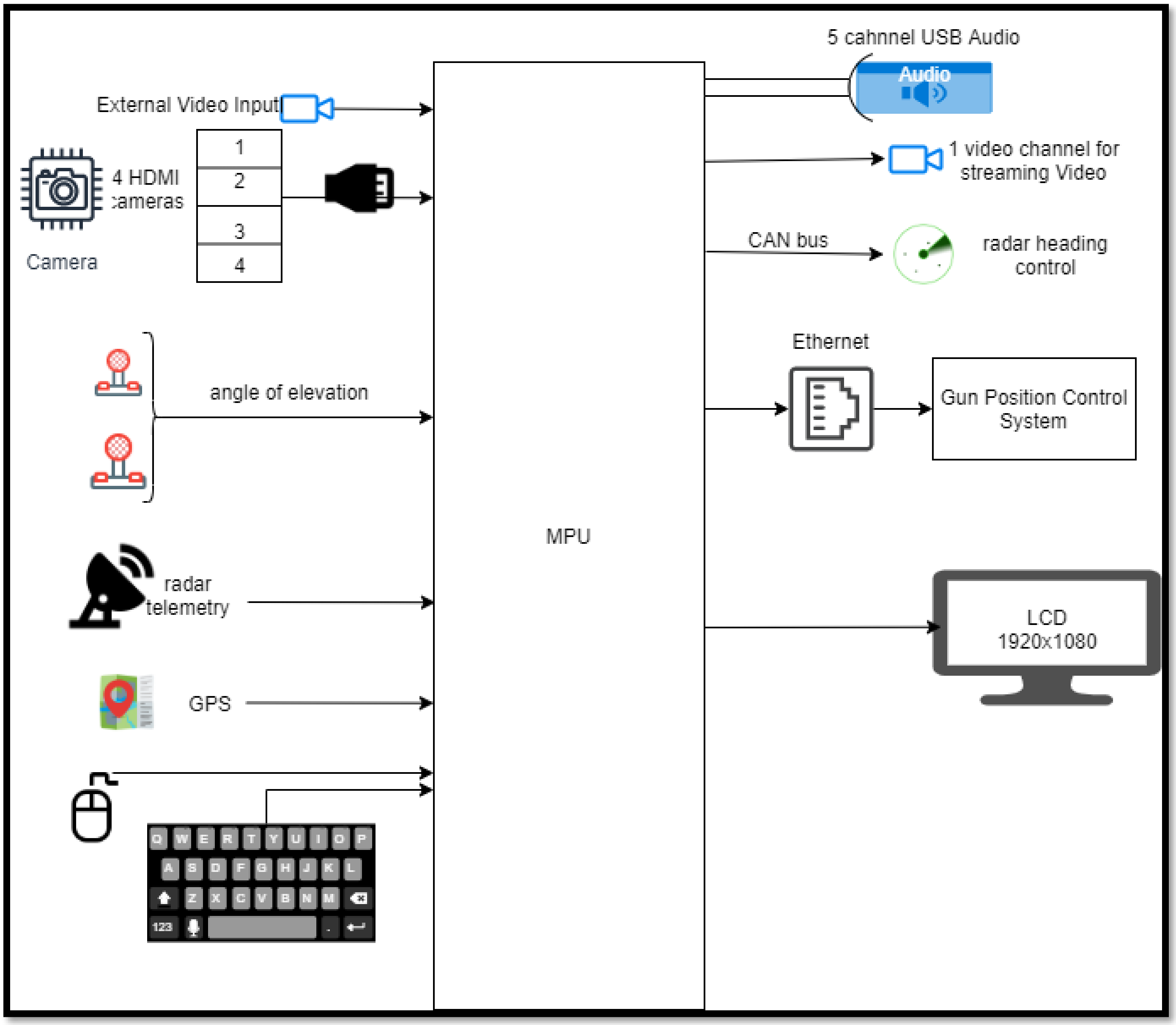
- it supports RTOS, so the soft real time system requirement and hard real time system requirement can be satisfied.
- It is Low cost and easy to manage.
- It does support MMU and have multicore support.
- It supports other various platforms and can be used across various chips.
- It supports Thumb 2 instruction set which reduced code size and improved the execution time.

Cons:

- Windows embedded compact requires more memory than other OS.
- Licensing is required to write applications.
- It has cross platform compatibility issues.
- Difficult to upgrade.
- Troubleshooting is difficult.
- Not scalable due to hardware limitations.

- For the given embedded system, **QnX is best option** as it provides more reliability and security which is required for military standard products. Though the cost is high in case of **military products focus should be on security and reliability** rather than cost. From above listed pros and cons, QnX has minimum cons.

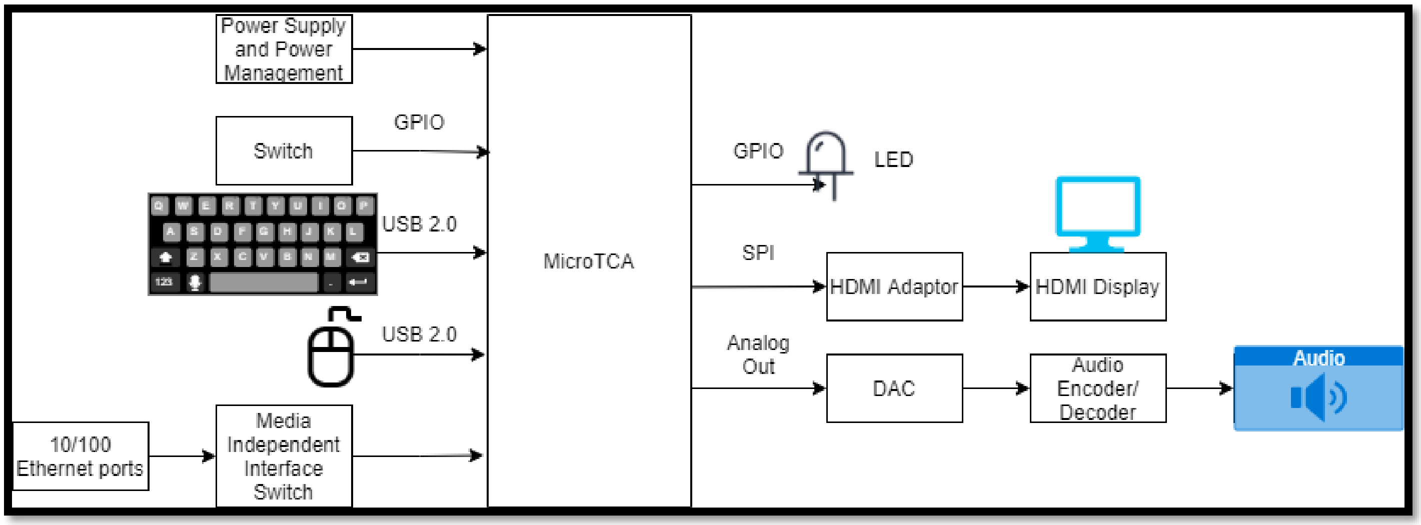
2.b



2.c

I would choose MPU and it would be a good choice to build this type of Architecture as the requirement of external peripherals can be satisfied with MPU. Graphics can be used with MPU and also MPU supports CAN bus and other required communication protocols.

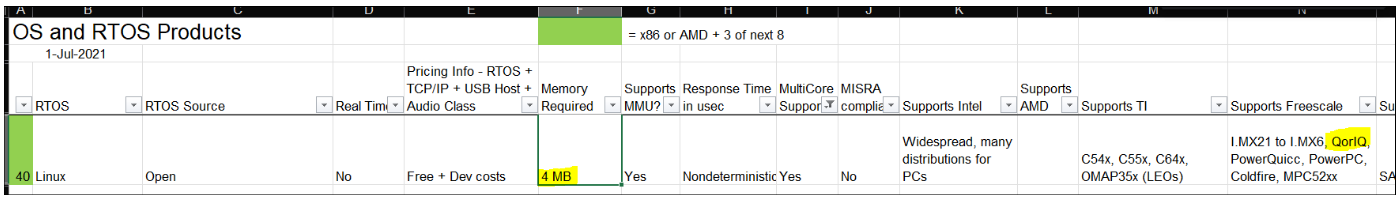
3.a



3.b

- It should be based on a **MicroTCA chassis solution** for the reasons mentioned in 3.e.
- The break-even point for build vs buy for both the solutions comes greater than 24 months. Hence, Avaya should go with the buy option to get more profit from the start.
- If we compare the profits from an Embedded PC vs MicroTCA solutions, MicroTCA gives more profit.
- The system should be based on MicroTCA chassis solutions based on the calculations and reasons explained in the 3.e.

3.c



OS and RTOS Products

1-Jul-2021

= x86 or AMD + 3 of next 8

	RTOS	RTOS Source	Real Tim	Pricing Info - RTOS + TCP/IP + USB Host + Audio Class	Memory Required	Supports MMU?	Response Time in usec	MultiCore Suppor	MISRA complia	Supports Intel	Supports AMD	Supports TI	Supports Freescale	Su
40	Linux	Open	No	Free + Dev costs	4 MB	Yes	Nondeterministic	Yes	No	Widespread, many distributions for PCs		C54x, C55x, C64x, OMAP35x (LEOs)	I.MX21 to I.MX6, QorIQ, PowerQuicc, PowerPC, Coldfire, MPC52xx	SA

I would select **Linux.**

As discussed with Prof today, OS selection is based on processor selection and processor selection is based on OS selection.

- Based on the processor selection done in 3.d, Linux supports QorlQ processors.
- Linux satisfies the Memory requirements for Freescale LS1046ASE8MQA QorlQ.
- Linux has good support for Networking protocols.
- Linux is open-source OS so it is available at no cost.
- For Avaya's VoIP system, Linux satisfies its requirements and so I think Linux will be a perfect fit for this system.

3.d

As given in the requirements, each call requires about 20 DMIPS to process for the G.729 Coded and echo cancellation. For best case of 200 calls we need to accommodate 4000 MIPS, for 500 calls 10000MIPS, for 1000 calls 20000 MIPS and for worse case of 2000 calls, we need to accommodate 40000 MIPS.

- In first Option**: xN Freescale i.MX7 Dual (Nx 2280 DMIPS, $12.72, No FLASH, 768kB SRAM, USB, 2x Gigabit Ethernet)**
    - To achieve max of 40000 MIPS, we need N = 2.
      The cost for this processor = 2 * 12.72 = $25.44
    - To achieve max of 10000 MIPS, we need N = 5.
      The cost for this processor = 5 * 12.72 = $63.6
    - To achieve max of 20000 MIPS, we need N = 9.
      The cost for this processor = 9 * 12.72 = $114.48
    - To achieve max of 40000 MIPS, we need N = 18.
      The cost for this processor = 18 * 12.72 = $228.96

    - The average N for this processor is 9 and the average cost of it is $108.12. So, average total cost will be 973.08.

- In second Option**: xN Freescale LS1046ASE8MQA QorIQ (Nx 22560 DMIPS, $53.53, 2000kB FLASH, 2M SRAM, USB, 7x Gig Ethernet)**
    - To achieve max of 4000 MIPS, we need N = 1.
      The cost for this processor = 1 * 53.53 = $53.53
    - To achieve max of 10000 MIPS, we need N = 1.
      The cost for this processor = 1 * 53.53 = $53.53
    - To achieve max of 20000 MIPS, we need N = 1.
      The cost for this processor = 1 * 53.53 = $53.53
    - To achieve max of 40000 MIPS, we need N = 2.
      The cost for this processor = 2 * 53.53 = $107.06

    - The average N for this processor is 2 and the average cost of it is $66.92. So, average total cost will be 133.83.

- In third Option: **xN Texas Inst. AM3894BCYG150 (Nx 3000 DMIPS $48.30, no FLASH, 576kB SRAM, USB, Gigabit Ethernet)**
    - To achieve as max of 4000 MIPS, we need N = 2.
    The cost for this processor = 2 * 48.30 = $96.6
    - To achieve as max of 10000 MIPS, we need N = 4.
      The cost for this processor = 4 * 48.30 = $193.2
    - To achieve as max of 20000 MIPS, we need N = 7.
      The cost for this processor = 7 * 48.30 = $338.1
    - To achieve as max of 40000 MIPS, we need N = 14.
      The cost for this processor = 14 * 48.30 = $676.2

    - The average N for this processor is 7 and the average cost of it $326.025. So, average total cost will be $2,282.175.

- In fourth option: **xN Texas Inst. AM3352BZCED60 (Nx 1000 MIPS, $8.12, 175kB FLASH, 128kB SRAM, USB, Gig Ethernet)**
    - To achieve as max of 4000 MIPS, we need N = 4.
      The cost for this processor = 4 * 8.12 = $32.48
    - To achieve as max of 10000 MIPS, we need N = 10.
      The cost for this processor = 10 * 8.12 = $81.2
    - To achieve as max of 20000 MIPS, we need N = 20.
      The cost for this processor = 20 * 8.12 = $162.4
    - To achieve as max of 40000 MIPS, we need N = 40.
      The cost for this processor = 40 * 8.12 = $324.8

    - The average N for this processor is 19 and the average cost of it $150.22. So, average total cost will be $2,854.18.

- In fifth option: **x1 Intel Core i7-7500U (49360 MIPS, $393, no FLASH, 4 MB SRAM, USB, Gigabit Ethernet)**

    - To achieve as max of 4000 MIPS, we need N = 1.
      The cost for this processor = 1 * 393 = $393
    - To achieve as max of 10000 MIPS, we need N = 1.
      The cost for this processor = 1 * 393 = $393
    - To achieve as max of 20000 MIPS, we need N = 1.
      The cost for this processor = 1 * 393 = $393
    - To achieve as max of 40000 MIPS, we need N = 1.
      The cost for this processor = 1 * 393 = $393

    - The average N for this processor is 1 and the average cost of it $393. So, average total cost will be $393.

Based on above calculations, **Freescale LS1046ASE8MQA QorIQ seems the best option**. The overall average cost considering all MIPS and Ethernet requirements is lower in Freescale LS1046ASE8MQA QorIQ processor.

3.e

Embedded pc solution:
YC = FC * T = 1000k*T
YPR = GP/U * V
For 200 lines, GP/U = 0, YPR = $0
For 500 lines, GP/U = 4500-3000 = 1500, YPR = 1500 * 250 = $375k
For 1000 lines, GP/U = 7000-3000 = 4000, YPR = 4000 * 250 = $1000k

For 2000 lines, GP/U = 12000 - 3000 = 9000, YPR = 9000*250 = $2250k
Total **YPR = $3625k**

YI = YPR * YTP – YC = 3625k * T – 1000k * T **= 2625k * T**
Profit for first Year = 2625k * 12 = $31500k

Cost savings per month = (YUC – DUC) * V = (3000-1000)*1000 = $2000k

DI = V * (GP/U + (YUC – DUC)) * (T-12) - FC * T – NRE
  = (3625k + 2000k)*(T-12) – 1000kT – 4000k
  = 5625kT – 67500k – 1000kT – 4000k
  = 4625kT – 71500k

DI >= YI
4625kT – 71500k = 2625kT
2000kT = 71500k
T = 35.75 months

Micro TCA solution:

YC = FC * T = 1000k*T
YPR = GP/U * V
For 200 lines, YPR = (3000 – (2500+200)) * 250 = $75k
For 500 lines, YPR = (4500 - (2500+200)) * 250 = $450k
For 1000 lines, YPR = (7000 – (2500+400)) * 250 = $1025k
For 2000 lines, YPR = (12000 - 3300) * 250 = $2175k
Total **YPR = $3725k**

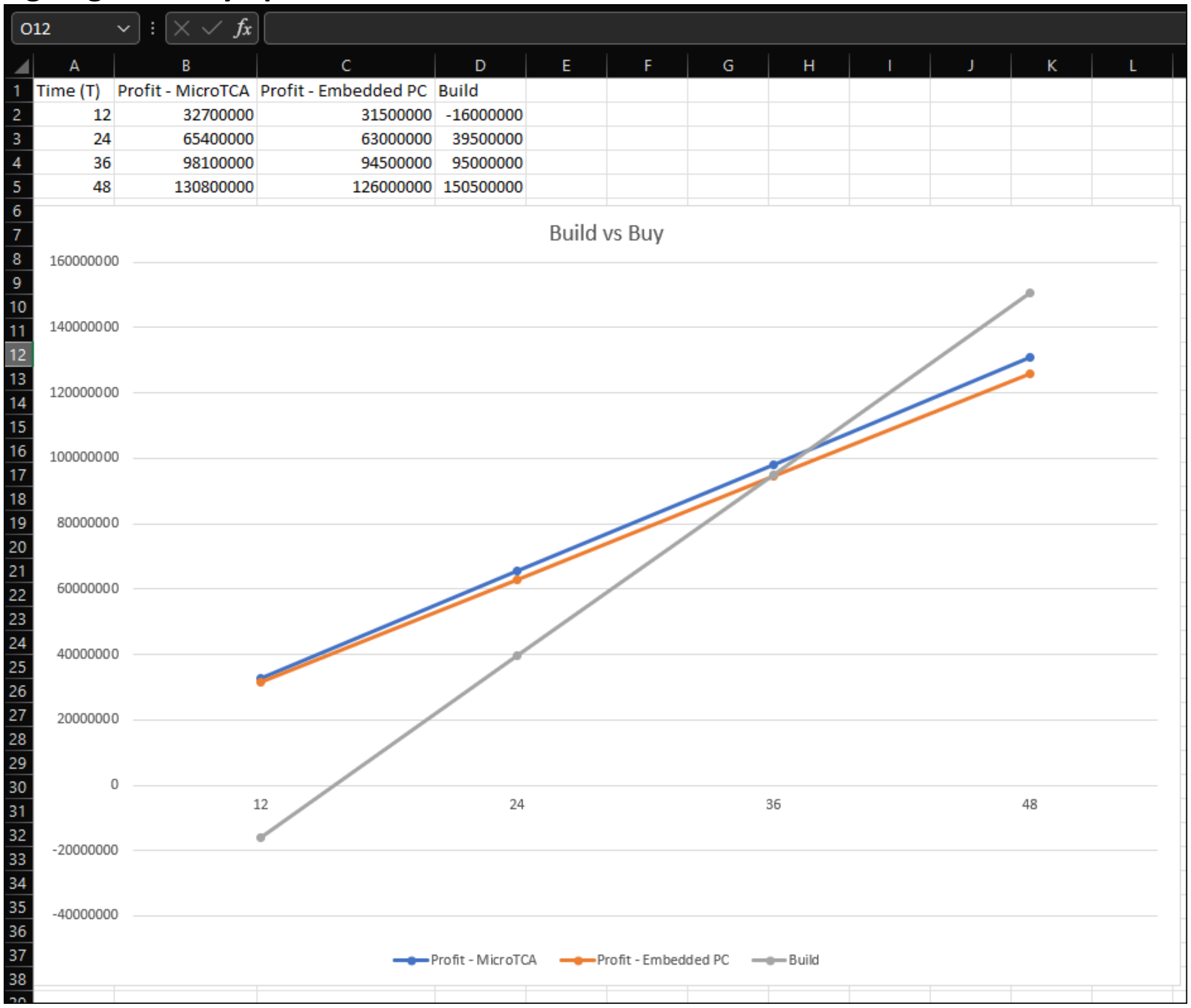
YI = YPR * YTP – YC = 3725k * T – 1000k * T **= 2725k * T**
Profit for first Year = 2725k * 12 = $32700k

Cost savings per month = (YUC – DUC) * V
For 200 lines, (2700-1000)*250 = $425k
For 500 lines, (2700-1000)*250 = $425k
For 1000 lines, (2900-1000)*250 = $475k
For 2000 lines, (3300-1000)*250 = $575k
Total cost savings per month = $1900k

DI = V * (GP/U + (YUC – DUC)) * (T-12) - FC * T – NRE
  = (3725k + 1900k)*(T-12) – 1000kT – 4000k
  = 5625kT – 67500k – 1000kT – 4000k
  = 4625kT – 71500k

DI >= YI
4625kT – 71500k = 2725kT
1900kT = 71500k
T = 37.63 months

I am going with Buy option.



O12

	A	B	C	D
1	Time (T)	Profit - MicroTCA	Profit - Embedded PC	Build
2	12	32700000	31500000	-16000000
3	24	65400000	63000000	39500000
4	36	98100000	94500000	95000000
5	48	130800000	126000000	150500000

In **Buy** option we can see the break-even point for Embedded PC is ~36 months and for MicroTCA it is ~38 months – not much difference. Now from the graph it is clearly shows that eventually the Profit from MicroTCA is more compared to the Embedded PC before the break-even point and even after the break-even point. So, according to me, Avaya should go with MicroTCA solution.

Problems: 14.4, 14.23

14.4 The binary search algorithm has a worst-case complexity of O(log2N) when N is the size of the search range. Provide complexity analysis to show that this is correct.

Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise, narrow it to the upper half. Repeatedly check until the value is found, or the interval is empty.

Sorted Array of 10 elements: 2, 5, 8, 12, 16, 23, 38, 56, 72, 91

- Iteration 1:

Array: 2, 5, 8, 12, 16, 23, 38, 56, 72, 91
Select the middle element. (Here 16)
Since 23 is greater than 16, so we divide the array into two halves and consider the sub-array after element 16.
Now this subarray with the elements after 16 will be taken into next iteration.

- Iteration 2:

Array: 23, 38, 56, 72, 91
Select the middle element. (Now 56)
Since 23 is smaller than 56, so we divide the array into two halves and consider the sub-array before element 56.
Now this subarray with the elements before 56 will be taken into next iteration.

- Iteration 3:

Array: 23, 38
Select the middle element. (Now 23)
Since 23 is the middle element. So, the iterations will now stop.

Let's say the iteration in Binary Search terminates after k iterations. In the above example, it terminates after 3 iterations, so here k = 3
At each iteration, the array is divided by half. So, let's say the length of array at any iteration is n

At Iteration 1, Length of array = n
At Iteration 2, Length of array = n/2
At Iteration 3, Length of array = (n/2)/2 = $n/2^2$
Therefore, after Iteration k, Length of array = $n/2^k$

Also, we know that, after k iterations, the length of array becomes 1

Therefore
Length of array = $n/2^k$ = 1=> n = 2k

Applying log function on both sides: => log2 (n) = log2 (2k) => log2 (n) = k log2 (2)

As (log a(a) = 1)
Therefore,
=> k = log2(n)
Ref: https://www.geeksforgeeks.org/complexity-analysis-of-binary-search/

14.23 A colleague has designed an embedded application that utilizes a timer that is supposed to interrupt every 5 ms. She is debugging the code and claims that she cannot determine whether the timer is working properly. Suggest a way she can instrument the code to enable her to determine whether the program is entering the interrupt service routine and, if so, at what interval.

While keeping the best practices in writing ISRs, one can modify a piece of software (specifically the ISR) to make sure that she can verify ISR's expected behavior. It is necessary to keep the ISR short so we can not add a significant piece of code.

She can modify a code in below listed ways to determine ISR working or not and find interval:

- A simple way could be to set flag variables to call a function in the main code which sets /clears GPIO functionality.
- The best thing could be an LED.
- A simple print statement when this function is called from the main loop would make sure that the timer interrupt is firing, and the mechanism is set up correctly (One needs to careful with print statements and ISRs)
- A logic analyzer could be helpful in determining if the correct timing is achieved and that the duration of 5ms is achieved.
- It possible to check the values in the timer load register as well if JTAG debugging is supported and register values can be read.
- The next thing would be to make sure that the correct value is loaded in the timer register.