EEE3095/6S Term 3 Quiz

30 August 2023

Duration: 45 minutes Total: 50 marks

Empl ID: _____

Instructions:

- This is a closed-book test.
- There is only one part to this test.
- You must use a pen to complete the test — pencil will only be marked for drawings. Values indicated on drawings must be written in pen. Please show all of your working clearly — method and approach matter.
- Keep all your answers within the allocated blocks; anything outside of these blocks may not be marked.
- A formula sheet appears at the end of this paper.

Question	Available Marks	Grade
1	10	
2	20	
3	20	
Total:	50	

Part A

Empl ID: | | | | | | | |

Question:	1	2	3	Total
Points:	10	20	20	50
Score:				

Question 1: Multiple Choice [10 marks]

a. If a Serial Peripheral Interface (SPI) protocol is configured to operate in SPI Mode 2, the logic level of the clock signal is: (2)

√ **high while idling**
◯ low while idling
◯ always high
◯ always alternating

b. The 32-bit value 0x13A04B71 needs to be stored in memory; if using Big Endian, which value would be stored at the smallest memory address? (2)

◯ 0b01110001
◯ 0b00100001
√ **0b00010011**
◯ 0b00010111

c. How would a dynamic, asynchronous real-time system function? (2)

◯ Predictable task arrival with periodic execution
◯ Predictable task arrival with aperiodic execution
◯ Unpredictable task arrival with periodic execution
√ **Unpredictable task arrival with aperiodic execution**

d. An execution environment can have more than one runtime environment. True or false? (2)

√ **True**
◯ False

e. An Application Binary Interface (ABI) is said to be at "source-code level" and relates to the interface between different software components. True or false? (2)

◯ True
√ **False**

Question 2: ARM Processing [20 marks]

a. Define the terms **RISC** and **CISC** and explain the main difference between them. (3)

Solution: RISC: Reduced Instruction Set Computer
CISC: Complex Instruction Set Computer
CISC has more instructions available and thus allows for more complex processing, requiring less code/software than RISC

b. Two well-known optimisation strategies are Small Function In-lining (SFI) and Loop Unrolling (LUR). Briefly describe each of these with at least one advantage associated with the optimisation. (4)

Solution: SFI is all about taking a small function and implements it directly into the calling function instead; saves code space and potentially reduces call/return overhead.
LUR is used to cut down the number of iterations in a loop; reduces looping overhead (such as condition checks) and allows more aggressive instruction scheduling.

c. Mixing ARM Assembly code with C code requires the use of the same Application Binary Interface (ABI), and this defines the calling convention for our functions. Answer the following questions:

(i) What is the difference between an ABI and an API (Application Programming Interface)? (2)

Solution: An API is higher level and defines the interface between software components at the source code level; on the other hand, the ABI is lower level and defines the binary/machine code connection to modules in the application.

(ii) A calling convention can be split up into two components; name these parts and provide a brief description for each one. (2)

Solution: Function prologue: happens at the start of the function and sets up the stack for local variables
Function epilogue: happens at the end of the function and frees up local variables before returning to the calling function

(iii) List two reasons why a programmer may want to combine ARM Assembly and C code. (2)

Solution: Any two of the following:

- Interfacing an Assembly program with C libraries (or vice-versa)
- Performance – writing speed-critical key parts of a program in Assembly, and leaving the rest in C, may boost overall performance
- Size – human-written Assembly can be smaller than compiler-generated code
- Doing something very fast and specialised (through Assembly) as a tiny portion of your (larger) C program

d. Given the following ARM Assembly code, what would be the base-10 value in register **r0** after each line of code? Briefly explain your working. (7)

```
mov r0, #16
mov r1, 0b1000
```

```
add r0, r0, r1
asr r0, #2
lsr r0, #1
mul r1, r0, r1
mov r0, 0x11
```

Solution: First mov operation sets r0 = 16
Second mov operation does not affect r0, so r0 = 16
add operation sets r0 = r0 + r1 = 16 + 8 = 24
asr operation sets r0 = $r0/2^2 = 6$
lsr operation sets r0 = $r0/2^1 = 3$
mul operation does not affect r0, so r0 = 3
Third mov operation sets r0 = 0x10 = $(11)_{16} = (17)_{10} = 17$

Question 3: Embedded Communications [20 marks]

a. The 7-bit data word 0b1010111 is to be transmitted with a parity bit appended; what would be the final 8-bit data word that is transmitted in both odd- and even-parity systems? (2)

Solution: The 7-bit data contains 5 ones, so:

- in an odd parity system, the full 8-bit data would be 0b10101110 (a zero is appended for a total of 5 ones)
- in an even parity system, the full 8-bit data would be 0b10101111 (a one is appended for a total of 6 ones)

b. SPI uses a Slave Select line to initiate communication with a specific slave device, whereas I2C (Inter-Integrated Circuit) uses a different means of selecting the slave device with which the master communicates. Explain how this is done, and describe what would happen if an error occurs during this slave selection process. (2)

Solution: The master transmits a 7-bit address on the SDA line to indicate the address of the slave with which it will be communicating. If an error occurs, or the 7-bit address is invalid, the SDA line will be left High by all to indicate a NACK bit (not acknowledged by slave device).

c. RS-485 uses a "twisted cable" configuration; what is the purpose of this? (2)

Solution: RS-485 use twisted cables so that they pick up less noise (by reducing interference from external electromagnetic fields), and this is achieved by minimising the gap between the two wires.

d. The following questions relate to the SPI communication protocol.

(i) Figure 1 shows the basic structure of the SPI protocol's interface. Provide names for the four missing labels. (2)

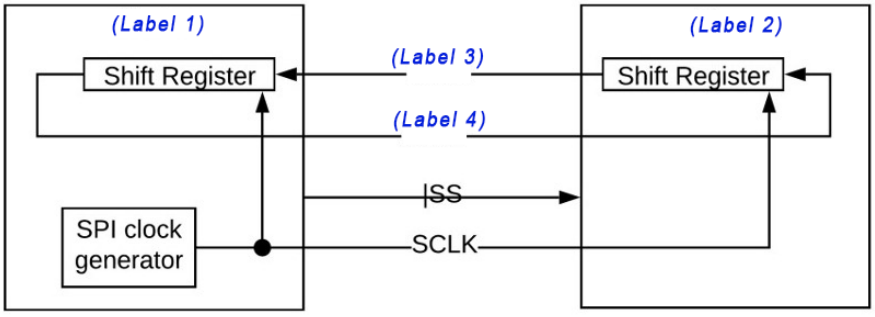


Figure 1: SPI interface with missing labels

Solution: Label 1: Master
Label 2: Slave
Label 3: MISO
Label 4: MOSI

(ii) SPI is said to be a full-duplex protocol. Explain what this means and make reference to Figure 1 in your answer. (2)

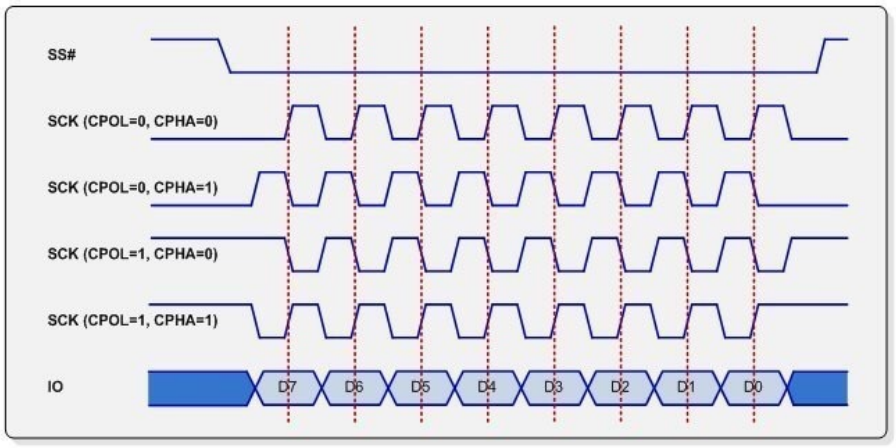
Solution: Both the master and slave devices can communicate simultaneously to each other as there are separate data lines for transmission from master to slave (MOSI) and from slave to master (MISO), as illustrated by the directional arrows between the two devices in Figure 1.

(iii) The four SPI Modes are dependent on the Clock Phase (CPHA) and Clock Polarity (CPOL). Explain these two terms. (2)

Solution: CPHA: Controls whether data is sampled on the first or second clock edge of every cycle
CPOL: Indicates the logic level of the clock signal when idling

(iv) Based on the above, draw a basic SPI timing diagram to illustrate the differences between all four SPI Modes; you only need to draw the clock lines and SS. (4)

Solution: The following figure shows the correct clock lines for SPI Mode 0 to SPI Mode 3; all four clock lines must be shown with their corresponding SPI Mode and sample timing, and the SS line should be indicated simply to illustrate communication being started. IO line is not required.



(v) Draw a diagram to illustrate how we could connect three **independent** slave devices to a single master device using SPI. (4)

Solution: The following figure shows the final result; the student needs to indicate all three SS lines (with different labels), and all connections should be accurate – namely the shared MOSI/MISO/SCLK lines and the independent SS lines.

END OF TEST

Appendix B: ARM Assembly Language Cheatsheet

Memory access instructions

```
LDR Rd, [Rn]            ; load 32-bit number at [Rn] to Rd
LDR Rd, [Rn,#off]       ; load 32-bit number at [Rn+off] to Rd
STR Rt, [Rn]            ; store 32-bit Rt to [Rn]
STR Rt, [Rn,#off]       ; store 32-bit Rt to [Rn+off]
PUSH {Rt}               ; push 32-bit Rt onto stack
POP  {Rd}               ; pop 32-bit number from stack into Rd

MOV{S} Rd, <op2>        ; set Rd equal to op2
MOV Rd, #im16           ; set Rd equal to im16, im16 is 0 to 65535
MVN{S} Rd, <op2>        ; set Rd equal to -op2
```

Branch instructions

```
B label                 ; branch to label Always
BEQ label               ; branch if Z == 1 Equal
BNE label               ; branch if Z == 0 Not equal
BCS label               ; branch if C == 1 Higher or same, unsigned ≥
BHS label               ; branch if C == 1 Higher or same, unsigned ≥
BCC label               ; branch if C == 0 Lower, unsigned <
BLO label               ; branch if C == 0 Lower, unsigned <
BMI label               ; branch if N == 1 Negative
BPL label               ; branch if N == 0 Positive or zero
BVS label               ; branch if V == 1 Overflow
BVC label               ; branch if V == 0 No overflow
BHI label               ; branch if C==1 and Z==0 Higher, unsigned >
BLS label               ; branch if C==0 or Z==1 Lower or same, unsigned ≤
BGE label               ; branch if N == V Greater than or equal, signed ≥
BLT label               ; branch if N != V Less than, signed <
BGT label               ; branch if Z==0 and N==V Greater than, signed >
BLE label               ; branch if Z==1 or N!=V Less than or equal, signed ≤
BX Rm                   ; branch indirect to location specified by Rm
BL label                ; branch to subroutine at label
BLX Rm                  ; branch to subroutine indirect specified by Rm
```

Logical instructions

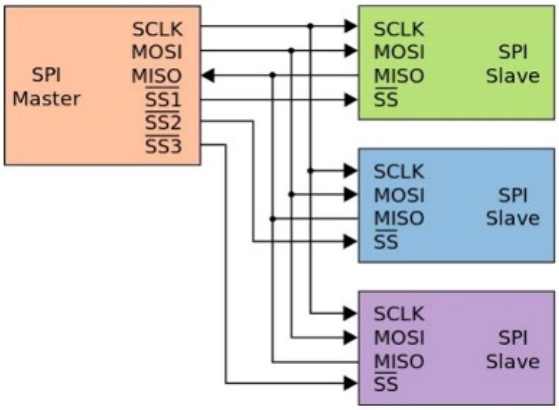
```
AND{S} {Rd,} Rn, <op2>  ; Rd=Rn&op2 (op2 is 32 bits)
ORR{S} {Rd,} Rn, <op2>  ; Rd=Rn|op2 (op2 is 32 bits)
EOR{S} {Rd,} Rn, <op2>  ; Rd=Rn^op2 (op2 is 32 bits)
BIC{S} {Rd,} Rn, <op2>  ; Rd=Rn&(~op2) (op2 is 32 bits)
ORN{S} {Rd,} Rn, <op2>  ; Rd=Rn|(~op2) (op2 is 32 bits)
LSR{S} Rd, Rm, Rs       ; logical shift right Rd=Rm>>Rs (unsigned)
LSR{S} Rd, Rm, #n       ; logical shift right Rd=Rm>>n (unsigned)
ASR{S} Rd, Rm, Rs       ; arithmetic shift right Rd=Rm>>Rs (signed)
ASR{S} Rd, Rm, #n       ; arithmetic shift right Rd=Rm>>n (signed)
LSL{S} Rd, Rm, Rs       ; shift left Rd=Rm<<Rs (signed, unsigned)
LSL{S} Rd, Rm, #n       ; shift left Rd=Rm<<n (signed, unsigned)
```

Arithmetic instructions

```
ADD{S} {Rd,} Rn, <op2>   ; Rd = Rn + op2
ADD{S} {Rd,} Rn, #im12   ; Rd = Rn + im12, im12 is 0 to 4095
SUB{S} {Rd,} Rn, <op2>   ; Rd = Rn - op2
SUB{S} {Rd,} Rn, #im12   ; Rd = Rn - im12, im12 is 0 to 4095
RSB{S} {Rd,} Rn, <op2>   ; Rd = op2 - Rn
RSB{S} {Rd,} Rn, #im12   ; Rd = im12 – Rn
CMP Rn, <op2>            ; Rn – op2 sets the NZVC bits
MUL{S} {Rd,} Rn, Rm      ; Rd = Rn * Rm signed or unsigned
```

Notes

Ra Rd Rm Rn Rt represent 32-bit registers
value any 32-bit value: signed, unsigned, or address
{S} if S is present, instruction will set condition codes
#im12 any value from 0 to 4095
#im16 any value from 0 to 65535
{Rd,} if Rd is present Rd is destination, otherwise Rn
#n any value from 0 to 31
#off any value from -255 to 4095
label any address within the ROM of the microcontroller
op2 the value generated by <op2>


SPI
Master
SCLK
MOSI
MISO
SS1
SS2
SS3
SCLK
MOSI
MISO
SS
SPI
Slave
SCLK
MOSI
MISO
SS
SPI
Slave
SCLK
MOSI
MISO
SS
SPI
Slave