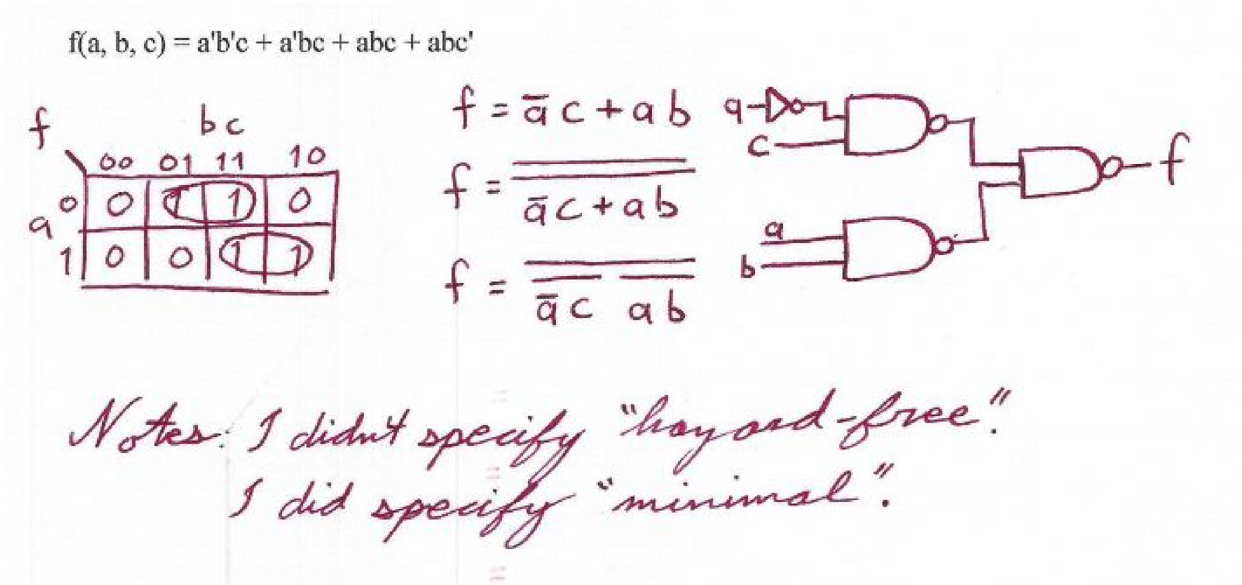
Solutions to EECS 373 Midterm Exam
Winter 2017

1)



2)

i. Considering all pairwise interactions, how do the number of interactions between components in an embedded system increase as a function of the number of components (n)? ________n(n-1)/2________

ii. In general, how does debugging time change as a function of the number of interactions that may introduce bugs? Circle the correct answer.

a) Sublinearly
b) Linearly
c) Superlinearly

iii. Using two phrases, each of which has six or fewer words, point out two primary approaches to debugging that are synergistic when used together.

1) Testing to gather data.
2) Analysis/reasoning to design better tests.

3) For each description, fill in the blank with the letter associated with the correct tool.
A. objdump
B. as
C. objcopy
D. ld
E. make
F. gcc
G. nm

__D__Combines object files and libraries into a single file, resolving symbols in the process.
__B__Converts human-readable assembly language code into machine-readable object files.
__F__Converts human-readable C language code into machine-readable object files.
__A__Displays instructions and data in object files in human-readable form.
__G__Displays symbol table in object or executable, including symbol resolution status.
__C__Translates object files from one format to another, potentially stripping debugging information and symbols in the process.
__E__Builds a presence and modification time based dependency graph transitively connecting
build sources and build targets and executes build rules to traverse the graph, thus producing
build targets.

4)

```
module blinky(
input PCLK, input PRESERN,
input PSEL, input PENABLE,
input [7:0] PADDR, output PREADY,
output PSLVERR, input PWRITE,
input [31:0] PWDATA, output [31:0] PRDATA
output light );
reg [31:0] count, width, period;
reg light, on;

assign PREADY=1'b1;
assign PSLVERR=1'b0;

always@(posedge PCLK) begin

	if(PSEL && PENABLE && PWRITE && (PADDR==8'd0))
		on <= PWDATA;
	if(PSEL && PENABLE && PWRITE && (PADDR==8'd4))
		period<= PWDATA;
		width <= period / 2;

	if(PSEL && PENABLE && ~PWRITE)
		PRDATA <= period;

	If (~PRESERN)
		on <= 32'd0;

	if(~PRESERN || (count>period) )
		count<=32'd0;
	else
		count <=count+32'd1;

	if (count<width && on)
		light <=1'b0;
	else
		light <= 1'b1;
end
endmodule
```

5)

i. Arguments that are passed into subroutines are placed in registers r0-r3. If there is not enough room there, the extra arguments are placed on the stack.

ii. The result of a function is placed into r0 or registers r0 and r1 before returning. Note: “r0-r3 and stack” was accepted as an answer for the second and third blanks because there are very specific cases in which they are used.

iii. When a subroutine is called, the return address is placed in the link register.

iv. The PC holds the address of the next instruction to be executed.

v. For this course, we are using ARM in little endian mode.

vi. Registers r4-r8, r10, r11, sp need to be saved by a subroutine before being used, while registers r0-r3 need to be saved by a function before calling a subroutine. Note: for vi. an answer which included r9 was accepted because it is required for certain systems, however it is not typical.

vii. To return from a subroutine, the instruction bx lr must be used.

6)

```
uint32_t goblue(uint32_t x[], uint32_t y[], int n ) {
	int i;
	int sumx = 0;
	int sumy = 0;
	for(i=0; i<n; i++){
		if(x[i] > y[i])
			sumx = conv(x[i])+sumx;
		else
			sumy = conv(y[i])+sumy;
	}
	return (sumy - sumx);
}
```

```
goblue:
	push {r4,r5,r6,r7,r8,lr}
	mov r4,r0 @x
	mov r5,r1 @y
	mov r6,r2 @i=n
	mov r7,#0 @sumx
	mov r8,#0 @sumy
	ldr r0,[r4],#0 @get x
	ldr r1,[r5],#0 @get y

for:
	cmp r0,r1 @x[i] > y[i]
	bgt xgty
	mov r0,r1 @ x < y
```

```
        bl conv @conv value
        add r8,r8,r0 @ sumy
        b cont
xgty:
        bl conv @conv value
        add r7,r7,r0 @sumx
cont:
        ldr r0,[r4],#4 @get x and post increment to next value
        ldr r1,[r5],#4 @get y and post increment to next value
        add r6,-1 @i--
        cmp r6,0
        bgt for @i == 0?

        sub r0,r8,r7 @
        pop {r4,r5,r6,r7,r8,lr}
        bx lr
```

7)

```
void Timer_ISR(void) {
  uint32_t * GPIO1 =    (uint32_t *)  0x12345678;
  uint32_t * compare =  (uint32_t *)  0x876543210;
  uint32_t * overflow = (uint32_t *)  0x876543214;
  uint32_t * status =   (uint32_t *)  0x876543218;
  volatile uint32_t * duty_cycle = (uint32_t *) 0x45671234;

  // set overflow
  *overflow = 10000000/100; // aka 100,000

  // set output
  if(*status & 0x1) {
    *GPIO1 = 0x0;
  }
  else if (*status & 0x2) {
    *GPIO1 = 0x1;
    *compare = duty_cycle / 100.0 * 10000000/10;
  }
}
```

8)

Part 1:

```
int initGPIO(int gpioNum, int IO_mode) {

	volatile uint32_t *BASE_ADDR = (volatile uint32_t *)GPIO_x_CFG;

	if (IO_mode == INPUT_MODE) {
		if(*(BASE_ADDR + gpioNum) & 0x2) return 0;
		*(BASE_ADDR + gpioNum) |= 0x2;
		*(BASE_ADDR + gpioNum) &= 0xFFFFFFFE;
	} else if (IO_mode == OUTPUT_MODE) {
		if(*(BASE_ADDR + gpioNum) & 0x1) return 0;
		*(BASE_ADDR + gpioNum) |= 0x1;
		*(BASE_ADDR + gpioNum) &= 0xFFFFFFFD;
	}
	return 1;
}
```

Part 2:

```
__attribute__ ((interrupt)) void Fabric_IRQHandler( void ) {
	initGPIO(0, INPUT_MODE);
	wait_1ms();
	volatile uint32_t *IN_BASE = (volatile uint32_t *) GPIO_IN;
	int result = *(IN_BASE) & (0x1);
	initGPIO(1, OUTPUT_MODE);
	if (result) {
		setGPIO(1, 0);
	} else {
		setGPIO(1, 1);
	}
}
```