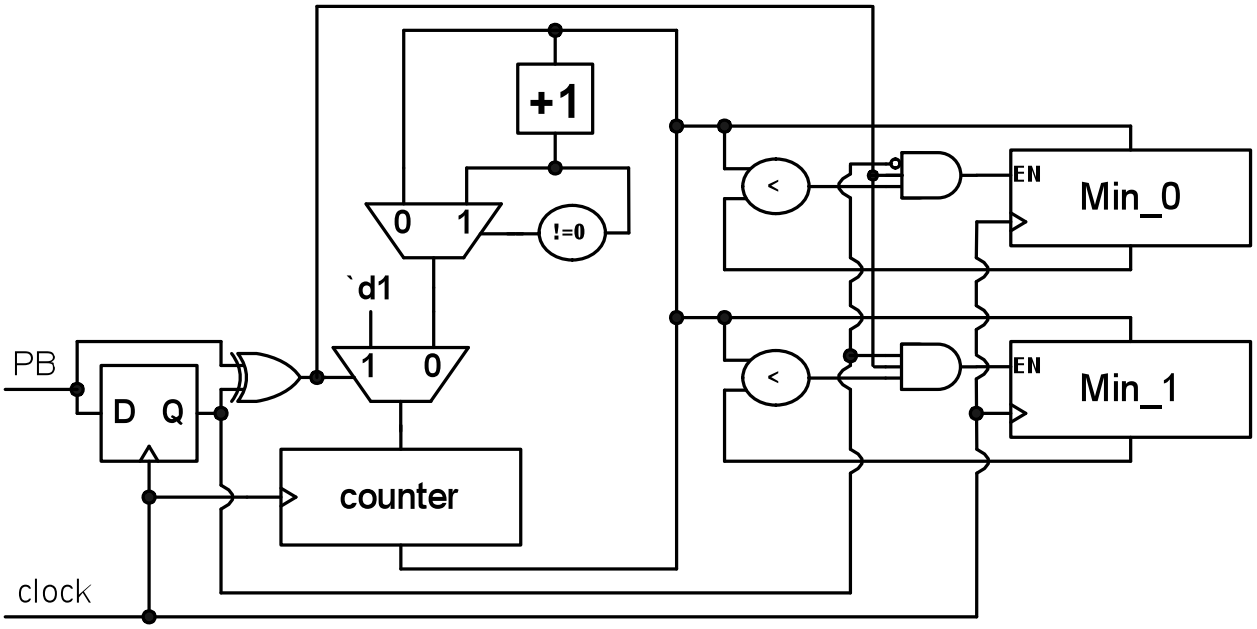
Problem 1: The basic idea is to monitor the push button signal and measure the runs of 0s and 1s respectively. Note, whenever a new run of 0s or 1s starts, the first value of the counter must be ‘d1 (i.e., we already have the first sample). When an edge on the push-button signal occurs, then next state of the counter will change to ‘d1 and the present state of the counter needs to be compared against two registers that store the min values (Min_0 and Min_1 in the figure below).

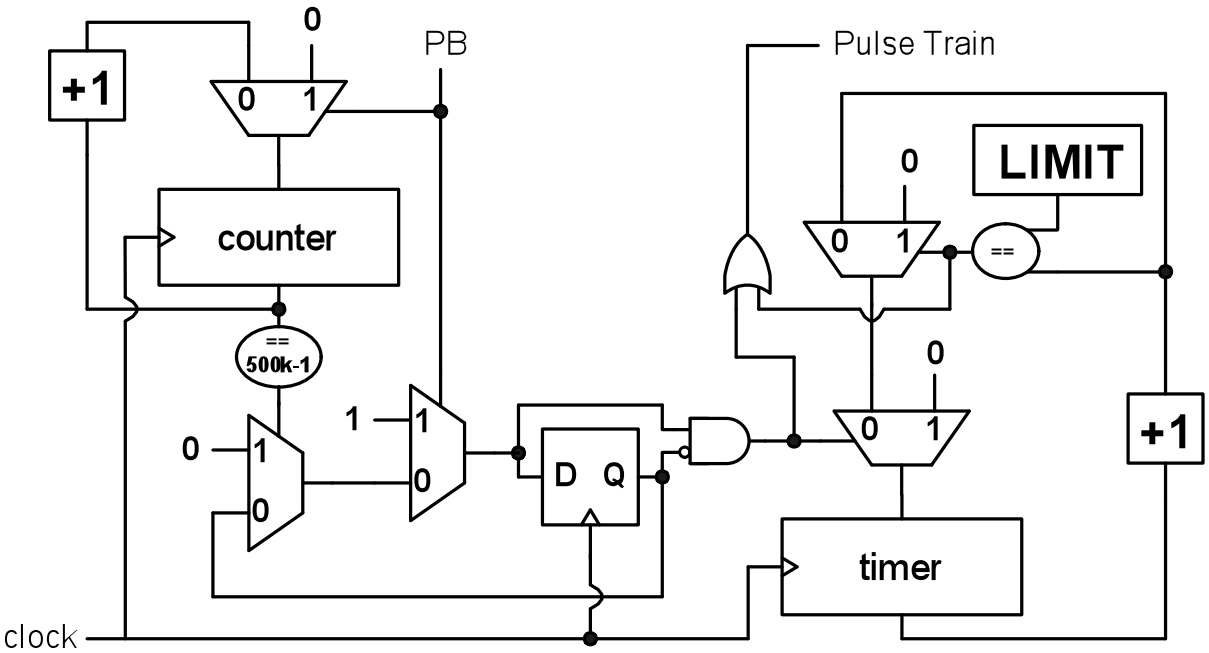
The Min_0 register is updated when the following conditions occur: an edge is detected on the push button; we have just finished a run of 0s and if the minimum value in the Min_0 register is greater than the present state of the counter. The same principles are applied to updating register Min_1.

Note, because the run of 0s or 1s can be excessive when the push button is released or pressed for a very long time, we decide saturate the counter to its maximum value. Although in the implementation below, we do not specify explicitly the bitwidth of the counter, this parameter should be given as part of the spec (e.g., having runs of 0s or 1s larger than 20 ms will be highly unlikely for most switches).

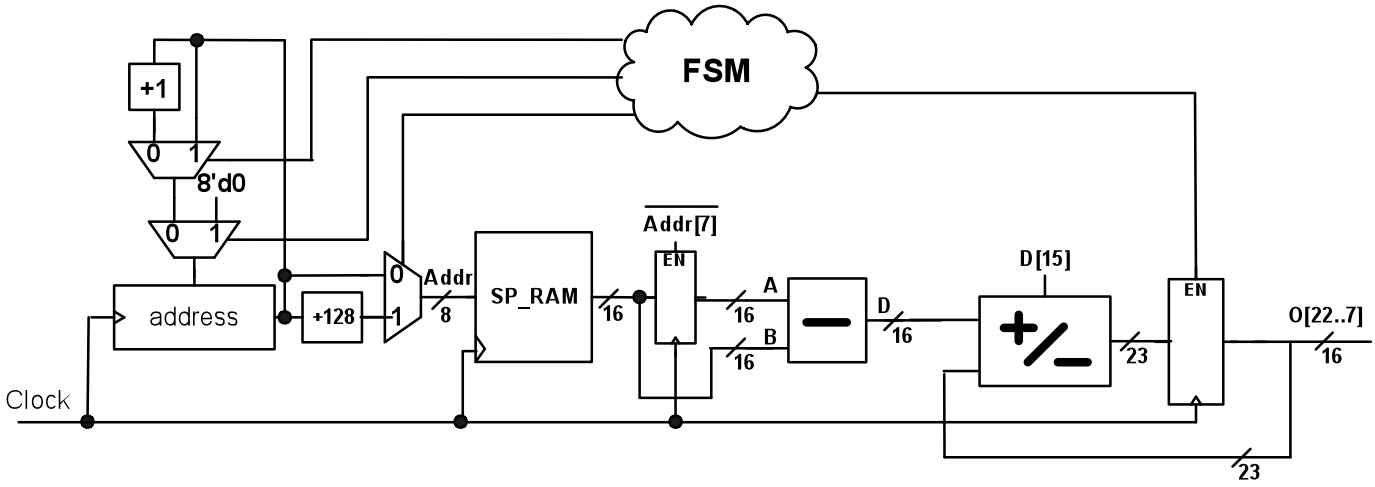


Problem 2: The debouncing in this implementation is done using a reference counter. Because the debounced signal is lowered after having a run of 0s on the push button larger than 10 ms, the counter needs to be compared against 499,999 to reset the debounced signal. Due to our implementation choice, there is no need to reset the counter when it reaches 499,999. The counter can continue counting and it will "eventually" roll-over. Of course, to save resources the number of flip-flops in the counter should be allocated to count up to 499,999, i.e., 19 bits.

The reason why there is no need to reset the counter is because the debounced signal will be set to a 1 each time PB is 1 and reset to 0 whenever the counter reaches 499,999. In any other case it will keep its value (see the flip-flop driven by two 2-to-1 muxes on which edge detection is done). The output of this flip-flop will be used to generated the first pulse, as well as to reset the timer. The timer will count until it will reach the user-specified limit, at which point it will reset itself and generate a new pulse.

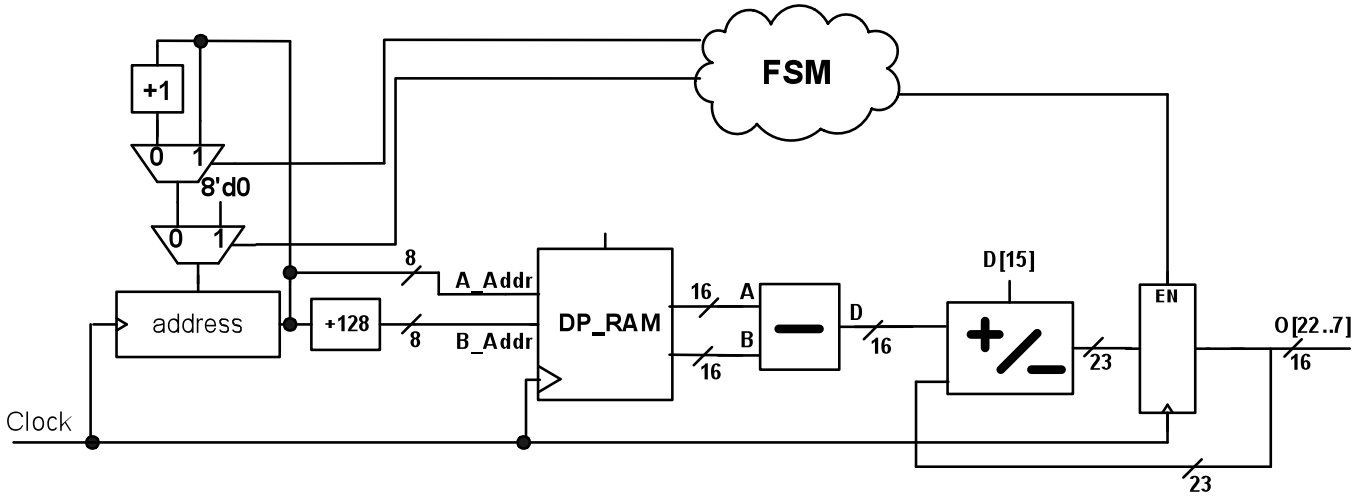


Problem 3: The data path and the state table for the single-port RAM implementation are given below. Note we employ an adder/subtractor for accumulation. When the sign bit of the difference between A[i] and B[i] is 1, then instead of taking its absolute value we subtract it from (instead of adding it to) the accumulated sum. We use the sign bit as an add/subtract signal (reminder: this signal is fed into the input carry, as well as the XOR that take 1s complement "on-the-fly"). The result is divided by 128, by removing the 7 least significant bits.



Clock	0	1	2	3	4	5	6	...
state	S_IDLE	S_READ_A	S_READ_B	S_READ_A	S_READ_B	S_READ_A	S_READ_B	
Addr		0	128	1	129	2	130	
A				A[0]		A[1]		
B				B[0]		B[1]		
D				A[0]-B[0]=D[0]		A[1]-B[1]=D[1]		
O					|D[0]|+0=O[0]		|D[1]|+O[0]=O[1]	

The data path and the state table for the dual-port RAM implementation are given below.



Clock	0	1	2	3	4	5	...
state	S_IDLE	S_READ	S_READ	S_READ	S_READ	S_READ	
A_Addr		0	1	2	3	4	
B_Addr		128	129	130	131	132	
A			A[0]	A[1]	A[2]	A[3]	
B			B[0]	B[1]	B[2]	B[3]	
D			A[0]-B[0]=D[0]	A[1]-B[1]=D[1]	A[2]-B[2]=D[2]	A[3]-B[3]=D[3]	
O				|D[0]|+0=O[0]	|D[1]|+O[0]=O[1]	|D[2]|+O[1]=O[2]	

Problem 4: Note, the memory is byte addressable. The short int is represented on two bytes, the float is represented on 4 bytes, the my_struct is represented on 12 bytes and the my_union is represented on 8 bytes. The discussion provided below is for a 32-bit machine (for a 64-bit machine, or a 32-bit machine where doubles are 4-byte aligned, `my_struct_pointer` will be represented on 16 bytes). If void_pointer points to *address*, then the following will occur:

```
short_int_pointer += 5;  //  will change the short_int_pointer to address+5*2
float_pointer += 4;      // will change the float_pointer to address+4*4
my_struct_pointer++;     // will change the my_struct_pointer to address+12

// the my_union_pointer will be assigned my_struct_pointer-8=address+4
my_union_pointer = (union my_union *)my_struct_pointer - 1;
```

Based on the above, the program execution will return the following messages:

A short message for this problem
ssage for this problem
for this problem
age for this problem
ort message for this problem

Problem 5: See the enclosed code.

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_WORD_SIZE  10

char *extract_reversed_word(unsigned int word_index)
{
     char *message = "A short    message for this problem";

     char *return_pointer;
     static char string_buffer[MAX_WORD_SIZE];
     return_pointer = &string_buffer[sizeof(string_buffer)-1];
     *return_pointer = '\0';

     unsigned int word_position = 0, i = 0;
     while (word_position < word_index)
     {
          while (message[i++] != ' ');
          if (message[i] != ' ')
               word_position++;
     }
     while (message[i] != ' ' && message[i] != '\0')
          *--return_pointer = message[i++];
     return return_pointer;
}

int main(void)
{
     unsigned int i=0;
     for (; i<6; i++)
          printf("%s\n", extract_reversed_word(i));
     return 0;
}
```

Problem 6: See the enclosed code.

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_VALUE 10

struct vector {
     short int size;
     short int *vec;
};

struct vector2 {
     short int size;
     struct vector *vec;
};

struct vector3 {
     short int size;
     struct vector2 *vec;
};

int main(void) {

     int i, j, k;
     srandom(0);
     struct vector3 v3;
     v3.size = (short int) (random() % MAX_VALUE) + 1;
     v3.vec = (struct vector2 *)malloc(sizeof(struct vector2) * v3.size);

     for (i=0; i<v3.size; i++)
     {
          v3.vec[i].size = (short int) (random() % MAX_VALUE) + 1;
          v3.vec[i].vec = (struct vector *)malloc(sizeof(struct vector) * v3.vec[i].size);
          for (j=0; j<v3.vec[i].size; j++)
          {
               v3.vec[i].vec[j].size = (short int) (random() % MAX_VALUE) + 1;
               v3.vec[i].vec[j].vec = (short int *)malloc(sizeof(short int) * v3.vec[i].vec[j].size);
               for (k=0; k<v3.vec[i].vec[j].size; k++)
                    v3.vec[i].vec[j].vec[k] = (short int) (random() % MAX_VALUE) + 1;
          }
     }

     int sum = 0;
     int count = 0;
     for (i=0; i<v3.size; i++)
          for (j=0; j<v3.vec[i].size; j++)
          {
               for (k=0; k<v3.vec[i].vec[j].size; k++)
                    sum += v3.vec[i].vec[j].vec[k];
               count += v3.vec[i].vec[j].size;
          }

     printf("Average = %f\n", (float)sum/(float)count);

     for (i=0; i<v3.size; i++)
     {
          for (j=0; j<v3.vec[i].size; j++)
               free(v3.vec[i].vec[j].vec);
          free(v3.vec[i].vec);
     }

     free(v3.vec);

     return 0;
}
```

Problem 7: See the enclosed code.

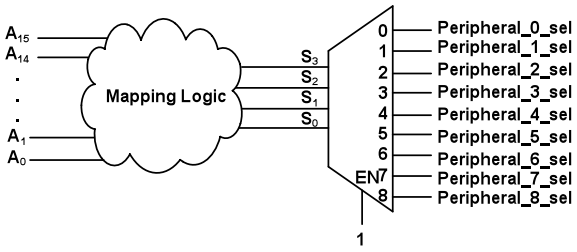
```c
#include <stdio.h>
#include <stdlib.h>

#define SIZE 10

struct my_struct {
     int i;
     float f;
};

union my_union {
     struct my_struct s;
     double d;
};

int compare_struct(const void *left, const void *right)
{
     struct my_struct operand1, operand2;
     operand1 = ((union my_union *)left)->s;
     operand2 = ((union my_union *)right)->s;

     if (operand1.i < operand2.i)
          return -1;
     else if (operand1.f > operand2.f)
          return 1;
     return 0;
}

int compare_double(const void *left, const void *right)
{
     if ((*(union my_union *)left).d == (*(union my_union *)right).d)
          return 0;
     else if ((*(union my_union *)left).d < (*(union my_union *)right).d)
          return -1;
     return 1;
}

void find_min_max (union my_union my_union_array[], int (*my_union_compare)(),
                        union my_union *my_union_min, union my_union *my_union_max)
{
     *my_union_max = *my_union_min = my_union_array[0];
     int i;
     for (i=1; i<SIZE; i++)
     {
          if (my_union_compare(my_union_max, &my_union_array[i]) < 0)
               *my_union_max = my_union_array[i];
          if (my_union_compare(my_union_min, &my_union_array[i]) > 0)
               *my_union_min = my_union_array[i];
     }
}
```

```
int main(void) {

     int (*my_union_compare)(const void *, const void *);
     union my_union my_union_array[SIZE];
     union my_union my_union_max, my_union_min;
     srandom(0);

     int i;
     for (i=0; i<SIZE; i++)
     {
          my_union_array[i].s.i = (int)(random() % SIZE);
          my_union_array[i].s.f =  (float)(random() % SIZE);
     }

     my_union_compare = &compare_struct;
     find_min_max(my_union_array, my_union_compare, &my_union_min, &my_union_max);
     printf("min value is %d %f ", my_union_min.s.i, my_union_min.s.f);
     printf("max value is %d %f\n", my_union_max.s.i, my_union_max.s.f);

     for (i=0; i<SIZE; i++)
          my_union_array[i].d = (double)(random() % SIZE);
     my_union_compare = &compare_double;
     find_min_max(my_union_array, my_union_compare, &my_union_min, &my_union_max);
     printf("min value is %f ", my_union_min.d);
     printf("max value is %f\n", my_union_max.d);

     return 0;
}
```

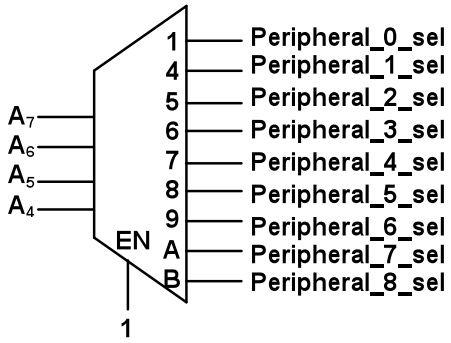
Problem 8:

At a first glance, it would be required to build the mapping logic between the address and the select signals used for the decoder and multiplexers. However, on closer inspection of the memory map it can be seen that every single peripheral can be uniquely identified *ONLY* by the address lines A_7 to A_4. Hence A_7 can be mapped directly to S_3, ..., and A_4 mapped directly to S_0. Note, the decoding/muxing logic also changes as shown at the bottom of the page.

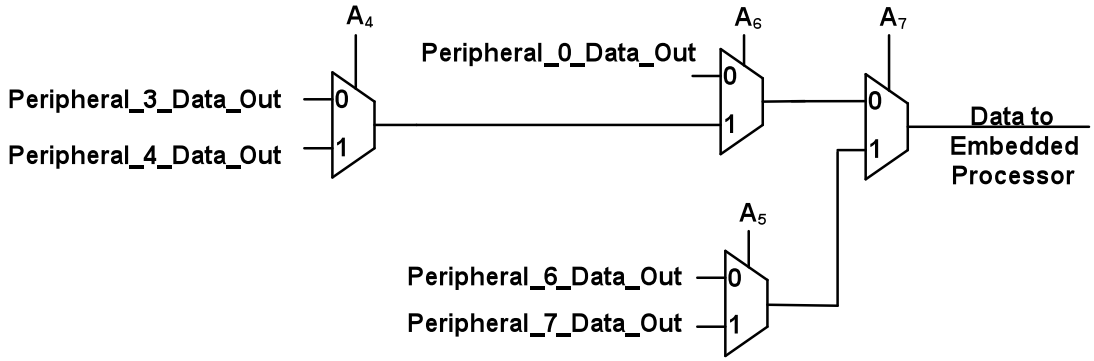
Address bus																Selection signals			
A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	S_3	S_2	S_1	S_0
0	0	1	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0
							.....				.						...		
0	0	1	0	0	0	1	1	0	0	0	1	1	1	1	1	0	0	0	0
							.....				.								
0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1
							.....				.						...		
0	0	0	1	0	0	1	0	0	1	0	0	1	1	1	1	0	0	0	1
0	0	0	1	0	0	1	0	0	1	0	1	0	0	0	0	0	0	1	0
							.....				.						...		
0	0	0	1	0	0	1	0	0	1	0	1	1	1	1	1	0	0	1	0
0	0	0	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0	1	1
							.....				.						...		
0	0	0	1	0	0	1	0	0	1	1	0	1	1	1	1	0	0	1	1
0	0	0	1	0	0	1	0	0	1	1	1	0	0	0	0	0	1	0	0
							.....				.						...		
0	0	0	1	0	0	1	0	0	1	1	1	1	1	1	1	0	1	0	0
0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1
							.....				.						...		
0	0	0	1	0	0	1	0	1	0	0	0	1	1	1	1	0	1	0	1
0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	1	1	0
							.....				.						...		
0	0	0	1	0	0	1	0	1	0	0	1	1	1	1	1	0	1	1	0
0	0	0	1	0	0	1	0	1	0	1	0	0	0	0	0	0	1	1	1
							.....				.						...		
0	0	0	1	0	0	1	0	1	0	1	0	1	1	1	1	0	1	1	1
0	0	0	1	0	0	1	0	1	0	1	1	0	0	0	0	1	0	0	0
							.....				.						...		
0	0	0	1	0	0	1	0	1	0	1	1	1	1	1	1	1	0	0	0



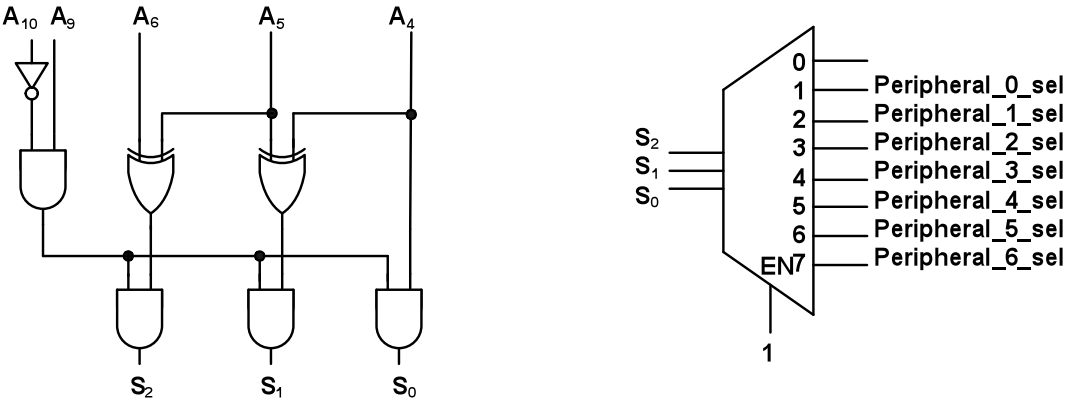
For the decoding logic below, the outputs of the 4-to-16 decoder that are not used will be optimized away (i.e., the logic will not be implemented).



For the multiplexer network we also exploit the fact that peripherals, 1, 2, 5 and 8 are output-only in which case they will not necessary to drive the data to the embedded processor.



Problem 9: There are only 7 peripherals with the select signals generated, as shown by the decoding logic below. As it can be noted, the output zero of the decoder is never used. Hence, the only values for which a select signal is generated is when at least one of the $S_2\,S_1\,S_0$ signals is a logic 1. For this to be satisfied, the logic gates at the bottom of the mapping logic must be enabled. Hence, $A_{10}A_9$ must be 01. Because bits $A_6A_5A_4$ are the only ones that are used in the mapping logic, they are the only ones that also need to be reversed engineered (note, as part of the problem statement we know that address lines that are not used in the simplified implementation of the mapping logic are assumed to be 0 when computing the memory map).



Based on the above mapping logic, we can see that when the AND gates are enabled, i.e., $A_{10}A_9$ =01, A_4=S_0. Because S_1 = A_5 XOR A_4, we can rewrite A_5 = S_1 XOR A_4 = S_1 XOR S_0. Similarly because S_2 = A_6 XOR A_5, we can rewrite A_6 = S_2 XOR A_5 = S_2 XOR S_1 XOR S_0. Hence, for peripheral 0 to be selected, because $S_2S_1S_0$ = 001, we must have $A_6A_5A_4$ = 111 ...

Following the same line of reasoning for any combination of $S_2S_1S_0$ (for which a select signal is generated), we will have the following base addresses on 16 bits: peripheral 0 – 0x0270, peripheral 1 – 0x0260, peripheral 2 – 0x0210, peripheral 3 – 0x0240, peripheral 4 – 0x0230, peripheral 5 – 0x0220, peripheral 6 – 0x0250.

Problem 10: There are several ways to tackle to this problem. However, what they all have in common is the need to design a custom interrupt controller which prioritizes the requests from 3 peripherals and passes the interrupt to the embedded processor. This interrupt controller can be viewed as a separate peripheral, whose inputs are the interrupt requests from the 3 peripherals that cannot connect their interrupt request signal directly to the embedded processor. For example, if the embedded processor has 4 interrupt inputs, the interrupt requests from peripherals 0, 1, and 2 can be connected directly to the interrupt inputs of the embedded processor. However, interrupt requests from peripherals 3, 4 and 5 will be connected to the data inputs of the custom interrupt controller and the interrupt request signal from the interrupt controller will be connected to interrupt input 3 of the embedded processor.

The difference between different implementations is in who is in charge of clearing the interrupt flags in each of the peripherals, as well as in the interrupt controller. One option is to run a single IOWR in the interrupt service routine for the interrupt controller, which based on the value which was reset it will decide in hardware which of its input peripherals will need the edge detect register cleared. The second option is to clear not only the flag in the edge register in the interrupt controller through an IOWR, but also use an additional IOWR that clears the peripheral that is connected to the interrupt controller. This option will require less hardware, however, the interrupt service routine will require more software and the time spent on clearing the interrupt flags will eventually increase.

Problem 11: See the enclosed code.

```
#include "io.h"
#include "system.h"
#include "stddef.h"
#include "alt_types.h"
#include "sys/alt_stdio.h"
#include "sys/alt_irq.h"
#include "priv/alt_busy_sleep.h"

int main()
{
    alt_u8 i, count;
    alt_u16 sw_val, led_value = 0;

    alt_putstr("Problem 11!\n");

    while (1) {
        // count switches SWITCH_GRPA
        sw_val = IORD(SWITCH_GRPA_I_BASE, 0);
        count = 0;
        for (i = 0; i < 9; i++)
            count += ((sw_val >> i) & 0x1);
        if (count > 4) led_value |= 0x1;    // turn on
        else led_value &= 0x1FE;            // turn off

        // count switches SWITCH_GRPB
        sw_val = IORD(SWITCH_GRPB_I_BASE, 0);
        count = 0;
        for (i = 0; i < 9; i++)
            count += ((sw_val >> i) & 0x1);
        if (count > 4) led_value |= 0x100;  // turn on
        else led_value &= 0x0FF;            // turn off

        IOWR(LED_GREEN_O_BASE, 0, led_value);
    }

    return 0;
}
```

Problem 12: See the enclosed code.

```
void led_count(
    alt_u16 sw_val, alt_u16 mask,
    alt_u16 *led_value
) {
     // this implements the counting function
    alt_u8 i, count;

    count = 0;
    for (i = 0; i < 9; i++)
        count += ((sw_val >> i) & 0x1);
    if (count > 4) *led_value |= mask;
    else *led_value &= (mask ^ 0x1FF);

    IOWR(LED_GREEN_O_BASE, 0, *led_value);
}

void SW_GRPA_interrupt(alt_u16 *led_value) {
    alt_u16 sw_val = IORD(SWITCH_GRPA_I_BASE, 0);
    led_count(sw_val, 0x1, led_value);
    IOWR(SWITCH_GRPA_I_BASE, 3, 0x0);
}

void SW_GRPB_interrupt(alt_u16 *led_value) {
    alt_u16 sw_val = IORD(SWITCH_GRPB_I_BASE, 0);
    led_count(sw_val, 0x100, led_value);
    IOWR(SWITCH_GRPB_I_BASE, 3, 0x0);
}

int main()
{
    volatile alt_u16 led_value = 0;

    alt_putstr("Problem 12!\n");

    IOWR(SWITCH_GRPA_I_BASE, 3, 0x0); // Edge capture register
    IOWR(SWITCH_GRPA_I_BASE, 2, 0x1FF); // IRQ mask
    alt_irq_register(SWITCH_GRPA_I_IRQ, (void *)(&led_value), (void*)SW_GRPA_interrupt );

    IOWR(SWITCH_GRPB_I_BASE, 3, 0x0); // Edge capture register
    IOWR(SWITCH_GRPB_I_BASE, 2, 0x1FF); // IRQ mask
    alt_irq_register(SWITCH_GRPB_I_IRQ, (void *)(&led_value), (void*)SW_GRPB_interrupt );

    SW_GRPA_interrupt(&led_value); // initialize for SW_GRPA
    SW_GRPB_interrupt(&led_value); // initialize for SW_GRPB

    while (1);

    return 0;
}
```

Problem 13: See the enclosed code.

```
#include "io.h"
#include "system.h"
#include "stddef.h"
#include "alt_types.h"
#include "sys/alt_stdio.h"
#include "sys/alt_irq.h"
#include "priv/alt_busy_sleep.h"

int main()
{
    alt_u8 counter = 0, i, j;
    alt_u16 sw_val, mask = 0;

    alt_putstr("Problem 13!\n");

    while (1) {
        sw_val = IORD(SWITCH_GRPA_I_BASE, 3);
        if (sw_val != 0) {
            j = 0;
            for (i = 0; i < 9; i++)
                j += ((sw_val >> i) & 0x1);
            if (j > 1) {
               // more than 1 switch changed - reset
                counter = 0;
                mask = 0;
            } else {
               // one switch changed
                if (sw_val == mask) counter++;
                else {
                    // new switch changed, reset counter
                    counter = 1;
                    mask = sw_val;
                }
            }

            if (counter == 3) {
                i = 0;
                while ((mask >> i) != 1) i++;
                alt_printf("Input %x switched three times in a row\n", i);
                mask = 0;
                counter = 0;
            }
            alt_busy_sleep(10000);
            IOWR(SWITCH_GRPA_I_BASE, 3, 0x0);
        }
    }

    return 0;
}
```

Problem 14: See the enclosed code.

```
#include "io.h"
#include "system.h"
#include "stddef.h"
#include "alt_types.h"
#include "sys/alt_stdio.h"
#include "sys/alt_irq.h"
#include "priv/alt_busy_sleep.h"

void SW_GRPA_interrupt(void) {
    static alt_u8 counter = 0;
    static alt_u16 mask = 0;
    alt_u8 i, j;
    alt_u16 sw_val;

    sw_val = IORD(SWITCH_GRPA_I_BASE, 3);
    if (sw_val != 0) {
        j = 0;
        for (i = 0; i < 9; i++)
            j += ((sw_val >> i) & 0x1);
        if (j > 1) {
          // more than 1 switch changed - reset
            counter = 0;
            mask = 0;
        } else {
          // one switch changed
            if (sw_val == mask) counter++;
            else {
                // new switch changed, reset counter
                counter = 1;
                mask = sw_val;
            }
        }

        if (counter == 3) {
            i = 0;
            while ((mask >> i) != 1) i++;
            alt_printf("Input %x switched three times in a row\n", i);
            mask = 0;
            counter = 0;
        }
        alt_busy_sleep(10000);
        IOWR(SWITCH_GRPA_I_BASE, 3, 0x0);
    }
}

int main()
{
    alt_putstr("Problem 14!\n");

    IOWR(SWITCH_GRPA_I_BASE, 3, 0x0); // Edge capture register
    IOWR(SWITCH_GRPA_I_BASE, 2, 0x1FF); // IRQ mask
    alt_irq_register(SWITCH_GRPA_I_IRQ, NULL, (void*)SW_GRPA_interrupt );

    while (1);

    return 0;
}
```

Problem 15: See the enclosed code.

```c
#include "io.h"
#include "system.h"
#include "stddef.h"
#include "alt_types.h"
#include "sys/alt_stdio.h"
#include "sys/alt_irq.h"
#include "priv/alt_busy_sleep.h"

int main()
{
    alt_u8 i;
    alt_u16 xor;

    alt_putstr("Problem 15!\n");

    IOWR(LED_GREEN_O_BASE, 0, 0x0);
    while (1) {
        if (IORD(SWITCH_GRPA_I_BASE, 0) & 0x1) {
            // generate XOR
            xor = IORD(SWITCH_GRPB_I_BASE, 0);
            for (i = 1; i < 9; i++)
                xor ^= ( xor >> i ) & 0x1;
            xor &= 0x1;
            IOWR(LED_GREEN_O_BASE, 0, xor);
            xor = 0;
        } else {
            // generate pulses
            xor ^= 0x1FF;
            IOWR(LED_GREEN_O_BASE, 0, xor);
            alt_busy_sleep(
                1000 * IORD(SWITCH_GRPB_I_BASE, 0) );
        }
    }

    return 0;
}
```

Problem 16: See the enclosed code.

```
#include "io.h"
#include "system.h"
#include "stddef.h"
#include "alt_types.h"
#include "sys/alt_stdio.h"
#include "sys/alt_irq.h"
#include "priv/alt_busy_sleep.h"

void SW_GRPA_interrupt(alt_u8 *mode) {
     // decide mode based on SWITCH_GRPA
    *mode = (IORD(SWITCH_GRPA_I_BASE, 0) & 0x1);

    if (*mode) {
     // XOR mode, turn interrupts on
        IOWR(SWITCH_GRPB_I_BASE, 2, 0x1FF);
        SW_GRPB_interrupt();
    } else {
     // pulse mode, turn interrupts off
        IOWR(SWITCH_GRPB_I_BASE, 2, 0x0);
    }
     // clear edge detect register
    IOWR(SWITCH_GRPA_I_BASE, 3, 0x0);
}

void SW_GRPB_interrupt(void) {
    alt_u8 i;
    alt_u16 xor;
    // perform XOR
    xor = IORD(SWITCH_GRPB_I_BASE, 0);
    for (i = 1; i < 9; i++)
        xor ^= ( xor >> i ) & 0x1;
    xor &= 0x1;
    IOWR(LED_GREEN_O_BASE, 0, xor);
     // clear edge detect register
    IOWR(SWITCH_GRPB_I_BASE, 3, 0x0);
}

int main()
{
    volatile alt_u8 mode;
    alt_u16 value = 0;

    alt_putstr("Problem 16!\n");

    IOWR(SWITCH_GRPA_I_BASE, 3, 0x0); // Edge capture register
    IOWR(SWITCH_GRPA_I_BASE, 2, 0x1); // IRQ mask
    alt_irq_register(SWITCH_GRPA_I_IRQ, (void *)(&mode), (void*)SW_GRPA_interrupt );

    IOWR(SWITCH_GRPB_I_BASE, 3, 0x0); // Edge capture register
    IOWR(SWITCH_GRPB_I_BASE, 2, 0x0); // IRQ mask
    alt_irq_register(SWITCH_GRPB_I_IRQ, NULL, (void*)SW_GRPB_interrupt );

    SW_GRPA_interrupt(&mode);
    while (1) {
        if (mode == 0) {
          // generate pulses
            value ^= 0x1FF;
            IOWR(LED_GREEN_O_BASE, 0, value);
            alt_busy_sleep(
                1000 * IORD(SWITCH_GRPB_I_BASE, 0) );
        } else value = 0;
    }
    return 0;
}
```

Problem 17: See the enclosed code.

```
#include "io.h"
#include "system.h"
#include "stddef.h"
#include "alt_types.h"
#include "sys/alt_stdio.h"
#include "sys/alt_irq.h"
#include "priv/alt_busy_sleep.h"

#define DISARMED 0
#define ARMED 1
#define ALARM_Son 2
#define ALARM_Soff 3

int main()
{
    alt_u8 state;

    alt_putstr("Problem 17!\n");

    state = DISARMED;
    IOWR(LED_GREEN_O_BASE, 0, 0x0);

    while (1) {
     // case statement for state machine
        switch (state) {
            case DISARMED : // waiting for e = 1
                if (IORD(SWITCH_GRPA_I_BASE, 0) & 0x1)
                    state = ARMED;
                break;
            case ARMED : // waiting for e = 0 or w/d = 1
                if ((IORD(SWITCH_GRPA_I_BASE, 0) & 0x1) == 0)
                    state = DISARMED;
                else if ((IORD(SWITCH_GRPB_I_BASE, 0) & 0x3) != 0)
                    state = ALARM_Son;
                break;
            case ALARM_Son : // generate on pulse of alarm
                IOWR(LED_GREEN_O_BASE, 0, 0x1);
                alt_busy_sleep(900*1000);
                if ((IORD(SWITCH_GRPA_I_BASE, 0) & 0x1) == 0) {
                    IOWR(LED_GREEN_O_BASE, 0, 0x0);
                    state = DISARMED;
                }
                else state = ALARM_Soff;
                break;
            case ALARM_Soff : // generate off pulse of alarm
                IOWR(LED_GREEN_O_BASE, 0, 0x0);
                alt_busy_sleep(600*1000);
                if ((IORD(SWITCH_GRPA_I_BASE, 0) & 0x1) == 0)
                    state = DISARMED;
                else state = ALARM_Son;
                break;
        }
    }

    return 0;
}
```

Problem 18: See the enclosed code.

```
#include "io.h"
#include "system.h"
#include "stddef.h"
#include "alt_types.h"
#include "sys/alt_stdio.h"
#include "sys/alt_irq.h"
#include "priv/alt_busy_sleep.h"

#define DISARMED 0
#define ARMED 1
#define ALARM_Son 2
#define ALARM_Soff 3

void SW_GRPA_interrupt(alt_u8 *state) {
    if ((IORD(SWITCH_GRPA_I_BASE, 0) & 0x1) == 0)
        *state = DISARMED; // disarm if e = 0
    else if (
        (*state == DISARMED) &
        (IORD(SWITCH_GRPA_I_BASE, 0) & 0x1)
    ) *state = ARMED; // arm if disarmed and e = 1
    IOWR(SWITCH_GRPA_I_BASE, 3, 0x0);
}

void SW_GRPB_interrupt(alt_u8 *state) {
    if (
        (*state == ARMED) &
        ((IORD(SWITCH_GRPB_I_BASE, 0) & 0x3) != 0)
    ) *state = ALARM_Son; // sound alarm if w|d when armed
    IOWR(SWITCH_GRPB_I_BASE, 3, 0x0);
}

int main()
{
    volatile alt_u8 state = DISARMED;

    alt_putstr("Problem 18!\n");

    IOWR(SWITCH_GRPA_I_BASE, 3, 0x0); // Edge capture register
    IOWR(SWITCH_GRPA_I_BASE, 2, 0x1); // IRQ mask
    alt_irq_register(SWITCH_GRPA_I_IRQ, (void *)(&state), (void*)SW_GRPA_interrupt );

    IOWR(SWITCH_GRPB_I_BASE, 3, 0x0); // Edge capture register
    IOWR(SWITCH_GRPB_I_BASE, 2, 0x3); // IRQ mask
    alt_irq_register(SWITCH_GRPB_I_IRQ, (void *)(&state), (void*)SW_GRPB_interrupt );

    IOWR(LED_GREEN_O_BASE, 0, 0x0);
    while (1) {
        if (state == ALARM_Son) {
          // generate on pulse of alarm
            state = ALARM_Soff;
            IOWR(LED_GREEN_O_BASE, 0, 0x1);
            alt_busy_sleep(900*1000);
        } else if (state == ALARM_Soff) {
          // generate off pulse of alarm
            state = ALARM_Son;
            IOWR(LED_GREEN_O_BASE, 0, 0x0);
            alt_busy_sleep(600*1000);
        } else IOWR(LED_GREEN_O_BASE, 0, 0x0);
    }

    return 0;
}
```

Problem 19: See the enclosed code.

```
#include "io.h"
#include "system.h"
#include "stddef.h"
#include "alt_types.h"
#include "sys/alt_stdio.h"
#include "sys/alt_irq.h"
#include "priv/alt_busy_sleep.h"

#define NUM_SAMPLES 30

int main()
{
    alt_u8 count = 0, flag = 0;
    alt_u16 sw_val;

    alt_putstr("Problem 19!\n");
    IOWR(LED_GREEN_O_BASE, 0, 0x0);
    sw_val = IORD(SWITCH_GRPB_I_BASE, 0);

    while (1) {
        if ((IORD(SWITCH_GRPB_I_BASE, 0) ^ sw_val) & 0x1) {
               // change on SWITCH_GRPB lsb
            sw_val = IORD(SWITCH_GRPA_I_BASE, 0);
//            alt_printf("count = %x, flag = %x, sw_val = %x\n",
//                count, flag, sw_val);

            // check temperature based on state
            if (flag == 0) {
                if (sw_val < 18) count++;
                else count = 0;
            } else {
                if (sw_val > 23) count++;
                else count = 0;
            }

            // check if consecutive samples reached
            if (count == NUM_SAMPLES) {
                flag ^= 0x1;
                IOWR(LED_GREEN_O_BASE, 0, flag);
                count = 0;
            }

            // update value of SWITCH_GRPB
            sw_val = IORD(SWITCH_GRPB_I_BASE, 0);
//            alt_busy_sleep(10000);
        }
    }

    return 0;
}
```

Problem 20: See the enclosed code.

```
#include "io.h"
#include "system.h"
#include "stddef.h"
#include "alt_types.h"
#include "sys/alt_stdio.h"
#include "sys/alt_irq.h"
#include "priv/alt_busy_sleep.h"

#define NUM_SAMPLES 30

void SW_GRPB_interrupt(void) {
    static alt_u8 count = 0, flag = 0;
    alt_u16 sw_val = 0;

    sw_val = IORD(SWITCH_GRPA_I_BASE, 0);
//    alt_printf("count = %x, flag = %x, sw_val = %x\n",
//        count, flag, sw_val);

    // check temperature based on state
    if (flag == 0) {
        if (sw_val < 18) count++;
        else count = 0;
    } else {
        if (sw_val > 23) count++;
        else count = 0;
    }

    // check if consecutive samples reached
    if (count == NUM_SAMPLES) {
        flag ^= 0x1;
        IOWR(LED_GREEN_O_BASE, 0, flag);
        count = 0;
    }

//    alt_busy_sleep(10000);
     // clear edge detect register
    IOWR(SWITCH_GRPB_I_BASE, 3, 0x0);
}

int main()
{
    IOWR(SWITCH_GRPB_I_BASE, 3, 0x0); // Edge capture register
    IOWR(SWITCH_GRPB_I_BASE, 2, 0x1); // IRQ mask
    alt_irq_register(SWITCH_GRPB_I_IRQ, NULL, (void*)SW_GRPB_interrupt );

    alt_putstr("Problem 20!\n");
    IOWR(LED_GREEN_O_BASE, 0, 0x0);

    while (1);

    return 0;
}
```

Problem 21: See the enclosed charts.

(a)

Task	Priority	e	p
A	1	2	10
B	2	3	11
C	3	4	12

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU	A	A	B	B	B	C	C	C	C	X	A	A	B	B	B	C	C	C	C	X
Arrived	A, B, C										A	B	C							
Preempted (OS ticks left)																				

(b)

Task	Priority	e	p
A	1	2	10
B	2	3	11
C	3	4	12

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU1	A	A	C	C	C	C	X	X	X	X	A	A	C	C	C	C	X	X	X	X
CPU2	B	B	B	X	X	X	X	X	X	X	X	B	B	B	X	X	X	X	X	X
Arrived	A, B, C										A	B	C							
Preempted (OS ticks left)																				

(c)

Task	Priority	e	p
A	1	2	10
B	2	3	11
C	3	4	12
D	4	5	11
E	5	5	12

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU1	A	A	C	C	C	C	E	E	E	E	E	B	B	B	D	D	D	D	D	X
CPU2	B	B	B	D	D	D	D	D	X	X	A	A	C	C	C	C	E	E	E	E
Arrived	A, B, C, D, E										A	B, D	C, E							
Preempted (OS ticks left)																				

Problem 22: See the enclosed charts.

(a)

Task	Priority	e	p
A	1	2	10
B	2	3	11
C	3	4	12

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU	A	A	B	B	B	C	C	C	C	X	A	A	B	B	B	C	C	C	C	X
Arrived	A, B, C										A	B	C							

(b)

Task	Priority	e	p
A	1	2	10
B	2	3	11
C	3	4	12

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU1	A	A	C	C	C	C	X	X	X	X	A	A	C	C	C	C	X	X	X	X
CPU2	B	B	B	X	X	X	X	X	X	X	X	B	B	B	X	X	X	X	X	X
Arrived	A, B, C										A	B	C							

(c)

Task	Priority	e	p
A	1	2	10
B	2	3	11
C	3	4	12
D	4	5	11
E	5	5	12

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU1	A	A	C	C	C	C	E	E	E	E	E	B	B	B	D	D	D	D	D	X
CPU2	B	B	B	D	D	D	D	D	X	X	A	A	C	C	C	C	E	E	E	E
Arrived	A, B, C, D, E										A	B, D	C, E							

Problem 23: See the enclosed charts.

(a)

Task	Priority	e	p
A	5	3	9
B	4	2	10
C	3	2	11

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU	C	C	B	B	A	A	A	X	X	A	B	C	C	B	A	A	X	X	A	A
Arrived	A, B, C									A	B	C							A	
Preempted (OS ticks left)											A (2)	B (1)								

(b)

Task	Priority	e	p
A	5	3	9
B	4	2	10
C	3	2	11

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU1	C	C	A	A	A	X	X	X	X	A	A	C	C	X	X	X	X	X	X	X
CPU2	B	B	X	X	X	X	X	X	X	X	B	B	A	X	X	X	X	X	A	A
Arrived	A, B, C									A	B	C							A	
Preempted (OS ticks left)												A (1)								

(c)

Task	Priority	e	p
A	5	3	9
B	4	2	10
C	3	2	11
D	2	4	9
E	1	5	11

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU1	E	E	E	E	E	B	B	X	X	A	B	E	E	E	E	E	A	A	A	A
CPU2	D	D	D	D	C	C	A	A	A	D	D	D	D	C	C	B	X	X	D	D
Arrived	A, B, C, D, E									A, D	B	C, E							A, D	
Preempted (OS ticks left)											A (2)	B (1)								

Problem 24: See the enclosed charts.

(a)

Task	Priority	e	p
A	5	3	9
B	4	2	10
C	3	2	11

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU	C	C	B	B	A	A	A	X	X	A	A	A	C	C	B	B	X	X	A	A
Arrived	A, B, C									A	B	C							A	

(b)

Task	Priority	e	p
A	5	3	9
B	4	2	10
C	3	2	11

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU1	C	C	A	A	A	X	X	X	X	A	A	A	C	C	X	X	X	X	A	A
CPU2	B	B	X	X	X	X	X	X	X	X	B	B	X	X	X	X	X	X	X	X
Arrived	A, B, C									A	B	C							A	

(c)

Task	Priority	e	p
A	5	3	9
B	4	2	10
C	3	2	11
D	2	4	9
E	1	5	11

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU1	E	E	E	E	E	B	B	X	X	A	A	A	E	E	E	E	E	X	A	A
CPU2	D	D	D	D	C	C	A	A	A	D	D	D	D	C	C	B	B	X	D	D
Arrived	A, B, C, D, E									A, D	B	C, E							A, D	

Problem 25: See the enclosed charts.

(a)

Task	Priority	e	i
A	1	2	6
B	2	3	6
C	3	4	6

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU	A	A	B	B	B	C	C	C	A	A	C	B	B	B	X	X	A	A	C	C
Arrived	A, B, C								A			B					A	C		
Preempted (OS ticks left)									C (1)											

(b)

Task	Priority	e	i
A	1	2	6
B	2	3	6
C	3	4	6

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU1	A	A	C	C	C	C	X	X	A	A	X	X	C	C	C	C	A	A	X	X
CPU2	B	B	B	X	X	X	X	X	X	B	B	B	X	X	X	X	X	X	B	B
Arrived	A, B, C								A	B			C				A		B	
Preempted (OS ticks left)																				

(c)

Task	Priority	e	i
A	1	2	6
B	2	3	6
C	3	4	6
D	4	5	6
E	5	6	6

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU1	A	A	C	C	C	C	E	E	E	B	B	B	C	C	C	C	A	A	B	B
CPU2	B	B	B	D	D	D	D	D	A	A	E	E	E	X	D	D	D	D	D	E
Arrived	A, B, C, D, E								A	B			C		D		A		B	E
Preempted (OS ticks left)										E (3)										

Problem 26: See the enclosed charts.

(a)

Task	Priority	e	i
A	1	2	6
B	2	3	6
C	3	4	6

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU	A	A	B	B	B	C	C	C	C	A	A	B	B	B	X	C	C	C	C	A
Arrived	A, B, C								A			B				C		A		

(b)

Task	Priority	e	i
A	1	2	6
B	2	3	6
C	3	4	6

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU1	A	A	C	C	C	C	X	X	A	A	X	X	C	C	C	C	A	A	X	X
CPU2	B	B	B	X	X	X	X	X	X	B	B	B	X	X	X	X	X	X	B	B
Arrived	A, B, C								A	B			C				A		B	

(c)

Task	Priority	e	i
A	1	2	6
B	2	3	6
C	3	4	6
D	4	5	6
E	5	6	6

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU1	A	A	C	C	C	C	E	E	E	E	E	E	C	C	C	C	A	A	E	E
CPU2	B	B	B	D	D	D	D	D	A	A	B	B	B	X	D	D	D	D	D	B
Arrived	A, B, C, D, E								A	B			C		D		A		E	B

Problem 27: See the enclosed charts.

(a)

Task	Priority	e	i
A	5	3	6
B	4	2	8
C	3	2	9

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
CPU	C	C	B	B	A	A	A	X	X	X	X	C	C	B	B	A	A	A	X	X	X
Arrived	A, B, C											C	B	A							
Preempted (OS ticks left)																					

(b)

Task	Priority	e	i
A	5	3	6
B	4	2	8
C	3	2	9

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU1	C	C	A	A	A	X	X	X	X	X	X	C	C	X	X	X	X	X	X	X
CPU2	B	B	X	X	X	X	X	X	X	X	B	B	A	A	A	X	X	X	X	X
Arrived	A, B, C										B	A, C								
Preempted (OS ticks left)																				

(c)

Task	Priority	e	i
A	5	3	6
B	4	2	8
C	3	2	9
D	2	4	5
E	1	5	6

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU1	E	E	E	E	E	B	B	X	X	X	X	E	E	E	E	E	B	B	D	D
CPU2	D	D	D	D	C	C	A	A	A	D	D	D	D	X	X	C	C	A	A	A
Arrived	A, B, C, D, E									D		E				A, B, C			D	
Preempted (OS ticks left)																				

Problem 28: See the enclosed charts.

(a)

Task	Priority	e	i
A	5	3	6
B	4	2	8
C	3	2	9

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU	C	C	B	B	A	A	A	X	X	X	X	C	C	B	B	A	A	A	X	X
Arrived	A, B, C											C	B	A						

(b)

Task	Priority	e	i
A	5	3	6
B	4	2	8
C	3	2	9

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU1	C	C	A	A	A	X	X	X	X	X	X	C	C	X	X	X	X	X	X	X
CPU2	B	B	X	X	X	X	X	X	X	X	B	B	A	A	A	X	X	X	X	X
Arrived	A, B, C										B	A, C								

(c)

Task	Priority	e	i
A	5	3	6
B	4	2	8
C	3	2	9
D	2	4	5
E	1	5	6

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU1	E	E	E	E	E	B	B	X	X	X	X	E	E	E	E	E	B	B	D	D
CPU2	D	D	D	D	C	C	A	A	A	D	D	D	D	X	X	C	C	A	A	A
Arrived	A, B, C, D, E									D		E				A, B, C			D	

Problem 29: See the enclosed charts.

(a)

Task	Priority	e1	i1	e2	i2
A	1	1	4	1	2
B	2	1	3	1	3
C	3	2	2	2	4

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU	A1	B1	C1	C1	X	A2	B2	C2	A1	C2	B1	X	X	A2	B2	C1	A1	C1	B1	X
Arrived	A1, B1, C1					A2, B2	C2		A1		B1			A2	B2, C1		A1		B1	
Preempted (OS ticks left)									C2 (1)								C1 (1)			

(b)

Task	Priority	e1	i1	e2	i2
A	1	1	4	1	2
B	2	1	3	1	3
C	3	2	2	2	4

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU1	A1	C1	C1	X	X	A2	X	X	A1	X	X	X	B2	A2	X	X	A1	X	X	X
CPU2	B1	X	X	X	B2	C2	C2	X	B1	X	X	C1	C1	X	X	C2	B1	C2	X	X
Arrived	A1, B1, C1				B2	A2, C2			A1, B1			C1	B2	A2		C2	A1, B1			
Preempted (OS ticks left)																	C2 (1)			

(c)

Task	Priority	e1	i1	e2	i2
A	1	1	4	1	2
B	2	1	3	1	3
C	3	2	2	2	4
D	4	1	2	2	3
E	5	2	3	3	3

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU1	A1	C1	C1	X	D2	C2	C2	X	A1	X	D1	C1	C1	A2	E1	E1	A1	X	X	E2
CPU2	B1	D1	E1	E1	B2	A2	D2	E2	B1	E2	E2	X	B2	D2	D2	C2	B1	C2	D1	X
Arrived	A1, B1, C1, D1, E1				B2, D2	A2, C2		E2	A1, B1		D1	C1	B2	A2, D2	E1	C2	A1, B1		D1	E2
Preempted (OS ticks left)						D2 (1)			E2 (2)								C2 (1)			

Problem 30: See the enclosed charts.

(a)

Task	Priority	e1	i1	e2	i2
A	1	1	4	1	2
B	2	1	3	1	3
C	3	2	2	2	4

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU	A1	B1	C1	C1	X	A2	B2	C2	C2	A1	B1	X	X	C1	C1	A2	B2	C2	C2	A1
Arrived	A1, B1, C1					A2, B2	C2		A1		B1			C1	A2, B2			C2	A1	

(b)

Task	Priority	e1	i1	e2	i2
A	1	1	4	1	2
B	2	1	3	1	3
C	3	2	2	2	4

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU1	A1	C1	C1	X	X	A2	X	X	A1	X	X	X	B2	A2	X	X	A1	B1	X	X
CPU2	B1	X	X	X	B2	C2	C2	X	B1	X	X	C1	C1	X	X	C2	C2	X	X	X
Arrived	A1, B1, C1				B2	A2, C2			A1, B1			C1	B2	A2		C2	A1, B1			

(c)

Task	Priority	e1	i1	e2	i2
A	1	1	4	1	2
B	2	1	3	1	3
C	3	2	2	2	4
D	4	1	2	2	3
E	5	2	3	3	3

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU1	A1	C1	C1	X	D2	D2	C2	C2	A1	B1	D1	X	C1	C1	B2	E1	E1	C2	C2	X
CPU2	B1	D1	E1	E1	B2	A2	X	E2	E2	E2	X	X	X	A2	D2	D2	A1	X	B1	D1
Arrived	A1, B1, C1, D1, E1				B2, D2	A2, C2		E2	A1, B1	D1			C1	A2, B2, D2, E1			A1, C2		B1	D1

Problem 31: See the enclosed chart.

Task	e	d	Arrive
A	5	10	0
B	2	7	2
C	3	8	4

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
CPU	S	A	S	A	S	A	A	A	B	B	C	C	C	X	X
Arrived	A		B		C										
Preempted (OS ticks left)			A (4)		A (3)										
Priority															
1	A		A		A										
2			B		B			B							
3					C			C		C					

Problem 32: See the enclosed chart.

Task	e	d	Arrive
A	5	10	0
B	2	7	2
C	3	8	4

deadline not met

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
CPU	S	A	S	B	S	C	C	C	B	A	A	A	A	X	X
Arrived	A		B		C										
Preempted (OS ticks left)			A (4)		B (1)										
Priority															
1	A		B		C										
2			A		B			B							
3					A			A	A						

Problem 33: See the enclosed chart.

Task	e	d	Arrive
A	5	10	0
B	2	7	2
C	3	8	4

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
CPU	S	A	S	B	S	B	A	A	A	A	C	C	C	X	X
Arrived	A		B		C										
Preempted (OS ticks left)			A (4)		B (1)										
Priority															
1	A		B		B										
2			A		A	A									
3					C	C				C					
Deadline															
A	10														
B			9												
C					12										

Problem 34: See the enclosed chart.

Task	e	d	Arrive
A	5	10	0
B	2	7	2
C	3	8	4

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
CPU	S	A	S	A	S	A	A	A	B	B	C	C	C	X	X
Arrived	A		B		C										
Preempted (OS ticks left)			A (4)		A (3)										
OS ticks left															
A	5	4	4	3	3	2	1	0							
B			2	2	2	2	2	2	1	0					
C					3	3	3	3	3	3	2	1	0		
Priority															
1	A		A		A										
2			B		B			B							
3					C			C		C					
Slack															
A	5		4		3										
B			5		3			0							
C					5			2		0					

Problem 35: See the enclosed charts.

(a)

Task	Priority	e	i	Arrive	Mutex
A	3	5	0	0	Y
B	2	4	0	2	Y
C	1	3	0	4	Y

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
CPU	A	A	A	A	A	C	C	C	B	B	B	B	X	X	X
Arrived	A		B		C										
Mutex owner	A					C			B						
Preempted (OS ticks left)															
Priority															
A	3	3	0		0										
B			2		2	2			2						
C					1	1									

(b)

Task	Priority	e	i	Arrive	Mutex
A	3	5	0	0	Y
B	2	4	0	2	Y
C	1	3	0	4	N

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
CPU	A	A	A	A	A	C	C	C	B	B	B	B	X	X	X
Arrived	A		B		C										
Mutex owner	A								B						
Preempted (OS ticks left)															
Priority															
A	3	3	0		0										
B			2		2	2			2						
C					1	1									

(c)

Task	Priority	e	i	Arrive	Mutex
A	3	5	0	0	Y
B	2	4	0	2	N
C	1	3	0	4	Y

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
CPU	A	A	B	B	A	A	A	C	C	C	B	B	X	X	X
Arrived	A		B		C										
Mutex owner	A							C							
Preempted (OS ticks left)			A (3)		B (2)										
Priority															
A	3		3		0										
B			2		2			2			2				
C					1			1							

Problem 36: See the enclosed charts.

(a)

Task	Priority	e	i	Arrive	Mutex
A	1	3	0	0	Y
B	2	4	0	2	Y
C	3	5	0	4	Y

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
CPU	A	A	A	B	B	B	B	C	C	C	C	C	X	X	X
Arrived	A		B		C										
Mutex owner	A			B				C							
Preempted (OS ticks left)															
Priority															
A	1		1												
B			2	2	2										
C					3			3							

(b)

Task	Priority	e	i	Arrive	Mutex
A	1	3	0	0	Y
B	2	4	0	2	Y
C	3	5	0	4	N

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
CPU	A	A	A	B	B	B	B	C	C	C	C	C	X	X	X
Arrived	A		B		C										
Mutex owner	A			B											
Preempted (OS ticks left)															
Priority															
A	1		1												
B			2	2	2										
C					3			3							

(c)

Task	Priority	e	i	Arrive	Mutex
A	1	3	0	0	Y
B	2	4	0	2	N
C	3	5	0	4	Y

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
CPU	A	A	A	B	B	B	B	C	C	C	C	C	X	X	X
Arrived	A		B		C										
Mutex owner	A							C							
Preempted (OS ticks left)															
Priority															
A	1		1												
B			2	2	2										
C					3			3							

Problem 37: See the enclosed charts.

(a)

Task	Priority	e	i	Arrive	Mutex
A	3	5	0	0	Y
B	2	4	0	2	Y
C	1	3	0	4	Y

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
CPU	A	A	A	A	A	C	C	C	B	B	B	B	X	X	X
Arrived	A		B		C										
Mutex owner	A					C			B						
Preempted (OS ticks left)															
Priority															
A	0		0		0										
B			2		2	2			0						
C					1	0									

(b)

Task	Priority	e	i	Arrive	Mutex
A	3	5	0	0	Y
B	2	4	0	2	Y
C	1	3	0	4	N

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
CPU	A	A	A	A	A	C	C	C	B	B	B	B	X	X	X
Arrived	A		B		C										
Mutex owner	A								B						
Preempted (OS ticks left)															
Priority															
A	0		0		0										
B			2		2	2			0						
C					1	1									

(c)

Task	Priority	e	i	Arrive	Mutex
A	3	5	0	0	Y
B	2	4	0	2	N
C	1	3	0	4	Y

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
CPU	A	A	A	A	A	C	C	C	B	B	B	B	X	X	X
Arrived	A		B		C										
Mutex owner	A					C									
Preempted (OS ticks left)															
Priority															
A	0		0		0										
B			2		2	2			2						
C					1	0									

Problem 38: See the enclosed charts.

(a)

Task	Priority	e	i	Post	Pend
A	1	2	20	B, C	
B	2	4	0	D, E	B
C	3	3	0	F, G	C
D	4	3	0		D
E	5	2	0		E
F	6	4	0		F
G	7	2	0		G

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU	A	A	B	B	B	B	C	C	C	D	D	D	E	E	F	F	F	F	G	G
Arrived	A to G																			
Preempted (OS ticks left)																				
Mailbox																				
B			V																	
C			V																	
D							V													
E							V													
F										V										
G										V										

(b)

Task	Priority	e	i	Post	Pend
A	1	2	20	B, C	
B	2	4	0	D, E	B
C	3	3	0	F, G	C
D	4	3	0		D
E	5	2	0		E
F	6	4	0		F
G	7	2	0		G

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU	A	A	B	B	B	B	C	C	C	D	D	D	E	E	F	F	F	F	G	G
Arrived	A to G																			
Preempted (OS ticks left)																				
Mailbox																				
B		V																		
C		V																		
D				V																
E				V																
F								V												
G								V												

(c)

Task	Priority	e	i	Post	Pend
A	1	2	20	B, C	
B	2	4	0	D, E	B
C	3	3	0	F, G	C
D	4	3	0		D
E	5	2	0		E
F	6	4	0		F
G	7	2	0		G

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU	A	A	B	B	B	B	C	C	C	D	D	D	E	E	F	F	F	F	G	G
Arrived	A to G																			
Preempted (OS ticks left)																				
Mailbox																				
B			V																	
C			V																	
D					V															
E					V															
F									V											
G									V											

Problem 39: See the enclosed charts.

(a)

Task	Priority	e	i	Post	Pend
A	7	2	20	B, C	
B	6	4	0	D, E	B
C	5	3	0	F, G	C
D	4	3	0		D
E	3	2	0		E
F	2	4	0		F
G	1	2	0		G

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU	A	A	C	C	C	G	G	F	F	F	F	B	B	B	B	E	E	D	D	D
Arrived	A to G																			
Preempted (OS ticks left)																				
Mailbox																				
B			V																	
C			V																	
D																V				
E																V				
F						V														
G						V														

(b)

Task	Priority	e	i	Post	Pend
A	7	2	20	B, C	
B	6	4	0	D, E	B
C	5	3	0	F, G	C
D	4	3	0		D
E	3	2	0		E
F	2	4	0		F
G	1	2	0		G

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU	A	C	G	G	F	F	F	F	C	C	B	E	E	D	D	D	B	B	B	A
Arrived	A to G											B (3)								
Preempted (OS ticks left)		A (1)	C (2)																	
Mailbox																				
B		V																		
C		V																		
D												V								
E												V								
F			V																	
G			V																	

(c)

Task	Priority	e	i	Post	Pend
A	7	2	20	B, C	
B	6	4	0	D, E	B
C	5	3	0	F, G	C
D	4	3	0		D
E	3	2	0		E
F	2	4	0		F
G	1	2	0		G

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU	A	A	C	C	G	G	F	F	F	F	C	B	B	E	E	D	D	D	B	B
Arrived	A to G																			
Preempted (OS ticks left)					C (1)									B (2)						
Mailbox																				
B			V																	
C			V																	
D														V						
E														V						
F					V															
G					V															

Problem 40: See the enclosed chart.

Task	Priority	e	i	Post	Pend
A	7	2	20	1B, 2C	
B	6	4	0	1D, 2E	B
C	5	3	0	1F, 2G	C
D	4	3	0		D
E	3	2	0		E
F	2	4	0		F
G	1	2	0		G

OS tick	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU	A	B	D	D	D	B	E	E	B	B	A	C	F	F	F	F	C	G	G	C
Arrived	A to G												C (2)					C (1)		
Preempted (OS ticks left)		A (1)	B (3)				B (2)													
Mailbox																				
B		V																		
C												V								
D			V																	
E							V													
F													V							
G																		V		