Chapter 10
Embedded Systems

10.1. Using Port A as input and Port B as output, the following program can be used:

```
#define   PAIN      (volatile unsigned char *) 0xFFFFFFF0
#define   PADIR     (volatile unsigned char *) 0xFFFFFFF2
#define   PBOUT     (volatile unsigned char *) 0xFFFFFFF4
#define   PBDIR     (volatile unsigned char *) 0xFFFFFFF5
#define   PCONT     (volatile unsigned char *) 0xFFFFFFF7
#define   CNTM      (volatile unsigned char *) 0xFFFFFFD0
#define   CTCON     (volatile unsigned char *) 0xFFFFFFD8
#define   CTSTAT    (volatile unsigned char *) 0xFFFFFFD9

unsigned char table[16] = { 0x40, 0x79, 0x24, 0x30, 0x19, 0x12,
    0x02, 0x78, 0x00, 0x18, 0x08, 0x03, 0x46, 0x21, 0x06, 0x0E };
int current_value, high_digit, low_digit, disp_count;

void main()
{
    *PADIR = 0x0;                       /* Configure Port A as input. */
    *PBDIR = 0xFF;                      /* Configure Port B as output. */
    *PCONT = 0x0;                       /* Read inputs directly from pins. */
    disp_count = 0;
    *CNTM = 100000000;                  /* One-second delay periods. */
    *CTCON = 0x1;                       /* Start the timer. */
    while (1)
    {
        current_value = *PAIN;
        low_digit = current_value & 0x0F;
        high_digit = (current_value >> 4) & 0x0F;
        if (disp_count == 0)                        /* Display high digit. */
          *PBOUT = table[high_digit];
        else if (disp_count == 1)                   /* Display low digit. */
          *PBOUT = table[low_digit];
        else                                        /* Display the dash. */
          *PBOUT = 0x3F;
        if (*CTSTAT == 1)
        {
          *CTSTAT = 0;                      /* Clear the Status register. */
          if disp_count < 3)
            disp_count++;
          else
            disp_count = 0;
        }
    }
}
```

10.2. Use Port A as input and Port B as output, where Port B connects directly to the seven-segment display. The following program uses a display counter to control the sequencing of displayed digits. The counter is incremented in one-second intervals, and it operates as a modulo-4 counter. The timer circuit in the microcontroller is used to raise interrupt requests every second.

```
#define   PAIN      (volatile unsigned char *) 0xFFFFFFF0
#define   PADIR     (volatile unsigned char *) 0xFFFFFFF2
#define   PBOUT     (volatile unsigned char *) 0xFFFFFFF4
#define   PBDIR     (volatile unsigned char *) 0xFFFFFFF5
#define   PCONT     (volatile unsigned char *) 0xFFFFFFF7
#define   CNTM      (volatile unsigned char *) 0xFFFFFFD0
#define   CTCON     (volatile unsigned char *) 0xFFFFFFD8
#define   CTSTAT    (volatile unsigned char *) 0xFFFFFFD9
#define   IVECT     (volatile unsigned int *) 0x20
```

. . .continued on the next page

```
unsigned char table[16] = { 0x40, 0x79, 0x24, 0x30, 0x19, 0x12,
    0x02, 0x78, 0x00, 0x18, 0x08, 0x03, 0x46, 0x21, 0x06, 0x0E };
int current_value, high_digit, low_digit, disp_count;

void intserv();

void main()
{
    *PADIR = 0x0;                       /* Configure Port A as input. */
    *PBDIR = 0xFF;                      /* Configure Port B as output. */
    *PCONT = 0x0;                       /* Read inputs directly from pins. */
    disp_count = 0;

    /* Initialize the interrupt mechanism */
    *IVECT = (unsigned int *) &intserv; /* Set the interrupt vector. */
    asm ("MoveControl    PSR, #0x40");  /* Respond to IRQ interrupts. */
    *CNTM = 100000000;                  /* One-second delay periods. */
    *CTCON = 0x11;                      /* Start the timer in interrupt mode. */

    while (1)
    {
        current_value = *PAIN;
        low_digit = current_value & 0x0F;
        high_digit = (current_value >> 4) & 0x0F;
        if (disp_count == 0)                        /* Display high digit. */
          *PBOUT = table[high_digit];
        else if (disp_count == 1)                   /* Display low digit. */
          *PBOUT = table[low_digit];
        else                                        /* Display the dash. */
          *PBOUT = 0x3F;
    }
}

/* Interrupt service routine */
interrupt void intserv()
{
    *CTSTAT = 0;                        /* Clear the timeout bit. */
    if disp_count < 3)
        disp_count++;
    else
        disp_count = 0;
}
```

10.3. We will sample the value of the incoming square-wave signal at the rate of one MHz. Each new sample will be compared with the previous sample to detect the changes in the signal value from 0 to 1 and from 1 to 0, thus detecting the transitions in the waveform. We will use a sampling period of 100 one-μs intervals. During this period, there will be 20 transitions in a 100-kHz signal and 10 transitions in a 50-kHz signal. We will use a threshold of 15 transitions to distinguish between the 100-kHz and 50-kHz signals. The following program can be used:

```
#define   PAIN     (volatile unsigned char *) 0xFFFFFFF0
#define   PADIR    (volatile unsigned char *) 0xFFFFFFF2
#define   PBOUT    (volatile unsigned char *) 0xFFFFFFF4
#define   PBDIR    (volatile unsigned char *) 0xFFFFFFF5
#define   PCONT    (volatile unsigned char *) 0xFFFFFFF7
#define   CNTM     (volatile unsigned char *) 0xFFFFFFD0
#define   CTCON    (volatile unsigned char *) 0xFFFFFFD8
#define   CTSTAT   (volatile unsigned char *) 0xFFFFFFD9
```

. . .continued on the next page

```
unsigned int samples, transitions, x_new, x_prev;

void main()
{
    *PADIR = 0x0;                   /* Configure Port A as input. */
    *PBDIR = 0xFF;                  /* Configure Port B as output. */
    *PCONT = 0x0;                   /* Read inputs directly from pins. */
    samples = transitions = 0;
    *CNTM = 100;                    /* One μs delay. */
    *CTCON = 0x1;                   /* Start the timer. */
    while (1)
    {
        if (*CTSTAT == 1)
        {
          *CTSTAT = 0;              /* Clear the Status register. */
          x_new = *PAIN & 0x1;      /* New sample. */
          transitions = transitions + (x_new ^ x_prev);
          x_prev = x_new;
          samples++;
          if (samples ≥ 100)        /* End of sampling period? */
          {
            samples = 0;
            if (transitions == 0)
              *PBOUT = 0x40;        /* Display 0. */
            else if (transitions > 15)
              *PBOUT = 0x0E;        /* Display F. */
            else
              *PBOUT = 0x12;        /* Display S. */
            transitions = 0;
          }
        }
    }
}
```

10.4. We will sample the value of the incoming square-wave signal at the rate of one MHz. Each new sample will be compared with the previous sample to detect the changes in the signal value from 0 to 1 and from 1 to 0, thus detecting the transitions in the waveform. We will use a sampling period of 100 one-μs intervals. During this period, there will be 20 transitions in a 100-kHz signal and 10 transitions in a 50-kHz signal. We will use a threshold of 15 transitions to distinguish between the 100-kHz and 50-kHz signals. The following program can be used. It uses timer interrupts at one-μs intervals.

```
#define   PAIN     (volatile unsigned char *) 0xFFFFFFF0
#define   PADIR    (volatile unsigned char *) 0xFFFFFFF2
#define   PBOUT    (volatile unsigned char *) 0xFFFFFFF4
#define   PBDIR    (volatile unsigned char *) 0xFFFFFFF5
#define   PCONT    (volatile unsigned char *) 0xFFFFFFF7
#define   CNTM     (volatile unsigned char *) 0xFFFFFFD0
#define   CTCON    (volatile unsigned char *) 0xFFFFFFD8
#define   CTSTAT   (volatile unsigned char *) 0xFFFFFFD9
#define   IVECT    (volatile unsigned int *) 0x20
```

. . .continued on the next page

```
unsigned int samples, transitions, x_new, x_prev;

void intserv();

void main()
{
    *PADIR = 0x0;                          /* Configure Port A as input. */
    *PBDIR = 0xFF;                         /* Configure Port B as output. */
    *PCONT = 0x0;                          /* Read inputs directly from pins. */
    samples = transitions = 0;

    /* Initialize the interrupt mechanism */
    *IVECT = (unsigned int *) &intserv;    /* Set the interrupt vector. */
    asm ("MoveControl    PSR, #0x40");     /* Respond to IRQ interrupts. */
    *CNTM = 100;                           /* One-μs delay periods. */
    *CTCON = 0x11;                         /* Start the timer in interrupt mode. */

    while (1);                             /* Continuous loop. */
}

/* Interrupt service routine */
interrupt void intserv()
{
    *CTSTAT = 0;                           /* Clear the Status register. */
    x_new = *PAIN & 0x1;                   /* New sample. */
    transitions = transitions + (x_new ^ x_prev);
    x_prev = x_new;
    samples++;
    if (samples ≥ 100)                     /* End of sampling period? */
    {
     samples = 0;
     if (transitions == 0)
        *PBOUT = 0x40;                     /* Display 0. */
     else if (transitions > 15)
        *PBOUT = 0x0E;                     /* Display F. */
     else
        *PBOUT = 0x12;                     /* Display S. */
     transitions = 0;
    }
}
```

10.5. Connect character input to the serial port and the 7-segment display unit to parallel ports A and B. A possible program is:

```
#define   RBUF     (volatile unsigned char *) 0xFFFFFFE0
#define   SSTAT    (volatile unsigned char *) 0xFFFFFFE2
#define   PAOUT    (volatile unsigned char *) 0xFFFFFFF1
#define   PADIR    (volatile unsigned char *) 0xFFFFFFF2
#define   PBOUT    (volatile unsigned char *) 0xFFFFFFF4
#define   PBDIR    (volatile unsigned char *) 0xFFFFFFF5

unsigned char table[16] = { 0x40, 0x79, 0x24, 0x30, 0x19, 0x12,
    0x02, 0x78, 0x00, 0x18, 0x08, 0x03, 0x46, 0x21, 0x06, 0x0E };
unsigned int j, temp;

void main()
{
    /* Initialize the parallel ports */
    *PADIR = 0xFF;                      /* Configure Port A as output. */
    *PBDIR = 0xFF;                      /* Configure Port B as output. */

    /* Transfer the characters. */
    while (1)                           /* Infinite loop. */
    {
        while ((*SSTAT & 0x1) == 0);    /* Wait for a new character. */
        if (*RBUF == 'H')
        {
          while ((*SSTAT & 0x1) == 0);  /* Wait for the first digit. */
          j = *RBUF & 0xF;              /* Extract the BCD value. */
          temp = table[j];              /* Prepare 7-segment code for Port A. */
          while ((*SSTAT & 0x1) == 0);  /* Wait for the second digit. */
          j = *RBUF & 0xF;              /* Extract the BCD value. */
          *PBOUT = table[j];            /* Send the 7-segment code to Port B. */
          *PAOUT = temp;                /* Send the 7-segment code to Port A. */
        }
    }
}
```

10.6. Connect character input to the serial port and the 7-segment display unit to parallel ports A and B. A possible program is:

```
#define   RBUF     (volatile unsigned char *) 0xFFFFFFE0
#define   SCONT    (volatile unsigned char *) 0xFFFFFFE3
#define   PAOUT    (volatile unsigned char *) 0xFFFFFFF1
#define   PADIR    (volatile unsigned char *) 0xFFFFFFF2
#define   PBOUT    (volatile unsigned char *) 0xFFFFFFF4
#define   PBDIR    (volatile unsigned char *) 0xFFFFFFF5
#define   IVECT    (volatile unsigned int *) 0x20

unsigned char table[16] = { 0x40, 0x79, 0x24, 0x30, 0x19, 0x12,
    0x02, 0x78, 0x00, 0x18, 0x08, 0x03, 0x46, 0x21, 0x06, 0x0E };
unsigned int j;
unsigned char digits[2];                    /* Buffer for received BCD digits */
unsigned int k = 0;                         /* Set up to detect the first H */

interrupt void intserv();

void main()
{
    /* Initialize the parallel ports */
    *PADIR = 0xFF;                          /* Configure Port A as output */
    *PBDIR = 0xFF;                          /* Configure Port B as output */

    /* Initialize the interrupt mechanism */
    *IVECT = (unsigned int *) &intserv;     /* Set the interrupt vector. */
    asm ("MoveControl    PSR, #0x40");      /* Respond to IRQ interrupts. */
    *SCONT = 0x10;                          /* Enable receiver interrupts. */

    /* Transfer the characters */
    while (1);                              /* Infinite loop */
}

/* Interrupt service routine */
interrupt void intserv()
{
    if (k > 0)
    {
        j = *RBUF & 0xF;                    /* Extract the BCD value */
        k = k - 1;
        digits[k] = table[j];               /* Save 7-segment code for new digit */
        if (k == 0)
        {
          *PAOUT = digits[1];               /* Send first digit to Port A */
          *PBOUT = digits[0];               /* Send second digit to Port B */
        }
    else if (*RBUF == 'H') k = 2;
    }
}
```

10.7. Connect the parallel ports A and B to the four BCD to 7-segment decoders. Choose that PA_{7-4}, PA_{3-0}, PB_{7-4} and PB_{3-0} display the first, second, third and fourth received digits, respectively. Assume that all four digits arrive immediately after the character H has been received. The task can be achieved by the following program:

```
#define   RBUF     (volatile unsigned char *) 0xFFFFFFE0
#define   SSTAT    (volatile unsigned char *) 0xFFFFFFE2
#define   PAOUT    (volatile unsigned char *) 0xFFFFFFF1
#define   PADIR    (volatile unsigned char *) 0xFFFFFFF2
#define   PBOUT    (volatile unsigned char *) 0xFFFFFFF4
#define   PBDIR    (volatile unsigned char *) 0xFFFFFFF5

char temp;
char digits[4];                              /* Buffer for received digits. */
int i;

void main()
{
    /* Initialize the parallel ports. */
    *PADIR = 0xFF;                           /* Configure Port A as output. */
    *PBDIR = 0xFF;                           /* Configure Port B as output. */

    /* Transfer the characters. */
    while (1)                                /* Infinite loop. */
    {
        while ((*SSTAT & 0x1) == 0);         /* Wait for a new character. */
        if (*RBUF == 'H')
        {
          for (i = 3; i >= 0; i--)
          {
            while ((*SSTAT & 0x1) == 0);     /* Wait for the next digit. */
            digits[i] = *RBUF;               /* Save the new digit (ASCII). */
          }
          temp = digits[3] << 4;             /* Shift left first digit by 4 bits, */
          *PAOUT = temp | (digits[2] & 0xF); /*   append second and send to A. */
          temp = digits[1] << 4;             /* Shift left third digit by 4 bits, */
          *PBOUT = temp | (digits[0] & 0xF); /*   append fourth and send to B. */
        }
    }
}
```

10.8. Connect the parallel ports A and B to the four BCD to 7-segment decoders. Choose that PA_{7-4}, PA_{3-0}, PB_{7-4} and PB_{3-0} display the first, second, third and fourth received digits, respectively. Upon detecting the character H, the subsequent four digits have to be saved and displayed only when the fourth digit arrives. Interrupts are used to detect the arrival of both H and the four digits. Therefore, the interrupt service routine has to keep track of the received characters. Variable k is set to 4 when an H is detected, and it is decremented as the subsequent digits arrive. The following program may be used:

```
#define   RBUF     (volatile unsigned char *) 0xFFFFFFE0
#define   SCONT    (volatile unsigned char *) 0xFFFFFFE3
#define   PAOUT    (volatile unsigned char *) 0xFFFFFFF1
#define   PADIR    (volatile unsigned char *) 0xFFFFFFF2
#define   PBOUT    (volatile unsigned char *) 0xFFFFFFF4
#define   PBDIR    (volatile unsigned char *) 0xFFFFFFF5
#define   IVECT    (volatile unsigned int *) 0x20


char temp;
char digits[4];                          /* Buffer for received digits. */
int k;

interrupt void intserv();

void main()
{
    /* Initialize the parallel ports. */
    *PADIR = 0xFF;                       /* Configure Port A as output. */
    *PBDIR = 0xFF;                       /* Configure Port B as output. */
    /* Initialize the interrupt mechanism. */
    *IVECT = (unsigned int *) &intserv;  /* Set the interrupt vector. */
    asm ("MoveControl    PSR, #0x40");   /* Respond to IRQ interrupts. */
    *SCONT = 0x10;                       /* Enable receiver interrupts. */

    /* Transfer the characters. */
    k = 0;
    while (1);                           /* Infinite loop */
}

/* Interrupt service routine. */
interrupt void intserv()
{
    if (k > 0)
    {
        k = k - 1;
        digits[k] = *RBUF;               /* Save the new digit (ASCII). */
        if (k == 0)
        {
          temp = digits[3] << 4;         /* Shift left first digit by 4 bits, */
          *PAOUT = temp | (digits[2] & 0xF);  /*   append second and send to A. */
          temp = digits[1] << 4;         /* Shift left third digit by 4 bits */
          *PBOUT = temp | (digits[0] & 0xF);  /*   append fourth and send to B. */
        }
        else if (*RBUF == 'H')
          k = 4;
    }
}
```

10.9. Use a table to convert a received ASCII digit into a 7-segment code. Connect the bits S_a to S_g of all four registers to bits PA_{6-0} of Port A. Use bits PB_3 to PB_0 of Port B as Load signals for the registers displaying the first, second, third and fourth received digits, recpectively. The following program can be used:

```
#define   RBUF    (volatile unsigned char *) 0xFFFFFFE0
#define   SSTAT   (volatile unsigned char *) 0xFFFFFFE2
#define   PAOUT   (volatile unsigned char *) 0xFFFFFFF1
#define   PADIR   (volatile unsigned char *) 0xFFFFFFF2
#define   PBOUT   (volatile unsigned char *) 0xFFFFFFF4
#define   PBDIR   (volatile unsigned char *) 0xFFFFFFF5

unsigned char table[16] = { 0x40, 0x79, 0x24, 0x30, 0x19, 0x12,
    0x02, 0x78, 0x00, 0x18, 0x08, 0x03, 0x46, 0x21, 0x06, 0x0E };
char j;
char digits[4];                          /* Buffer for received digits. */
int i, k;

void main()
{
    /* Initialize the parallel ports */
    *PADIR = 0xFF;                       /* Configure Port A as output. */
    *PBDIR = 0xFF;                       /* Configure Port B as output. */

    /* Transfer the characters. */
    k = 0;
    while (1)                            /* Infinite loop. */
    {
        while ((*SSTAT & 0x1) == 0);     /* Wait for a new character. */
        if (*RBUF == 'H')
        {
          for (i = 3; i >= 0; i--)
          {
            while ((*SSTAT & 0x1) == 0); /* Wait for the next digit. */
            j = *RBUF & 0xF;             /* Extract the BCD value. */
            digits[i] = table[j];        /* Save 7-segment code for the digit. */
          }
          for (i = 0; i <= 3; i++)
          {
            *PAOUT = digits[i];          /* Send a digit to Port A. */
            *PBOUT = 1 << i;             /* Load the digit into its register. */
            *PBOUT = 0;                  /* Clear the Load signal. */
          }
        }
    }
}
```

10.10. Use a table to convert a received ASCII digit into a 7-segment code. Connect the bits S_a to S_g of all four registers to bits PA_{6-0} of Port A. Use bits PB_3 to PB_0 of Port B as Load signals for the registers displaying the first, second, third and fourth received digits, recpectively. Upon detecting the character H, the subsequent four digits have to be saved and displayed only when the fourth digit arrives. Interrupts are used to detect the arrival of both H and the four digits. Therefore, the interrupt service routine has to keep track of the received characters. Variable k is set to 4 when an H is detected, and it is decremented as the subsequent digits arrive. The following program accomplishes the task:

```
#define   RBUF     (volatile unsigned char *) 0xFFFFFFE0
#define   SCONT    (volatile unsigned char *) 0xFFFFFFE3
#define   PAOUT    (volatile unsigned char *) 0xFFFFFFF1
#define   PADIR    (volatile unsigned char *) 0xFFFFFFF2
#define   PBOUT    (volatile unsigned char *) 0xFFFFFFF4
#define   PBDIR    (volatile unsigned char *) 0xFFFFFFF5
#define   IVECT    (volatile unsigned int *) 0x20
```

. . .continued on the next page

```
unsigned char table[16] = { 0x40, 0x79, 0x24, 0x30, 0x19, 0x12,
    0x02, 0x78, 0x00, 0x18, 0x08, 0x03, 0x46, 0x21, 0x06, 0x0E };
char j;
char digits[4];                         /* Buffer for received digits. */
int i, k;

interrupt void intserv();

void main()
{
    /* Initialize the parallel ports. */
    *PADIR = 0xFF;                      /* Configure Port A as output. */
    *PBDIR = 0xFF;                      /* Configure Port B as output. */

    /* Initialize the interrupt mechanism. */
    *IVECT = (unsigned int *) &intserv;     /* Set the interrupt vector. */
    asm ("MoveControl    PSR, #0x40");      /* Respond to IRQ interrupts. */
    *SCONT = 0x10;                      /* Enable receiver interrupts. */

    /* Transfer the characters. */
    k = 0;
    while (1);                          /* Infinite loop */
}

/* Interrupt service routine. */
interrupt void intserv()
{
    if (k > 0)
    {
        j = *RBUF & 0xF;                /* Extract the BCD value. */
        k = k - 1;
        digits[k] = table[j];           /* Save 7-segment code for new digit. */
        if (k == 0)
        {
          for (i = 0; i <= 3; i++)
          {
            *PAOUT = digits[i];         /* Send a digit to Port A. */
            *PBOUT = 1 << i;            /* Load the digit into its register. */
            *PBOUT = 0;                 /* Clear the Load signal. */
          }
        }
    else if (*RBUF == 'H')
        k = 4;
    }
}
```

10.11. Connect the two 7-segment displays to Port A. Use the 3 bits of Port B to connect to the switches and LED as shown in Figure 10.13. It is necessary to modify the conversion and display portions of the program in Figure 10.14.

The end of the program in Figure 10.14 should be:

```
            /* Compute the total count */
            total_count = (0xFFFFFFFF − counter_value);

            /* Convert count to time */ ;
            actual_time = total_count / 1000000;          /* Time in hundredths of seconds */
            tenths = actual_time / 10;
            hundredths = actual_time − tenths * 10;

            *PAOUT = ((tenths << 4) | hundredths);    /* Display the elapsed time */
        }
    }
```