

EMBEDDED SYSTEM DESIGN (E3-257)

LAB ASSIGNMENT – 4

Explanation of Code

Apart from the functionalities implemented until lab 3 following new additions were made:

➔ Initialized the seven-segment display:

Configured the port A and port B pins correctly to keep active and display the count and cycles of colour changes.

```
39 void SSD_init(void)
40 {
41     SYSCFG_RCGC2_R |= 0x00000003;    // enable clock to GPIOA, GPIOB at clock gating control register
42     // Enable the GPIO pins
43     // For PORTB, all pins are used to set 7 segment display
44     // For PORTA, pins 7 to 4 are used for selecting one of the four 7 segment display
45     GPIO_PORTA_DIR_R |= 0xF0;        // PA4 to PA7 set to output
46     GPIO_PORTB_DIR_R |= 0xFF;        // PB0 to PB7 set to output
47     // GPIO_PORTC_DIR_R |= 0xF0;
48     // enable the GPIO pins for digital function
49     GPIO_PORTA_DEN_R |= 0xF0;        // enabling PA4 to PA7
50     GPIO_PORTB_DEN_R |= 0xFF;        // enabling PB0 to PB8
51     // GPIO_PORTC_DEN_R |= 0xF0;
52 }
```

- ➔ Kept the display steady to parallelly out 3 digits using very minimal delay between flickering to meet persistence of vision as below
- ➔ Display at Zeroth place reflects the colour code going from 0 for green to 6 till white and updating the 3rd display by 1 to account for finishing one cycle.
- ➔ Similarly, as the cycle updates to 2-digit count 4th place is updates once 3rd display crosses 9. Same is shown below in code:

```
734
735     for(int i =0;i<20;i++){
736         GPIO_PORTA_DATA_R &= ~(0xF0);
737         GPIO_PORTA_DATA_R |= 0x10;
738         GPIO_PORTB_DATA_R = 0;
739         GPIO_PORTB_DATA_R = digitPattern[colour_mode];
740
741
742     for(int i =0;i<20;i++){
743         GPIO_PORTA_DATA_R &= ~(0xF0);
744         GPIO_PORTA_DATA_R |= 0x80;
745         GPIO_PORTB_DATA_R = 0;
746         GPIO_PORTB_DATA_R = digitPattern[count_2];
747
748
749     for(int i =0;i<20;i++){
750
751         GPIO_PORTA_DATA_R &= ~(0xF0);
752         GPIO_PORTA_DATA_R |= 0x40;
753         GPIO_PORTB_DATA_R = 0;
754         GPIO_PORTB_DATA_R = digitPattern[count_1];
755
756
757     for(int i =0;i<20;i++){
758
759         if (colour_mode == 6)
760         {
761             count_1 = count_1 + 1;
762             GPIO_PORTA_DATA_R &= ~(0xF0);
763             GPIO_PORTA_DATA_R |= 0x40;
764             GPIO_PORTB_DATA_R = 0;
765             GPIO_PORTB_DATA_R = digitPattern[count_1];
766             colour_mode = 0;
767
768         }
769
770         if(count_1 > 9)
771         {
772             count_2 = count_2 + 1;
773             GPIO_PORTA_DATA_R &= ~(0xF0);
774             GPIO_PORTA_DATA_R |= 0x80;
775             GPIO_PORTB_DATA_R = 0;
776             GPIO_PORTB_DATA_R = digitPattern[count_2];
777             count_1 = 0;
778             colour_mode = 0;
779             for(int i =0;i<20;i++){
780
781
782         }
783
```

➔ **Defining the new states**

As per the task requirement the following new states were added to be reflected as console commands as well as key presses.

➔ Given below is the addition to code to accept these states via console commands:

```

448 else if ((strcmp(cmnd.type, "stop")==0))
449 {
450
451     flag = 1;
452     colour_mode = 7;
453     stop_stat = 1;
454     printstring("Valid Entry\n\r");
455     check= 1;
456 }
457
458 else if ((strcmp(cmnd.type, "start")==0))
459 {
460     blink_mode = 0;
461     flag = 0;
462     stop_stat = 0;
463     colour_mode = 0;
464     factor = 2 ;
465     count_2 = 0;
466     count_1 =0 ;
467     printstring("Valid Entry\n\r");
468     check= 1;
469 }
470
471
472 else if ((strcmp(cmnd.type, "pause")==0))
473 {
474
475     flag = 1;
476     pause_stat = 1;
477     //colour_mode = 7;
478     printstring("Valid Entry\n\r");
479     check= 1;
480 }
481
482 else if ((strcmp(cmnd.type, "resume")==0))
483 {
484     pause_stat = 0;
485     flag = 0;
486     //colour_mode = 7;
487     printstring("Valid Entry\n\r");
488     check= 1;
489 }

```

➔ **Handling the new command length and input style from console:**

The previous UART console assignment the command lengths were fixed, but the addition of new commands required to play around with string manipulation to adjust and accept them differently as shown below:

```
338
339 if(b==6)
340 {
341     for(int i = 0; i<6; i++)
342         cmd.type[i] = full_cmd[i];
343
344     printstring("Option: ");
345
346     for(int i = 0; i<6; i++)
347         UART0_Transmitter(cmd.type[i]);
348     UART0_Transmitter('\n');
349     UART0_Transmitter('\r');
350 }
351
352 else
353 {for(int i = 0; i<5; i++)
354 cmd.type[i] = full_cmd[i];
355
356 printstring("Option: ");
357
358 for(int i = 0; i<5; i++)
359     UART0_Transmitter(cmd.type[i]);
360 UART0_Transmitter('\n');
361 UART0_Transmitter('\r');
362 }
```

- ➔ Since the new commands “stop”, “pause”, “start”, “resume” are not having any value to accompany as in command “color” and “blink” we have just assigned zeros to cmd.data[] item of struct declared initially.

```
366 if((pause_stat == 0))
367 {printstring("Value: ");
368 for(int i = 0; i<(b-5); i++)
369 {
370     cmd.data[i] = full_cmd[i+5];
371     UART0_Transmitter(cmd.data[i]);
372 }
373 UART0_Transmitter('\n');
374 UART0_Transmitter('\r');
375 }
376
377 else if((pause_stat == 1))
378 {
379     printstring("Value: ");
380     for(int i = 0; i<(b-5); i++)
381     {
382         cmd.data[i] = '\0';
383         UART0_Transmitter(cmd.data[i]);
384     }
385     UART0_Transmitter('\n');
386     UART0_Transmitter('\r');
387 }
```

```

388
389 if(pause_stat == 0)
390 {if((stop_stat == 0))
391 {printstring("Value: ");
392 for(int i = 0; i<(b-5); i++)
393 {
394     cmd.data[i] = full_cmd[i+5];
395     UART0_Transmitter(cmd.data[i]);
396 }
397 UART0_Transmitter('\n');
398 UART0_Transmitter('\r');
399
400 }
401
402 else if((stop_stat == 1))
403 {
404     printstring("Value: ");
405     for(int i = 0; i<(b-5); i++)
406     {
407         cmd.data[i] = '\0';
408         UART0_Transmitter(cmd.data[i]);
409     }
410     UART0_Transmitter('\n');
411     UART0_Transmitter('\r');
412 }
413 }

```

➔ Using keypad to toggle between states

➔ We add the same state definitions to after keypress is checked for and toggle state is realised for either of the first two switches in the training kit:

```

669 if(isKeyPressed())
670 {
671     for(int i = 0; i<100; i++){
672         data = readkey();
673
674         if(data == 'A')
675         {
676             if(!stat_key_1)
677             {
678                 stat_key_1 = 1;
679                 flag = 1;
680                 colour_mode = 7;
681                 printstring("Stop\n\r");
682                 stop_stat = 1;
683                 check = 1;
684             }
685
686             else
687             {
688                 stat_key_1 = 0;
689                 stop_stat = 0;
690                 blink_mode = 0;
691                 flag = 0;
692                 colour_mode = 0;
693                 factor = 2 ;
694                 count_2 = 0;
695                 count_1 = 0 ;
696                 printstring("Start\n\r");
697                 check = 1;
698             }
699         }
700 }

```

```
704         else if(data == 'B')
705         {
706             if(!stat_key_2)
707             {
708                 stat_key_2 = 1;
709
710                 flag = 1;
711                 //colour_mode = 7;
712                 printstring("Pause\n\r");
713                 check= 1;
714
715             }
716
717             else
718             {
719                 stat_key_2 = 0;
720                 flag = 0;
721                 //colour_mode = 7;
722                 printstring("Resume\n\r");
723                 check= 1;
724
725             }
726
727         }
728
729         while(isKeyPressed());
730     }
731
732
```

→ Ensuring the States don't change by push button or console if its in "pause" or "stop" state until "start"/"resume" comes in

Using certain flag assignments, we keep checking what state it is while the GPIO interrupts are happening if the flag indicating a pause/ stop state is high we try to maintain the colour_mode and blink rate or turn them off completely respectively.

```
if((strcmp(cmnd.type, "pause")==0)|| (stat_key_2 == 1)|| (stat_key_1 == 1))
{
    colour_mode = temp;
    //pause_stat = 1;
}
else if((strcmp(cmnd.type, "resume")==0)|| (strcmp(cmnd.type, "start")==0)|| (stat_key_2 == 0)|| (stat_key_1 == 0))
{

```

```
blink_mode = 0;
if((strcmp(cmnd.type, "pause")==0)|| (stat_key_2 == 1)|| (stat_key_1 == 1))
{
    flag = 1;
}
else if ((strcmp(cmnd.type, "resume")==0)|| (strcmp(cmnd.type, "start")==0)|| (stat_key_1 == 1)|| (stat_key_2 == 0)|| (stat_key_1 == 0))
{
    if(j==0)
    {
        flag = 0;
    }
}
```
