20/03/2017

Embedded Systems Assignment 1

ASSIGNMENT TITLE: 4 DIGIT DIGITAL COMBINATION LOCK

STUDENT NAME: CIARA POWER
STUDENT NUMBER: 20072488

Table of Contents

System Description 2
Inputs 2
Outputs 2
Normal Operation 3
Error Handling 3
Event Response List 4
Statechart 5
Next-State Table 6
Outline Architecture 7
General Structure 8
Complete List of Apps Used 8
App Configuration 9
Digital_IO: Button_1/ Button_2/ Button_3/ Button_4/ Button_Reset 9
Digital_IO: Lock_LED/ Error_LED 9
Timer: Minutes_Timer 10
Timer: Seconds_Timer 10
Timer: Debounce_Timer 11
Counter: PushButton_Count 11
Interrupt: Minutes_Timer 12
Interrupt: Seconds_Timer 12
Pin Allocation 13
Source Code Files Appendix 14
File: Main.c 14

System Description

This assignment involves developing a digital combination lock for the XMC1100 micro controller using applications such as “DAVE”, “KEIL”, and “MICRIUM µC PROBE”.

The lock is a 4 digit combination lock, and will use inputs and outputs from the microcontroller pins, with some inputs set up in the Probe tool.

Inputs

The inputs are as follows:

- **4 numeric pushbutton inputs -**
  These inputs are set up in the Probe Tool. Each button corresponds to a number – 1,2,3,4. These are used to enter in combination sequences. Each button input is 0 while not being pressed, and are put to 1 when being pressed. They become 0 once again when the user releases the button.
- **1 reset pushbutton input –**
  This input is set up in the Probe tool also. As with the numeric pushbutton inputs, it is 1 when being pressed, and 0 when released.

All of the above pushbutton inputs are connected to Digital I/O input pins on the microcontroller. Each pushbutton input needs to be debounced for 20ms when activated also.

Outputs

The outputs are as follows:

- **A lock output –**
  This output is connected to P1.0 on the microcontroller, which is the red LED found on the top-left of the chip. This is at logic 1 (LED ON) when the lock is activated, and logic 0 (LED OFF) when the lock is open.
- **An error output –**
  This output is connected to P1.1 on the microcontroller, which is the red LED found on the top-right of the chip. This goes to logic 1 (LED ON) in various error situations, and is at logic 0 (LED OFF) otherwise.

Normal Operation

- The lock is initially open, and reset to the default combination.
- The default combination for the lock initially is 1234.
- When the lock is open, if the correct 4 digit combination sequence is entered, the lock will close.
- When the lock is closed, if the correct 4 digit combination sequence is entered, the lock will open.
- To set a new combination for the lock, press the RESET button (putting the lock into RESET MODE), followed by the new 4 digit combination. When the new combination is entered, the lock will stay in its current state (open/closed).
- The lock output is activated (logic 1) when the lock is closed, otherwise logic 0.

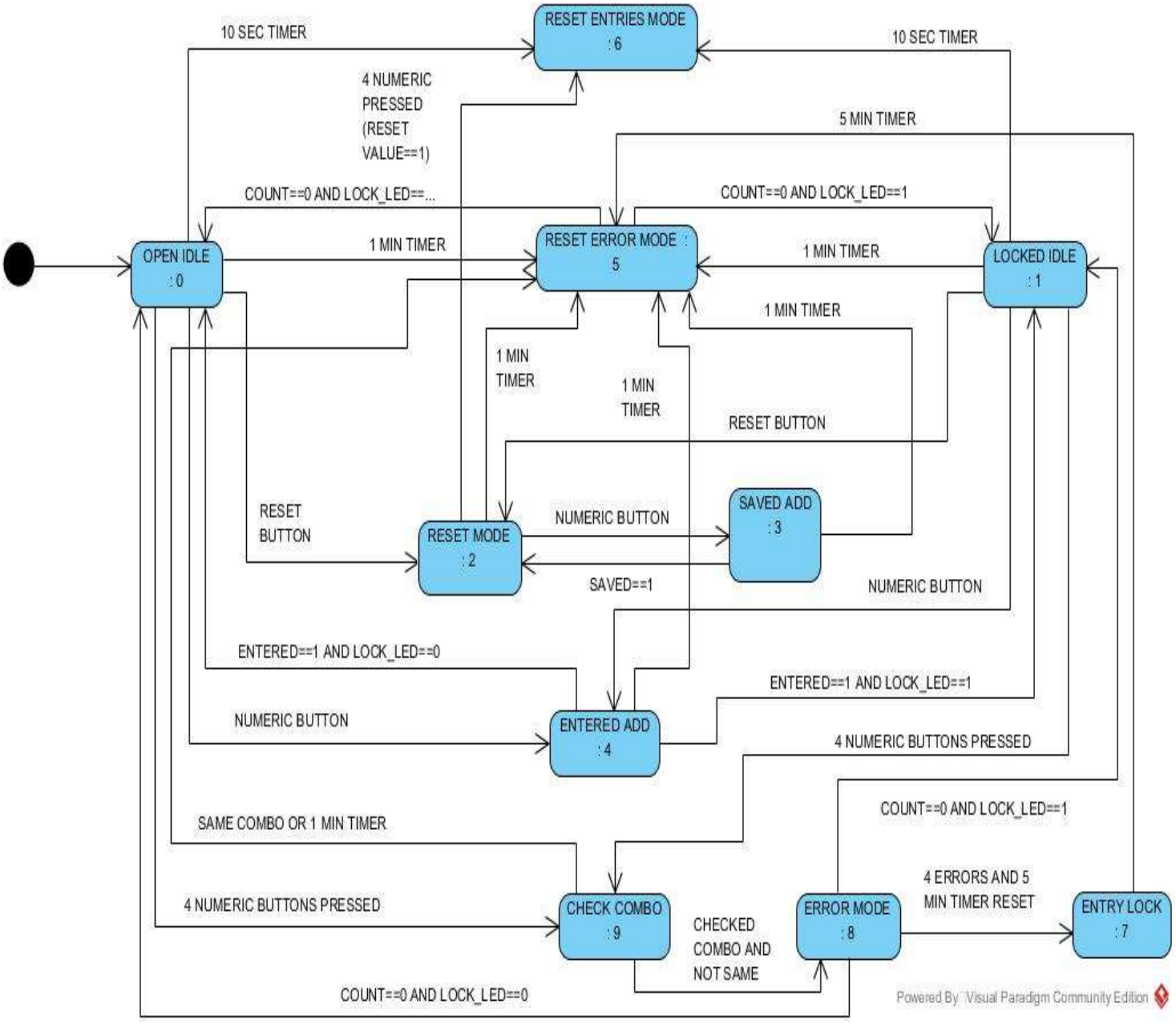
Error Handling

- If a full 4 digit combination sequence is not entered within 10 seconds, the user input sequence is disregarded and the lock returns to an idle state.
- If an invalid combination is entered, the error output LED flashes for 1 second (a single flash).
- If more than 3 unsuccessful combination sequences are entered within 1 minute, then the error output LED flashes 3 times (1 second each time), and any further attempts are prevented for 5 minutes.

Event Response List

EVENTS		RESPONSE
0	No buttons pressed AND unlocked	Lock moves to the Open Idle state
1	RESET button pressed	Lock moves to the Reset Mode state
2	4 buttons pressed in reset mode	Lock moves to the Reset Entries Mode state
3	Numeric button pressed	Lock moves to the Entered Add state
4	Numeric button put into entered combo AND unlocked	Lock moves to the Open Idle state
5	Entered combo is correct	Lock moves to the Reset Error Mode state
6	Numeric button pressed put into saved combo	Lock moves to the Reset Mode state
7	4 buttons pressed	Lock moves to the Check Combo Mode state
8	Entered combo checked and is incorrect	Lock moves to the Error Mode state
9	4 errors AND 5 Min Timer has not started	Lock moves to the Entry Lock Mode state
10	5 Min Timer Completes	Lock moves to the Reset Error Mode state
11	No buttons pressed AND locked	Lock moves to the Locked Idle state
12	10 Second Timer Completes	Lock moves to the Reset Entries Mode state
13	1 Min Timer Completes	Lock moves to the Reset Error Mode state
14	Numeric button put into entered combo AND locked	Lock moves to the Locked Idle state

Statechart

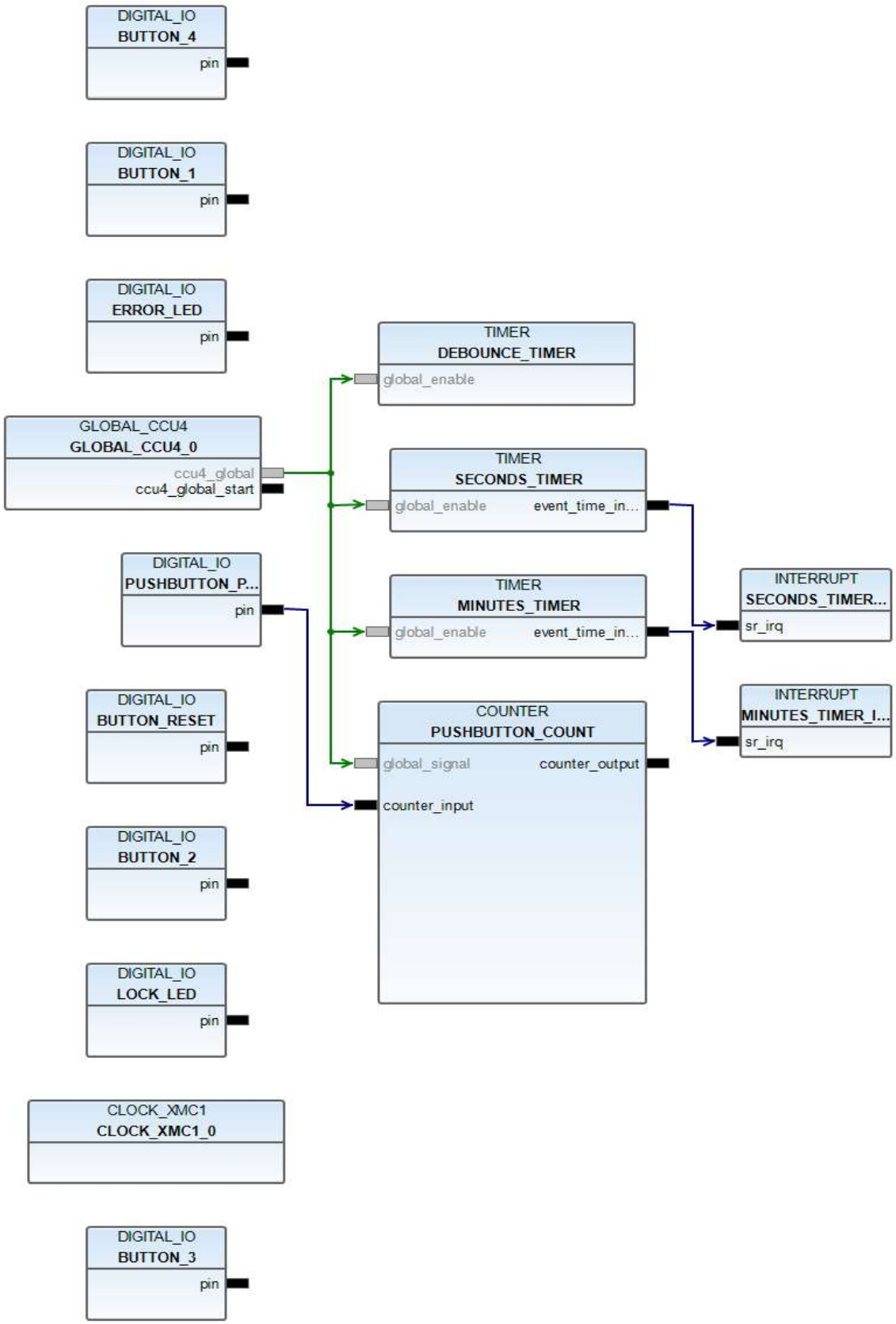

RESET ENTRIES MODE
: 6
10 SEC TIMER
10 SEC TIMER
4 NUMERIC
PRESSED
(RESET
VALUE==1)
5 MIN TIMER
COUNT==0 AND LOCK_LED==...
COUNT==0 AND LOCK_LED==1
OPEN IDLE
: 0
1 MIN TIMER
RESET ERROR MODE
5
1 MIN TIMER
LOCKED IDLE
: 1
1 MIN TIMER
1 MIN
TIMER
1 MIN
TIMER
RESET BUTTON
RESET
BUTTON
RESET MODE
: 2
NUMERIC BUTTON
SAVED ADD
: 3
SAVED==1
NUMERIC BUTTON
ENTERED==1 AND LOCK_LED==0
ENTERED==1 AND LOCK_LED==1
NUMERIC BUTTON
ENTERED ADD
: 4
4 NUMERIC BUTTONS PRESSED
COUNT==0 AND LOCK_LED==1
SAME COMBO OR 1 MIN TIMER
4 ERRORS AND 5
MIN TIMER RESET
4 NUMERIC BUTTONS PRESSED
CHECK COMBO
: 9
CHECKED
COMBO AND
NOT SAME
ERROR MODE
: 8
ENTRY LOCK
: 7
COUNT==0 AND LOCK_LED==0
Powered By Visual Paradigm Community Edition

Next-State Table

EVENTS

CURRENT STATE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Open Idle: ***0***	-1	2	-1	4	-1	-1	-1	9	-1	-1	-1	-1	6	5	-1
Locked Idle: ***1***	-1	2	-1	4	-1	-1	-1	9	-1	-1	-1	-1	6	5	-1
Reset Mode: ***2***	-1	-1	6	3	-1	-1	-1	-1	-1	-1	-1	-1	-1	5	-1
Saved Add: ***3***	-1	-1	-1	-1	-1	-1	2	-1	-1	-1	-1	-1	-1	5	-1
Entered Add: ***4***	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	5	1
Reset Error Mode: ***5***	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1
Reset Entries Mode: ***6***	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	5	-1
Entry Lock: ***7***	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	5	-1	-1	-1	-1
Error Mode: ***8***	0	-1	-1	-1	-1	-1	-1	-1	-1	7	-1	1	-1	-1	-1
Check Combo: ***9***	-1	-1	-1	-1	-1	5	-1	-1	8	-1	-1	-1	-1	5	-1

Outline Architecture



General Structure

- There is 3 timers used in this state machine, all connected to the CCU4. The Minutes and Seconds timers are connected to an interrupt.
- There is a counter to count the High-Low transitions of a Digital IO App, which is high when any of the numeric pushbuttons are high.
- There is 4 numeric pushbutton Digital IO Apps, and a reset pushbutton Digital IO App, not connected to any other apps.
- There is 2 Digital IO Apps configured as outputs for the LED lights on the microcontroller, also not connected to any other apps.
- There is a PushButton_Pressed Digital IO app which acts as an input to the Counter mentioned above. The Counter can only take one input, but for this system there is 4 button inputs that need to be counted, so this extra Digital IO app enables the counter to increment. This is set to high after all input debouncing, when the system detects a high-low transition of an input (can be any of the numeric button input pins). This enables the counter to increment for each button press.

Complete List of Apps Used

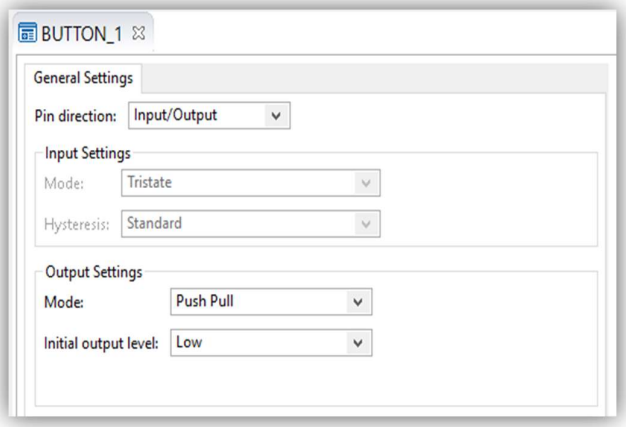
Resource Mapping | Pin Allocator | Signal Assignment | APPs

APP Instance Name	APP
BUTTON_1	DIGITAL_IO
BUTTON_2	DIGITAL_IO
BUTTON_3	DIGITAL_IO
BUTTON_4	DIGITAL_IO
BUTTON_RESET	DIGITAL_IO
CLOCK_XMC1_0	CLOCK_XMC1
CPU_CTRL_XMC1_0	CPU_CTRL_XMC1
DEBOUNCE_TIMER	TIMER
ERROR_LED	DIGITAL_IO
GLOBAL_CCU4_0	GLOBAL_CCU4
LOCK_LED	DIGITAL_IO
MINUTES_TIMER	TIMER
MINUTES_TIMER_INTERRUPT	INTERRUPT
PUSHBUTTON_COUNT	COUNTER
PUSHBUTTON_PRESSED	DIGITAL_IO
SECONDS_TIMER	TIMER
SECONDS_TIMER_INTERRUPT	INTERRUPT

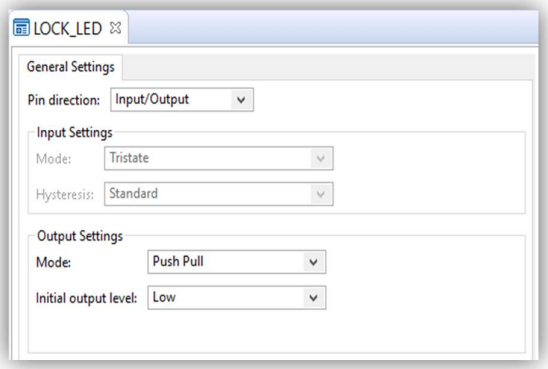
App Configuration

Digital_IO: Button_1/ Button_2/ Button_3/ Button_4/ Button_Reset

The Digital IO Apps for the listed Buttons are set up as Input/Outputs. This is to enable the pins to be set to High/Low. The Initial Output Level is set to Low, as the buttons are not being pressed upon initialisation.



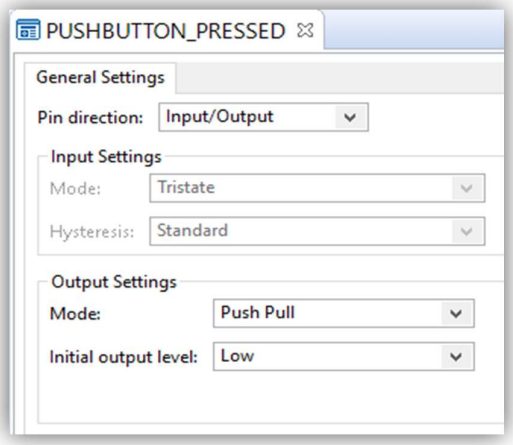
Digital_IO: Lock_LED/ Error_LED



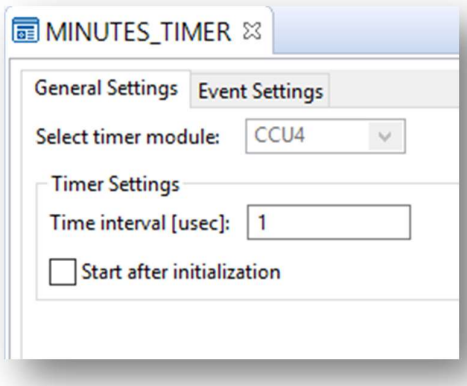
The Digital IO Apps for the listed LEDs are set up as Input/Outputs. This is to enable the pins to be set to High/Low. The Initial Output Level is set to Low for both LEDs, as the Lock is unlocked, and no errors have occurred upon initialisation.

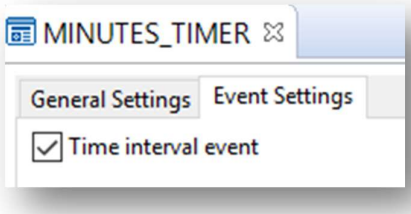
Digital_IO: Pushbutton_Pressed

The Digital IO App for Pushbutton_Pressed is set up as an Input/Output. This is to enable the pin to be set to High/Low. The Initial Output Level is set to Low, as the buttons are not being pressed upon initialisation. This acts as the input for the Counter app used to count numeric button presses.



Timer: Minutes_Timer

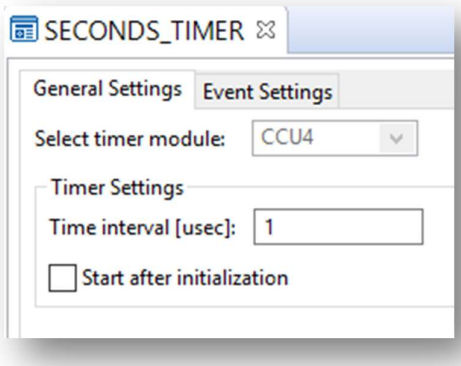


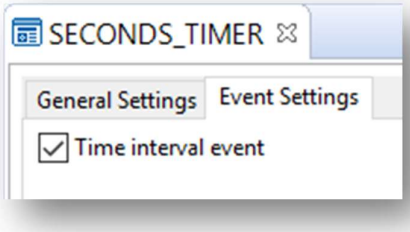


The Minutes_Timer is configured initially to have a time interval of 1µsec, because the interval is manually set in the code before each use. This timer has an event after each interval is reached. It is used to time the 5 minute lockdown interval when more than three incorrect combinations are entered, and also to time the 1 minute interval within which no more than 3 incorrect combinations can be input.

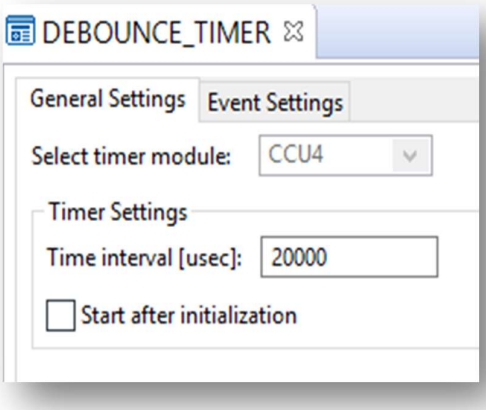
Timer: Seconds_Timer

The Seconds_Timer is configured initially to have a time interval of 1µsec, because the interval is manually set in the code before each use. This timer has an event after each interval is reached. It is used to time the 10 sec interval within which the user must enter a full 4-digit combination sequence, and also to time the 1 second interval used for the error LED flashes.





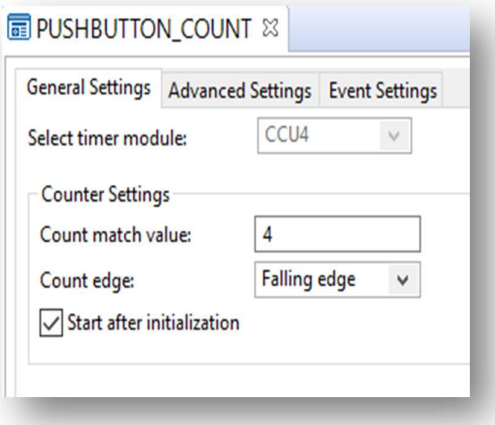
Timer: Debounce_Timer



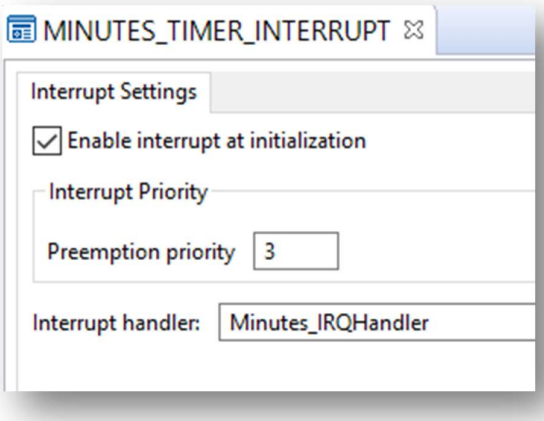
The Debounce_Timer is configured initially to have a time interval of 20000µsec, because the specified debounce time interval is 20ms. This timer does not have an event after the interval is reached. It is used to time an interval when a button input is set to high, so that any noise that transitions the pin input to high is filtered out. If the input is still high after the debounce time interval, then it is not noise being detected, the button is being pressed down, and the relevant actions for a high pushbutton pin input can take place.

Counter: PushButton_Count

The PushButton_Count is configured to count the falling edge of the Digital_IO PushButton_Pressed App, which is high when a numeric button is pressed. The count match value is 4, but the count match interrupt is not used in this app. It starts after initialisation as the lock is initially available to read user pushbutton inputs.



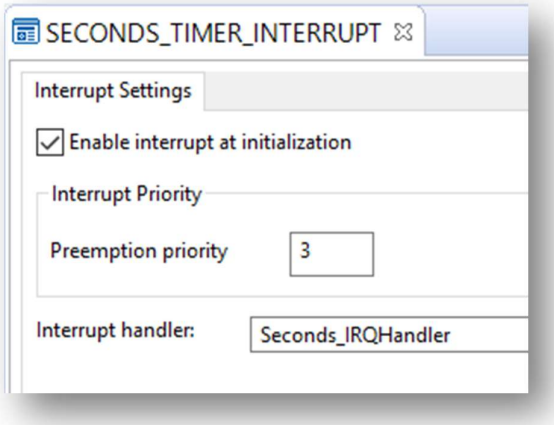
Interrupt: Minutes_Timer



The Minutes_Timer_Interrupt has an input from the Minutes_Timer output. This interrupt is enabled at initialisation, which means it can interrupt the program at any stage when the program is running, without being manually started in the code. The method that deals with this interrupt is the Minutes_IRQHandler.

Interrupt: Seconds_Timer

The Seconds_Timer_Interrupt has an input from the Seconds_Timer output. This interrupt is enabled at initialisation, which means it can interrupt the program at any stage when the program is running, without being manually started in the code. The method that deals with this interrupt is the Seconds_IRQHandler.



Pin Allocation

The table below shows the microcontroller pin allocation for each app in this state machine design.

APP Instance Name	Resource	Port-Pin
BUTTON_1		
	pin	(P0.5)
BUTTON_2		
	pin	(P0.6)
BUTTON_3		
	pin	(P0.7)
BUTTON_4		
	pin	(P0.8)
BUTTON_RESET		
	pin	(P0.9)
ERROR_LED		
	pin	(P1.1)
LOCK_LED		
	pin	(P1.0)
PUSHBUTTON_PRESSED		
	pin	(P0.12)

The ports assigned for the pushbutton inputs were any available ports on the microcontroller suitable for input/output functionality.

The ERROR_LED and LOCK_LED are assigned to ports P1.1 and P1.0, respectively. These ports are hardware assigned, as this is where the LED lights on the microcontroller are found.

The PUSHBUTTON_PRESSED app is assigned to port P0.12. This is the app acting as the input for the counter, which is set to high when any of the numeric button inputs are detected as high after a debounce period.

Source Code Files Appendix

File: Main.c

```
/*
 * main.c
 *
 *  Created on: 2017 Feb 28 23:59:24
 *  Author: Ciara Power
 */



#include <DAVE.h>                 //Declarations from DAVE Code Generation
(includes SFR declaration)

int32_t button1=0;     // buttons for GUI
int32_t button2=0;
int32_t button3=0;
int32_t button4=0;
int32_t reset=0;



int32_t count;      //variables for GUI
int32_t locked=0;   // either 1 for locked, or 0 for unlocked, used for GUI
bitmap of lock
int32_t resetValue=0;    // either 1 for in reset mode or 0 for not in
reset mode, used in GUI
int32_t errorCount=0;    //how many errors have occurred
int32_t errorFlashCount=0;   //how many flashes have occurred of the error
LED( ON/OFF is 2 "flashes")
int32_t thirtySecCount=0;  // how many 30 sec intervals have passed on
minutes_timer


int saved_combo[4]={1,2,3,4};   //the combination saved on lock
int entered_combo[4];   //user entry

     //table for events and next states
int nsTable[10][20]={{-1,2,-1,4,-1,-1,-1,9,-1,-1,-1,-1,6,5,-1},
                     {-1,2,-1,4,-1,-1,-1,9,-1,-1,-1,-1,6,5,-1},
                    {-1,-1,6,3,-1,-1,-1,-1,-1,-1,-1,-1,-1,5,-1},
                    {-1,-1,-1,-1,-1,-1,2,-1,-1,-1,-1,-1,-1,5,-1},
                    {-1,-1,-1,-1,0,-1,-1,-1,-1,-1,-1,-1,-1,5,1},
                    {0,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,-1,-1},
                    {0,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,5,-1},
                    {-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,5,-1,-1,-1,-1},
                    {0,-1,-1,-1,-1,-1,-1,-1,-1,-1,7,-1,1,-1,-1,-1},
                    {-1,-1,-1,-1,-1,5,-1,-1,8,-1,-1,-1,-1,5,-1}};
int currentState=0;
int event;
int entered=0;   // variable to indicate if user entry button value was
entered into entered_combo array
int saved=0; // variable to indicate if user entry button value was entered
into saved_combo array
int same=0;   // entered_combo and saved_combo comparison variable
int checked=0;   //if entered_combo was checked against saved_combo
int tenSecTimer=0;   // if tenSecTimer has completed this equals 1
```

```
/**

 * @brief main() - Application entry point
 *
 * <b>Details of function</b><br>
 * This routine is the application entry point. It is invoked by the device
startup code. It is responsible for
 * invoking the APP initialization dispatcher routine - DAVE_Init() and
hosting the place-holder for user application
 * code.
 */

int getEvent(){    //method to get the event number occurred
if (thirtySecCount<10 && errorFlashCount==8) return NULL;  // while in
state of user entry lock stay in same state

else if(thirtySecCount==10) return 10;   // 5 minutes has passed

// 4 errors and 5 min timer hasnt started yet
else if(errorCount==4 && thirtySecCount==0) return 9;

// 1 minute has passed and less than 4 errors
else if((thirtySecCount==2) && (errorCount!=4)) return 13;

else if(tenSecTimer==1) return 12;   // if 10 secs have passed

else if(DIGITAL_IO_GetInput(&BUTTON_1)==1 ||
DIGITAL_IO_GetInput(&BUTTON_2)==1 || DIGITAL_IO_GetInput(&BUTTON_3)==1 ||
DIGITAL_IO_GetInput(&BUTTON_4)==1) return 3;  //button pressed

else if(same==4) return 5;  //same combinations

//not same combinations after they've been checked
else if(same!=4 && checked==1) return 8;

// the user entry button value was put into saved_combo array
else if(saved==1) return 6;

// reset button was pressed
else if(DIGITAL_IO_GetInput(&BUTTON_RESET)==1) {
COUNTER_ResetCounter(&PUSHBUTTON_COUNT); return 1;}

else if((COUNTER_GetCurrentCount(&PUSHBUTTON_COUNT)==4) && (resetValue==1))
return 2;   // 4 buttons were entered while in reset mode

// user entry button value was entered into entered_combo array and
unlocked
else if(entered==1 && DIGITAL_IO_GetInput(&LOCK_LED)==0 ) return 4;

// user entry button value was entered into entered_combo array and locked
else if(entered==1 && DIGITAL_IO_GetInput(&LOCK_LED)==1) return 14;

// if 4 buttons were entered
else if(COUNTER_GetCurrentCount(&PUSHBUTTON_COUNT)==4) return 7;

// if no buttons entered and unlocked
else if(COUNTER_GetCurrentCount(&PUSHBUTTON_COUNT)==0 &&
DIGITAL_IO_GetInput(&LOCK_LED)==0) return 0;

// if no buttons entered and locked
```

```
else if(COUNTER_GetCurrentCount(&PUSHBUTTON_COUNT)==0 &&
DIGITAL_IO_GetInput(&LOCK_LED)==1) return 11;

else return NULL;    // if none of the above events occurred
}

//gets next state with the given state and event that has occurred
int getNextState(int currentState,int nsTable[10][20],int event){
if (nsTable[currentState][event] != -1)
return nsTable[currentState][event];
// state can transition to a next   state with the current event

else return currentState;
// current state doesnt have entry for current event so stay in current
state
}

void combosEqual()    //test if enetered_combo == saved_combo
{
      same=0;
      for(int i=0;i<4;i++){
         if(saved_combo[i]==entered_combo[i]){
            same++;
            //if combos are the same this variable will end up to be 4
          }
      }
      checked=1;  //combo was checked
}

void openIdleMode(){
      DIGITAL_IO_SetOutputLow(&LOCK_LED);   //unlocked
      resetValue=0;  // resetMode is off
      locked=0;  //unlocked
      entered=0;  //item not just entered into combo
      same=0;   // combos similarity reset to 0
}

void lockedIdleMode(){
  DIGITAL_IO_SetOutputHigh(&LOCK_LED); //locked
  locked=1;  //locked
      entered=0;  // item not just entered into combo
      resetValue=0; // resetMode is off
      same=0;   // combos similarity reset to 0
}

void resetMode(){
      resetValue=1;   //in reset mode
      saved=0;   // item was not just added into saved_combo
}

void savedAddMode(){
// digital io that will increment counter
DIGITAL_IO_SetOutputHigh(&PUSHBUTTON_PRESSED);
count=COUNTER_GetCurrentCount(&PUSHBUTTON_COUNT);// to display count on gui
      if(DIGITAL_IO_GetInput(&BUTTON_1)==1){
              saved_combo[count]=1;
        }
            else if(DIGITAL_IO_GetInput(&BUTTON_2)==1){
                  saved_combo[count]=2;
            }
            else if(DIGITAL_IO_GetInput(&BUTTON_3)==1){
```

```
                saved_combo[count]=3;
            }
            else if(DIGITAL_IO_GetInput(&BUTTON_4)==1){
                saved_combo[count]=4;
            }
            saved=1;  //item just added to saved_combo
}

void enteredAddMode(){
// digital io that will increment counter
DIGITAL_IO_SetOutputHigh(&PUSHBUTTON_PRESSED);
count=COUNTER_GetCurrentCount(&PUSHBUTTON_COUNT);// to display count on gui

// if its the first button pressed, start the 10 second timer for user
input
if(count==0){
            TIMER_SetTimeInterval(&SECONDS_TIMER,1000000000);
            TIMER_Start(&SECONDS_TIMER);
            }
            if(DIGITAL_IO_GetInput(&BUTTON_1)==1){
              entered_combo[count]=1;
            }
            else if(DIGITAL_IO_GetInput(&BUTTON_2)==1){
                entered_combo[count]=2;
            }
            else if(DIGITAL_IO_GetInput(&BUTTON_3)==1){
                entered_combo[count]=3;
            }
            else if(DIGITAL_IO_GetInput(&BUTTON_4)==1){
                entered_combo[count]=4;
            }
            entered=1;   // item just added to entered_combo
}

void resetEntriesMode(){
      COUNTER_ResetCounter(&PUSHBUTTON_COUNT);
      count=COUNTER_GetCurrentCount(&PUSHBUTTON_COUNT);

// reset the ten second timer variable indicating it has/hasnt completed
      tenSecTimer=0;

      if(same==4) // if combos were just checked and found the same
      {
            checked=0;   // reset to unchecked
      }
}

void resetErrorMode(){
      if (same==4){     // if combos were checked and are the same
            resetEntriesMode();     //reset entries
      }
            same=0;
            errorCount=0;
            errorFlashCount=0;

// the 1 minute timer is reset , so this variable counting 30 sec intervals
is reset too
            thirtySecCount=0;
}
```

```
void entryLockMode(){
// this is <8 when the errors entered has not reached 4 yet
      while(errorFlashCount<8);

// if no 30 secs have passed in minutes_timer
      if (thirtySecCount==0){
//start the timer for 30 secs
      TIMER_SetTimeInterval(&MINUTES_TIMER,3000000000);
      TIMER_Start(&MINUTES_TIMER);
      }
}

void errorMode(){
    checked=0;
    errorCount++;
    DIGITAL_IO_SetOutputHigh(&ERROR_LED);   // turn on error LED
    TIMER_SetTimeInterval(&SECONDS_TIMER,100000000); //start 1 sec timer
    TIMER_Start(&SECONDS_TIMER);
   if(errorCount==1){  // if first error, start the minute timer at 30 secs
      TIMER_SetTimeInterval(&MINUTES_TIMER,3000000000);
      TIMER_Start(&MINUTES_TIMER);
   }
}

void checkComboMode(){
      TIMER_Stop(&SECONDS_TIMER);    // stop the 10 sec user input timer
      TIMER_Clear(&SECONDS_TIMER);
      TIMER_ClearEvent(&SECONDS_TIMER);
      combosEqual();  //check if equal
      resetEntriesMode();  //reset entries
      if(same==4){  //if same
            DIGITAL_IO_ToggleOutput(&LOCK_LED);
      }
}


      void debounceTimer(void){
      TIMER_Start(&DEBOUNCE_TIMER);

     //stays here while timer is actively timing (20ms)
      while(!TIMER_GetInterruptStatus(&DEBOUNCE_TIMER));

      TIMER_ClearEvent(&DEBOUNCE_TIMER);
      TIMER_Clear(&DEBOUNCE_TIMER);
      }


void Minutes_IRQHandler(void){
// if less than 5 mins have passed OR  1 minute has passed AND total error
flashes havent occured

if(thirtySecCount>=10 || (thirtySecCount==2 && errorFlashCount!=8))){
   TIMER_Stop(&MINUTES_TIMER);
    TIMER_Clear(&MINUTES_TIMER);
            TIMER_ClearEvent(&MINUTES_TIMER);
}
else{
thirtySecCount++;
```

```
                if(thirtySecCount==2 && errorFlashCount!=8){
                      TIMER_Stop(&MINUTES_TIMER);
    TIMER_Clear(&MINUTES_TIMER);
            TIMER_ClearEvent(&MINUTES_TIMER);
            }
  }
}


      void Seconds_IRQHandler(void){
   // error light is on and not all erros have occured
            if(DIGITAL_IO_GetInput(&ERROR_LED)==1 && errorCount<4){
            TIMER_Stop(&SECONDS_TIMER);
            TIMER_Clear(&SECONDS_TIMER);
            TIMER_ClearEvent(&SECONDS_TIMER);
            errorFlashCount++;
            DIGITAL_IO_ToggleOutput(&ERROR_LED);
            }
            else if(errorCount==4){   //all errors have occured
            DIGITAL_IO_ToggleOutput(&ERROR_LED);
            errorFlashCount++;
            if (errorFlashCount==8){   // all error flashes have occured
            TIMER_Stop(&SECONDS_TIMER);
            TIMER_Clear(&SECONDS_TIMER);
            TIMER_ClearEvent(&SECONDS_TIMER);
                  }
            }
// 10 secs have passed ( nothing to do with error timing) so timer is
stopped
            else{
            TIMER_Stop(&SECONDS_TIMER);
            TIMER_Clear(&SECONDS_TIMER);
            TIMER_ClearEvent(&SECONDS_TIMER);
            tenSecTimer=1;    // variable for 10 secs set to 1
            }
      }

int main(void)
{
  DAVE_STATUS_t status;

  status = DAVE_Init();           /* Initialization of DAVE APPs  */

  if(status != DAVE_STATUS_SUCCESS)
  {
    /* Placeholder for error handler code. The while loop below can be
replaced with an user error handler. */
    XMC_DEBUG("DAVE APPs initialization failed\n");

    while(1U)
    {

    }
  }

  /* Placeholder for user application code. The while loop below can be
replaced with user application code. */


  while(1U)
```

```
  {  //check each gui button and set pin high for each if pressed, and
debounce
            if(button1==1){
                  DIGITAL_IO_SetOutputHigh(&BUTTON_1);
                  debounceTimer();
            }
            else if(button2==1){
                  DIGITAL_IO_SetOutputHigh(&BUTTON_2);
                  debounceTimer();
            }
            else if (button3==1 ){
                  DIGITAL_IO_SetOutputHigh(&BUTTON_3);
                  debounceTimer();
      }
            else if(button4==1){
                  DIGITAL_IO_SetOutputHigh(&BUTTON_4);
                  debounceTimer();
            }
   else if(reset==1){
             DIGITAL_IO_SetOutputHigh(&BUTTON_RESET);
             debounceTimer();
       }
       else{   //if no button on gui is being pressed, make sure all pins
are low and update count on gui
             DIGITAL_IO_SetOutputLow(&BUTTON_RESET);
             DIGITAL_IO_SetOutputLow(&BUTTON_1);
             DIGITAL_IO_SetOutputLow(&BUTTON_2);
             DIGITAL_IO_SetOutputLow(&BUTTON_3);
             DIGITAL_IO_SetOutputLow(&BUTTON_4);
             DIGITAL_IO_SetOutputLow(&PUSHBUTTON_PRESSED);
             count=COUNTER_GetCurrentCount(&PUSHBUTTON_COUNT);
       }


       event=getEvent();
       currentState=getNextState(currentState,nsTable,event);

       // jump to the next current state
       if(currentState==0) openIdleMode();
       else if(currentState==1) lockedIdleMode();
       else if(currentState==2) resetMode();
       else if(currentState==3) savedAddMode();
       else if(currentState==4) enteredAddMode();
       else if(currentState==5) resetErrorMode();
       else if(currentState==6) resetEntriesMode();
       else if(currentState==7) entryLockMode();
       else if(currentState==8) errorMode();
       else if(currentState==9) checkComboMode();

}
      }
```