

```

1  /*****
2  * @file      main.c
3  * @version   V3.00
4  * $Revision: 3 $
5  * $Date: 14/01/28 11:44a $
6  * @brief     M051 Series GPIO Driver Sample Code
7  *
8  * @note
9  * Copyright (C) 2013 Nuvoton Technology Corp. All rights reserved.
10 *****/
11 #include <stdio.h>
12 #include "M051Series.h"
13
14
15 #define PLL_CLOCK          50000000
16
17 volatile uint32_t g_au32TMRINTCount[1] = {0};
18 volatile uint32_t u32InitCount;
19
20
21 /**
22 * @brief      Port2/Port3/Port4 IRQ
23 *
24 * @param      None
25 *
26 * @return     None
27 *
28 * @details    The Port2/Port3/Port4 default IRQ, declared in startup_M051Series.s.
29 */
30 void GPIO_P2P3P4_IRQHandler(void)
31 {
32     //display=GPIO_GET_INT_FLAG(P2, BIT4);
33     /* To check if P4.5 interrupt occurred */
34     if(GPIO_GET_INT_FLAG(P2, BIT4))
35     {
36         GPIO_CLR_INT_FLAG(P2, BIT4);
37         printf("\n P2.4 INT occurred.\n");
38     }
39     else
40     {
41         /* Un-expected interrupt. Just clear all PORT2, PORT3 and PORT4 interrupts */
42         P2->ISRC = P2->ISRC;
43         P3->ISRC = P3->ISRC;
44         P4->ISRC = P4->ISRC;
45         printf("Un-expected interrupts.\n");
46     }
47 }
48
49 void TMR0_IRQHandler(void)
50 {
51     if(TIMER_GetIntFlag(TIMER0) == 1)
52     {
53         /* Clear Timer1 time-out interrupt flag */
54         TIMER_ClearIntFlag(TIMER0);
55         //P00 = !P00;
56         //printf("\n\nuart1: 115200\n");
57     }
58 }
59
60 /**
61 * @brief      Timer2 IRQ
62 *
63 * @param      None
64 *
65 * @return     None
66 *
67 * @details    The Timer1 default IRQ, declared in startup_M051Series.s.
68 */
69 volatile uint32_t capture_value[52][2];
70 volatile uint32_t capture_data[52];
71 volatile uint32_t capture_count=0;
72 volatile uint32_t capture_value_count=0;
73 volatile uint32_t capture_T1=0;
74 volatile uint32_t capture_T2=0;
75 volatile uint32_t capture_T3=0;
76 volatile uint32_t display = 0;
77 volatile uint32_t TIMER_CAPTURE_EDGE_flag=0;
78 volatile uint32_t u32AdcIntFlag=0;

```

```

78
79 void TMR2_IRQHandler(void)
80 {
81     if(TIMER_GetCaptureIntFlag(TIMER2) == 1)
82     {
83         /* Clear Timer2 capture interrupt flag */
84         TIMER_ClearCaptureIntFlag(TIMER2);
85
86         if(TIMER_CAPTURE_EDGE_flag==0) { TIMER_CAPTURE_EDGE_flag=1;TIMER_EnableCapture(TIMER2,
TIMER_CAPTURE_FREE_COUNTING_MODE, TIMER_CAPTURE_FALLING_EDGE);}
87         else { TIMER_CAPTURE_EDGE_flag=0;TIMER_EnableCapture(TIMER2,
TIMER_CAPTURE_FREE_COUNTING_MODE, TIMER_CAPTURE_RISING_EDGE); }
88
89
90         switch(capture_count-capture_count/3*3)
91         {
92             case 0:
93                 capture_T1 = TIMER_GetCaptureData(TIMER2);
94                 break;
95
96             case 1:
97                 capture_T2 = TIMER_GetCaptureData(TIMER2);
98                 break;
99
100             case 2:
101                 capture_T3 = TIMER_GetCaptureData(TIMER2);
102                 if(capture_value_count==1)
103                     display=0;
104                 if(capture_T2 > capture_T1)
105                 {
106                     capture_value[capture_value_count][0] = (capture_T2-capture_T1);
107                     capture_value[capture_value_count][0] = capture_value[capture_value_count][0]
/12;
108                 }
109                 else
110                 { capture_value[capture_value_count][0] = (0xffffffff+
capture_T2-capture_T1); //width}
111                 capture_value[capture_value_count][0] =
capture_value[capture_value_count][0]/12; //width}
112             }
113
114             if(capture_T3 > capture_T2 )
115             { capture_value[capture_value_count][1] = (capture_T3-capture_T2); //low
116               capture_value[capture_value_count][1] =
capture_value[capture_value_count][1]/12;
117             }
118             else
119             { capture_value[capture_value_count][1] = (0xffffffff+
capture_T3-capture_T2); //low
120               capture_value[capture_value_count][1] =
capture_value[capture_value_count][1]/12;
121             }
122
123
124             capture_value_count++;
125
126             capture_count++;
127             capture_T1 = capture_T3;
128
129             if(capture_value_count==51)
130                 capture_value_count=0;
131
132             break;
133
134             default:break;;
135         }
136
137         capture_count++;
138     }
139     // if(TIMER_GetIntFlag(TIMER2)==1)
140 }
141
142
143 //uint32_t g_u32AdcIntFlag=0;
144 volatile uint32_t NUM=0;
145 volatile uint32_t ConversionData[4];
146 volatile uint32_t i32ConversionData=0;

```

```
147 void ADC_IRQHandler(void)
148 {
149
150     ADC_CLR_INT_FLAG(ADC, ADC_ADF_INT); /* clear the A/D conversion flag */
151     ConversionData[NUM] = ADC_GET_CONVERSION_DATA(ADC, 4);
152     NUM++;
153     if(NUM==4)
154     { NUM=0;u32AdcIntFlag = 1;
155       i32ConversionData=(ConversionData[0]+ConversionData[1]+ConversionData[2]+ConversionData[3])>>2;
156     }
157 }
158
159 void GPIO_Init(void)
160 {
161     /* Configure P2.4 as Input mode and enable interrupt by FALLING edge trigger */
162     GPIO_SetMode(P2, BIT4, GPIO_PMD_INPUT);
163     GPIO_EnableInt(P2, 4, GPIO_INT_FALLING);
164     NVIC_EnableIRQ(GPIO_P2P3P4_IRQn);
165     GPIO_ENABLE_DEBOUNCE(P2, BIT4); //去抖
166     GPIO_SET_DEBOUNCE_TIME(GPIO_DBCLKSRC_LIRC, GPIO_DBCLKSEL_16); //16x0.1ms (10k)
167
168     /* Configure P0.0 as OUTPUT mode*/
169     GPIO_SetMode(P0, BIT0, GPIO_PMD_OUTPUT);
170
171
172     GPIO_SetMode(P4, BIT0, GPIO_PMD_INPUT); //t2ex
173     //GPIO_EnableInt(P2, 4, GPIO_INT_FALLING);
174     // NVIC_EnableIRQ(GPIO_P2P3P4_IRQn);
175     // GPIO_ENABLE_DEBOUNCE(P4, BIT0); //去抖
176     // GPIO_SET_DEBOUNCE_TIME(GPIO_DBCLKSRC_LIRC, GPIO_DBCLKSEL_16); //16x0.1ms (10k)
177
178
179
180     //GPIO_EnableInt(P2, 4, GPIO_INT_FALLING);
181     //NVIC_EnableIRQ(GPIO_P2P3P4_IRQn);
182     //GPIO_ENABLE_DEBOUNCE(P2, BIT4); //去抖
183     // GPIO_SET_DEBOUNCE_TIME(GPIO_DBCLKSRC_LIRC, GPIO_DBCLKSEL_16); //16x0.1ms (10k)
184
185 }
186
187
188 void ADC_Init(void)
189 {
190
191     CLK_EnableModuleClock(ADC_MODULE);
192
193     /* ADC clock source is 22.1184MHz, set divider to 7, ADC clock is 22.1184/7 MHz */
194     CLK_SetModuleClock(ADC_MODULE, CLK_CLKSEL1_ADC_S_HXT, CLK_CLKDIV_ADC(96));
195     //CLK_CLKSEL1_ADC_S_HIRC
196     // CLK_EnableModuleClock(ADC_MODULE);
197
198
199     /* Disable the P1.0 - P1.4 digital input path to avoid the leakage current */
200     GPIO_DISABLE_DIGITAL_PATH(P1, BIT4);
201     GPIO_SetMode(P1, BIT4, GPIO_PMD_INPUT);
202
203     /* Configure the P1.0 - P1.3 ADC analog input pins */
204     SYS->P1_MFP &= ~(SYS_MFP_P14_Msk);
205     SYS->P1_MFP |= SYS_MFP_P14_AIN4 ;
206
207     //SYS_ResetModule(ADC_RST);
208     ADC_SET_INPUT_CHANNEL(ADC, BIT4);
209
210     /* Set the ADC operation mode as CONTINUOUS, input mode as single-end and enable the analog
input channel 4 */
211     //ADC_Open(ADC, ADC_ADCR_DIFFEN_SINGLE_END, ADC_ADCR_ADMD_SINGLE, 0x1 << 4);
212     ADC_Open(ADC, ADC_ADCR_DIFFEN_SINGLE_END, ADC_ADCR_ADMD_CONTINUOUS, BIT4);
213     /* Power on ADC module */
214     ADC_POWER_ON(ADC);
215
216     /* clear the A/D interrupt flag for safe */
217     ADC_CLR_INT_FLAG(ADC, ADC_ADF_INT);
218     ADC_DisableHWTrigger(ADC);
219
220
221     /* Enable the ADC interrupt */
222     ADC_EnableInt(ADC, ADC_ADF_INT);
```

```

223         NVIC_EnableIRQ(ADC_IRQn);
224     }
225
226     void ADC_Process(void)
227     {
228
229         ADC_START_CONV(ADC);
230         while(1)
231         {
232
233             CLK_SysTickDelay(2000000);
234             // P00 = !P00;
235
236             /* Reset the ADC interrupt indicator and Start A/D conversion */
237             //u32AdcIntFlag = 0;
238             // ADC_START_CONV(ADC);
239             //ADC_DisableInt(ADC, ADC_ADF_INT);
240             /* Wait ADC interrupt (g_u32AdcIntFlag will be set at IRQ_Handler function)*/
241             if(u32AdcIntFlag == 1)
242             {
243
244                 /* Disable the ADC interrupt */
245                 //ADC_DisableInt(ADC, ADC_ADF_INT);
246                 u32AdcIntFlag = 0;
247                 /* Get the conversion result of the ADC channel 4 */
248                 i32ConversionData = ADC_GET_CONVERSION_DATA(ADC, 4);
249                 printf("Conversion result of channel 2: 0x%X (%d)\n\n", i32ConversionData,
4960*i32ConversionData/4095);
250                 P00 = !P00;
251             }
252
253         }
254
255     }
256
257
258
259     void SYS_Init(void)
260     {
261
262         /*-----*/
263         /* Init System Clock */
264         /*-----*/
265
266         /* Enable Internal RC 22.1184MHz clock */
267         CLK_EnableXtalRC(CLK_PWRCON_OSC22M_EN_Msk);
268
269         /* Waiting for Internal RC clock ready */
270         CLK_WaitClockReady(CLK_CLKSTATUS_OSC22M_STB_Msk);
271
272         /* Switch HCLK clock source to Internal RC and HCLK source divide 1 */
273         CLK_SetHCLK(CLK_CLKSEL0_HCLK_S_HIRC, CLK_CLKDIV_HCLK(1));
274
275         /* Enable external XTAL 12MHz clock */
276         CLK_EnableXtalRC(CLK_PWRCON_XTL12M_EN_Msk);
277
278         /* Waiting for external XTAL clock ready */
279         CLK_WaitClockReady(CLK_CLKSTATUS_XTL12M_STB_Msk);
280
281         /* Set core clock as PLL_CLOCK from PLL */
282         CLK_SetCoreClock(PLL_CLOCK);
283
284         /* Peripheral clock source */
285         CLK->CLKSEL1 = CLK_CLKSEL1_TMR2_S_HXT;
286
287         SystemCoreClockUpdate();
288
289     }
290
291     void UART1_Init(void)
292     {
293

```

```

/*-----*/
-----*/
294     /* Init
UART                                                                    */
295
/*-----*/
-----*/
296     /* Enable UART module clock */
297     CLK_EnableModuleClock(UART1_MODULE);
298
299     /* Select UART module clock source */
300     CLK_SetModuleClock(UART1_MODULE, CLK_CLKSEL1_UART_S_HXT,
CLK_CLKDIV_UART(1)); //CLK_CLKSEL1_UART_S_HXT或则CLK_CLKSEL1_UART_S_PLL
301
302     /* Set P1 multi-function pins for UART1 RXD and TXD */
303     SYS->P1_MFP &= ~(SYS_MFP_P13_Msk | SYS_MFP_P12_Msk);
304     SYS->P1_MFP |= (SYS_MFP_P12_RXD1 | SYS_MFP_P13_TXD1);
305
306     /* Reset UART */
307     SYS_ResetModule(UART1_RST);
308
309     /* Configure UART1 and set UART1 Baudrate */
310     UART_Open(UART1, 115200);
311
312 }
313
314
315 void Timer_Init(void)
316 {
317     /* Enable peripheral clock */
318     CLK_EnableModuleClock(TMR0_MODULE);
319
320     /* Open Timer0 frequency to 1 Hz in periodic mode, and enable interrupt */
321     TIMER_Open(TIMER0, TIMER_PERIODIC_MODE, 2000);
322
323     TIMER_EnableInt(TIMER0);
324
325     /* Enable Timer0 NVIC */
326     NVIC_EnableIRQ(TMR0_IRQn);
327
328     /* Start Timer0 counting */
329     TIMER_Start(TIMER0);
330
331
332     //init T2EX-----
333     //Init I/O Multi-function
334     SYS->P4_MFP &= ~(SYS_MFP_P40_Msk );
335     SYS->P4_MFP |= SYS_MFP_P40_T2EX;
336
337     /* Enable peripheral clock */
338     CLK_EnableModuleClock(TMR2_MODULE);
339
340     /* Initial Timer1 default setting */
341     TIMER_Open(TIMER2, TIMER_CONTINUOUS_MODE, 1);
342     //TIMER_Open(TIMER2, TIMER_PERIODIC_MODE, 1000);
343
344     /* Configure Timer1 setting for external counter input and capture function */
345     TIMER_SET_PRESCALE_VALUE(TIMER2, 0);
346     TIMER_SET_CMP_VALUE(TIMER2, 0xffffffff); //2*24次方/12000=1.3s
347     //TIMER_EnableEventCounter(TIMER2, TIMER_COUNTER_FALLING_EDGE);
348     TIMER_EnableCapture(TIMER2, TIMER_CAPTURE_FREE_COUNTING_MODE, TIMER_CAPTURE_RISING_EDGE);
349     TIMER_EnableCaptureInt(TIMER2);
350     //TIMER_EnableInt(TIMER2);
351     //TIMER_EnableCaptureDebounce(TIMER2);
352     /* Enable Timer1 NVIC */
353     NVIC_EnableIRQ(TMR2_IRQn);
354
355     /* Clear Timer2 interrupt counts to 0 */
356     //u32InitCount = g_au32TMRINTCount[1] = 0;
357
358     /* Start Timer2 counting */
359     TIMER_Start(TIMER2);
360
361 }
362
363 /*-----*/
-----*/

```

```
364  /* MAIN
function                                                                    */
365  /*-----*/
366  int main(void)
367  {
368  //    int32_t i32Err;
369
370      /* Unlock protected registers */
371      SYS_UnlockReg();
372
373      /* Init System, peripheral clock and multi-function I/O */
374      SYS_Init();
375
376      /* Lock protected registers */
377      SYS_LockReg();
378
379      /* Init UART0 for printf */
380      UART1_Init();
381
382      Timer_Init();
383
384      GPIO_Init();
385
386      ADC_Init();
387
388      printf("\n\nCPU @ %d Hz\n", SystemCoreClock);
389      printf("\n\nTimer0: Clock source 12 MHz; Periodic mode; Enable interrupt; 2 interrupt
tick/sec.\n");
390      printf("\n\nuart1: 115200\n");
391      //ADC_START_CONV(ADC);
392
393
394      //    printf("+-----+\n");
395      //    printf("|      P1.2(Output) and P4.1(Input) Sample Code      |\n");
396      //    printf("+-----+\n\n");
397
398      while(1)
399      {
400          ADC_Process();
401          //P00 = !P00;
402          //CLK_SysTickDelay(2000000);
403          //CLK_SysTickDelay(100000);
404          // CLK_SysTickDelay(100000);
405          // CLK_SysTickDelay(100000);
406          // CLK_SysTickDelay(100000);
407          // CLK_SysTickDelay(100000);
408          // CLK_SysTickDelay(100000);
409          // CLK_SysTickDelay(100000);
410          // CLK_SysTickDelay(100000);
411          // CLK_SysTickDelay(100000);
412
413
414
415
416      } //while(1)
417
418
419  }
420
421
422
423  /*** (C) COPYRIGHT 2013 Nuvoton Technology Corp. ***/
424
```