```c
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "LPC54114_cm4.h"
#include "fsl_debug_console.h"


/* FreeRTOS kernel includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"

#include "fsl_usart_freertos.h"
#include "fsl_usart.h"

#include "fsl_ctimer.h"
/* TODO: insert other definitions and declarations here. */

#define CTIMER CTIMER0                  /* Timer 0 */
#define LM0 kCTIMER_Match_0         //  J1[19]                  PWM Pin connected to left motor PIN1
#define RM0 kCTIMER_Match_1         //  J2[18]                  PWM Pin connected to left motor PIN2
#define LM1 kCTIMER_Match_2         //  J1[16]                  PWM Pin connected to right motor PIN1
#define RM1 kCTIMER_Match_0         //  J2[17]                  PWM Pin connected to right motor PIN2


#define DEMO_USART USART0
#define DEMO_USART_IRQHandler FLEXCOMM0_IRQHandler
#define DEMO_USART_IRQn FLEXCOMM0_IRQn


/* Task priorities. */
#define uart_task_PRIORITY (configMAX_PRIORITIES - 1)
#define USART_NVIC_PRIO 5

static void uart_task(void *pvParameters);              //Task responsible for receiving data from beaglebone
static void Drive_task(void *pvParameters);             //This Task is used to drive motor
static void Ultrasonic_Task(void *pvParameters);        //This Task associate with Ultarsonic Sensor to avoid obstracle
static void Object_Search();                            //This Task used to search teh object


void MotorsSetup();                                     // It set up PWM to drive motors
void Move();                                            // It is used to move in forward direction
void Turn_SlowLeft();                                   // It is used to turn left with slow speed
void Turn_SlowRight();                                  // It is used to turn right with slow speed
void Turn_Left();                                       // It is used to turn left
```

```
52 void Turn_Right();                                    // It is used to turn right
53 void Stop();                                         // It is used to Stop robot
54 void Reverse();                                      // It is used to reverse the robot
55 float Front_Obstarcle();                              // It gives the distance of front
   obstacle
56 float Rear_Obstarcle();                               // It gives the distance of rear obstacle
57 void Search();                                       // It drive the motors to search object
58 void Circle();                                       // It drive the motors to make circle
59
60
61 uint8_t background_buffer[100];                       // For receiving data from Beaglebone in
   background
62 uint8_t recv_buffer[1];
63
64 usart_rtos_handle_t handle;                           // USART handle
65 struct _usart_handle t_handle;
66
67
68 /*This structure contain the configurations of USART
69  * USART runs at 9600 baudrate 8N1
70  */
71
72 struct rtos_usart_config usart_config = {
73     .baudrate    = 9600,
74     .parity      = kUSART_ParityDisabled,
75     .stopbits    = kUSART_OneStopBit,
76     .buffer      = background_buffer,
77     .buffer_size = sizeof(background_buffer),
78 };
79
80 /* Queue Handle  */
81
82
83 xQueueHandle Obj_track= NULL;
84 xQueueHandle queue1= NULL;
85
86 /* Task Handles   */
87
88 TaskHandle_t Uart_Task_Handle=NULL;
89 TaskHandle_t Ultrasonic_Task_Handle=NULL;
90 TaskHandle_t Drive_task_Handle=NULL;
91 TaskHandle_t Object_Search_Handle=NULL;
92
93
94 int main(void)
95
96 {
97
98
99
100     CLOCK_AttachClk(BOARD_DEBUG_UART_CLK_ATTACH);
101     SYSCON->ASYNCAPBCTRL = 1;
102
103
104
105     BOARD_InitBootPins();
106     BOARD_InitBootClocks();
107     BOARD_InitBootPeripherals();
108
109
```

```
110     MotorsSetup();
111 
112     /* Creating Queues for InterTask communication */
113 
114     queue1=xQueueCreate(1,sizeof(uint8_t));
115     Obj_track=xQueueCreate(1,sizeof(uint8_t));
116 
117 
118     /* Creating Tasks for rtos */
119 
120     if (xTaskCreate(uart_task, "Uart_task", configMINIMAL_STACK_SIZE + 10, NULL, 3
121             ,&Uart_Task_Handle) != pdPASS)
122         {
123             PRINTF("Task creation failed!.\r\n");
124             while (1)
125                 ;
126         }
127 
128 
129     if (xTaskCreate(Drive_task, "Robot_driving_task", configMINIMAL_STACK_SIZE + 10, NULL,
    2,&Drive_task_Handle) != pdPASS)
130                 {
131                     PRINTF("Task creation failed!.\r\n");
132                     while (1)
133                         ;
134                 }
135 
136     if (xTaskCreate(Ultrasonic_Task, "Ultrasonic_Task", configMINIMAL_STACK_SIZE + 10, NULL,
    4,&Ultrasonic_Task_Handle) != pdPASS)
137                             {
138                                 PRINTF("Task creation failed!.\r\n");
139                                 while (1)
140                                     ;
141                             }
142     if (xTaskCreate(Object_Search, "Object_Search", configMINIMAL_STACK_SIZE + 10, NULL,
    0,&Object_Search_Handle) != pdPASS)
143                     {
144                         PRINTF("Task creation failed!.\r\n");
145                         while (1);
146 
147 
148                 }
149 
150 
151 
152         vTaskStartScheduler();
153         for (;;)
154             ;
155     }
156 
157 
158 
159 
160 
161 static void uart_task(void *pvParameters)
162  {
163 
164     vTaskSuspend(Object_Search_Handle);
165     int error,status=0;
166     uint8_t send;
```

```
167     size_t n            = 0;
168     usart_config.srcclk = BOARD_DEBUG_UART_CLK_FREQ;
169     usart_config.base   = DEMO_USART;
170
171     NVIC_SetPriority(DEMO_USART_IRQn, USART_NVIC_PRIO);
172
173
174
175     USART_RTOS_Init(&handle, &t_handle, &usart_config);
176
177
178     while(1)
179     {
180         /* Receive the data form USART */
181
182         error=USART_RTOS_Receive(&handle, recv_buffer, sizeof(recv_buffer), &n);
183         //printf("%c",recv_buffer);
184         if (error == kStatus_USART_RxRingBufferOverrun)
185                {
186             printf("Buffer overrun");
187
188                }
189
190
191         if (n > 0)
192         {
193             send=recv_buffer[0];
194
195             if(send=='F')
196            {
197                 //send1=recv_buffer[0];
198                 if(status==0)
199                 {
200                 vTaskSuspend(Drive_task_Handle);
201                 vTaskResume(Object_Search_Handle);
202                 //printf("drive suspend\n");
203                 //printf("search resume\n");
204                 status=1;
205                 }
206                 xQueueSend(Obj_track,&send,10);
207
208
209            }
210              else if(send=='S')
211               {
212                  Stop();
213                 vTaskSuspend(Object_Search_Handle);
214
215               }
216
217               else
218                {
219                   if(status==1)
220                   {
221                     vTaskSuspend(Object_Search_Handle);
222                     vTaskResume(Drive_task_Handle);
223                     //printf("drive resume\n");
224                     //printf("search suspend\n");
225                       status=0;
226
```

```
227                 }
228
229               xQueueSend(queue1,&send,10);
230              }
231
232
233
234           //printf("%c",recv_buffer);
235         }
236     }
237
238 }
239
240
241
242 static void Drive_task(void *pvParameters)
243 {
244     uint8_t recv;
245
246        while(1){
247            xQueueReceive(queue1,&recv,10);
248            if(recv=='M')
249            {
250                Move(90);
251                //printf("moving\n");
252                recv='n';
253            }
254            else if(recv=='L')
255            {
256                Turn_Left();
257                //printf("left\n");
258                recv='n';
259            }
260            else if(recv=='R')
261            {
262                Turn_Right();
263                //printf("right\n");
264                recv='n';
265            }
266            else if(recv=='S')
267            {
268                Stop();
269                //printf("stop");
270                recv='n';
271
272            }
273            else if(recv=='B')
274            {
275                Stop();
276                vTaskDelay(5);
277                 Reverse();
278                // printf("reverse\n");
279                recv='n';
280            }
281
282            else if(recv=='l')
283            {
284                Turn_SlowLeft();
285                // printf("reverse\n");
286                 recv='n';
```

```
                        }
            else if(recv=='r')
            {
                Turn_SlowRight();
                // printf("reverse\n");
                 recv='n';
            }
            else if(recv=='F')
            {
                xQueueSend(Obj_track,&recv,10);
                recv='n';
            }
            }
}


static void Ultrasonic_Task(void *pvParameters)
{
   float Front_obs,Rear_obs;


    while(1)


   {
        Front_obs=Front_Obstarcle();
        Rear_obs=Rear_Obstarcle();
        if(Front_obs<10)
        {  Stop();
            vTaskSuspend(Uart_Task_Handle);
            vTaskSuspend(Drive_task_Handle);
            vTaskSuspend(Object_Search_Handle);

            vTaskDelay(10);
            Reverse();
            vTaskDelay(150);
            /*Turn_Left();
            vTaskDelay(200);
            Turn_Right();
            vTaskDelay(200);*/
            Stop();
            vTaskResume(Uart_Task_Handle);
            vTaskResume(Drive_task_Handle);
            vTaskResume(Object_Search_Handle);
           }

        if(Rear_obs<10)
        {  Stop();
            vTaskSuspend(Uart_Task_Handle);
            vTaskSuspend(Object_Search_Handle);
            vTaskResume(Drive_task_Handle);
            vTaskDelay(5);
            Move(90);
            vTaskDelay(150);
           // Turn_Left();
            //vTaskDelay(200);
            //Move();
            //vTaskDelay(80);
            //Turn_Right();
```

```
347             //vTaskDelay(100);
348             Stop();
349             vTaskResume(Uart_Task_Handle);
350             vTaskResume(Object_Search_Handle);
351             vTaskResume(Drive_task_Handle);
352            }
353
354
355
356
357
358
359    }
360
361 }
362
363
364 static void Object_Search(void *pvParameters)
365 {
366     uint8_t Obj_recv;
367    while(1)
368    {
369        printf("finding\n");
370    xQueueReceive(Obj_track,&Obj_recv,10);
371    if(Obj_recv=='F')
372    {
373    Circle();
374    Move(75);
375    vTaskDelay(300);
376    Search();
377    Stop();
378
379
380    }
381    Obj_recv='n';
382 }}
383
384
385
386
387
388
389 void MotorsSetup()
390 {
391            ctimer_config_t config;
392            uint32_t srcClock_Hz;
393            srcClock_Hz = CLOCK_GetFreq(kCLOCK_BusClk);
394
395
396
397            CTIMER_GetDefaultConfig(&config);
398
399
400            CTIMER_Init(CTIMER, &config);
401            CTIMER_Init(CTIMER1, &config);
402            CTIMER_Init(CTIMER2, &config);
403            CTIMER_Init(CTIMER3, &config);
404
405            CTIMER_SetupPwm(CTIMER,LM0,0,20000,srcClock_Hz,NULL);
406            CTIMER_SetupPwm(CTIMER,LM1,0,20000,srcClock_Hz,NULL);
```

```
407             CTIMER_SetupPwm(CTIMER,RM0,0,20000,srcClock_Hz,NULL);
408             CTIMER_SetupPwm(CTIMER1,RM1,0,20000,srcClock_Hz,NULL);
409             CTIMER_StartTimer(CTIMER);
410             CTIMER_StartTimer(CTIMER1);
411 }
412
413
414 void Move(int speed)
415 {
416        CTIMER_UpdatePwmDutycycle(CTIMER, LM0, speed);
417        CTIMER_UpdatePwmDutycycle(CTIMER, LM1, 0);
418        CTIMER_UpdatePwmDutycycle(CTIMER, RM0, speed);
419        CTIMER_UpdatePwmDutycycle(CTIMER1, RM1, 0);
420
421
422 }
423
424
425 void Turn_SlowLeft()
426 {
427        CTIMER_UpdatePwmDutycycle(CTIMER, LM0, 0);
428        CTIMER_UpdatePwmDutycycle(CTIMER, LM1, 0);
429        CTIMER_UpdatePwmDutycycle(CTIMER, RM0, 70);
430        CTIMER_UpdatePwmDutycycle(CTIMER1, RM1, 0);
431
432 }
433
434
435 void Turn_SlowRight()
436 {
437        CTIMER_UpdatePwmDutycycle(CTIMER, LM0, 70);
438        CTIMER_UpdatePwmDutycycle(CTIMER, LM1, 0);
439        CTIMER_UpdatePwmDutycycle(CTIMER, RM0, 0);
440        CTIMER_UpdatePwmDutycycle(CTIMER1, RM1,0);
441 }
442
443
444 void Turn_Left()
445 {
446        CTIMER_UpdatePwmDutycycle(CTIMER, LM0, 0);
447        CTIMER_UpdatePwmDutycycle(CTIMER, LM1, 0);
448        CTIMER_UpdatePwmDutycycle(CTIMER, RM0,90);
449        CTIMER_UpdatePwmDutycycle(CTIMER1, RM1,0);
450 }
451
452
453
454 void Turn_Right()
455
456 {
457        CTIMER_UpdatePwmDutycycle(CTIMER, LM0, 90);
458        CTIMER_UpdatePwmDutycycle(CTIMER, LM1, 0);
459        CTIMER_UpdatePwmDutycycle(CTIMER, RM0, 0);
460        CTIMER_UpdatePwmDutycycle(CTIMER1, RM1,0);
461
462 }
463
464
465
466 void Stop()
```

```
{
       CTIMER_UpdatePwmDutycycle(CTIMER, LM0,0);
       CTIMER_UpdatePwmDutycycle(CTIMER, LM1,0);
       CTIMER_UpdatePwmDutycycle(CTIMER, RM0,0);
       CTIMER_UpdatePwmDutycycle(CTIMER1,RM1,0);
}



void Reverse()
{
       CTIMER_UpdatePwmDutycycle(CTIMER, LM0, 0);
       CTIMER_UpdatePwmDutycycle(CTIMER, LM1, 80);
       CTIMER_UpdatePwmDutycycle(CTIMER, RM0, 0);
       CTIMER_UpdatePwmDutycycle(CTIMER1,RM1, 80);

}



void Search()
{
    CTIMER_UpdatePwmDutycycle(CTIMER, LM0, 75);
    CTIMER_UpdatePwmDutycycle(CTIMER, LM1, 0);
    CTIMER_UpdatePwmDutycycle(CTIMER, RM0, 0);
    CTIMER_UpdatePwmDutycycle(CTIMER1,RM1, 75);
    vTaskDelay(750);

}
void Circle()
{
   CTIMER_UpdatePwmDutycycle(CTIMER, LM0, 90);
   CTIMER_UpdatePwmDutycycle(CTIMER, LM1, 0);
   CTIMER_UpdatePwmDutycycle(CTIMER, RM0, 70);
   CTIMER_UpdatePwmDutycycle(CTIMER1,RM1, 0);
   vTaskDelay(1000);

}




float Front_Obstarcle()
{

       float Front_time,Front_distance;
       GPIO_PinWrite(BOARD_Front_trig_GPIO,BOARD_Front_trig_PORT,BOARD_Front_trig_PIN,1);
       vTaskDelay(10);
       GPIO_PinWrite(BOARD_Front_trig_GPIO,BOARD_Front_trig_PORT,BOARD_Front_trig_PIN,0);

       while(GPIO_PinRead(BOARD_Front_echo_GPIO,BOARD_Front_echo_PORT,BOARD_Front_echo_PIN)==0);

       CTIMER_StartTimer(CTIMER2);

       while(GPIO_PinRead(BOARD_Front_echo_GPIO,BOARD_Front_echo_PORT,BOARD_Front_echo_PIN)==1);
       CTIMER_StopTimer(CTIMER2);

       Front_time= CTIMER_GetTimerCountValue(CTIMER2);

```

```
527         Front_distance =(0.0343*(Front_time/96))/2;
528 
529         CTIMER_Reset(CTIMER2);
530 
531         return Front_distance;
532 
533 }
534 
535 
536 float Rear_Obstarcle()
537 {       float Rear_time,Rear_distance;
538 
539         GPIO_PinWrite(BOARD_Rear_trig_GPIO,BOARD_Rear_trig_PORT,BOARD_Rear_trig_PIN,1);
540         vTaskDelay(10);
541         GPIO_PinWrite(BOARD_Rear_trig_GPIO,BOARD_Rear_trig_PORT,BOARD_Rear_trig_PIN,0);
542 
543         while(GPIO_PinRead(BOARD_Rear_echo_GPIO,BOARD_Rear_echo_PORT,BOARD_Rear_echo_PIN)==0);
544 
545         CTIMER_StartTimer(CTIMER3);
546 
547         while(GPIO_PinRead(BOARD_Rear_echo_GPIO,BOARD_Rear_echo_PORT,BOARD_Rear_echo_PIN)==1);
548         CTIMER_StopTimer(CTIMER3);
549 
550         Rear_time= CTIMER_GetTimerCountValue(CTIMER3);
551         Rear_time=Rear_time/96;
552         Rear_distance =(0.0343*Rear_time)/2;
553 
554         CTIMER_Reset(CTIMER3);
555 
556 
557         return Rear_distance;
558 }
559 
```