```
 1 /*
 2  *  Nickolas Schiffer
 3  *  CMPE 146 S19
 4  *  Lab: UART: part0
 5  */
 6
 7 /**
 8  * @file
 9  * @brief This is the application entry point.
10  */
11
12 #include <stdio.h>
13 #include <LPC17xx.h>
14 #include "utilities.h"
15 #include <tasks.hpp>
16 #include <switches.hpp>
17 #include <LED_Display.hpp>
18 #include <UART/UART_0_1_2_3.hpp>
19
20
21 #define LOW  false
22 #define HIGH true
23
24 enum operation {
25     add,
26     sub,
27     mult
28 };
29 uint8_t dig1 = 0;
30 uint8_t dig2 = 0;
31 operation op = add;
32
33
34 SemaphoreHandle_t rx_sem = NULL;
35
36 void vALU(void *pvParameters){
37     auto led = LED_Display::getInstance();
38     auto uart3 = LabUart();
39     uart3.Initialize(LabUart::U3, LabUart::b38400, true, LabUart::f8_bit);
40     uint8_t d1, d2, result = 0;
41     operation op_recv;
42     while(1){
43         d1 = (uint8_t)uart3.Receive();
44         d2 = (uint8_t)uart3.Receive();
45         op_recv = (operation)uart3.Receive();
46         printf("ALU recv d1: %u, d2: %u, op: %u\n", d1, d2, op_recv);
47         switch(op_recv){
```

```
48              case add:
49                  result = d1 + d2;
50                  break;
51              case sub:
52                  if (d1 < d2){
53                      printf("no negative results please\n");
54                      result = 0;
55                      break;
56                  }
57                  result = d1 - d2;
58                  break;
59              case mult:
60                  result = d1 * d2;
61                  break;
62          }
63          led.setNumber(result);
64          uart3.Transmit((char)(result / 10));
65          uart3.Transmit((char)(result % 10));
66          printf("ALU result: %u\n", result);
67
68      }
69  }
70
71  void vControl_Unit(void *pvParameters){
72      auto led = LED_Display::getInstance();
73
74      LabUart uart2 = LabUart();
75      uart2.Initialize(LabUart::U2, LabUart::b38400, true, LabUart::f8_bit);
76      uint8_t result1, result2;
77      while(1){
78          if (xSemaphoreTake(rx_sem, portMAX_DELAY)){
79              printf("started\n");
80              uart2.Transmit((char)dig1);
81              uart2.Transmit((char)dig2);
82              uart2.Transmit((char)op);
83
84              result1 = (uint8_t)uart2.Receive();
85              result2 = (uint8_t)uart2.Receive();
86              printf("received: %d, %d\n", result1, result2);
87              led.setNumber((result1 * 10) + result2);
88              printf("CU Result %d\n",(result1 * 10) + result2);
89              vTaskDelay(500);
90          }
91
92      }
93  }
94
```

```
 95 void vInterfaceControl(void *pvParamters){
 96     auto sw = Switches::getInstance();
 97     auto led = LED_Display::getInstance();
 98     led.init();
 99     led.clear();
100     sw.init();
101     bool sw_prev_states[4] = {HIGH};
102     bool sw_current_states[4] = {HIGH};
103 
104     while (1){
105         for (int i = 0; i < 4; i++){
106             sw_prev_states[i] = sw_current_states[i];
107             sw_current_states[i] = sw.getSwitch(i + 1);
108             if ((sw_current_states[i] == HIGH) && (sw_prev_states[i] == LOW)){
109                 switch(i){
110                     case 0:
111                         dig1 = (dig1 + 1) % 10;
112                         printf("dig1: %d\n",dig1);
113                         led.setNumber((dig1 * 10) + dig2);
114                         break;
115                     case 1:
116                         dig2 = (dig2 + 1) % 10;
117                         printf("dig2: %d\n",dig2);
118                         led.setNumber((dig1 * 10) + dig2);
119                         break;
120                     case 2:
121                         switch(op){
122                             case add:
123                                 op = sub;
124                                 break;
125                             case sub:
126                                 op = mult;
127                                 break;
128                             case mult:
129                                 op = add;
130                                 break;
131                         }
132                         printf("op: %d\n",op);
133                         break;
134                     case 3:
135                         xSemaphoreGive(rx_sem);
136                         vTaskDelay(500);
137                         break;
138 
139                 }
140             }
141         }
```

```
142     }
143 }
144
145 int main(){
146
147     rx_sem = xSemaphoreCreateBinary();
148
149     scheduler_add_task(new terminalTask(PRIORITY_HIGH));
150     xTaskCreate(vInterfaceControl,"InterfaceControl",1000, NULL,PRIORITY_LOW,
    NULL);
151     xTaskCreate(vControl_Unit,"CU",1000, NULL,PRIORITY_LOW, NULL);
152     xTaskCreate(vALU,"ALU",1000, NULL,PRIORITY_LOW, NULL);
153
154     scheduler_start();
155     return -1;
156 }
157
```

```
 1 /*
 2  * UART_0_1_2_3.hpp
 3  *
 4  *  Created on: Mar 9, 2019
 5  *      Author: nickschiffer (nick@schiffer.us)
 6  */
 7
 8 #ifndef UART_0_1_2_3_HPP_
 9 #define UART_0_1_2_3_HPP_
10
11 #include <LPC17xx.h>
12 #include <char_dev.hpp>
13
14 #define QUEUE_SIZE 100
15 #define ENQUEUE_TIMEOUT 1
16 #define DEQUEUE_TIMEOUT portMAX_DELAY
17
18
19 class LabUart// : public CharDev
20 {
21  public:
22     enum UART_Device {
23         U0,
24         U1,
25         U2,
26         U3
27     };
28
29     enum BAUD_Rate {
30         b600    = 600,
31         b1200   = 1200,
32         b2400   = 2400,
33         b4800   = 4800,
34         b9600   = 9600,
35         b14400  = 14400,
36         b19200  = 19200,
37         b38400  = 38400,
38         b56000  = 56000,
39         b57600  = 57600,
40         b115200 = 115200
41     };
42
43     enum Frame_Size {
44         f5_bit,
45         f6_bit,
46         f7_bit,
47         f8_bit
```

```
48      };
49
50      enum Stop_Bit {
51          s1_bit,
52          s2_bit
53      };
54
55      enum Parity_Mode {
56          pNone,
57          pOdd,
58          pEven,
59          pForced_1,
60          pForced_2
61      };
62
63      enum UART_Init_Result{
64          Success,
65          Invalid_Parity_Config,
66          No_Device_Initialized,
67          Unspecified_Error
68      };
69
70      LabUart(){};
71      ~LabUart();
72
73      /*
74       * Initializes UARTn interface.
75       * @param device is the UART_Device enumeration of the desired device.
76       * @param baud_rate is the BAUD_Rate enumeration of the desired BAUD rate.
77       * @param rx_interrupt determines whether an interrupt will occur upon RX.
78       * @param frame_size is the Frame_Size enumeration of the desired frame
    size.
79       * @param break_control determines whether or not break control will
    occur.
80       * @param parity_enable determines whether parity bits will be generated.
81       * @param parity_select is the Parity_Select enumeration for desired
    parity behavior.
82       * @return returns UART_Init_Result::Success if Initialization is
    successful,
83       *     other UART_Init_Result status if Initialization has failed
84       */
85      int Initialize(UART_Device device = U2, BAUD_Rate baud_rate = b38400,
86              bool rx_interrupt_enable = true, Frame_Size frame_size = f8_bit,
    Stop_Bit stop_bit = s1_bit,
87              bool break_control = false, bool parity_enable = false,
    Parity_Mode parity_mode = pNone);
88      /*
```

```
 89      * Transmits character on initialized device
 90      * @param c char to be transmitted
 91      * @return returns 1 if successful, 0 if failed
 92      */
 93     int Transmit(char c);
 94
 95     /*
 96      * Returns next char in RX FIFO
 97      * If RX interrupt is not enabled, this will block until a character is
   available
 98      * @return oldest char in RX FIFO
 99      */
100     char Receive();
101
102
103
104
105
106
107  private:
108     UART_Device this_device = U2;
109     bool initialized = false;
110     bool intr_enabled = false;
111
112     static void uart0_rx_intr();
113     static void uart1_rx_intr();
114     static void uart2_rx_intr();
115     static void uart3_rx_intr();
116
117     static QueueHandle_t u0_rx_queue;
118     static QueueHandle_t u1_rx_queue;
119     static QueueHandle_t u2_rx_queue;
120     static QueueHandle_t u3_rx_queue;
121
122
123 };
124
125
126
127 #endif /* UART_2_3_HPP_ */
128
```

```cpp
/*
 * UART_0_1_2_3.cpp
 *
 *  Created on: Mar 9, 2019
 *      Author: nickschiffer (nick@schiffer.us)
 */

#include <UART/UART_0_1_2_3.hpp>

QueueHandle_t LabUart::u0_rx_queue;
QueueHandle_t LabUart::u1_rx_queue;
QueueHandle_t LabUart::u2_rx_queue;
QueueHandle_t LabUart::u3_rx_queue;

LabUart::~LabUart()
{
    //TODO disable and cleanup device
}

int LabUart::Initialize(UART_Device device, BAUD_Rate baud_rate,
        bool rx_interrupt_enable, Frame_Size frame_size, Stop_Bit stop_bit,
  bool break_control,
        bool parity_enable, Parity_Mode parity_mode)
{

    switch (device){
        case U0:
        case U1:
        case U2:
        case U3:{
            /*
             * Power on UART Device
             */
            switch(device){
                case U0:
                    LPC_SC->PCONP |= (1 << pconp_uart0);
                    break;
                case U1:
                    LPC_SC->PCONP |= (1 << pconp_uart1);
                    break;
                case U2:
                    LPC_SC->PCONP |= (1 << pconp_uart2);
                    break;
                case U3:
                    LPC_SC->PCONP |= (1 << pconp_uart3);
                    break;
            }
```

```
47
48              /*
49               * Select Peripheral Clock Divider
50               * Select 01 for CCLK/1
51               */
52                  switch(device){
53                      case U0:
54                          LPC_SC->PCLKSEL0 &= ~(1 << 7);
55                          LPC_SC->PCLKSEL0 |=  (1 << 6);
56                          break;
57                      case U1:
58                          LPC_SC->PCLKSEL0 &= ~(1 << 9);
59                          LPC_SC->PCLKSEL0 |=  (1 << 8);
60                          break;
61                      case U2:
62                          LPC_SC->PCLKSEL1 &= ~(1 << 17);
63                          LPC_SC->PCLKSEL1 |=  (1 << 16);
64                          break;
65                      case U3:
66                          LPC_SC->PCLKSEL1 &= ~(1 << 19);
67                          LPC_SC->PCLKSEL1 |=  (1 << 18);
68                          break;
69                  }
70
71              /*
72               * Select TX and RX pins
73               */
74                  switch(device){
75                      case U0:
76                          LPC_PINCON->PINSEL0 |=  (1 << 4);
77                          LPC_PINCON->PINSEL0 &= ~(1 << 5);
78                          LPC_PINCON->PINSEL0 |=  (1 << 6);
79                          LPC_PINCON->PINSEL0 &= ~(1 << 7);
80                          break;
81                      case U1:
82                          LPC_PINCON->PINSEL4 |=  (1 << 1);
83                          LPC_PINCON->PINSEL4 &= ~(1 << 0);
84                          LPC_PINCON->PINSEL4 |=  (1 << 3);
85                          LPC_PINCON->PINSEL4 &= ~(1 << 2);
86                          break;
87                      case U2:
88                          LPC_PINCON->PINSEL4 |=  (1 << 17);
89                          LPC_PINCON->PINSEL4 &= ~(1 << 16);
90                          LPC_PINCON->PINSEL4 |=  (1 << 19);
91                          LPC_PINCON->PINSEL4 &= ~(1 << 18);
92                          break;
93                      case U3:
```

```
 94                      LPC_PINCON->PINSEL9 |=  (0b11 << 24);
 95                      LPC_PINCON->PINSEL9 |=  (0b11 << 26);
 96                      break;
 97              }
 98          /*
 99           * Enable pullup on TX pin (not recommended on RX pin)
100           */
101              switch(device){
102                  case U0:
103                      LPC_PINCON->PINMODE0 &= ~(0b11 << 4);
104                      break;
105                  case U1:
106                      LPC_PINCON->PINMODE4 &= ~(0b11 << 0);
107                      break;
108                  case U2:
109                      LPC_PINCON->PINMODE4 &= ~(0b11 << 16);
110                      break;
111                  case U3:
112                      LPC_PINCON->PINMODE9 &= ~(0b11 << 24);
113                      break;
114              }
115          /*
116           * Select requested frame size
117           */
118              switch (frame_size){
119                  case f5_bit:
120                      /*
121                       * Set frame size to 5 (00)
122                       */
123                      switch(device){
124                          case U0:
125                              LPC_UART0->LCR &= ~(0b11 << 0);
126                              break;
127                          case U1:
128                              LPC_UART1->LCR &= ~(0b11 << 0);
129                              break;
130                          case U2:
131                              LPC_UART2->LCR &= ~(0b11 << 0);
132                              break;
133                          case U3:
134                              LPC_UART3->LCR &= ~(0b11 << 0);
135                              break;
136                      }
137                      break;
138                  case f6_bit:
139                      /*
140                       * Set frame size to 6 (01)
```

```
141                      */
142                     switch(device){
143                         case U0:
144                             LPC_UART0->LCR &= ~(1 << 1);
145                             LPC_UART0->LCR |=  (1 << 0);
146                             break;
147                         case U1:
148                             LPC_UART1->LCR &= ~(1 << 1);
149                             LPC_UART1->LCR |=  (1 << 0);
150                             break;
151                         case U2:
152                             LPC_UART2->LCR &= ~(1 << 1);
153                             LPC_UART2->LCR |=  (1 << 0);
154                             break;
155                         case U3:
156                             LPC_UART3->LCR &= ~(1 << 1);
157                             LPC_UART3->LCR |=  (1 << 0);
158                             break;
159                     }
160                     break;
161                 case f7_bit:
162                     /*
163                      * Set frame size to 7 (10)
164                      */
165                     switch(device){
166                         case U0:
167                             LPC_UART0->LCR &= ~(1 << 0);
168                             LPC_UART0->LCR |=  (1 << 1);
169                             break;
170                         case U1:
171                             LPC_UART1->LCR &= ~(1 << 0);
172                             LPC_UART1->LCR |=  (1 << 1);
173                             break;
174                         case U2:
175                             LPC_UART2->LCR &= ~(1 << 0);
176                             LPC_UART2->LCR |=  (1 << 1);
177                             break;
178                         case U3:
179                             LPC_UART3->LCR &= ~(1 << 0);
180                             LPC_UART3->LCR |=  (1 << 1);
181                             break;
182                     }
183                     break;
184                 case f8_bit:
185                     /*
186                      * Set frame size to 7 (11)
187                      */
```

```
188                     switch(device){
189                         case U0:
190                             LPC_UART0->LCR |= (0b11 << 0);
191                             break;
192                         case U1:
193                             LPC_UART1->LCR |= (0b11 << 0);
194                             break;
195                         case U2:
196                             LPC_UART2->LCR |= (0b11 << 0);
197                             break;
198                         case U3:
199                             LPC_UART3->LCR |= (0b11 << 0);
200                             break;
201                     }
202                     break;
203                 default:
204                     return Unspecified_Error;
205             }
206         /*
207          * Select requested number of stop bits
208          */
209             switch (stop_bit){
210                 case s1_bit:
211                     /*
212                      * 1 stop bit
213                      */
214                     switch(device){
215                         case U0:
216                             LPC_UART0->LCR &= ~(1 << 2);
217                             break;
218                         case U1:
219                             LPC_UART1->LCR &= ~(1 << 2);
220                             break;
221                         case U2:
222                             LPC_UART2->LCR &= ~(1 << 2);
223                             break;
224                         case U3:
225                             LPC_UART3->LCR &= ~(1 << 2);
226                             break;
227                     }
228                     break;
229                 case s2_bit:
230                     /*
231                      * 2 stop bits
232                      */
233                     switch(device){
234                         case U0:
```

```
235                             LPC_UART0->LCR |=  (1 << 2);
236                             break;
237                         case U1:
238                             LPC_UART1->LCR |=  (1 << 2);
239                             break;
240                         case U2:
241                             LPC_UART2->LCR |=  (1 << 2);
242                             break;
243                         case U3:
244                             LPC_UART3->LCR |=  (1 << 2);
245                             break;
246                     }
247                     break;
248                 default:
249                     return Unspecified_Error;
250             }
251         /*
252          * Enable Parity and parity mode if requested
253          */
254             if (parity_enable){
255                 switch(device){
256                     case U0:
257                         LPC_UART0->LCR |= (1 << 3);
258                         break;
259                     case U1:
260                         LPC_UART1->LCR |= (1 << 3);
261                         break;
262                     case U2:
263                         LPC_UART2->LCR |= (1 << 3);
264                         break;
265                     case U3:
266                         LPC_UART3->LCR |= (1 << 3);
267                         break;
268                 }
269                 switch (parity_mode){
270                     case pOdd:
271                         /*
272                          * Odd parity. Number of 1s in the transmitted
273                          * character and the attached parity bit will be
    odd.
274                          */
275                         switch(device){
276                             case U0:
277                                 LPC_UART0->LCR &= ~(0b11 << 4);
278                                 break;
279                             case U1:
280                                 LPC_UART1->LCR &= ~(0b11 << 4);
```

```
281                             break;
282                         case U2:
283                             LPC_UART2->LCR &= ~(0b11 << 4);
284                             break;
285                         case U3:
286                             LPC_UART3->LCR &= ~(0b11 << 4);
287                             break;
288                     }
289                     break;
290                 case pEven:
291                     /*
292                      * Even Parity. Number of 1s in the transmitted
293                      * character and the attached parity bit will be
    even.
294                      */
295                     switch(device){
296                         case U0:
297                             LPC_UART0->LCR &= ~(0b11 << 4);
298                             LPC_UART0->LCR |=  (1 << 4);
299                             break;
300                         case U1:
301                             LPC_UART1->LCR &= ~(0b11 << 4);
302                             LPC_UART1->LCR |=  (1 << 4);
303                             break;
304                         case U2:
305                             LPC_UART2->LCR &= ~(0b11 << 4);
306                             LPC_UART2->LCR |=  (1 << 4);
307                             break;
308                         case U3:
309                             LPC_UART3->LCR &= ~(0b11 << 4);
310                             LPC_UART3->LCR |=  (1 << 4);
311                             break;
312                     }
313                     break;
314                 case pForced_1:
315                     /*
316                      * Forced "1" stick parity.
317                      */
318                     switch(device){
319                         case U0:
320                             LPC_UART0->LCR &= ~(0b11 << 4);
321                             LPC_UART0->LCR |=  (1 << 5);
322                             break;
323                         case U1:
324                             LPC_UART1->LCR &= ~(0b11 << 4);
325                             LPC_UART1->LCR |=  (1 << 5);
326                             break;
```

```
327                         case U2:
328                             LPC_UART2->LCR &= ~(0b11 << 4);
329                             LPC_UART2->LCR |=  (1 << 5);
330                             break;
331                         case U3:
332                             LPC_UART3->LCR &= ~(0b11 << 4);
333                             LPC_UART3->LCR |=  (1 << 5);
334                             break;
335                     }
336                     break;
337                 case pForced_2:
338                     /*
339                      * Forced "0" stick parity.
340                      */
341                     switch(device){
342                         case U0:
343                             LPC_UART0->LCR |= (0b11 << 4);
344                             break;
345                         case U1:
346                             LPC_UART1->LCR |= (0b11 << 4);
347                             break;
348                         case U2:
349                             LPC_UART2->LCR |= (0b11 << 4);
350                             break;
351                         case U3:
352                             LPC_UART3->LCR |= (0b11 << 4);
353                             break;
354                     }
355                     break;
356                 default:
357                     return Invalid_Parity_Config;
358             }
359         }
360         else {
361             switch(device){
362                 case U0:
363                     LPC_UART0->LCR &= ~(1 << 3);
364                     break;
365                 case U1:
366                     LPC_UART1->LCR &= ~(1 << 3);
367                     break;
368                 case U2:
369                     LPC_UART2->LCR &= ~(1 << 3);
370                     break;
371                 case U3:
372                     LPC_UART3->LCR &= ~(1 << 3);
373                     break;
```

```
374                     }
375                     if (parity_mode != pNone)
376                         return Invalid_Parity_Config;
377                 }
378             /*
379              * Enable Break Control if Requested
380              */
381                 switch(device){
382                     case U0:
383                         break_control ? (LPC_UART0->LCR |= (1 << 6)) :
    (LPC_UART0->LCR &= ~(1 << 6));
384                         break;
385                     case U1:
386                         break_control ? (LPC_UART1->LCR |= (1 << 6)) :
    (LPC_UART1->LCR &= ~(1 << 6));
387                         break;
388                     case U2:
389                         break_control ? (LPC_UART2->LCR |= (1 << 6)) :
    (LPC_UART2->LCR &= ~(1 << 6));
390                         break;
391                     case U3:
392                         break_control ? (LPC_UART3->LCR |= (1 << 6)) :
    (LPC_UART3->LCR &= ~(1 << 6));
393                         break;
394                 }
395             /*
396              * Set requested baud rate (see baud rate formula)
397              */
398                 uint32_t pclk;
399                 uint8_t clk_div;
400                 switch(device){
401                     case U0:
402                         clk_div = ((LPC_SC->PCLKSEL1 >> 6) & 0x3);
403                         break;
404                     case U1:
405                         clk_div = ((LPC_SC->PCLKSEL0 >> 8) & 0x3);
406                         break;
407                     case U2:
408                         clk_div = ((LPC_SC->PCLKSEL1 >> 16) & 0x3);
409                         break;
410                     case U3:
411                         clk_div = ((LPC_SC->PCLKSEL1 >> 18) & 0x3);
412                         break;
413                 }
414                 switch (clk_div){
415                     case clkdiv_1:
416                         pclk = sys_get_cpu_clock();
```

```
417                         break;
418                     case clkdiv_2:
419                         pclk = sys_get_cpu_clock() >> 1;
420                         break;
421                     case clkdiv_4:
422                         pclk = sys_get_cpu_clock() >> 2;
423                         break;
424                     case clkdiv_8:
425                         pclk = sys_get_cpu_clock() >> 3;
426                         break;
427                     default:
428                         return Unspecified_Error;
429                 }
430                 uint16_t dl = (uint16_t)(pclk / ((uint32_t)baud_rate << 4));
431                     /*
432                      * Set DLAB bit to 1 to access DLL and DLM registers
433                      */
434                         switch (device){
435                             case U0:
436                                 LPC_UART0->LCR |= (1 << 7);
437                                 break;
438                             case U1:
439                                 LPC_UART1->LCR |= (1 << 7);
440                                 break;
441                             case U2:
442                                 LPC_UART2->LCR |= (1 << 7);
443                                 break;
444                             case U3:
445                                 LPC_UART3->LCR |= (1 << 7);
446                                 break;
447                             default:
448                                 return Unspecified_Error;
449                         }
450                     /*
451                      * Set DLM and DLL registers
452                      */
453                         switch (device){
454                             case U0:
455                                 LPC_UART0->DLL = (dl & 0xFF);
456                                 LPC_UART0->DLM = ((dl >> 8) & 0xFF);
457                                 break;
458                             case U1:
459                                 LPC_UART1->DLL = (dl & 0xFF);
460                                 LPC_UART1->DLM = ((dl >> 8) & 0xFF);
461                                 break;
462                             case U2:
463                                 LPC_UART2->DLL = (dl & 0xFF);
```

```
464                             LPC_UART2->DLM = ((dl >> 8) & 0xFF);
465                             break;
466                         case U3:
467                             LPC_UART3->DLL = (dl & 0xFF);
468                             LPC_UART3->DLM = ((dl >> 8) & 0xFF);
469                             break;
470                         default:
471                             return Unspecified_Error;
472                     }
473             /*
474              * Enable RX Interrupt if requested
475              */
476                 /*
477                  * Enable RX Interrupt (need to set DLAB to 0 first)
478                  */
479                     switch (device){
480                         case U0:
481                             LPC_UART0->LCR &= ~(1 << 7);
482                             if (rx_interrupt_enable){
483                                 LPC_UART0->IER |=  (1 << 0);
484                                 NVIC_EnableIRQ(UART0_IRQn);
485                                 isr_register(UART0_IRQn, uart0_rx_intr);
486                                 u0_rx_queue = xQueueCreate(QUEUE_SIZE,
    sizeof(char));
487                                 intr_enabled = true;
488 
489                             }
490                             else
491                                 LPC_UART0->IER &= ~(1 << 0);
492                             break;
493                         case U1:
494                             LPC_UART1->LCR &= ~(1 << 7);
495                             if (rx_interrupt_enable){
496                                 LPC_UART1->IER |=  (1 << 0);
497                                 NVIC_EnableIRQ(UART1_IRQn);
498                                 isr_register(UART1_IRQn, uart1_rx_intr);
499                                 u1_rx_queue = xQueueCreate(QUEUE_SIZE,
    sizeof(char));
500                                 intr_enabled = true;
501                             }
502                             else
503                                 LPC_UART1->IER &= ~(1 << 0);
504                             break;
505                         case U2:
506                             LPC_UART2->LCR &= ~(1 << 7);
507                             if (rx_interrupt_enable){
508                                 LPC_UART2->IER |=  (1 << 0);
```

```
509                             NVIC_EnableIRQ(UART2_IRQn);
510                             isr_register(UART2_IRQn, uart2_rx_intr);
511                             u2_rx_queue = xQueueCreate(QUEUE_SIZE,
   sizeof(char));
512                             intr_enabled = true;
513                         }
514                         else
515                             LPC_UART2->IER &= ~(1 << 0);
516                         break;
517                     case U3:
518                         LPC_UART3->LCR &= ~(1 << 7);
519                         if (rx_interrupt_enable){
520                             LPC_UART3->IER |=  (1 << 0);
521                             NVIC_EnableIRQ(UART3_IRQn);
522                             isr_register(UART3_IRQn, uart3_rx_intr);
523                             u3_rx_queue = xQueueCreate(QUEUE_SIZE,
   sizeof(char));
524                             intr_enabled = true;
525                         }
526                         else
527                             LPC_UART3->IER &= ~(1 << 0);
528                         break;
529                     default:
530                         return Unspecified_Error;
531                 }
532             /*
533              * Start FIFO Queues and reset TX & RX Buffers
534              */
535                 switch (device){
536                     case U0:
537                         LPC_UART0->FCR |= (1 << 0);
538                         LPC_UART0->FCR |= ((1 << 1) | (1 << 2));
539                         break;
540                     case U1:
541                         LPC_UART1->FCR |= (1 << 0);
542                         LPC_UART1->FCR |= ((1 << 1) | (1 << 2));
543                         break;
544                     case U2:
545                         LPC_UART2->FCR |= (1 << 0);
546                         LPC_UART2->FCR |= ((1 << 1) | (1 << 2));
547                         break;
548                     case U3:
549                         LPC_UART3->FCR |= (1 << 0);
550                         LPC_UART3->FCR |= ((1 << 1) | (1 << 2));
551                         break;
552                     default:
553                         return Unspecified_Error;
```

```
554                 }
555             /*
556              * Enable RX Status Line Interrupts
557              */
558                 switch (device){
559                     case U0:
560                         LPC_UART0->IER |= (1 << 2);
561                         break;
562                     case U1:
563                         LPC_UART1->IER |= (1 << 2);
564                         break;
565                     case U2:
566                         LPC_UART2->IER |= (1 << 2);
567                         break;
568                     case U3:
569                         LPC_UART3->IER |= (1 << 2);
570                         break;
571                     default:
572                         return Unspecified_Error;
573                 }
574             this_device = device;
575             initialized = true;
576             return Success;
577         }
578         default:
579             return No_Device_Initialized;
580     }
581 }
582
583 int LabUart::Transmit(char c)
584 {
585     switch(this_device){
586         case U0:
587             while (!((LPC_UART0->LSR >> 5) & 1));
588             LPC_UART0->THR = c;
589             break;
590         case U1:
591             while (!((LPC_UART1->LSR >> 5) & 1));
592             LPC_UART1->THR = c;
593             break;
594         case U2:
595             while (!((LPC_UART2->LSR >> 5) & 1));
596             LPC_UART2->THR = c;
597             break;
598         case U3:
599             while (!((LPC_UART3->LSR >> 5) & 1));
600             LPC_UART3->THR = c;
```

```
601             break;
602         default:
603             return No_Device_Initialized;
604 
605     }
606     return Success;
607 }
608 
609 char LabUart::Receive()
610 {
611     char c = NULL;
612     if (intr_enabled){
613         switch(this_device){
614             case U0:
615                 xQueueReceive(u0_rx_queue, &c, DEQUEUE_TIMEOUT);
616                 break;
617             case U1:
618                 xQueueReceive(u1_rx_queue, &c, DEQUEUE_TIMEOUT);
619                 break;
620             case U2:
621                 xQueueReceive(u2_rx_queue, &c, DEQUEUE_TIMEOUT);
622                 break;
623             case U3:
624                 xQueueReceive(u3_rx_queue, &c, DEQUEUE_TIMEOUT);
625                 break;
626         }
627     return c;
628     }
629     else{
630         switch(this_device){
631             case U0:
632                 while (!(LPC_UART0->LSR & 1));
633                     c = LPC_UART0->RBR;
634                 break;
635             case U1:
636                 while (!(LPC_UART1->LSR & 1));
637                     c = LPC_UART1->RBR;
638                 break;
639             case U2:
640                 while (!(LPC_UART2->LSR & 1));
641                     c = LPC_UART2->RBR;
642                 break;
643             case U3:
644                 while (!(LPC_UART3->LSR & 1));
645                     c = LPC_UART3->RBR;
646                 break;
647         }
```

```
648     return c;
649     }
650 }
651
652 void LabUart::uart0_rx_intr()
653 {
654     char c = LPC_UART0->RBR;
655     xQueueSend(u0_rx_queue, &c, ENQUEUE_TIMEOUT);
656 }
657
658 void LabUart::uart1_rx_intr()
659 {
660     char c = LPC_UART1->RBR;
661     xQueueSend(u1_rx_queue, &c, ENQUEUE_TIMEOUT);
662 }
663
664 void LabUart::uart2_rx_intr()
665 {
666     char c = LPC_UART2->RBR;
667     xQueueSend(u2_rx_queue, &c, ENQUEUE_TIMEOUT);
668 }
669
670 void LabUart::uart3_rx_intr()
671 {
672     char c = LPC_UART3->RBR;
673     xQueueSend(u3_rx_queue, &c, ENQUEUE_TIMEOUT);
674 }
675
```