RFID Reading App-note

This tutorial explains how to take the **ID-20LA** made by **ID-Innovations** and read the tag numbers through RS-232-UART, routed through GPIO, to the monitor. **Note:** This tutorial is not 100% complete, as the tag numbers being read in currently are not consistent, although this will be worked out shortly.

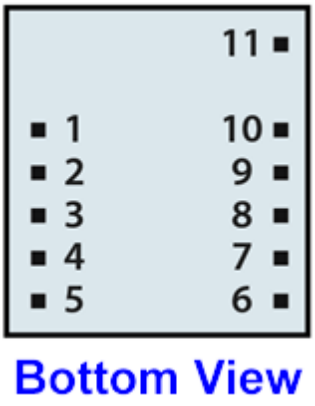
Brief Outline:

[0] Hook up all pins from GPIO to reader

[1] Set up Q-sys components
[2] Modify Generated VHDL top-level code
[3] Write C-code for testing
[4] Run/Test with RFID tag

Details:

[0] Hook up all pins from GPIO to reader



1. GND
2. RES (Connect to pin 11[VCC])
3. No Connection
4. No Connection
5. No Connection
6. Tag in Range (No Connection needed)
7. Format Select (Connect to pin 1[GND] to select ASCII format.)
8. Data 1 (Connect to GPIO pin [IO_B33]. Physically, pin 38.)
9. No Connection
10. No Connection
11. 3.3V VCC

→ Using the ribbon cable and a header, connect all pins as described above.

[1] Set up Q-sys components

→ You will need to set up the Altera DE2 board with the components you will be using. We have chosen to use the **Nios II/e** (efficient) processor, although you may choose any one you would like. This can be found in the **Embedded Processor** library.
→ You will also need many of the same components used in the Lab1/2 in the beginning of this course.
→ Add the RS-232 UART component, found in: **University Program -> Communications** library. Make sure you change the **Baud Rate to 9600**, as this is the same rate as the RFID reader we are using here.
→ Once these have been added, continue as usual by clicking the **Generate** button in the generate tab.

[2] Modify Generated VHDL top-level code

→ Move over to the HDL tab, and copy the port lines for the RS-232 receiver over to the Quartus **VHDL top-level**, which looks similar to this:

rs232_0_external_interface_RXD : in std_logic :=X;

This needs to be added to the niosii_system component port, as well as the port map in the architecture.
→ Once these lines have been added you will need to add a **GPIO entity**. You may choose to use GPIO_0 or GPIO_1; it doesn't matter, as long as you're consistent. The entity will be added exactly like this:

GPIO_1 : inout std_logic_vector(35 downto 0);

→ You then need to re route the UART to utilize the GPIO rather than the RS-232 since we are using a component that requires 2.8-5V inputs, whereas the RS-232 DB-9 cables can range in voltage from 3-15V. The **GPIO output is 3.3V**, which is within spec for the reader. To set up the GPIO to be the receiver of the UART, change the signal in the port map to look this this. This uses pin 33 in the GPIO map (physically pin 38).

rs232_0_external_interface_RXD => GPIO_1(33);

→ Then you will **compile** the code and **program** the board.

[3] Write C-code for testing

→ Create a new "NIOII Application and BSP from Template". Then select the SOPC file generated from the Quartus top-level code. Name it, and choose the "Hello MicroC/OS-II" Template.

→ The following C-code was used to receive data from the reader:

```
#include <stdio.h>
#include "includes.h"
#include "sys/alt_irq.h"
#include "alt_types.h"
#include "altera_up_avalon_rs232.h"
#include "altera_up_avalon_rs232_regs.h"

/* Definition of Task Stacks */
#define   TASK_STACKSIZE       2048
OS_STK task1_stk[TASK_STACKSIZE];
OS_STK task2_stk[TASK_STACKSIZE];

/* Definition of Task Priorities */

#define TASK1_PRIORITY      1
#define TASK2_PRIORITY      2

#define Q_SIZE 30

#define WRITE_FIFO_EMPTY 0x80
#define READ_FIFO_EMPTY 0x0

OS_EVENT* queue_handler;
void* queue[Q_SIZE];

void task1(void* pdata) {

         alt_u16 read_FIFO_used;
         alt_u8 data_r8;
         int enter = 0;
         unsigned p_error;
         alt_up_rs232_dev* rs232_dev;
         rs232_dev = alt_up_rs232_open_dev("/dev/rs232_0");
         if (rs232_dev == NULL)
            printf("error\n");
         else
            printf("no error\n");
         alt_up_rs232_enable_read_interrupt(rs232_dev);

         while (1) {
            read_FIFO_used =
alt_up_rs232_get_used_space_in_read_FIFO(rs232_dev);
            if (read_FIFO_used > READ_FIFO_EMPTY) {

                  int i;
                  for (i = 0; i < read_FIFO_used; i++) {
                        alt_up_rs232_read_data(rs232_dev, &data_r8,
&p_error);
                        printf("%x", data_r8);
                        int i = OSQPost(queue_handler, data_r8);
                        if (i < 0) {
                              printf("Error posting");
                        }
                  }
                  printf("\n");
            }
```

```
        }
}

void task2(void* pdata) {
        while (1) {
           int err;
           char a = OSQPend(queue_handler, 0, &err);
           printf("%x\n", a);
        }
}
/* The main function creates two task and starts multi-tasking */
int main(void) {

        queue_handler = OSQCreate(queue, Q_SIZE);

        OSTaskCreateExt(task1, NULL, (void *) &task1_stk[TASK_STACKSIZE
- 1],
                  TASK1_PRIORITY, TASK1_PRIORITY, task1_stk,
TASK_STACKSIZE, NULL, 0);

        OSTaskCreateExt(task2, NULL, (void *) &task2_stk[TASK_STACKSIZE
- 1],
                  TASK2_PRIORITY, TASK2_PRIORITY, task2_stk,
TASK_STACKSIZE, NULL, 0);
        OSStart();
        return 0;
}
```

[4] Run/Test with RFID tag

→ To ensure no issues, **Generate BSP** before you **build** and **compile** the code.

→ After it has been built and compiled, simply press the **Run** button in eclipse and place a tag within range of the reader, and you can see the ASCII values received from the reader, output to the console.

Repeat Note: This tutorial is not 100% complete, as the tag numbers being read in currently are not consistent, although this will be worked out shortly.

[Appendix]

The data output by the reader is a total of 16 bytes with a Start Transmit byte of 0x02, 10 ASCII bytes for data and 2 ASCII bytes for checksum, 2 bytes we don't need to worry about, and an End Transmit byte of 0x03.

The tag ID's look similar to this: 0C000621A58E, where the last byte (8E) is the checksum.

0C = 00001100
00 = 00000000
06 = 00000110
21 = 00100001
A5 = 10100101

CHECKSUM = 10001110 (8E) → Therefore the data was received without corruption.

→ This is a link to the ID low voltage reader series:
http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/ID/ID-2LA, ID-12LA, ID-20LA(2013-4-10).pdf