RCX Library manual

Joakim Lindgren
dat95jln@idt.mdh.se

1 Contents

Joakim Lindgren
jln99023@student.mdh.se

2 Introduction

This document describes the hardware library to access different devices on the Lego Mindstorm Microcontroller (RCX) [2]. The library is included in the Asterix Framework. It is available to the application programmer of a Asterix system a make it easier to access the library. The library has been designed in a way that it does not uses any interrupt, and therefor is the overhead of the library included in the tasks execution times analysis. There are function that make use of the ROM memory function on the RCX. The documented is devided into a sections that describes all devices that can be controlled. In each section constants,are presented, as well as function descriptions for routine associated with the device.

Joakim Lindgren
jln99023@student.mdh.se

3 Access the buttons

3.1 Constants

```
#define VIEW_BUTTON 2
#define PRGM_BUTTON 4
#define RUN_BUTTON  1
#define NON_BUTTON  0
```

3.2 Read the buttons

Syntax: static inline void readButtons (short *ptr)

Semantic: Checks which button that is pressed. The function is non-blocking and the **ptr* can indicate that no button was pressed.

Input: *ptr*, a pointer to a short where which button pressed is indicated.

Output: -

Calls: ROM(0x1fb6)

Comments: -

4 Access the communication protocol

To access the communication protocol that is presented in the document "Obelix Up-loader Tool" [1] the following functions and constant are available to the application programmer or the kernel.

4.1 Constants

```
#define PACKAGE_SIZE     65536
```

4.2 Send a package

Syntax: void packageSend(char *data, unsigned short dsize)

Semantic: This function sends a stream of data. It splitting up the data into chucks of 16 bytes and using the data link layer to send the chuck in frames. In the first frame

Joakim Lindgren
jln99023@student.mdh.se

that is sent the size of the following data is put. The function is blocking till the whole package is sent.

Input: *data*, a pointer to a data stream to send.
dsize, the size of the data stream to send. Max size is PACKAGE_SIZE.

Output: -

Calls: DLSend

Comments: Note that the function does not return any error code. The reason is that the function keeps trying to contact the receiver of the package without any timeout.

4.3 Send a memory dump

Syntax: void memSend(char *data, unsigned short dsize)

Semantic: Send a stream of memory. the different to the packageSend function is that this function also sends information about the start address.

Input: *data*, a pointer to the memory to send.
dsize, the size of the memory to send.

Output: -

Calls: packageSend

Comments: Note that the size is a real size and not a offset in memory. For instance if dsize is 0xff , we send 255 bytes of data, and not a memory space of 0xff(256 bytes).

5 Routines to access the display

To make a change on the display become visible the refreshDisplay has to be called afterwards.

5.1 Constants

The display segments that are not part of numbers:

```
#define LCD_STANDING            0x3006
#define LCD_WALKING             0x3007
#define LCD_SENSOR_0_VIEW       0x3008
#define LCD_SENSOR_0_ACTIVE     0x3009
```


Joakim Lindgren
jln99023@student.mdh.se

```
#define LCD_SENSOR_1_VIEW       0x300a
#define LCD_SENSOR_1_ACTIVE     0x300b
#define LCD_SENSOR_2_VIEW       0x300c
#define LCD_SENSOR_2_ACTIVE     0x300d
#define LCD_MOTOR_0_VIEW        0x300e
#define LCD_MOTOR_0_REV         0x300f
#define LCD_MOTOR_0_FWD         0x3010
#define LCD_MOTOR_1_VIEW        0x3011
#define LCD_MOTOR_1_REV         0x3012
#define LCD_MOTOR_1_FWD         0x3013
#define LCD_MOTOR_2_VIEW        0x3014
#define LCD_MOTOR_2_REV         0x3015
#define LCD_MOTOR_2_FWD         0x3016
#define LCD_DATALOG             0x3018
#define LCD_DOWNLOAD            0x3019
#define LCD_UPLOAD              0x301a
#define LCD_BATTERY             0x301b
#define LCD_RANGE_SHORT         0x301c
#define LCD_RANGE_LONG          0x301d
#define LCD_ALL                 0x3020
```

These constants is used in the setLcdNumber function:

```
#define LCD_DECIMAL_0           0x3002
#define LCD_DECIMAL_1           0x3003
#define LCD_DECIMAL_2           0x3004
#define LCD_DECIMAL_3           0x3005
#define LCD_PROGRAM             0x3017
#define LCD_SIGNED              0x3001
#define LCD_UNSIGNED            0x301f
```

5.2 Set a LCD segment on the display

Syntax: static inline void setLcdSegment (short code)

Semantic: Turn on the specific segment on the RCX.

Input: *code*, one of the previous shown constants.

Output: -

Calls: ROM(0x1b62)

Comments: -

Joakim Lindgren
jln99023@student.mdh.se

5.3 Clear a LCD segment on the display

Syntax: static inline void clearLcdSegment (short code)

Semantic: Turn off the specific segment on the RCX.

Input: *code*, one of the previous shown constants

Output: -

Calls: ROM(0x1e4a)

Comments: -

5.4 Print a decimal number on the display

Syntax: static inline void setLcdNumber (short code, short value, short pointcode)

Semantic: Prints a decimal number on the display. By the arguments code and pointcode the format can be adjusted. The display can show number between –9999 to 9999.

Input: *code*, tell is the number will be displayed unsigned or signed. Also used when the program number field to be set. Then the value must be 0 to 9.
value, the value to display.
pointcode, tell how many decimal to show.

Output: -

Calls: ROM(0x1ff2)

Comments: -

5.5 Clear the display window

Syntax: static inline void clearDisplay (void)

Semantic: Clear the all display segments.

Input: -

Output: -

Joakim Lindgren
jln99023@student.mdh.se

Calls: ROM(0x27ac)

Comments: -

5.6 Update the display

Syntax: static inline void refreshDisplay (void)

Semantic: Makes a change visible on the display. The function is naturally used after the other function in this section.

Input: -

Output: -

Calls: ROM(0x27c8)

Comments: -

5.7 Print a hexadecimal value

Syntax: extern void printHex(unsigned short data)

Semantic: This function prints a hexadecimal value on the LCD, using 4 digits. This means that values up to 16 bits unsigned int can be printed.

Input: *data*, a value to print.

Output: -

Calls: cputc_native

Comments: This function was created to make it easier to display a whole 16 bits integer

6 Access the motors

6.1 Constants

```
#define MOTOR_FORWARD     1
#define MOTOR_BACKWARD    2
#define MOTOR_STOP        3
#define MOTOR_FLOAT       4
```


Joakim Lindgren
jln99023@student.mdh.se

```
#define MOTOR_A         0x2000
#define MOTOR_B         0x2001
#define MOTOR_C         0x2002
```

6.2 Change the state of a motor

Syntax: void setMotor(short code, short dir)

Semantic: This function change the state of a motor. E.g. turn it on and off.

Input: *code*, which motor to affect. (E. i. MOTOR_A)
dir, This field tells what to do with the specific motor. The motor can be set in following states; running forward, running backward, stop, and float.

Output: -

Calls: -

Comments: The difference between stop and float is that the stop state makes the motor be "locked" in position. A motor in float state is easy to rotate by hand.

7 Access the sensors

7.1 Constants

```
#define BLACK 0xdc00
#define WHITE 0x9c00


#define SENSOR_1 0x1000
#define SENSOR_2 0x1001
#define SENSOR_3 0x1002
#define SENSOR_4 0x1003                 /* battery voltage */
```

7.2 Initiate the sensors

Syntax: static inline void initSensors (void)

Semantic: Configure port 6 bit 0, 1, and 2 to output.

Input: -

Joakim Lindgren
jln99023@student.mdh.se

Output: -

Calls: ROM(0x1498)

Comments: This function must be called before any other function that deals with the sensors.

7.3 Set a sensor active

Syntax: void setSensorActive (short code)

Semantic: Turn on the power to the sensor.

Input: code, which sensor to activate, e. i. SENSOR_1.

Output: -

Calls: -

Comments: This function is used when accessing a light sensor.

7.4 Set a sensor passive

Syntax: void setSensorPassive (short code)

Semantic: Turn off the power to the sensor.

Input: code, which sensor to activate, e. i. SENSOR_1.

Output: -

Calls: -

Comments: This function is used when accessing a light sensor.

7.5 Read a sensor value

Syntax: void readSensor(short code, short *raw)

Semantic: The function starts a A/D conversion and returns the digital value. It only sample the specific sensor input.

Input: *code*, which sensor to read a value from, e. i. SENSOR_1.
raw, a pointer to a 16bits integer to put the sampled value.

Joakim Lindgren
jln99023@student.mdh.se

Output: -

Calls: -

Comments: -

8 Access the serial primitives

8.1 Constants

#define BLOCKED 0
#define NONBLOCKED 1

8.2 Initiate the serial port

Syntax: void serialInit(void)

Semantic: Sets the control parameters for the serial transmission. E.I. sets 2400 baud, turn on the receiver and transmission mode on the UART.

Input: -

Output: -

Calls: serial_shutdown
carrier_init
lnp_logical_range

Comments: -

8.3 Shut down the serial port

Syntax: void serialShutdown(void)

Semantic: Turn off the functionality to the serial port.

Input: -

Output: -

Calls: lnp_logical_range

Joakim Lindgren
jln99023@student.mdh.se

Comments: -

8.4 Read the serial port

Syntax: int readSerial(char *buf, int length, int block)

Semantic: Reads the serial port in either blocking or non-blocking mode. In blocking mode, the routine returns when *length* bytes has been read. In non-blocking mode, the routine returns immediately. Is a bytes was read the function returns 1, otherwise 0.

Input: *buf*, a pointer to a buffer to put the read data.
length, the length of the data to read.
block, tells if the function will be blocking or non-blocking.

Output: *int*, in blocking no does not matter. In non-blocking mode, 1 of a byte was read else 0.

Calls: serialError

Comments: -

8.5 Send data

Syntax: int writeSerial(char *buf, int length)

Semantic: Sends a given data stream in the serial port.

Input: *buf*, a pointer to the data stream to send.
length, the length of the data stream to send.

Output: int, does not mean anything(- yet;-).

Calls: -

Comments: -

Joakim Lindgren
jln99023@student.mdh.se

9 References

[1] “Obelix Up-loader Tool” , J. Lindgren, University of Mälardalen, 2000

[2] “RCX Internals”, Kekoa Proudfoot.

Joakim Lindgren
jln99023@student.mdh.se