I2C read commands / blocks

register	bits	description	type	NXT block
0	0	battery low warning	logic	I2C Multiple Read
	1	bumber left	logic	
	2	bumber right	logic	
	3	IR-RC5 received	logic	
	4	IR-RC5 ready to send	logic	
	5	ACS: obstacle left	logic	
	6	ACS: obstacle right	logic	
	7	motor state changed	logic	
1	0	power on	logic	I2C Multiple Read
	1	ACS active	logic	
	2	watchdog timer	logic	
	3	watchdog request	logic	
	4	watchdog active	logic	
2	0	motor activity completed	logic	I2C Multiple Read
	1	motor running	logic	
	2	motor over-current	logic	
	3..4	direction (FWD=0, BWD=1, LEFT=2, RIGHT=3)	number	
3	0..7	motor power left (0..210)	number	I2C Multiple Read
4	0..7	motor power right (0..210)	number	
5	0..7	motor speed left	number	RP6 Motor Speed
6	0..7	motor speed right	number	
7	0..7	target speed left	number	I2C Multiple Read
8	0..7	target speed right	number	
9+10	0..15	distance left (L+H)	number	RP6 Distance
11+12	0..15	distance right (L+H)	number	
13+14	0..15	brightness left (L+H)	number	RP6 Brightness
15+16	0..15	brightness right (L+H)	number	
17+18	0..15	motor current left (L+H)	number	I2C Multiple Read
19+20	0..15	motor current right (L+H)	number	
21+22	0..15	battery voltage level (L+H)	number	RP6 Battery
23+24	0..15	ADC0 (L+H) → 0..1023	number	I2C Multiple Read
25+26	0..15	ADC1 (L+H) → 0..1023	number	I2C Multiple Read
27	0..7	IR-RC5 address	number	RP6 Remote IR
28	0..7	IR-RC5 data	number	
29	0..7	state LEDs	number	I2C Multiple Read

I2C write commands / blocks

NXT block	description	command ID	parameter 1	parameter 2	parameter 3	parameter 4	parameter 5
RP6 LED	set LED status	3	bit mask LEDs (0x00=all off; 0x01=all on; >=bit 2 → mask)	-	-	-	-
RP6 Motor Stop	stop motor	4	-	-	-	-	-
RP6 Motor Run	start motor	5	speed left (0..255)	speed right (0..255)	-	-	-
RP6 Motor Move	drive fixed distance	7	speed (0..255)	direction (FWD 0, BWD 1, LEFT 2, RIGHT 3)	distance (HB) in encode pulses	distance (LB) in encode pulses	wait until finished?
RP6 Motor Rotate	rotate on spot	8	speed (0..255)	direction (FWD 0, BWD 1, LEFT 2, RIGHT 3)	angle (HB) in deg	angle (LB) in deg	wait until finished?
RP6 Distance Reset	reset distance measurement	13	-	-	-	-	-

module	function name	description
n.a.	void initRobotBase(void)	initialize microcontroller; must be called at start of main()!
UART	void writeChar(char ch)	send byte ch via UART
	void writeString(char *string)	send string in RAM via UART (terminated by '\n')
	Void writeString_P(STRING)	send string in flash (→ const) via UART (terminated by '\n')
	void writeStringLength(char *data, uint8_t length, uint8_t offset)	schicke length viele Bytes aus data über UART, beginnend mit offset
	void writeInteger(int16_t number, uint8_t base)	send integer with configurable base via UART. Supported bases are DEC, BIN, OCT, and HEX
	void writeIntegerLength(uint16_t number, uint8_t base, uint8_t length);	similar to writeInteger(), but with configurable number of digits. If number is shorter than length, number is padded with leaing zeroes. If number is longer than length, only last digits are sent
	char readChar(void)	read 1B from UART Rx ring buffer; read 0 if buffer is empty
	uint8_t getBufferLength(void)	read number of bytes in UART Rx ring buffer
	uint8_t readChars(char *buf, uint8_t num)	read num many bytes from UART Rx ring buffer to buf. Returns number of read bytes
Delay / time	void sleep(uint8_t time)	halt program for time*100us. Interrupts are handled in background
	void mSleep(uint16_t time)	halt program for time*1ms. Interrupts are handled in background
	void startStopwatchX(void) mit X=1..8	Start 16bit stop watch “X” (1..8) with tick every 1ms. The stop watch is not reset prior to start
	void stopStopwatchX(void)	Halt stop watch “X” (1..8). Counter value is not modified
	uint8_t isStopwatchXRunning(void)	return if stop watch “X” (1..8) is running (1=aktive; 0=stopped)
	void setStopwatchX(uint16_t preset)	init counter of 16bit stop watch “X” (1..8) to preset (in ms)
	uint16_t getStopwatchX(void)	return counter value of 16bit stop watch “X” (1..8) in 1ms
LEDs	void setLEDs(uint8_t leds)	set state of 6 status-LEDs (0x00=all off; 0x01=all on; >=Bit 2 → bit mask). Alternative use “statusLEDs.LEDx“ with x=0..5. Note: value is only written to shadow register!
	void updateStatusLEDs(void)	update state of 6 status-LEDs with value from shadow register
bumpers	uint8_t getBumperLeft(void)	read state of touch sensor left. Is hard-wired with a status-LED. Keep >=10ms between calls to getBumperX()
	uint8_t getBumperRight(void)	read state of touch sensor right. Is hard-wired with a status-LED. Keep >=10ms between calls to getBumperX()
	void task_Bumpers(void)	Bumper task for main(). Checks bumber status every 50ms and stores it in variables bumper_left and bumper_right. If defined, bumber handler is called on status change (see below)
	void BUMPERS_setStateChangedHandler(void (*bumperHandler)(void))	Define handler for function to call by task_Bumpers() on change of bumper status. Event handler should be kept as short as possible.
ADC / battery voltage, motor current, and light sensors	uint16_t readADC(uint8_t channel)	measure channel with 10bit ADC. Available channels: ADC_BAT, ADC_MCURRENT_R, ADC_MCURRENT_L, ADC_LS_L, ADC_LS_R, ADC_ADC0, ADC_ADC1
	void task_ADC(void)	ADC task for main(). Measures and stores ADC values in variables adcBat, adcMotorCurrentLeft, adcMotorCurrentRight, adcLSL, adcLSR, adc0, and adc1
AntiCollisionSystem (ACS)	void setACSPwrOff(void)	set power of ACS to OFF
	void setACSPwrLow(void)	set power of ACS to LOW
	void setACSPwrMed(void)	set power of ACS to MEDIUM
	void setACSPwrHigh(void)	set power of ACS to MAXIMUM
	void task_ACS(void)	ACS task for main(). Measured ACS status and stores it in of obstacle_left and obstacle_right
	void ACS_setStateChangedHandler(void (*acsHandler)(void))	Define handler for function to call by task_ACS() on change of ACS. Event handler should be kept as short as possible.
IRCOMM and RC5 (based on IR, as ACS → handling in task_ACS())	void IRCOMM_sendRC5(uint8_t addr, uint8_t data)	IR-RC5 send 6-bit data (plus “toggle bit”) to 5-bit address addr. Only bits 0..5 of data and bit 7 (MSB) are actually sent → 8-bit data requires 2 frames. Sending is automatically in task_ACS()
	void IRCOMM_setRC5DataReadyHandler(void (*rc5Handler)(RC5data_t))	Define handler for function to call by task_ACS() on reception via IR-RC5. Event handler should be kept as short as possible. Data type RC5data_t is a struct, which contains RC5 address bits (identifier), toggle bit (changes on each event), and key code (i.e. data). These can be accesses by rc5data.device, rc5data.toggle_bit, and rc5data.key_code
power saving	powerON()	Deactivate ACS (→ also IR-RC5), encoder, motor current-sense, and power-on LED (total approx. 10mA)
	powerOFF()	Activate ACS (→ also IR-RC5), encoder, motor current-sense, and power-on LED (total approx. 10mA)
motors (only set target values for task_motionControl())	void task_motionControl(void)	Motor task für main(). Controls the motors according to speed request. Regulates speed and acceleration, and controls over-current and failures → should be used instead of own routines
	void moveAtSpeed(uint8_t desired_speed_left, uint8_t desired_speed_right)	Set target speed in “encoder pulses per 200ms” for left and right motor in task_motionControl(). Set to 0 for deactivating the power modules after motion complete (for power-saving). For continous operation use speed<=160!
	uint8_t getDesSpeedLeft(void)	read target speed of left motor in “encoder pulses per 200ms”
	uint8_t getDesSpeedRight(void)	read target speed of right motor in “encoder pulses per 200ms”
	uint8_t getLeftSpeed(void)	read actual speed of left motor in “encoder pulses per 200ms”
	uint8_t getRightSpeed()	read actual speed of right motor in “encoder pulses per 200ms”
	void changeDirection(uint8_t dir)	Set new direction for task_motionControl(). Possible parameters are FWD, BWD, LEFT, and RIGHT. Motors are smoothly stopped, before direction is changed.
	uint8_t getDirection(void)	read current movement direction (FWD, BWD, LEFT or RIGHT)
	void move(uint8_t speed, uint8_t dir, uint16_t dist, uint8_t blocking)	drive straight for distance dist (in encoder pulses) with speed speed, direction dir (=FWD or BWD). Parameter blocking determines, if program is stopped until movement is finished (for true, task_motionControl() is obsolete). Convert mm to encoder pulses via macro DIST_MM(DISTANCE)
	uint8_t isMovementComplete(void)	return status of movement request (true=movement finished)
	void stop(void)	cancel all movement requests. Task motionControl() brakes motors smoothly
	void rotate(uint8_t speed, uint8_t dir, uint16_t angle,uint8_t blocking)	rotate on spot by angle (in deg) with speed speed, direction dir (=LEFT or RIGHT). Parameter blocking determines, if program is stopped until movement is finished (for true, task_motionControl() is obsolete)
misc	void task_RP6System(void)	Container for calling task_ADC(), task_ACS(), task_bumpers(), and task_motionControl(). For reasonable response, call every 10-50ms
I2C	void I2CTWI_initSlave(uint8_t address)	init I2C moduls as I2C slave with address addr (addr=0 → broadcast). Send and receive are handled in interrupts

variable name	description
uart_status	Is set to UART_BUFFER_OVERFLOW in case of UART ring buffer overflow. Configure buffer size via UART_RECEIVE_BUFFER_SIZE
bumper_left	State of bumber left. Is set by task_Bumpers() → has to be called regularly
bumper_right	State of bumber right. Is set by task_Bumpers() → has to be called regularly
adcBat	ADC result for battery voltage. Is set by task_ADC() → has to be called regularly
adcMotorCurrentLeft	ADC result for motor current left. Is set by task_ADC() → has to be called regularly
adcMotorCurrentRight	ADC result for motor current right. Is set by task_ADC() → has to be called regularly
adcLSL	ADC result for light sensor left. Is set by task_ADC() → has to be called regularly
adcLSR	ADC result for light sensor right. Is set by task_ADC() → has to be called regularly
adc0	ADC result for channel 0. Is set by task_ADC() → has to be called regularly
adc1	ADC result for channel 1. Is set by task_ADC() → has to be called regularly
obstacle_left	status ACS left. Is set by task_ACS() → has to be called regularly
obstacle_right	status ACS right. Is set by task_ACS() → has to be called regularly