**iRobot Create with Xilinx SDK Starter Guide**

**By Dan Eusebio**

Table of Contents

[Introduction 3](#_Toc414114962)

[Xilinx Platform Studio (XPS) Setup 4](#_Toc414114963)

[SDK Programming 4](#_Toc414114964)

[Move forward example 5](#_Toc414114965)

[Setting up the interrupt 6](#_Toc414114966)

[Read Sensor Data 7](#_Toc414114967)

[Putting it Together 10](#_Toc414114968)

# Introduction

This guide is meant to serve as a starting point for students who want to begin utilizing the iRobot create into their project with the Genesys Virtex 5. The iRobot Create is a device that can be seen as a basic shell of a Roomba. It is packed with collision sensors and cliff sensors, and has music playback. In order to communicate with the device, the iRobot Create includes a white serial to din connector as shown:



The RS232 connector here is female and so is the FPGA RS232 port so a female to male converter is required. The block that reads iRobot also has an led indicator to show that it is connected and arrows to indicate the direction of data flow.

The default baud rate of the iRobot Create is at 57600. This can also be scaled down to 19200 as dictated in the open interface manual, however, this guide uses 57600.

The way that the iRobot takes commands is that it must first be set on, and then you send it what they call opcodes. They’re opcodes here just like in CS33 because these values correspond to different commands. You just directly send the value to the device and it’ll read it. One way to get acquainted with this is to use real term.

* Connect the serial to USB converter to the iRobot connector and connect it to real term.
* Go to device manager and find the port that this now corresponds with.
* Take that port number, enter it into real term and change the baud rate to 57600.
* Open up the connection under the port tab.
* Make sure the iRobot is on and type in 128 131 135 and click send numbers.
* The iRobot should start its cover demo and start moving on its own. You can also send other commands as shown in the data sheet. Here 128 is start, 131 is sleep mode, and 135 is the cover demo.
* You will also see messages printed if you have the serial port connected and turn on the iRobot to further verify that your connection is correct.

# Xilinx Platform Studio (XPS) Setup

(This assumes the reader already knows how to setup base builder).

First, we must setup the UART IP Core correctly in order to communicate with the iRobot Create. We have two options for this:

* During the process for creating a new base builder project, continue till the Peripheral Configuration page. In Peripheral Configuration, click on RS232\_Uart\_1. Here you can keep it as uartlite or change it to uart16550.
  + If you keep it as uartlite, specify the baud rate now as 57600 and check the interrupt box.
  + If you use uart16550 instead, you can define the baud rate software side and is thus recommended.
* After this, the creation can be completed and the project can be exported to SDK.

# SDK Programming

You can now start sending commands to the iRobot Create in the Xilinx SDK Platform.

* First, make a new helloworld project.
  + We’ll simply just be treating helloworld.c as our main.c.
* You’ll want to include a few more headers as well.
  + #include “xparameters.h”
    - This header gives you access to all of the addresses you defined in XPS for your IP Cores. Without this, you would not be able to do much more than just process code and print to terminal in SDK.
  + #include “xuartns550\_l.h”
    - This header is for the uart16550 IP Core. It allows you to change the data direction as well as the baud rate.
    - Corresponds to xps\_uart16550 in XPS.
  + #include “xuartlite\_l.h”
    - This header is similar to xuartns550\_l.h but is for the uartlite IP Core.
    - Corresponds to xps\_uartlite in XPS
  + #include “xintc\_l.h”
    - This is to use the interrupt controller IP Core so that we can invoke interrupts for all the peripherals it was enabled for.
    - Corresponds to xps\_intc in XPS.
  + #include “xgpio.h”
    - This will let us control the LEDs and also every other GPIO peripheral we initiated in the FPGA.
* Now, we can write code to make the iRobot move.

# Move forward example

First if you are using uart16550, we have to set up the baud rate first. Here we use the function,

void XUartNs550\_SetBaud(u32 BaseAddress, u32 InputClockHz, u32 BaudRate);

This function can be found in xuartns550\_l.h. The base address of our UART can be found in xparameters.h.

* Usually it ends up being called in this format: XPAR\_RS232\_UART\_0\_BASEADDR:
  + Since the name of my IP Core was rs232\_uart\_0, it appears in the name.
* InputClkhz can also be found in the xparameters.h file and is usually denoted as XPAR\_RS232\_UART\_0\_CLOCK\_FREQ\_HZ.
  + If you remember, we set up the frequency in the base builder process and so this value should just be 125000000 (125Mhz).
* For baud rate, you can make your own definition or just input 57600.
* Result:
* XUartNs550\_SetBaud(XPAR\_RS232\_UART\_0\_BASEADDR,XPAR\_RS232\_UART\_0\_CLOCK\_FREQ\_HZ, UART\_BAUDRATE\_BOT);
* We also use:
  + XUartNs550\_SetLineControlReg(XPAR\_RS232\_UART\_0\_BASEADDR, XUN\_LCR\_8\_DATA\_BITS);
  + To set the byte and parity information.
  + XUN\_LCR\_DATA\_BITS
    - Can also be found in the xparameters.h file and sets the RS232 connector to 8 data bits, no parity and 1 stop bit.
* We can now use:
  + void XUartNs550\_SendByte(u32 BaseAddress, u8 Data);
  + To send the commands.

Code:

Int main(){

//UART0 -> General Serial Communication

XUartNs550\_SetBaud(XPAR\_RS232\_UART\_0\_BASEADDR,XPAR\_RS232\_UART\_0\_CLOCK\_FREQ\_HZ, UART\_BAUDRATE\_BOT); //set baudrate

XUartNs550\_SetLineControlReg(XPAR\_RS232\_UART\_0\_BASEADDR, XUN\_LCR\_8\_DATA\_BITS);

//Enable Control

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,128); //128 start command

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,132); //131 is safe mode, 132 is full mode

// Go forward

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,137); //drive command

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,0X00);

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,0xAA); //

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,0x80);

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,0);

// Go certain distance

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,156);

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,0);

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,5); //

// Stop

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,137);

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,0);

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,0);

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,0);

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,0);

In this code the first line of each block dictates the command. Opcode 128 tells the iRobot that we want to control it by sending it commands. We can then send it opcode 131, for safe mode, or 132, for full mode. Safe mode has some logic that forces the iRobot to go to an idle state and stop taking commands if certain sensors are actuated especially the cliff sensors (if you pick up the iRobot, these sensors will respond). Full mode bypasses that logic and will continue running despite sensors being on. This means that you can drive the iRobot down a ledge despite its cliff sensors activating.

Opcode 137 starts the drive sequence. This command must have 4 more opcodes sent with it. The next 2 opcodes define the speed at which the iRobot should drive. The first opcode represents the upper byte while the 2nd represents the lower byte. The opcodes can be written in hex as well to clarify. The velocity is in mm/s and when the opcodes are converted to decimal, the actual speed is revealed. Here we use 0x00AA = 170mm/s. The next two opcodes specify the turn radius. If we want the iRobot to drive straight we can send the two opcodes 128 and 0 (0x8000 in hex).

The next block dictates a wait distance command. We send opcode 156 and then two more bytes to represent the wait distance. Here the distance would be 5mm. Thus, the iRobot will continue until it reaches 5mm.

Afterwards, we just send another drive command that includes 4 bytes of 0s as a stop command.

And that’s how the iRobot moves forward. To move backwards, you simply have to specify a negative speed and a negative distance in two’s complement format. Turning in place and turning while moving are also possible and are specified in the open interface manual.

# Setting up the interrupt

By now it should be clear that it is important to know which header files are available to you in order to accurately control your peripherals. This link below will provide you with a data sheet on all the different types of IPs and their corresponding header files.

<http://courses.cs.washington.edu/courses/csep567/04sp/pdfs/xilinx_drivers.pdf>

Header files also will only be accessible if you have the correct IP Core activated on the FPGA.

The instructions for setting up the interrupt is best presented as code:

// Enable MicroBlaze Interrupts //

microblaze\_enable\_interrupts();

// Specify Interrupt Handler

XIntc\_RegisterHandler(XPAR\_XPS\_INTC\_0\_BASEADDR,XPAR\_XPS\_INTC\_0\_RS232\_UART\_0\_IP2INTC\_IRPT\_INTR,(XInterruptHandler)uart\_int\_handler,(void \*)XPAR\_RS232\_UART\_0\_BASEADDR);

// Start the interrupt controller

XIntc\_MasterEnable(XPAR\_INTC\_0\_BASEADDR);

// Enable UART interrupt in the interrupt controller

XIntc\_EnableIntr(XPAR\_XPS\_INTC\_0\_BASEADDR, XPAR\_RS232\_UART\_0\_IP2INTC\_IRPT\_MASK);

// Enable UART interrupt

XUartNs550\_EnableIntr(XPAR\_RS232\_UART\_0\_BASEADDR);

The process here is to first tell the FPGA to enable interrupts. Then link your specific interrupt handler to your interrupt device. Then start the interrupt controller, and then start the specific interrupt device in your controller. Thus, this process relies on the xintc\_l.h header which refers to the interrupt controller. The other values that you plug into the functions are taken from the xparameters.h file.

From xparameters.h:

/\* Definitions for peripheral RS232\_UART\_0 \*/

#define XPAR\_RS232\_UART\_0\_DEVICE\_ID 0

#define XPAR\_RS232\_UART\_0\_BASEADDR 0x83E20000

#define XPAR\_RS232\_UART\_0\_HIGHADDR 0x83E2FFFF

#define XPAR\_RS232\_UART\_0\_CLOCK\_FREQ\_HZ 125000000

…

#define XPAR\_INTC\_SINGLE\_BASEADDR 0x81800000

#define XPAR\_INTC\_SINGLE\_HIGHADDR 0x8180FFFF

#define XPAR\_INTC\_SINGLE\_DEVICE\_ID XPAR\_XPS\_INTC\_0\_DEVICE\_ID

#define XPAR\_XPS\_INTC\_0\_TYPE 0

#define XPAR\_RS232\_UART\_0\_IP2INTC\_IRPT\_MASK 0X000001

#define XPAR\_XPS\_INTC\_0\_RS232\_UART\_0\_IP2INTC\_IRPT\_INTR 0

#define XPAR\_RS232\_UART\_1\_IP2INTC\_IRPT\_MASK 0X000002

#define XPAR\_XPS\_INTC\_0\_RS232\_UART\_1\_IP2INTC\_IRPT\_INTR 1

#define XPAR\_XPS\_TIMEBASE\_WDT\_0\_WDT\_INTERRUPT\_MASK 0X000004

#define XPAR\_XPS\_INTC\_0\_XPS\_TIMEBASE\_WDT\_0\_WDT\_INTERRUPT\_INTR 2

#define XPAR\_XPS\_TIMEBASE\_WDT\_0\_TIMEBASE\_INTERRUPT\_MASK 0X000008

#define XPAR\_XPS\_INTC\_0\_XPS\_TIMEBASE\_WDT\_0\_TIMEBASE\_INTERRUPT\_INTR 3

#define XPAR\_XPS\_TIMER\_0\_INTERRUPT\_MASK 0X000010

#define XPAR\_XPS\_INTC\_0\_XPS\_TIMER\_0\_INTERRUPT\_INTR 4

Thus, everything needed can be found in the xparameters.h file.

From xintc\_l.h

void XIntc\_RegisterHandler(u32 BaseAddress, int InterruptId, XInterruptHandler Handler, void \*CallBackRef);

XIntc\_MasterEnable(BaseAddress)

XIntc\_EnableIntr(BaseAddress, EnableMask)

# Read Sensor Data

The sensor data can be read can be manually requested each time, but in this guide we use the continuous update instead. This command is referred to as stream in the manual. Data packets with the information we want will be sent across the RS232 line every 15ms. You can request any number of packets you want along with the type of packets. We’ll jump into the code to clarify.

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,148);

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,1);

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,7);

Opcode 148, indicates that you want to stream. The next opcode afterwards decides how many packets you want. Since we specify 1 here, we only want to send one more opcode to specify which data and 7 refers to the bumps and wheel drop sensor data.

Bumps and wheel drop sensor Data Format:



So we are only interested in the 5LSB.

Packet Stream Data:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 19 | 2 | 7 | What we want. | 182 |
| header | n-bytes | Packet ID | Packet Data | Checksum |

The image above shows how the data is sent by the packets. We have the header byte that helps determine the start of the packet. We are then sent a byte that represents the number of bytes till the checksum. Then the byte that describes what the data we asked for is follows. After, that we actually get the data we want. The rest are not important to save, only this byte is the fourth byte.

To decode this data we now set up our interrupt handler:

//globals

XGpio gp\_out; //to initialize the led’s

u8 sensor\_n = 0; //1byte values

u8 sensor\_id = 0;

u8 sensor\_data = 0;

u8 sensor\_cs = 0;

void uart\_int\_handler(void \*baseaddr\_p) {

u8 s\_header; //1 byte value

XGpio\_Initialize(&gp\_out, XPAR\_LEDS\_8BIT\_DEVICE\_ID);

XGpio\_SetDataDirection (&gp\_out, 1, 0x0);

XGpio\_DiscreteWrite (&gp\_out, 1,sensor\_data & 0x7F);

while (XUartNs550\_IsReceiveData(XPAR\_RS232\_UART\_0\_BASEADDR)) {

// read a character

switch(sensor\_byte\_count){

case 0: { //find header

s\_header = XUartNs550\_RecvByte(XPAR\_RS232\_UART\_0\_BASEADDR);

if(s\_header == 19){

sensor\_byte\_count++;

}

break;

}

case 1: { //n-bytes

sensor\_n = XUartNs550\_RecvByte(XPAR\_RS232\_UART\_0\_BASEADDR);

sensor\_byte\_count++;

break;

}

case 2: { //packet id

sensor\_id = XUartNs550\_RecvByte(XPAR\_RS232\_UART\_0\_BASEADDR);

sensor\_byte\_count++;

break;

}

case 3: { //sensor data we want

sensor\_data = XUartNs550\_RecvByte(XPAR\_RS232\_UART\_0\_BASEADDR);

sensor\_byte\_count++;

break;

}

case 4: { //checksum

sensor\_cs = XUartNs550\_RecvByte(XPAR\_RS232\_UART\_0\_BASEADDR);

sensor\_byte\_count = 0;

break;

}

}

}

}

In this code we also include the use of XGpio along with the LEDs on the FPGA. The set up for this is straightforward:

* Setup the instance for XGpio.
  + Here we call it gp\_out
* Initialize this instance to peripheral address
  + XGpio\_Initialize(&gp\_out, XPAR\_LEDS\_8BIT\_DEVICE\_ID);
    - Again the address value can be found in xparameters.h
* Determine the inputs and outputs.
  + Here the LEDs are viewed as a 1byte array.
  + XGpio\_SetDataDirection (&gp\_out, 1, 0x0);
    - Anything that is the second value here determines what is an input and output. So anything set to 1 is an input, and anything set to 0 is an output. In this case since we just want all the LEDs to be outputs we just use 0x0. If we want a particular LED configuration like InOutOutIn OutOutInIn, then we use 0x93 since in binary this is 1001 0011.
* We then write 1 to the values that correspond to the LED that we want to turn on.
  + XGpio\_DiscreteWrite (&gp\_out, 1,sensor\_data & 0x7F);
    - In this case we & 0x7F with sensor\_data because we want the sensor information to specify which LEDs turn on.
      * LED 0 = right bumper
      * LED 1 = left bumper
      * LED 2 = wheel drop right
      * LED 3 = wheel drop left
      * LED 4 = wheel drop caster (I think this is the casing or another one of the smaller wheels.

# Putting it Together

Here is the code for a sample project that will move forward forever until it bumps into something. It will also indicate on the FPGA LEDs which sensor is activated when it bumps into a wall or if the bumpers are pressed.

/\*-------------------------

iRobot Sample Code

This will make the iRobot Create move forward until the robot bumps into an object.

----------------------------\*/

#include <stdio.h>

#include <string.h>

#include "xgpio.h"

#include "xparameters.h"

#include "xutil.h"

#include "xuartlite\_l.h"

#include "platform.h"

#include "xuartns550\_l.h"

#include <xintc\_l.h>

#define UART\_BAUDRATE\_BOT 57600

void start();

void forward();

void sensor\_req();

u8 sensor\_n = 0;

u8 sensor\_id = 0;

u8 sensor\_data = 0;

u8 sensor\_cs = 0;

XGpio gp\_out;

void uart\_int\_handler(void \*baseaddr\_p) {

u8 s\_header;

XGpio\_Initialize(&gp\_out, XPAR\_LEDS\_8BIT\_DEVICE\_ID);

XGpio\_SetDataDirection (&gp\_out, 1, 0x0);

XGpio\_DiscreteWrite (&gp\_out, 1,sensor\_data & 0x7F);

while (XUartNs550\_IsReceiveData(XPAR\_RS232\_UART\_0\_BASEADDR)) {

// read a character

switch(sensor\_byte\_count){

case 0: { //check header

s\_header = XUartNs550\_RecvByte(XPAR\_RS232\_UART\_0\_BASEADDR);

if(s\_header == 19){

sensor\_byte\_count++;

}

break;

}

case 1: {

sensor\_n = XUartNs550\_RecvByte(XPAR\_RS232\_UART\_0\_BASEADDR);

sensor\_byte\_count++;

break;

}

case 2: { //get packet id

sensor\_id = XUartNs550\_RecvByte(XPAR\_RS232\_UART\_0\_BASEADDR);

sensor\_byte\_count++;

break;

}

case 3: { //get sensor data

sensor\_data = XUartNs550\_RecvByte(XPAR\_RS232\_UART\_0\_BASEADDR);

sensor\_byte\_count++;

break;

}

case 4: { //checksum

sensor\_cs = XUartNs550\_RecvByte(XPAR\_RS232\_UART\_0\_BASEADDR);

sensor\_byte\_count = 0;

break;

}

}

}

}

int main()

{

init\_platform();

//UART0 -> General Serial Communication

XUartNs550\_SetBaud(XPAR\_RS232\_UART\_0\_BASEADDR,XPAR\_RS232\_UART\_0\_CLOCK\_FREQ\_HZ, UART\_BAUDRATE\_BOT); //set baudrate

XUartNs550\_SetLineControlReg(XPAR\_RS232\_UART\_0\_BASEADDR, XUN\_LCR\_8\_DATA\_BITS);

Xuint8 rec\_data = 0XFF;

// Enable MicroBlaze Interrupts //

microblaze\_enable\_interrupts();

// Specify Interrupt Handler

XIntc\_RegisterHandler(XPAR\_XPS\_INTC\_0\_BASEADDR,XPAR\_XPS\_INTC\_0\_RS232\_UART\_0\_IP2INTC\_IRPT\_INTR,(XInterruptHandler)uart\_int\_handler,(void \*)XPAR\_RS232\_UART\_0\_BASEADDR);

// Start the interrupt controller

XIntc\_MasterEnable(XPAR\_INTC\_0\_BASEADDR);

// Enable UART interrupt in the interrupt controller

XIntc\_EnableIntr(XPAR\_XPS\_INTC\_0\_BASEADDR, XPAR\_RS232\_UART\_0\_IP2INTC\_IRPT\_MASK);

// Enable UART interrupt

XUartNs550\_EnableIntr(XPAR\_RS232\_UART\_0\_BASEADDR);

start();

sensor\_req();

while(1){

if(!(sensor\_data & 0x03)){ //robot will only move if sensors are not on

forward();

}

}

return 0;

}

void start(){

//Enable Control

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,128); //128 start command

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,132); //131 is safe mode, 132 is full mode

}

void forward(){

// Go forward

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,137); //drive command

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,0X00);

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,0xAA); //

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,0x80);

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,0);

// Go certain distance

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,156);

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,0);

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,5); //

// Stop

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,137);

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,0);

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,0);

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,0);

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,0);

}

void sensor\_req(){

//bumps and wheel drops data

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,148);

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,1);

XUartNs550\_SendByte(XPAR\_RS232\_UART\_0\_BASEADDR,7);

}