Robokid 2

Technical Notes

Jim Herd

Andy Haston

May 2022

Contents

[Robokid 2 1](#_Toc103809491)

[1 Introduction 3](#_Toc103809492)

[2 System 3](#_Toc103809493)

[2.1 Functional subsystems 3](#_Toc103809494)

[2.2 Technical subsystems 4](#_Toc103809495)

[3 Functional subsystems 6](#_Toc103809496)

[3.1 System management 6](#_Toc103809497)

[3.2 User interaction 8](#_Toc103809498)

[3.2.1 Joystick input 8](#_Toc103809499)

[3.2.2 Display 9](#_Toc103809500)

[3.2.3 Switches/LEDs 9](#_Toc103809501)

[3.2.4 Potentiometers 10](#_Toc103809502)

[3.2.5 Sound output 10](#_Toc103809503)

[4.2 Process structure of µP software : Simple I/O : GitHub Tag V1.0 11](#_Toc103809504)

[4.3 Process structure of µP software : Added sequencer : GitHub Tag V2.0 12](#_Toc103809505)

[4.4 Process structure of µP software : Added PID control : GitHub Tag V3.0 13](#_Toc103809506)

[Appendices 14](#_Toc103809507)

[Appendix A 14](#_Toc103809508)

[System constants and constraints 14](#_Toc103809509)

Figures

[Figure 1 General system structure 5](file:///C:\jth\Pico\jim_code\Robokid_Pico_FreeRTOS\Documents\Robokid_2_technical.docx#_Toc103797150)

Tables

[Table 1 Functional subsystems 3](#_Toc103797172)

[Table 2 Technical subsystems 4](#_Toc103797173)

[Table 3 Some Processor options 7](#_Toc103797174)

# 1 Introduction

This document brings together technical issues in the design of Robokid 2. He success of Robokid 1 means that many of the principles used in its design will be carried through to the new design. The fundamental change will be the use of more modern technology. A second document will look at the wider resource environment relevant to the running of a Robokid 2 project in schools.

# 2 System

## 2.1 Functional subsystems

The system is split into a number of functional subsystems

|  |  |
| --- | --- |
| Name | Responsibilities |
| System management | * Processor * Memory |
| User interaction | * USB SNES Gamepad * Push buttons * Potentiometers   + Allows user to input variable parameters     - eg. speed * LED displays * LCD display * Tone speaker |
| Actuation | * DC motors * Motor sensing |
| Comms Interfaces | * USB/FTDI Virtual com port * IR |
| Sensors | * Line sensors * Light sensors * Touch sensors |
| Power | * Battery * Charging * Condition sensing |
| Miscellaneous | * ?????? * RGB lighting |

Table 1 Functional subsystems

## 2.2 Technical subsystems

The above list of subsystems implies the need for the following technical components.

|  |  |
| --- | --- |
| Name | Responsibilities |
| Processor | * Microcontroller |
| USB | * SNES Gamepad (USB HID class) * Com port (USB CDC class) |
| Digital I/O | * Push buttons * LED displays * IR receiver |
| Analogue I/O | * IR line sensors * Potentiometers |
| PWM Output | * Motor control   + H-bridge control * Sound output   + Tone Speaker |
| Interfaces | * I2C/SPI   + LCD display * Serial   + FTDI Virtual Com Port |
| Power system | * Li-Ion battery   + Regulation   + Charging   + Monitoring |

Table 2 Technical subsystems

Figure 1 shows the connections between the subsystems and the central microcomputer.

User Output

User Input

Processor

SNES

Gamepad

I2C/SPI

LCD

display

USB

Push switches

A, B, C, and D

Digital

Digital

LED displays

A, B, C, and D

Analogue

3 line

sensors

PWM

Sounder

(speaker)

Analogue

3 variable

potentiometers

PWM

Digital

2 motor rotation sensors

(possible?)

2 DC motors

Serial

Serial to USB

Digital

IR link

USB

Analogue

Voltage monitor

USB

Charging

link

Li-Ion Battery system

Figure 1 General system structure

# 3 Functional subsystems

## 3.1 System management

The processor provides the “smarts” of the system, but does not need to be a top-range device as all likely activities are not intrinsically complex. Clearly, cost will be an issue and we will look to use a cost-effective device.

The following table gives some options and their pros and cons

|  |  |
| --- | --- |
| Device | Notes |
| Raspberry Pi zero | Bottom of the range of Raspberry Pi range of computers.  Pros   * Powerful for its size. * cost range : mid to high (relative to other options) * Wireless capability (see later discussion) * Commitment to longevity of production   Cons   * Volume supply issues * Needs SD card (could be lost) * Needs to be shutdown correctly to prevent SD card corruption * No analogue input * Not available as a chip, therefore would have to use the board. |
| Raspberry RP2040 | Recent Arm based microcontroller chip from the makers of the Raspberry Pi range of computers.  Pros   * Modern chip with full range of required interfaces * Available as a chip. * High clock rate with two cores * British designed chip * Cost range : low * Good documentation and software tools * Commitment to longevity of production * Family only has one device – easy decision.   Cons   * No wireless (see later) * Relatively new device * Family only has one device – no choice. |

|  |  |
| --- | --- |
| ST STM32 | 32-bit Arm based microcontrollers from ST Microelectronics  Pros   * Good range of devices. Plenty of choice. * Cost range : low to mid * Commitment to longevity of production   Cons   * Unsure about documentation and support * Extensive range of devices * Unsure about supply situation |
| ESP32 | Espressif developed microcontroller with embedded wireless and Bluetooth.  Pros   * Very capable device * Embedded wireless TCP/IP stack (see later)   Cons   * Complex to use as a chip (need to use module) * Cost range : mid * Unsure about documentation and support * Unsure about supply situation |
| Arduino | Pros   * Heavily used system in one-off applications. * Good support   Cons   * Little used in volume applications * Cost range : mid (probably have to use a module rather than a chip) |
| Others | ???????????????????? |

Table 3 Some Processor options

In a low performance, low quantity application like Robokid 2, there is unlikely to be a killer reason for choosing one device over another. The easiest option to discount is the Raspberry Pi Zero. The power on/off and SD card issues would not work well in a school classroom environment.

The list makes a number of references to wireless capability. The questions to be addressed : is it worthwhile paying the extra premium to have this capability? On a cursory consideration it would seem justifiable for the expenditure of and extra £2/£3 pounds. The ESP32 would seem an ideal solution. However, on consideration of the environment in which Robokid 2 will operate – the primary school classroom, there are a number of factors to consider

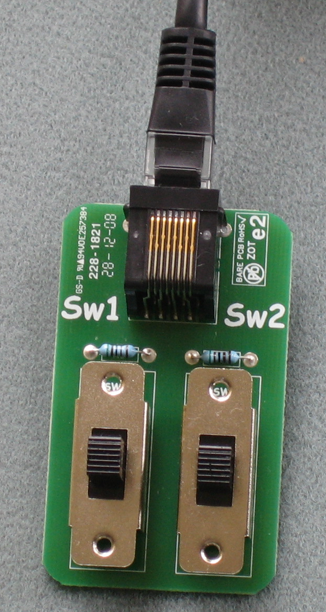
1. The majority of primary school teachers to not have a background in STEM/IT. In general they are very interested in the area, but have a limited knowledge.
2. The majority of primary schools do not have an in-house IT department to fix/solve any issues that may arise with either Wi-Fi or Bluetooth networking.
3. Schools are unlikely to have open Wi-Fi networks and education authority IT departments are unlike to grant access to tens of Wi-Fi devices without significant assessment on their part. In fact, I would think that access would be denied. I’m not sure Heriot-Watt IT would allow this!

For these reasons, I would suggest that Wi-Fi/Bluetooth should not be part of a schools robot which would remove the ESP32 device from consideration.

That leaves RP2040, STM32 and Arduino. I have used an Arduino in a small one-off project and it worked well, was quick to code and had lots of libraries. Beyond this, I have no experience of higher volume applications. My impression is that it is not heavily used for such applications. Most of the details about Arduino focus on boards rather than chips. For this reason I will remove the Arduino from my list.

Just RP2040 and STM32 left. Probably little to choose between these two. I am using an STM32 chip for another robot project which needs a CAN interface. However, I decided to use the RP2040 for my Robokid 2 test system. It is fast, 133MHz, dual core, plenty of memory, and cheap. As a UK designed chip, there may be some publicity traction in its use. From a technical perspective, my initial impressions are good.

|  |  |
| --- | --- |
| Proposal 1 | Build test system with the RP2040 chip. |



## 3.2 User interaction

### 3.2.1 Joystick input

Robokid 1 used a crude in-house designed “joystick” with two 3-position. This worked and had the benefit of simplicity and cheapness, but at the expense of flexibility.

More recent processors have USB interfaces that give the opportunity to use the processors as a host to a USB joystick or gamepad. The RP2040 has such an interface.



Initial experiments have been conducted with a SNES gamepad, as below

This is a relatively cheap option with a total of 12 digital buttons which will give more flexibility in the set of “joystick” activities.

|  |  |
| --- | --- |
| Proposal 2 | Build test to allow the use of a USB joystick/gamepad |

### 3.2.2 Display

The robot system needs some way of displaying information to the user. Robokid 1 used a simple two character 7-segment display which can show a reasonable representation of most the alphabet. The system incorporates a scrolling system to display simple text.

We need a better way to communicate with the user but within the constraints of cost and size. A good compromise option is the SSD1306 OLED display

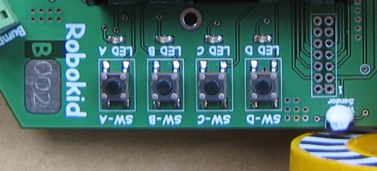
Although quite small (0.96 inch in the horizontal direction) the OLED technology gives a clear result. They have a display area of 128 by 64 pixels and access is at the pixel level. Thus, all user messages need to be built as a map of pixels before transfer to the display. These are very common devices and are supported by a number of open-source C libraries. The interface can be either I2C or SPI.

For the demo system I have modified a library to maximise the speed of update which provides a display area of 4 rows of 15 characters with vertical scrolling capability to show bigger messages. Both I2C and SPI interfaces have been tested.

|  |  |
| --- | --- |
| Proposal 3 | Use SSD1306 to provide the display |

### 3.2.3 Switches/LEDs

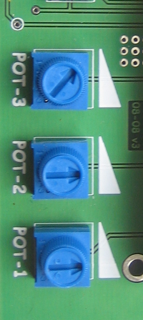
In Robokid 1, input from the user to set modes was with a set of four push button switches. This proved ideal, with each LED lit to highlight an active switch. IN the videos and literature, it was useful to specify Robokid operating modes as a string of ABCD button press values For example, sequence ADDDAA set Robokid into a line follow mode.



I would suggest that the same would be used on Robokid 2.

|  |  |
| --- | --- |
| Proposal 4 | Include 4 push switches and associated LEDs. |

### 3.2.4 Potentiometers

Switches can only give a Yes/No inputs. Sometimes we want the user to input a value, e.g. required speed. Without attachment to a computer, we need a different approach. In Robokid 1 we used three variable resistors, called potentiometers. The processor can read the potentiometer value and relate it to the quantity being specified. These potentiometers are linear therefore the user is presented with a well-behaved input.

|  |  |
| --- | --- |
| Proposal 5 | Include 3 potentiometers for parameter input |

### 3.2.5 Sound output

Some form of sound output provides good feedback to inputs and events. For example, acceptance of a button press, or detection of a black line when in bump mode. Given that there may be as many as 13 robots in a classroom, speech output could be easily drowned out by the general class noise. Loud buzzes would seem suitable.

There was a piezo buzzer on Robokid 1, but because it was not particularly loud and was placed on the underneath of the circuit board, it was not effective. Better placement and the use of and amplifer feeding a PCB speaker should give a more effective system.

|  |  |
| --- | --- |
| Proposal 6 | Include amplified PCB speaker giving tones and buzzes. |

## 4.2 Process structure of µP software : Simple I/O : GitHub Tag V1.0

The uP runs the MBED RTOS software. The initial test structure uses the following structure with three processes (threads) and two mailbox queues.

FPGA

Com

Port

High level

controller

P3

P2

Low Level controller (µP)

P1

Queue1

Queue2

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Program name** | **Notes** |
| **P1** | Process | read\_from\_HLcontrol\_task | * Read ASCII command string from controlling computer * Send FPGA related commands to FPGA command queue * Implement non-FPGA commands |
| **P2** | Process | write\_to\_HLcontrol\_task | * Take ASCII reply strings from reply queue and send to controlling computer |
| **P3** | Process | FPGA\_IO\_task | * Take FPGA command from FPGA command queue and Implement on the FPGA through the 8-bit bi-directional bus. |
| **Queue 1** | Mailbox | FPGA\_cmd\_queue | * FIFO queue of LLcontrol (uP) commands coded as ASCII strings |
| **Queue 2** | Mailbox | HLcontrol\_reply\_queue | * FIFO queue of binary coded FPGA register commands |

This structure provides the basic system to test and exercise the FPGA hardware. The ASCII command format allows easy access from a high level control computer e.g. PC, Raspberry Pi, etc.

Testing uses a windows laptop with a C# program (March 2021).

## 4.3 Process structure of µP software : Added sequencer : GitHub Tag V2.0

(30/3/21 Design only)

Added process (P4) to V1.0 design that can initiate and control a sequence of FPGA commands, e.g., execute timed reading of an encoder channel as part of testing. Works by injecting FPGA commands into “Queue 1” and receiving results through “Queue 4”. A specific PORT number would be assigned to this activity.

FPGA

Com

Port

High level

controller

P3

P2

Low Level controller (µP)

P1

Queue1

Queue2

P4

Queue3

Queue4

Additional objects to V1

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Program name** | **Notes** |
| **P4** | Process | sequencer\_task | * Implements sequence of FPGA commands, specifically for test purposes. |
| **Queue 3** | Mailbox | sequencer\_cmd\_queue | * FIFO queue of LLcontrol (uP) sequence commands |
| **Queue 4** | Mailbox | sequencer\_reply\_queue | * FIFO queue of results from executed FPGA commands |

## 4.4 Process structure of µP software : Added PID control : GitHub Tag V3.0

The third development is to add PID motor control capability. The system has the initial capability to run four independent PID loops to control four motors.

FPGA

Com

Port

High level

controller

P3

P2

Low Level controller (µP)

P1

Queue1

Queue2

P4

Queue3

Queue4

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Program name** | **Notes** |
| **P5x** | Process | PID\_task | * Implements a PID motor control algorithm with input from a quadrature encoder unit. |
| **Queue 5x** | Mailbox | sequencer\_cmd\_queue | * FIFO queue of LLcontrol (uP) sequence commands |
| **Queue 4** | Mailbox | sequencer\_reply\_queue | * FIFO queue of results from executed FPGA commands |

# Appendices

## Appendix A

### System constants and constraints