**Software**

Communication between the PC and MBED was needed for the game to operate on incoming sensor data and the servos to act on outgoing commands from the PC to the MBED, to the FPGA. The diagram below shows how the communication was structured. The same structure was used for maintenance mode and operate mode.

A close up of text on a white background

Description automatically generated

Visual C#

The games were written in C# programming language using visual studio in the windows forms application format. The skeleton project for each game was downloaded from GitHub, from there the code was edited to work with the MBED device.

Serial port connection was set up in the game by defining the name of the port which the MBED device was connected over. By default, this was “COM3” and the baud rate was set to 9600. The serial port was then opened upon loading of the game form.

A screenshot of a cell phone

Description automatically generated

Event handling was used to intake incoming data from the MBED device to the PC over the serial port. This worked better than while loops, continuously reading the incoming data which often slowed down the game and created logistical problems. The event was defined to be any new incoming data from the serial port.

Every time the event occurs it calls the object shown below. This requests colour sensor data and then calls upon a function which would uses the incoming data to determine direction. A loop is created in this way as every time a new event occurs, data is requested which in turn creates a new event as new data is present. The ‘Thread.sleep(200)’ line in the code is present to temporarily stop the loop to prevent crashes or overload of colour data. The distance sensor is read ten times before the game begins by parsing incoming string to an int32 variable and setting the range of allowed values between 0-10cm from the sensor.

A screenshot of a cell phone

Description automatically generated

Maintenance Mode

Under maintenance mode, the user could control the servos manually as well as receive data from the distance sensor, colour sensor or card reader in the text box. The form uses a combo box to select from available COM ports. Once opened, the form is fully operational with the MBED device. There are three positions options available to test for each servo; the face angle can be set to 0°, 90° or 180° and the prize motor can be set up, down or up then down.

A screenshot of a cell phone

Description automatically generated

MBED

Instead of sending all the sensor data at once from the embed and then parsing the serial port data in C#, it was easier to create a call-send system over the serial port. The MBED code was set up to only send the sensor data which it was being asked for by the PC. For example, to receive data from the colour sensor, the letter ‘h’ would have to be sent from the PC. This system was created to streamline our game experience as parsing the correct data while the serial port was open, caused latency.

This done by separating the sensor split it up into all incoming strings which could be called upon at the appropriate time in the code.

This was done by setting up the MBED code to only send relevant data back to the PC when it received the correct letter. This allowed less data being sent at any one time over the serial port which ultimately made the game run more smoothly.

The below table shows the commands used during serial port communication with C# and the MBED controller.

|  |  |
| --- | --- |
| Command | Function |
| ‘a’ | Turn Prize motor up and down (release one prize) |
| ‘b’ | Move prize motor up |
| ‘c’ | Move prize motor down |
| ‘d’ | Move totem head to left |
| ‘e’ | Move totem head to right |
| ‘f’ | Move totem head to middle |
| ‘g’ | Send distance sensor data to PC |
| ‘h’ | Send colour sensor data to PC |
| ‘i’ | Send card reader data to PC |

This call-send system worked in this project as only one type of sensor data needed to be read at a time during the game. This would not work if two or more types of data was needed at once.

During the build of the Totem, the TC3472 colour sensor needed to be re-calibrated constantly due to it’s sensitivity. It would react differently depending on its light environment which proved to be very challenging as it effected the operation of the games. This was overcome by selecting only two colour ranges to characterize each output colour. The two colour ranges that were selected would be the ones with the most contrast from all others. This allowed us to create a larger range of characterization over these colours and ultimately enabled them to have a larger variance without falling out of their characterization.