RetroChallenge 2021/10 competition entry: Using the Commodore 64 to control a remote control car.

A picture containing text, indoor

Description automatically generated

Description:

The Commodore 64 was released in 1982 and quickly became the best selling computer of all time. Many users treated the system as a games controller, but it was a real computer and could also be used as a microcontroller, using it’s ports like the User Port.

In this project I wanted to use this retro computer to control a remote control (RC) car. Using an inexpensive RC car and re-fitting it with an Arduino to allow connection to a radio transceiver (NRF24L01). On the Commodore 64 side I decided to use RS232, which on the C64 needs an adapter to bring it up to the voltages needed. The adapter released by Commodore was called the VIC-1011A.

Control of the RC car can be via a joystick and recorded (to a floppy disk) for later autonomous 'play back' by the C64. The C64 will use RS232 to a microcontroller with a wireless transceiver that can connect to the RC car. Commodore 64 control on an RC car By Steve Smit Started in September 2021

The code on the Arduinos used in this project is written using the IDE in the standard C++. The code on the Commodore 64 will initially be in BASIC, but will likely be re-written in either Forth (DurexForth) or assembler.

22nd September 2021 : First piece of progress, get C64 to send bytes representing the directions of the joystick by RS232 to a transceiver to a receiver to confirm 'commands' can be sent. Code on C64 written in DurexForth: <https://youtu.be/1RyrSl5RLjI>

A picture containing text, indoor, electronics

Description automatically generated

I purchased 1/14 scale RC ‘Monster Truck’ off-road 2WD car for just under AUD$40.

A picture containing text

Description automatically generated A red race car

Description automatically generated with medium confidence

I first tried to ‘hack’ the controller to see if perhaps this might mean I could leave the electronics in the car unmodified. This initially worked!:

<https://youtu.be/TNTKXlKmA1Y>

But, for no apparent reason, the ‘steering’ motor, while making a straining noise, now doesn’t seem to have the torque to turn the car anymore. Hacking the controller wasn’t ideal in any case, as I wanted to have more control via the Commodore 64. I decided to open the car up, firstly to see if I could address the steering motor issue, but also to replace the motor controller with a L298N dual h-bridge motor controller module and hook up an Arduino to do some testing.

A picture containing indoor, computer, floor, keyboard

Description automatically generated

Even directly connecting to the ‘steering’ motor didn’t fix the lack of movement issue. I have ordered a few replacement DC motors as an option to fix the issue. An alternative is to replace the steering mechanism with a servo motor. Here is how the current DC motor controls the steering:

A close-up of a car engine

Description automatically generated with medium confidence

While I wait for replacement items, I better get onto the code on the C64. I decided that the opening screen should offer the user 4 choices:

F1 = Operate RC car without recording

F3 = Operate RC car with recording

F5 = Show saved sequences

F7 = “Play” current sequence in memory , or something like this

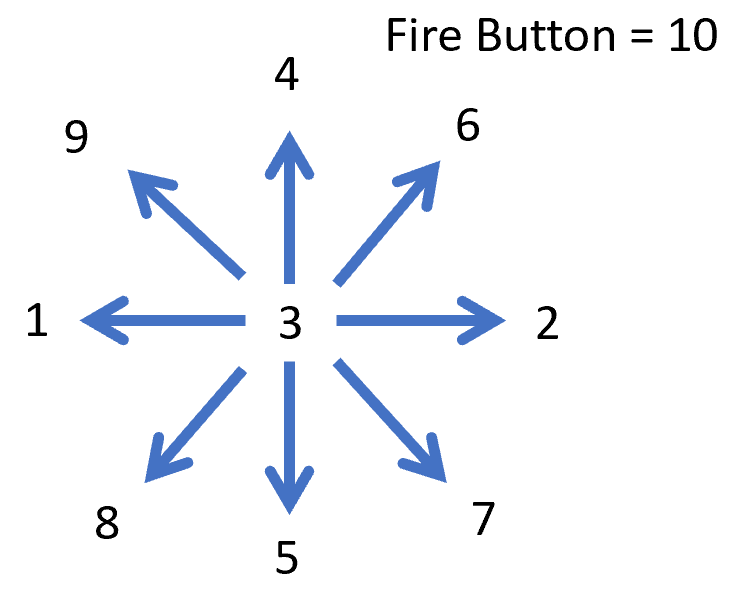
I realized that I didn’t know what byte is returned if someone presses the Function keys (F1, F3, etc.) so I wrote a quick basic program to get the values:

A blue screen with white text

Description automatically generated with low confidence

From this I see that F1 = 133, F3 = 134, F5 = 136 and F7 = 137.

For the joystick movements, I decided that each direction and the fire button be given a value that would be sent as a command to the RC car.

On a Commodore 64 the joystick in port 1 can be read by reading the contents of 56321 ($DC01). In BASIC this would be PEEK(56321).

I have chosen to write my code in assembler, so this is LDA $DC01

The joystick default is all 1s, i.e. 255 or $FF

Buttons (fire or a direction) pull bits low in the byte. Fire on it’s own is then = 239 or $EF

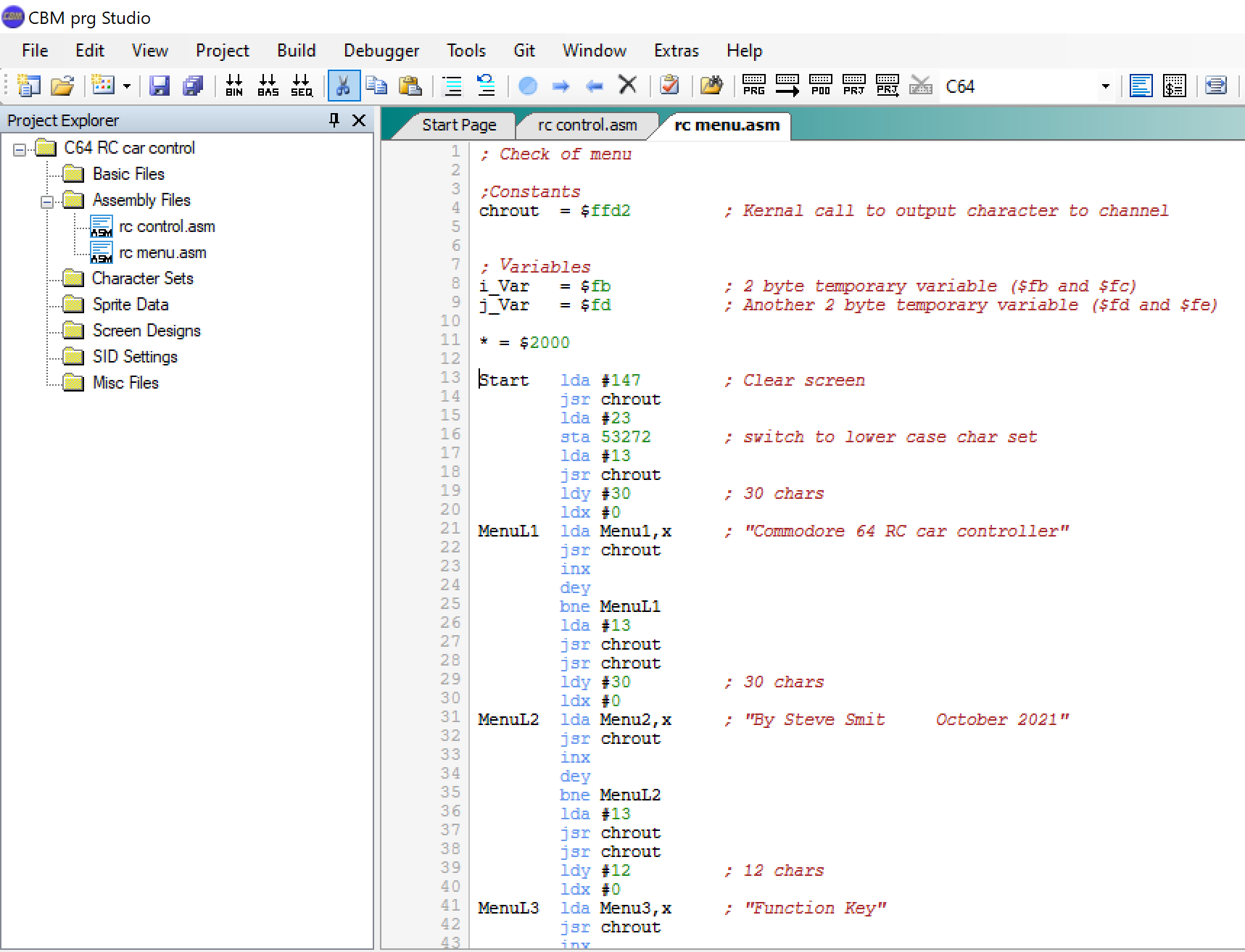
Up = 254, Down 253, Left 251, Right 247,

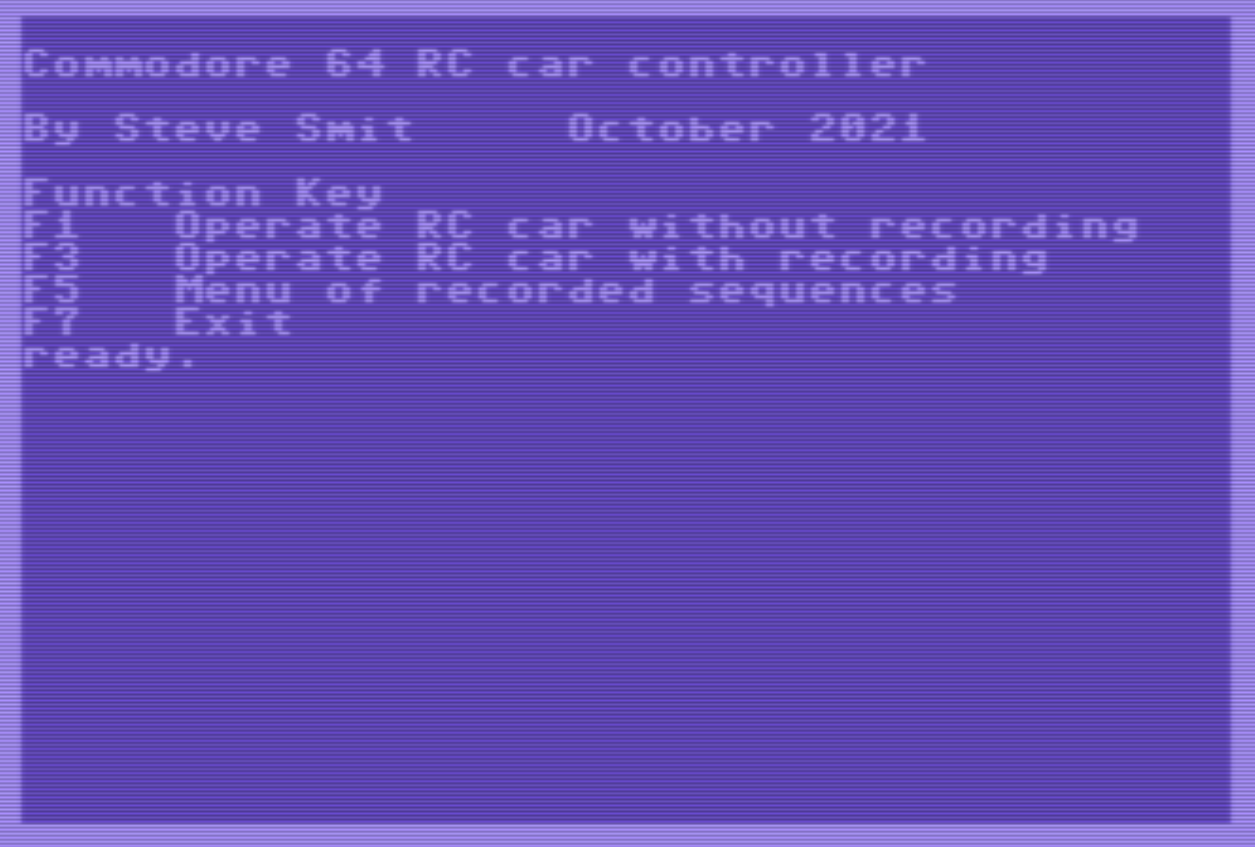
Up&Left 250, Up&Right 246, Down&Left 249,

and Down&Right 245. I then convert these to the commands in the diagram to the left.

27 September 2021 – Assembly programming

I’m still not sure what language to use on the Commodore 64, but I know that Assembly has the maximum level of control and speed, so l will make a start and see how I go. I’m using an IDE called CBM Prg Studio to enter the assembly that compiles the machine code:





Well, not the flashiest of menus, but it’s a start.

28 September 2021 – car steering motor repair

The replacement motor I ordered arrived and it fits perfectly and works! The Arduino sketch I used to test the car just spins the rear motor forward then backward, then confirms the front steering can be done. <https://youtu.be/1il9svvlH1M>

A close-up of a circuit board

Description automatically generated with low confidence

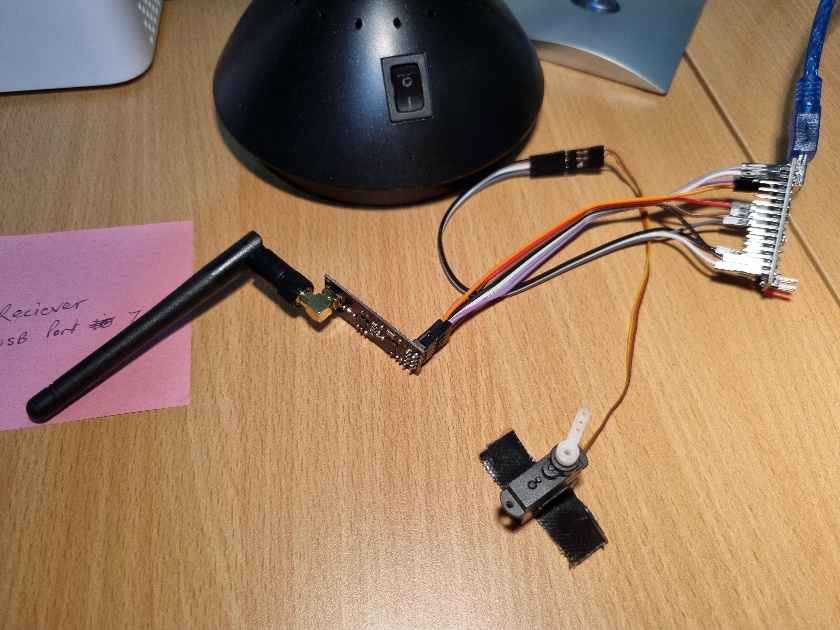
I still have a lot of coding to do on the Commodore 64, which I am persisting with Assembly language.

29 September 2021 – More machine coding

Now that I have a reasonable menu displayed, I need read the keyboard and act if one of the Function Keys is pressed.

1 October 2021

As the car has limited space, I decided to see if I could use an Arduino Nano in the car instead of the UNO. My first test was to see if I can send the commands for left, centre and right, including using a small servo to show this in action, and it worked! <https://youtu.be/7N0zTxdJpyA>

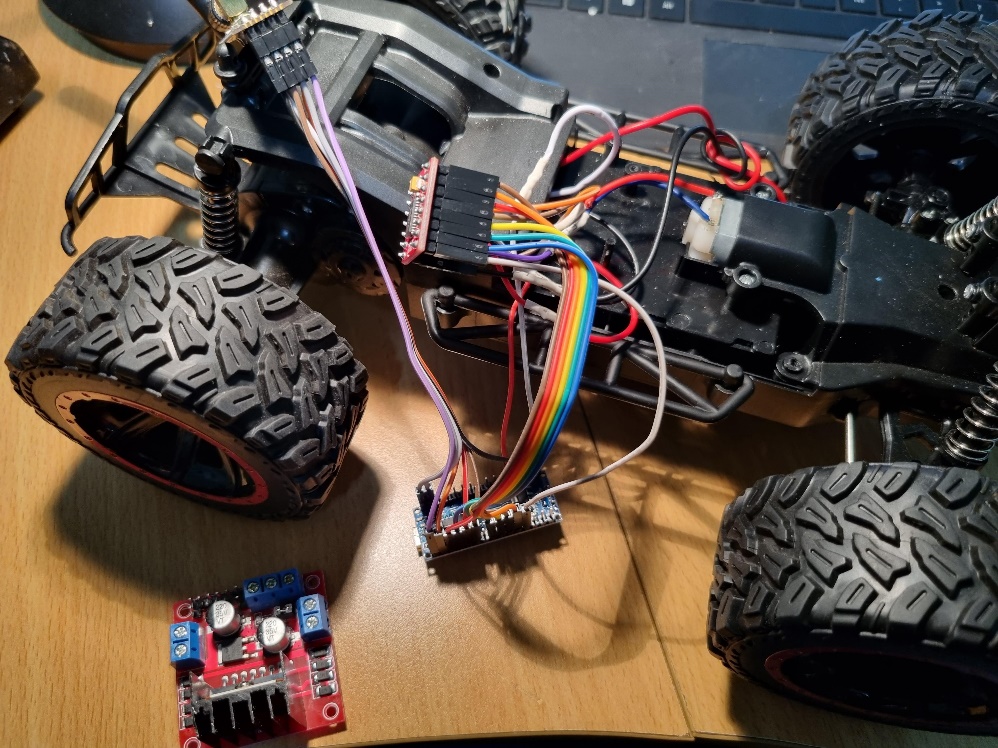


2nd October 2021 – Testing Nano in car with radio transceiver connected

After changing the Arduino UNO to an Arduino Nano, I did another test to confirm this works in the RC car. <https://youtu.be/uOMSnjM_ak8> . I’m pleased to see that it worked fine.

3rd October 2021 – Test change from L298N to TB6612FNG

I also decided that the L298N is a rather bulky device and loses around 1.4v from the battery, that I should see if I can use the more efficient and smaller TB6612FNG H-Bridge instead.



Of course, this also needed to be tested <https://youtu.be/IEmlqmNZZGI>

I then updated both the Arduino sketch and C64 programs to test also the forward and reverse control. I’m pleased to see that this works too! <https://youtu.be/RIUS3Q_FON8>

4th October 2021 – Code on Nano (in car) updated to support all directions

While my initial test for forward, reverse, left and right worked independently of each other, I also need to be able to support both moving forward while turning, as per the diagonal directions of joystick. I also wanted a way to avoid full power going to the front steering motor when it’s reached the end of travel. This got more complicated that I anticipated, as I need to use millisecond timers and keep track of duration of time during turning as well as when already in fully locked turned position where I wanted to reduce power (down from 250 to 150 for instance).

I used a flag I called TTFlag (which stood for Turning Time Flag) which would be false when the steering motor was in the centre position. If the flag wasn’t false, it meant that the steering motor was already turning under full power, so I needed to also read a variable that was measuring time in milliseconds to know when to drop power to the steering motor from full to hold (i.e. 250 down to 150). I am pleased to report that I succeeded in getting this sorted out ( <https://youtu.be/m04mfECTdz8> ). Now to re-charge the battery, secure the electronics and aerial and replace the top of the RC car, and give it a test on the floor.

5th October 2021 – First floor test (without securing the electronics or adding the top of the RC car)

OK, the RC car is working (sort of). All the controls are being properly interpreted, but there appears to be an excess of current for the motor controller. Fortunately, the motor controller has over-current/heating protection but this is resulting in jerky movement after less than 30 seconds of use. <https://youtu.be/hHUsokOsonM> . What I can be happy about is that all the other elements outside of the Commodore 64 seem to be working, so it’s just the code on the Commodore 64 that needs to be done now.

6th October 2021 – Coding on the Commodore 64

I decided to write my code in assembler. The Commodore 64 uses a MOS 6510 processor, which is essentially the same as the famous 6502 used in other systems like the Apple II, Atari 2600, BBC Micro and Nintendo Entertainment System, just to name a few. Assembler is more challenging, but the reward is that the code is small and fast. I used an IDE (Integrated Development Environment) called CBM prg Studio (<https://www.ajordison.co.uk/>), written by Arthur Jordison.

The program so far sets up the RS232 port at 1200 baud, displays the main menu and F1 (Operate RX car without recording) is functional <https://youtu.be/LmQ5uK1qb5w>

Graphical user interface

Description automatically generated with low confidence

October 7th 2021 – Testing assembler to control RC car without recording

I worked on and completed a new assembler code routine called “RC CAR.ASM”. The code now waits until there is activity on the joystick before sending the appropriate command out via RS232. On the screen is also shown characters representing the directions the user is moving the joystick. For this function, exit is by pressing the fire button. <https://youtu.be/aGeefbiyoig>

Now I need to move on to writing code that will store the sequence, including the timing between each movement of the joystick. The code will be very similar to above, except that I will allocate the fire button as start and stop of the sequence recording. To exit I will use something like the space bar. This next coding will be the most challenging, and I still have the ‘play-back’ function code to write after that (which shouldn’t be as difficult, I hope).