A Graphics Terminal with TTGO-VGA32

Martin Hepperle, July 2020

The TTGO VGA-32 is a small board which uses a solder-on ESP32 processor module. This processor is a quite powerful member of the ESP processor family and offers many interesting features including interaction with wireless networks. Similar to the ESP8266, its low cost makes it attractive choice for many hobby development boards. A set of libraries for the Arduino environment makes it easy to use many of its features.

1. Hardware

The TTGO VGA-32 is available in several incarnations, currently in versions V1.2 to V1.4. I used a V1.2 board.

It has interfaces to make a small multi-purpose computer system:

- Standard PS/2 keyboard,
- Standard PS/2 Mouse,
- Standard VGA Monitor,
- Power supply,
- Audio output,
- Serial Interface.

1.1. Serial Connection

The V1.2 board comes with a 4-pin pad arrangement, which accepts a pin header with 2 mm pitch. The sequence of the four pins equals that of many common TTL-RS-232C adapter boards.

Label	3.3V	DIO12	DIO02	GND
Purpose	VCC	RXD	TXD	GND

Table 1: Layout of the header pads on the TTGO-VGA32 board.

The four header pads as well as the FabGL examples (which use GPIO 2 and 12 for serial communication) suggest connecting these pads to a serial interface.

In fact everything worked fine - as long the GPIO pin 12 was connected after the system had been powered up.

However, if it is connected to RXD at boot time, the system won't start. It is not really practical to disconnect and connect the RXD line at each system start.

After reading the ESP32 datasheet I learned that the pin GPIO 12 is called MTDI at boot time. It determines the voltage used to drive the Flash memory (unconnected = Low = 3.3V). If this pin is connected to the RXD line of the idling TTL-RS232C converter, it is driven high and the Flash voltage is set to 1.8V. This is too low for the 3.3V system and prevents booting.

After soldering a wire to the neighboring GPIO 14 pad of the ESP32 module and connecting it (instead of GPIO 12) to RXD of the TTl-RS232C converter, the system started without any problems.

Conclusion: the pin header seems to be designed for other purposes, not as a serial interface. This is contrary to what the examples in the FabGL library and its layout suggest.

In the sketch I used `Serial` for all communication and replaced the initialization in the `setup()` routine like so:

```
// OLD: GPIO 12 blocks boot process
//  Serial.begin(BAUD_RATE, SERIAL_8N1, 12, 2);

// NEW: for TTGO-VGA32 use GPIO 14 (RX) and 2 (TX)
Serial.begin(BAUD_RATE, SERIAL_8N1, 14, 2);
```

Since then, everything worked as expected.

1.2. Power Connectors

Two connectors can be used to provide power:

- a Micro USB connector can be used to plug in a common 5V USB power supply,
- a small two pin connector can be used to connect to a 3,6V Lithium-Polymer battery.

There may be a charging circuit on the board, but you should use this only after careful monitoring the charginging case. There seems to exist an earlier production run of these board where a fuse was mounted instead of a diode, which could lead to overcharging a LiPo battery and subsequent fire.

1.3. Audio Connectors

Two audio connectors are mounted side by side and allow connecting the output signal of the 1W mono audio amplifier NS4150:

- a connector with two pins and a white plastic rim for an internal audio output device like a small loudspeaker,
- a jack for plugging in headphones or other audio systems.

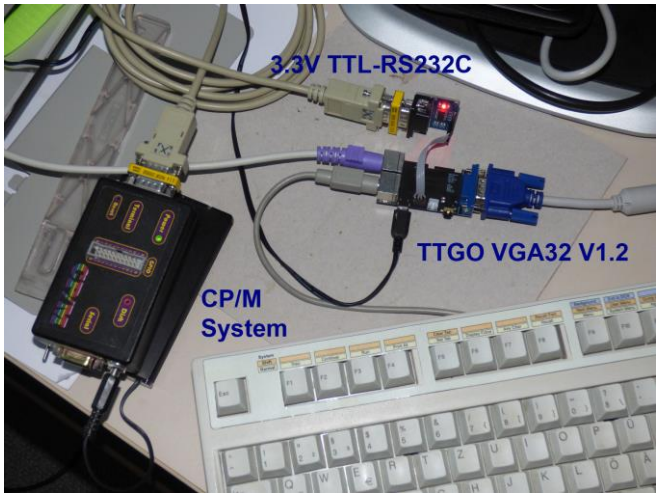

Figure 1 Connecting the power supply, keyboard, mouse, monitor and serial interface is easy.

2. Software

A very interesting piece of software for the ESP32 is the FabGL library which offers many useful functions for video and audio output. This library can be used with the VGA-32 board. One of the examples included with the FabGL library is a Terminal emulator which supports ANSI escape sequences.

I have taken this example and extended it to add Tektronix control sequences for monochrome vector graphics capability.

First I added an audible BELL because the FabGL library already has audio capabilities. In order to make error beeps audible I added code to recognize the ^G (BELL, 0x07) control character.

Secondly, I added an interpreter for Tektronix control sequences for monochrome vector graphics capability. For this purpose the Terminal class was sub-classed into a TekTerminal class and a handful of the private methods of the Terminal class had to be declared virtual to allow overriding them in the TekTerminal class. The result is a mixed emulation of an ANSI terminal which can be switched into Tektronix emulation mode.

To switch into Tektronix emulation the escape sequence

```
ESC [ ? 38 h
```

has to be used. This is a sequence also use by xterm to open its Tektronix window.

Switching back from Tektronix mode to ANSI mode is accomplished by the sequence

```
ESC ETX
```

Notes:

- Different line types (1…5) are translated into different intensity levels, e.g. 5 produces a faint green line.
- Graphical input GIN mode is not supported.
- The available resolution of 640 x 350 pixels is considerably coarser than that of a real Tektronix terminal. Therefore fine details will be lost.

2.1. Using a German Keyboard Layout

For using my German keyboard I had to switch to the desired layout by inserting a line after the PS2Controller has been started:

```
[...]
  PS2Controller.begin(PS2Preset::KeyboardPort0);
  /* switch layout to German */
#ifdef GERMAN
  PS2Controller.keyboard()->setLayout(&fabgl::GermanLayout);
#endif
[...]
```

There are still some smaller inconsistencies in this layout, e.g. with the numeric keypad and some extra keys, but it is very useable.

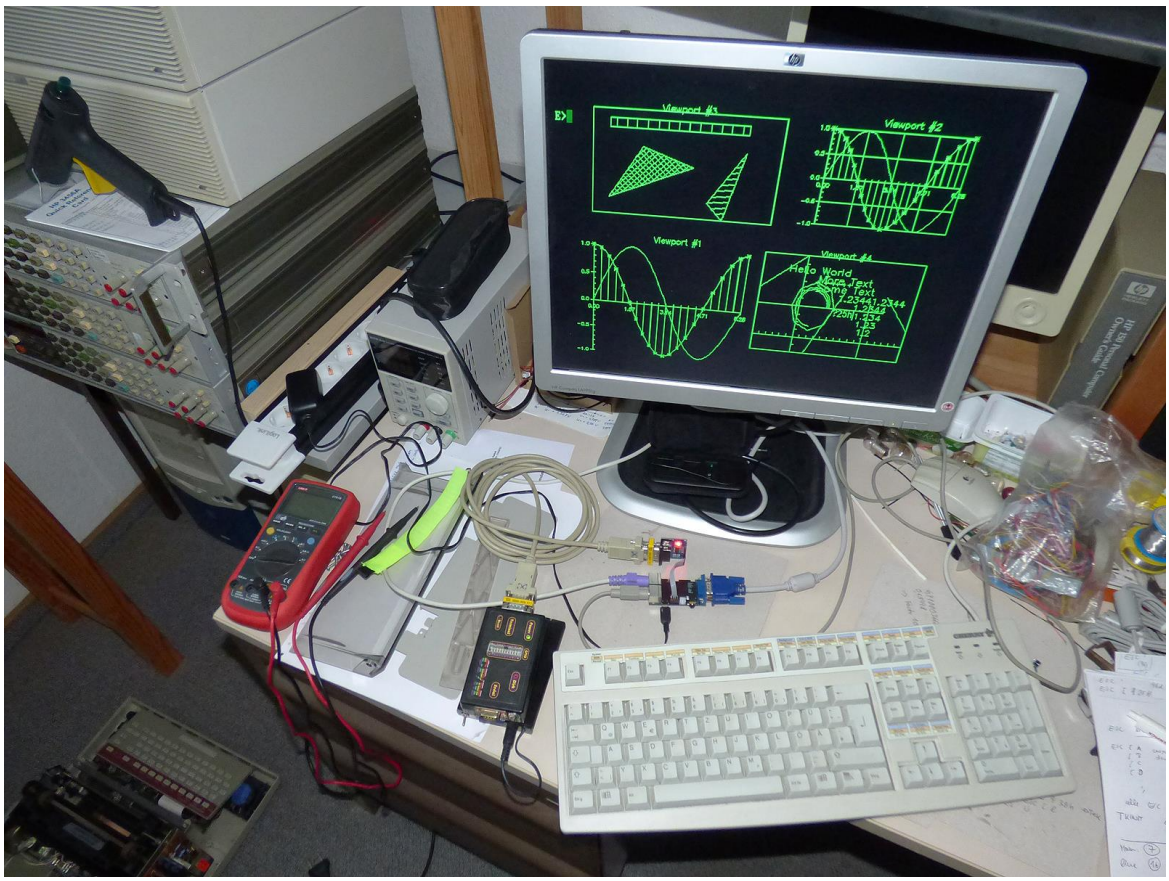

Figure 2 The terminal in action with one of my CP/M-80 GSX test programs.

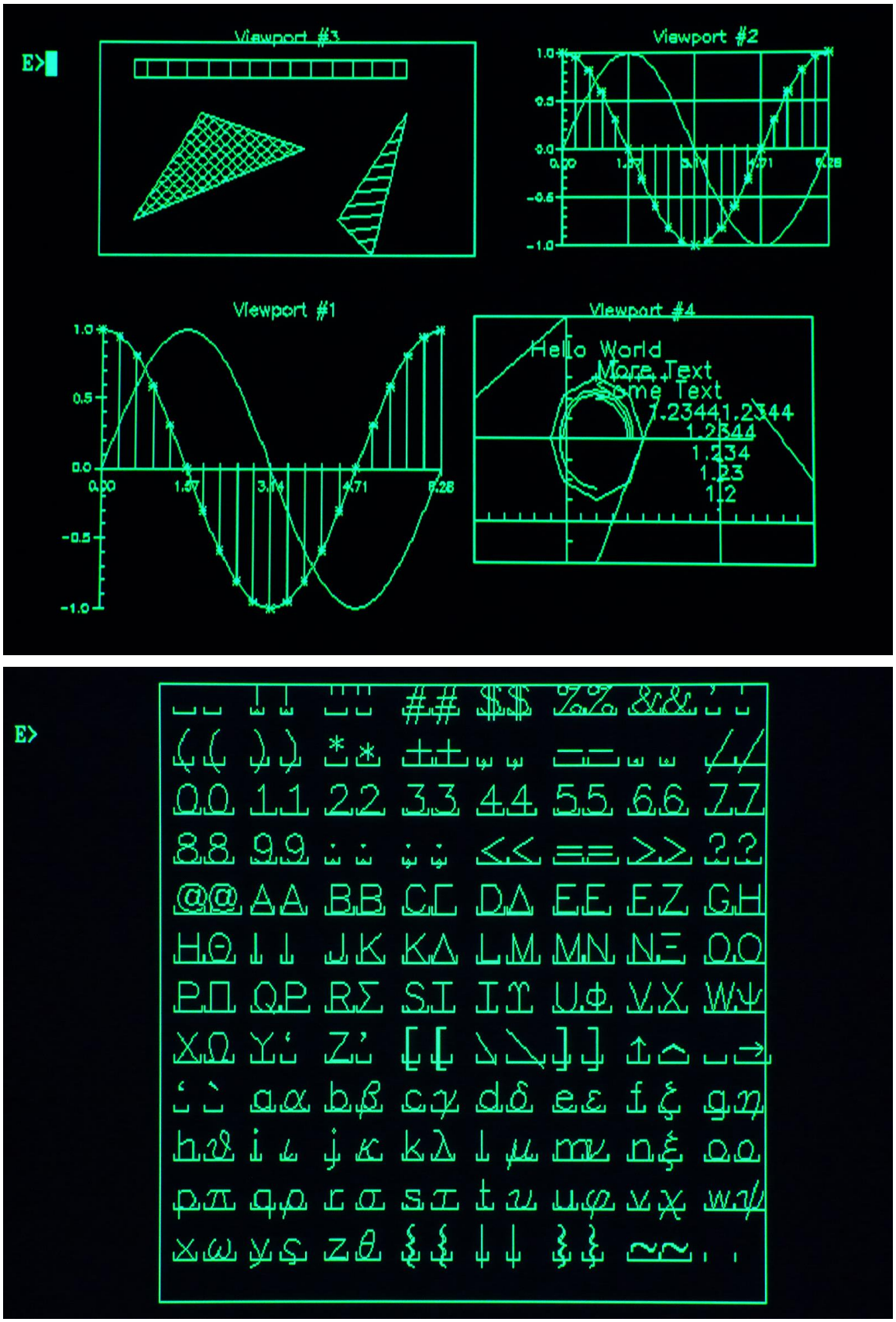

Figure 3 **Close-up view of the output in 640 x 350 VGA mode on a TFT screen.**

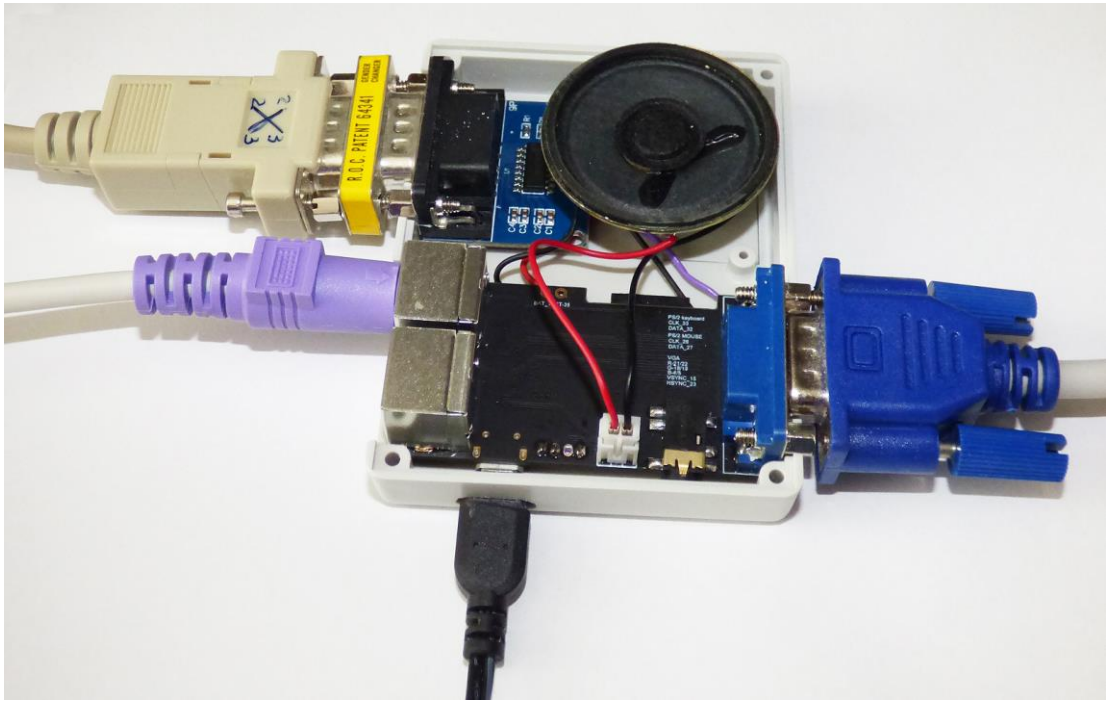

Figure 4 **The innards with VGA32 board, serial interface and loudspeaker.**

Figure 5 **The completed terminal for Tektronix and HP terminal emulation.**

2.2. ESP32 module and the TTGO-VGA32 Board

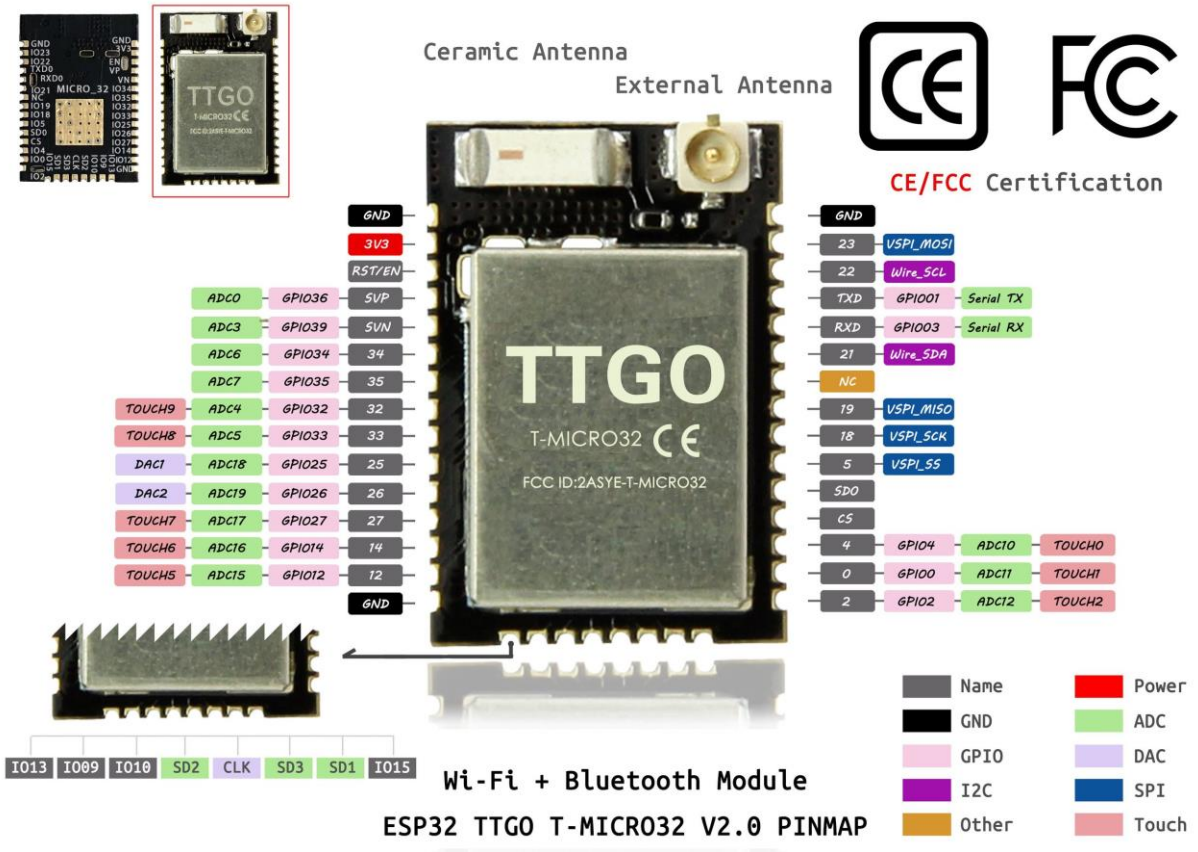

Figure 6 **This ESP32 module is soldered onto the TTGO VGA32 board. I soldered a wire to GPIO14 for RXD of the TTL-RS-232C converter, instead of using the GPIO12 on the 4-pin header.**

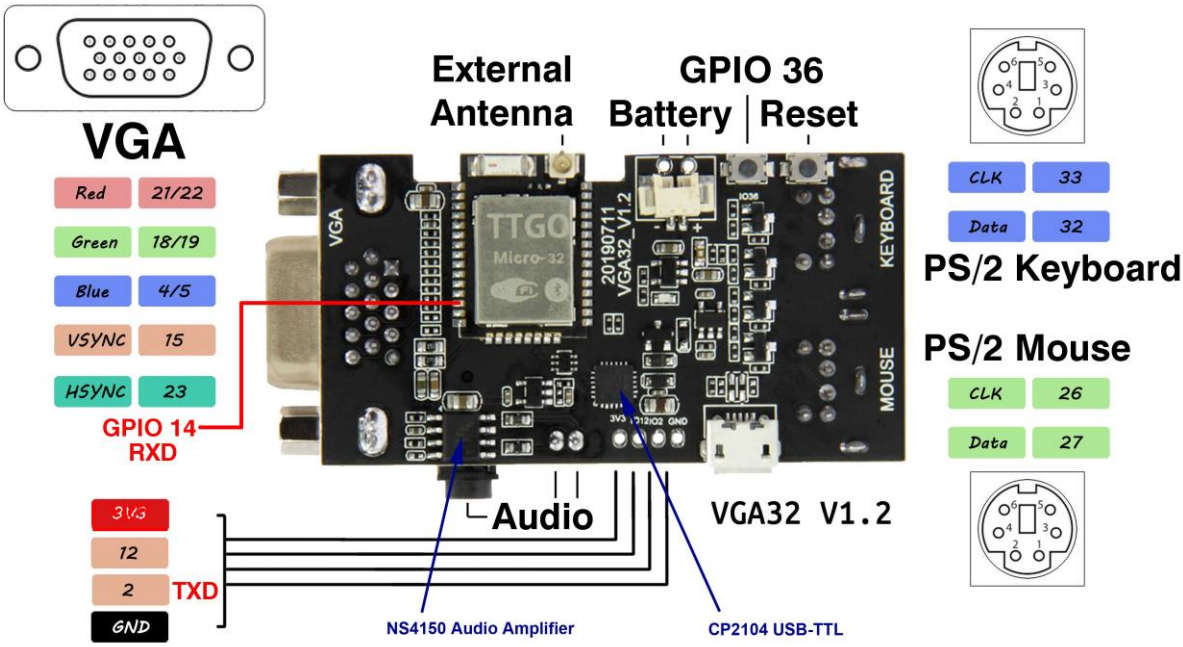

Figure 7 **TTGO VGA32 board Version 1.2 with GPIO 2 and an extra wire to GPIO 14 for the serial interface.**

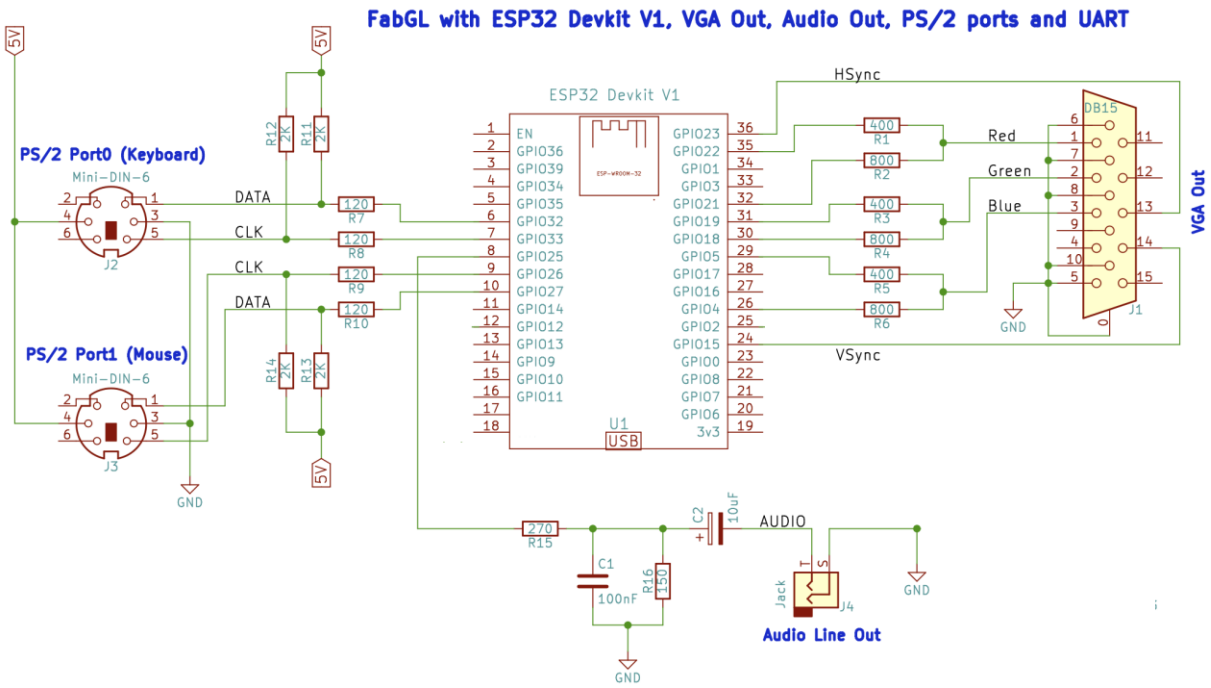

Figure 8 **Schematics proposed by the author of the FabGL library. The TTGO VGA32 Version 1.2 board follows this scheme and adds USB, audio amplifier and battery charging circuits.**

ESP32	Connection
GPIO 1	USB interface (TXD)
GPIO 2	to header (TXD)
GPIO 3	USB interface (RXD)
GPIO 4	VGA Blue
GPIO 5	VGA Blue
GPIO 12	4 pin header
GPIO 15	VGA Vsync
GPIO 18	VGA Green
GPIO 19	VGA Green
GPIO 21	VGA Red
GPIO 22	VGA Red

ESP32	Connection
GPIO 23	VGA Hsync
GPIO 25	Audio Out
GPIO 26	Mouse Data
GPIO 27	Mouse Clock
GPIO 32	Keyboard Data
GPIO 33	Keyboard Clock
GPIO 36	Button 2
RST	Button 1
GPIO 14	*solder wire (RXD)*

Table 2: **Pins used on the TTGO-VGA32 V 1.2 boards. GPIOs 1 and 3 are used for the USB-Serial connection of the bootloader. GPIOs 2 and 14 are used for the Serial connection.**

ESP32	Connection
GPIO 1	TXD / USB interface
GPIO 2	to header (TXD)
GPIO 3	RXD / USB interface
GPIO 4	VGA Blue
GPIO 5	VGA Blue
GPIO 12	to header
GPIO 13	to header
GPIO 14	to header (RXD)
GPIO 15	VGA Vsync
GPIO 18	VGA Green
GPIO 19	VGA Green

ESP32	Connection
GPIO 21	VGA Red
GPIO 22	VGA Red
GPIO 23	VGA Hsync
GPIO 25	Audio Out
GPIO 26	Mouse Data
GPIO 27	Mouse Clock
GPIO 32	Keyboard Data
GPIO 33	Keyboard Clock
GPIO 34	to header
GPIO 36	Button 2
RST	Button 1
GPIO 39	to header

Table 3: Pins used on the TTGO-VGA32 V1.4 boards. GPIOs 1 and 3 are used for the USB-Serial connection of the bootloader. GPIOs 2 and 14 are used for the Serial connection.

The two USB data lines are connected to the CP 2104 USB-TTL converter chip. This converts them to a 3.3V serial TTL signal which is connected to the TXD (GPIO 1) and RXD (GPIO 3) lines of the ESP32.

2.3. Buttons

Button 1 (close to the keyboard connector) on the TTGO VGA32 Board connects GND to RST and can be pressed to reset the system.

Button 2 (close to the battery connector) pulls GPIO 36 to GND and may be used for user code. It could be used to clear the Tektronix screen.

A simple HP 49G Demo Program

Even with a classical pocket calculator you can generate output on a Tektronix terminal. As a proof of concept I wrote a small program which mimics the Plot application of the HP 49G calculator.

These routines require that the regular HP 49G PLOT application is used to setup either a Function or a Polar plot. Besides defining the functions F(X) resp. R(X) the PLOT application must also be used to set up the plotting window.

The application stores the equations in the list EQ and the plot parameters and type in the list PPAR.

The following programs use these two data structures.

Currently, the program can plot either function plots Y=f(X) or polar plots R=f(angle).

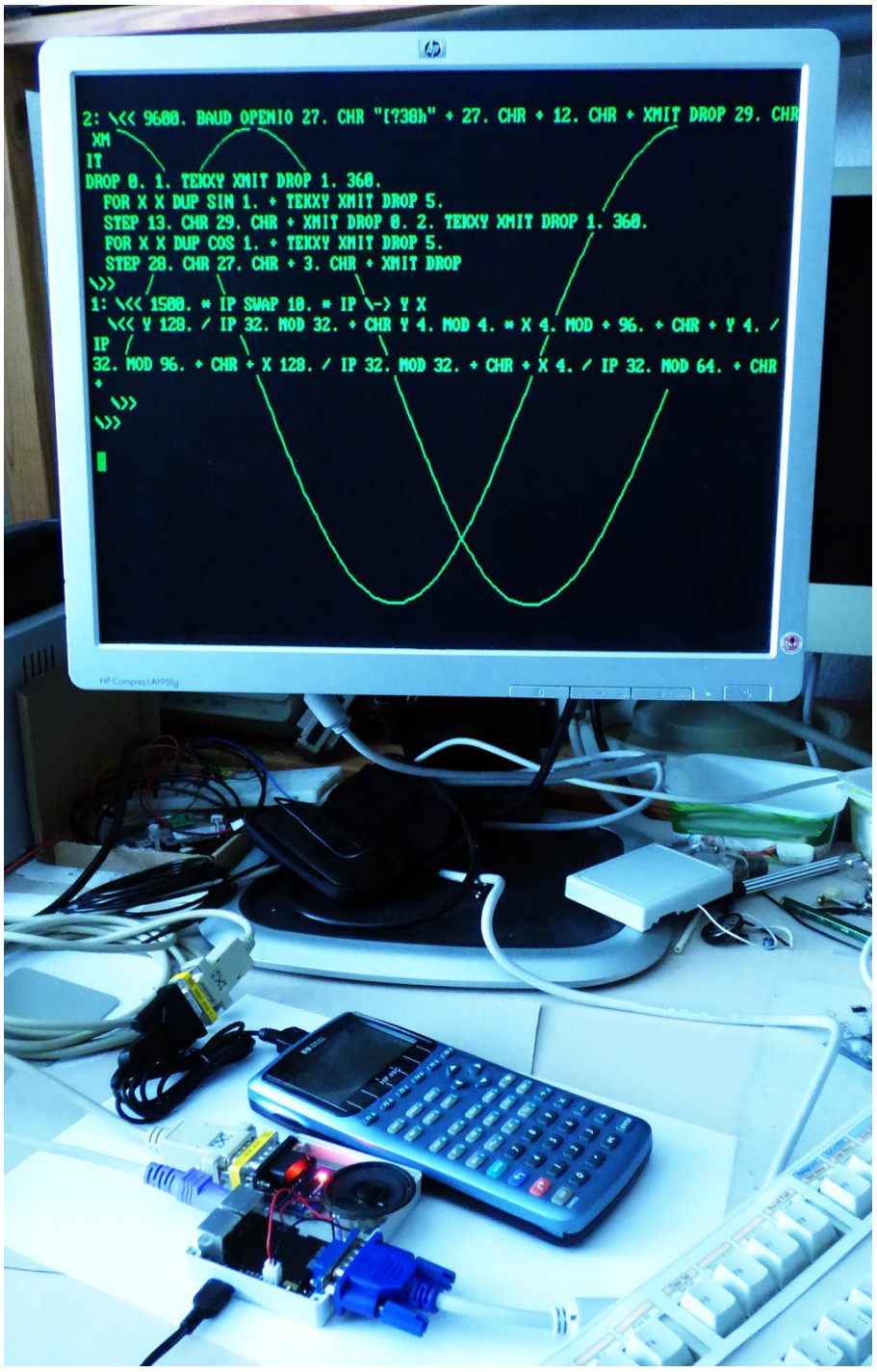

Figure 9 An test setup with an HP-49G calculator producing graphs in Tektronix format.

```
%%HP: T(3)A(D)F(.);
\<<
  TIME HMS\->
  TEKINI TEKEQ
  9600. BAUD OPENIO
  TEKGRAF
  PPAR DUP SIZE 1 - GET \->STR
  "FUNCTION" ==
  IF
  THEN TEKFUN
  ELSE TEKPOL
  END
  0 0 TEKMOVE
  DROP2 DROP2
  TEKEND
  ""
  1 TEQ SIZE
  FOR N
    N \->STR + ": " +
    TEQ N GET RCL \->STR DUP "'" POS 1 + 9999 SUB DUP "'" POS 1 - 1 SWAP SUB +
    "  " +
  NEXT
  PR1
  DROP 'TEQ' PURGE 'TEKPAR' PURGE
  TIME HMS\-> SWAP - \->HMS
\>>
```

Listing 1: Main Program TEKPLOT for creating Function or Polar plots.

```
%%HP: T(3)A(D)F(.);
\<<
  PPAR 1. GET PPAR 2. GET PPAR 1. GET - C\->R 2. \->LIST
  { 4095. 3072. } / INV LIST\-> DROP R\->C
  2. \->LIST 'TEKPAR' STO
\>>
```

Listing 2: Initialization program TEKINI which creates the temporary equation file TEKPAR from the PPAR file.

```
%%HP: T(3)A(D)F(.);
\<<
  { }
  1 { } EQ + SIZE
  FOR N
    { } EQ + N GET \->STR DUP "(" POS 1 - 1 SWAP SUB
    "'" + STR\-> +
  NEXT
  'TEQ' STO
\>>
```

Listing 3: Initialization program TEKEQ which creates the temporary equation file TEQ from the EQ file. The function names must be like F1(X) and the term starting at the opening parenthesis is stripped. We must force EQ to a list because it can be either a list or a single function name.

```
%%HP: T(3)A(D)F(.);
\<<
  TEKPAR 1. GET C\->R TEKPAR 2. GET C\->R
  PPAR 1. GET C\->R DROP 0. TEKMOVE
  PPAR 2. GET C\->R DROP 0. TEKDRAW
  0 PPAR 1. GET C\->R SWAP DROP TEKMOVE
  0 PPAR 2. GET C\->R SWAP DROP TEKDRAW
  PPAR 4 GET \-> DX
  \<<
    1 TEQ SIZE
    FOR N
      PPAR 1. GET C\->R DROP DUP TEQ N GET EVAL TEKMOVE
      PPAR 1. GET C\->R DROP PPAR 2. GET C\->R DROP
      FOR X
        X DUP TEQ N GET EVAL TEKDRAW
      DX STEP
    NEXT
  \>>
\>>
```

Listing 4: **Subroutine TEKPFUN for Function y = f(x) plots. The first lines draw the axes lines. The step size is taken from PPAR. The first line places the four transformation parameters 4:TX, 3:TY, 2:SX, 1:SY on the stack.**

```
%%HP: T(3)A(D)F(.);
\<<
  TEKPAR 1. GET C\->R TEKPAR 2. GET C\->R
  PPAR 1. GET C\->R DROP 0. TEKMOVE
  PPAR 2. GET C\->R DROP 0. TEKDRAW
  0 PPAR 1. GET C\->R SWAP DROP TEKMOVE
  0 PPAR 2. GET C\->R SWAP DROP TEKDRAW
  PPAR 4 GET \-> DX
  \<<
    1 TEQ SIZE
    FOR N
      0 TEQ N GET EVAL DUP 0 COS * SWAP 0 SIN * TEKMOVE
      PPAR 3 GET 2 GET PPAR 3 GET 3 GET
      FOR X
        X TEQ N GET EVAL DUP
        X COS * SWAP X SIN * TEKDRAW
      DX STEP
    NEXT
  \>>
\>>
```

Listing 5: **Subroutine TEKPOL for Polar R = f(angle) plots. The first lines draw the axes lines. The step size is taken from PPAR. The first line places the four transformation parameters 4:TX, 3:TY, 2:SX, 1: SY on the stack.**

```
%%HP: T(3)A(D)F(.);
\<<
  27. CHR "[?38h" + 27. CHR + 12. CHR + 7 CHR + XMIT DROP
\>>
```

Listing 6: **Subroutine TEKGRAF switches to Tektronix graphics mode..**

```
%%HP: T(3)A(D)F(.);
\<<
  28. CHR 27. CHR + 3. CHR + 7. CHR + XMIT DROP
\>>
```

Listing 7: **Subroutine TEKEND switches back to ANSI terminal mode.**

```
%%HP: T(3)A(D)F(.);
\<<
  TEKXY 29. CHR SWAP + XMIT DROP
\>>
```

Listing 8: **Subroutine TEKMOVE for moving the current pen to the position given on the stack (1:X, 2:Y). The stack must also contain the four transformation parameters 6: TX, 5: TY, 4: SX, 3: SY from TPAR.**

```
%%HP: T(3)A(D)F(.);
\<<
  TEKXY XMIT DROP
\>>
```

Listing 9: **Subroutine TEKDRAW for drawing a line from the current pen position to the position given on the stack (1:X, 2:Y). The stack must also contain the four transformation parameters 6:TX, 5:TY, 4:SX, 3:SY from TPAR.**

```
ASSEMBLE
  NIBASC /HPHP49-C/
                                                ( TEKXY.s )
                                                ( Bring X and Y into Tektronix format. )
                                                ( Stack also contains the transformation  )
                                                ( parameters extracted from TEKPAR. )
                                                ( These map from User space to device space, )
                                                ( which is integer in 4096x3072. )
RPL
::                                              ( 1:Y         2:X         3:SY 4:SX 5:YO 6:XO )
  CK2&Dispatch
  REALREAL                                      ( need at least 2 Reals on the stack )
  ::
    ( translate and scale Real X, Y values )
    5PICK %-                                    ( 1:Y-YO      2:X         3:SY 4:SX 5:YO 6:XO )
    3PICK %*                                    ( 1:[Y-YO]*SY 2:X         3:SY 4:SX 5:YO 6:XO )
    SWAP                                        ( 1:X         2:[Y-YO]*SY 3:SY 4:SX 5:YO 6:XO )
    BINT6 PICK %-                               ( 1:X-XO      2:[Y-YO]*SY 3:SY 4:SX 5:YO 6:XO )
    4PICK %*                                    ( 1:[X-XO]*SX 2:[Y-YO]*SY 3:SY 4:SX 5:YO 6:XO )
    ( now bring these 2 integer values IX, IY into Tektronix format: a 5-character string )
    COERCE2                                     ( convert IX, IY values to BINT )
                                                ( note: #/ leaves r and q on stack )
    BINT128 #/ BINT32 #+ #>CHR CHR>$            ( C4: 0.0.1.X12.X11.X10.X9.X8, to string  )
    SWAP BINT4 #/ BINT64 #+ #>CHR ROTSWAP >T$   ( C5: 0.1.0.X7.X6.X5.X4.X3, append to C4: C4&C5 )
    ROT BINT128 #/ BINT32 #+ #>CHR 4UNROLL      ( C1: 0.0.1.Y12.Y11.Y10.Y9.Y8 )
    BINT4 #/ BINT96 #+ #>CHR ROTSWAP >H$        ( C3: 0.1.1.Y7.Y6.Y5.Y4.Y3, prepend to C4&C5 )
    UNROT #2* #2* #+ BINT96 #+ #>CHR >H$        ( C2: 0.1.1.0.Y2.Y1.X2.X1, append to C1: C1&C2)
    SWAP >H$                                    ( finally: C1&C2&C3&C4&C5 )
    ( leaves a concatenated string of 5 characters: )
    ( C1: [0.0.1.Y12.Y11.Y10.Y9.Y8] )
    ( C2: & [0.1.1.0.Y2.Y1.X2.X1] )
    ( C3: & [0.1.1.Y7.Y6.Y5.Y4.Y3] )
    ( C4: & [0.0.1.X12.X11.X10.X9.X8] )
    ( C5: & [0.1.0.X7.X6.X5.X4.X3] )
  ;
;
```

Listing 10: **This routine TEKXY was initially written in UserRPL which was rather slow. It was then rewritten with the help from some HP-Forum members in SysRPL for speed. It expects the four transformation parameters from TEKPAR on the stack as well as the X and Y values in user coordinates. Another solution provided by forum members was written in assembler (not shown here).**

```
ASSEMBLE
  NIBASC /HPHP49-C/
                                                ( TEKXY.s )
                                                ( Bring X and Y into Tektronix format. )
                                                ( Stack also contains the transformation  )
                                                ( parameters extracted from TEKPAR. )
                                                ( These map from User space to device space, )
                                                ( which is integer in 4096x3072. )
RPL
::                                              ( 1:Y         2:X         3:SY 4:SX 5:YO 6:XO )
  CK2&Dispatch
  REALREAL                                      ( need at least 2 Reals on the stack )
  ::
    ( translate and scale Real X, Y values )
    5PICK %-                                    ( 1:Y-YO      2:X         3:SY 4:SX 5:YO 6:XO )
    3PICK %*                                    ( 1:[Y-YO]*SY 2:X         3:SY 4:SX 5:YO 6:XO )
    SWAP                                        ( 1:X         2:[Y-YO]*SY 3:SY 4:SX 5:YO 6:XO )
    6PICK %-                                    ( 1:X-XO      2:[Y-YO]*SY 3:SY 4:SX 5:YO 6:XO )
    4PICK %*                                    ( 1:[X-XO]*SX 2:[Y-YO]*SY 3:SY 4:SX 5:YO 6:XO )
    ( now truncate these to integer and bring them into Tektronix format: a 5-character string )
    COERCE2                                     ( convert IX, IY values to BINT )
    CODEM
      GOSBVL =POP2#
      C+C.A CSL.A
      ACEX.X A+A.A
      ASL.A ASL.M ASL.M
      ASR.A ASR.A
      ACEX.A
      ASL.M ASL.W A+C.A
      A+A.A C=A.X
      ASR.A ASR.A ASR.A
      C+C.A CSL.A
      A+C.A ASL.M
      LC 4020606020
      P=9
      A!C.WP
      GOSBVL =SAVPTR
      GOVLNG =PUSHhxsLoop
    ENDCODE
    # 02A2C
    CHANGETYPE
    ( leaves a concatenated string of 5 characters: )
    ( C1: [0.0.1.Y12.Y11.Y10.Y9.Y8] )
    ( C2: & [0.1.1.0.Y2.Y1.X2.X1] )
    ( C3: & [0.1.1.Y7.Y6.Y5.Y4.Y3] )
    ( C4: & [0.0.1.X12.X11.X10.X9.X8] )
    ( C5: & [0.1.0.X7.X6.X5.X4.X3] )
  ;
;
```

Listing 11: **This alternative routine TEKXY was written in Saturn assembler to minimize run time by Werner from the HP Forum. The test for the number of REAL parameters has not yet been adapted – six must be provided by the caller.**

```
%%HP: T(3)A(D)F(.);
{ Y1 R1 Y3 Y2 }
```

Listing 12: **The temporary file TEQ stores the stripped equation names; it is created by TEKEQ from the list in EQ. The actual name of each equation (Y… or R…) does not matter – they all take one parameter and return one result.**

```
%%HP: T(3)A(D)F(.);
{ (0.,0.) (11.375,3072.) }
```

Listing 13: **The temporary file TEKPAR stores the offset (TX,TY) and scaling factors (SX,SY) for TEKXY; it is created by TEKINI.**

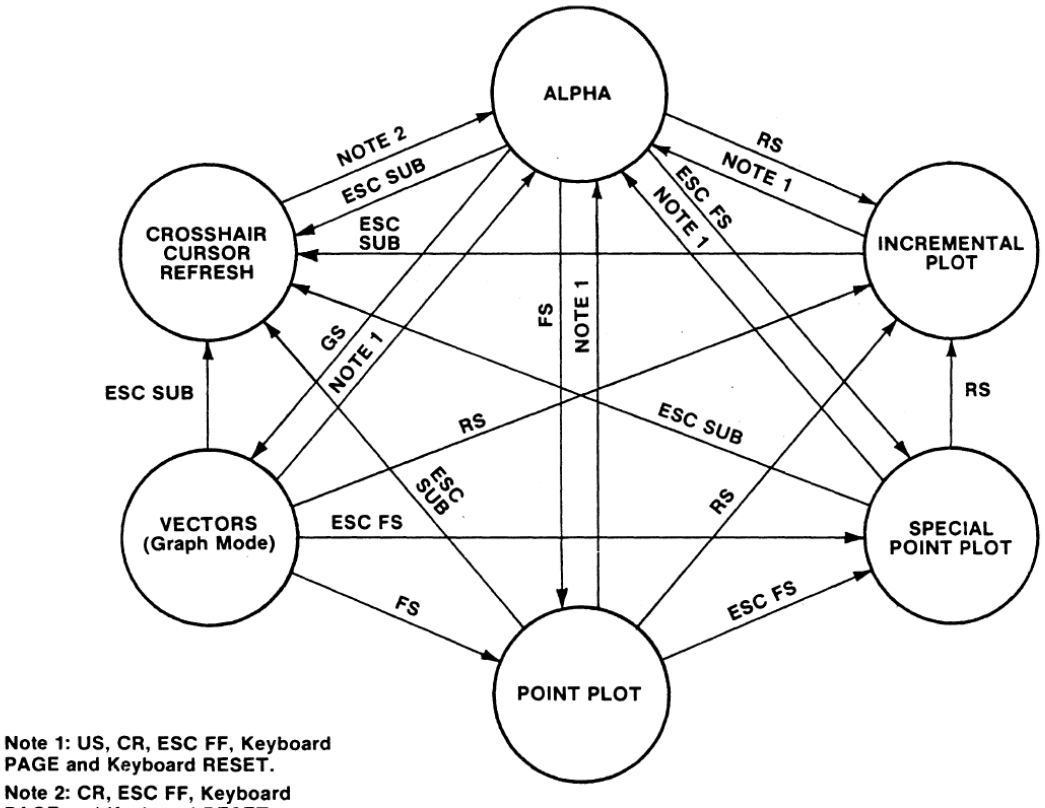

Figure 10 **Switching between the different display modes while in in Tektronix mode is accomplished by control characters or short escape sequences.**