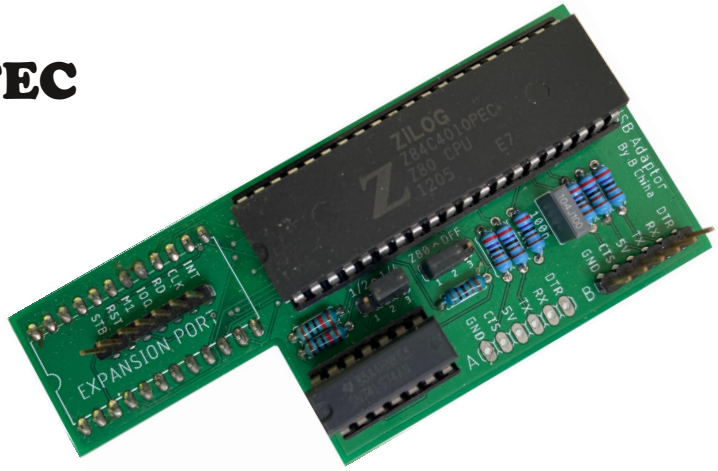
TEC JOURNAL

A tribute to the old magazine

Making OLD STUFF ... new

Serial Special

- **Connect Your TEC to the Zilog SIO**
- **Communicate with SPI and I2C Devices**
- **Bit Banging Serial for the TEC**
- **TEC PBC Revision History (with Pics)**
- **Modding Your TEC**
- **.....And More.....**



Editorial

Welcome to the New Talking Electronics Computer (TEC) Journal.

There has been some excellent TEC contributions and ideas over the last few years. Most of these 'posts' have been on the Facebook group. They seem to get lost forever, I thought it would be good to start a NEW magazine that collates these projects and makes them more permanent and visible.

This issue focuses on the Serial interface with the TEC. There has been some great work done on getting Serial Communication from your computer to the TEC. This is something I think is extremely valuable in these modern times if you are serious about programming the TEC.

For the next issues, it's up to YOU, The TEC Community! We want you to contribute and share your work, whether big or small on all things TEC related. To contribute, just add your work to the TEC Journal GitHub directory. When there are enough articles, I'll get around and publish them. There may be 2 to 4 issues a year, depending on how popular (or not) this concept is. I would also like to hear from the people who make up the TEC Community and find out why they are into the TEC.

Who am I? My name is Brian Chiha and like most of us discovered Talking Electronics when I was a teenager. I was able to build a TEC-1B back in the early nineties. I can't find it anymore, but one striking thing was that the Seven Segments Displays that I used had a different pinout. I had to wire wrap them from the board to their legs, creating some 3-D popout display. Fast forward about 30 years, I found the TEC-1D and built it. By then I was already a software engineer and had a better understanding of programming and electronics. I eventually went all out and built an LED 8x8, DAT, NVRam, SIO and Expansion board for the TEC. I also created a bunch of software, like TEC Invaders Mk2, A Maze Game and Game of Life.

I've designed the magazine to look and feel like the original TE mags. I hope you enjoy the read!

Brian.

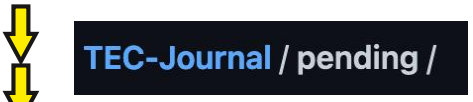
Table of Contents

How to Contribute

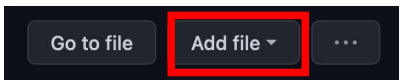
To get your work published in the TEC Journal, just upload your files/pictures/pdfs to the /pending directory of the TEC Journal GitHub page. This requires you to create a GitHub account and click on the 'Add File' button. If you prefer to format your article, use the 'Article-Template.docx' file and the fonts provided in the /template_and_fonts directory. Important: Put your name on the article to get credited!

Useful Links

* TEC 1 Resources Page: https://github.com/tec1group
* TEC Journal GitHub: https://github.com/tec1group/TEC-Journal
* TEC Facebook Page: https://www.facebook.com/groups/tec1z80
* TEC 1 Emulator: https://tec1group.github.io/wicked-tec1/
* Z80 Assembler and debugger: https://www.asm80.com/
* Z80 Reference: https://clrhome.org/table/



GitHub TEC-Journal pending directory

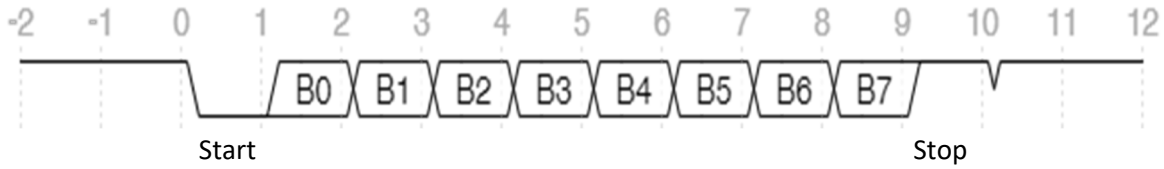


Add file button on GitHub

Bit Banging serial for the TEC

Bit banging a serial port is an easy and cheap way to connect your TEC to the outside world. In this article we will look at how bit banged serial transmit and receive can be implemented in Z80 assembly language.

By Craig RS Jones



Asynchronous Serial Character
8N1 = 8 bits, no parity, 1 stop bit, character length = 10 bits

An asynchronous serial byte or character is shown above, it's called **asynchronous** serial because there is no common clock signal between the receiver and the transmitter, instead each character or byte has start and stop bits added to provide synchronisation.
Each of the bits, including the start and stop bit are output for the same amount of time, this 'bit time' is equal to **1/ number of Bits Per Second(BPS) or Baud rate** . For example, at 4800 Baud the bit time would be 1/4800 = 208us.
The most important thing for successful transmission is to have the receiver and the transmitter using the same Baud rate!

Serial Transmit

There are three parts to the transmit code, the first is to send the start bit, the second to send the data and the last part to send the stop bit or bits.

1. Start Bit transmission

The FTDI USB to serial converter expects the RXI (Receive Input) to normally be High, so to send the start bit our code simply makes the data line Low and delays for one bit time.

2. Send the data byte

To send the data byte we must output each bit in turn for the bit time on the output pin. The serial data output is the Bit 6 position in the output register for the TEC so we have to shift all the bits in the data byte to the bit 6 position for output.

The RRC instruction shifts the bits in the register one bit to the right, shifting the bit in position 0 into the bit 7 position with each shift.

To start we put the loop or bit count in B, the first RRC instruction moves all the bits one position to the right, bit 0 moves into the now empty bit 7 position.

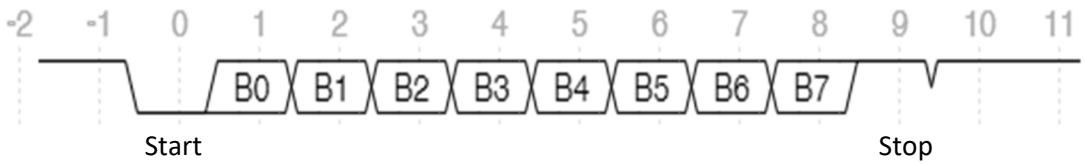
Now we just execute the loop 8 times, shifting each bit into the bit 6 position for output and then doing a bit time delay.

3. Stop Bit transmission

Finally, set the output pin high and output each stop bit for the bit time. The next character can then be sent, any additional delay here is usually called the 'inter-character delay'.

```
BAUD = delay constant for Baud rate

;-----------------------------------------
;transmit a serial character
;-----------------------------------------
; C = character to transmit
;
;
;send the start bit
;
        LD      a,$00
        OUT     (PORT),A      ;start bit = 0
        LD      HL,(BAUD)
        CALL    BITIME        ;one bit time delay
;
; set up to shift out the character
;
        LD      B,08H         ; 8 bits to transmit
        RRC     C
SHIFTOUT:
; move all the bits one position to the right
        RRC     C
        LD      A,C
        AND     40H
;output the bit and delay for the bit time
        OUT     (PORT),A
        LD      HL,(BAUD)
        CALL    BITIME        ;one bit time delay
;have we done all the bits?
        DJNZ    SHIFTOUT      ;jump if not finished
;
; send the stop bit (or bits)
;
        LD      A,$40
        OUT     (PORT),A      ;stop bit=1
        LD      HL,(BAUD)
        CALL    BITIME        ;send the stop bit
        CALL    BITIME        ;or two if required
        RET
```



Serial Receive Routine

Receiving a character is a little more complicated and starts with synchronising, waiting in a loop until the falling edge of the start bit arrives.

1. Start bit detection

Once the falling edge is detected we delay for only half a bit time, until time '0' in the diagram above. To calculate the delay time we divide the BAUD variable, which is the bit time delay value, by 2 by shifting the 16-bit HL register one bit to the right. The instruction, SRL (Logical shift right), shifts the byte in the H register one bit right, shifting Bit 0 into the carry, the next instruction, RR (Rotate Right through carry) shifts the bit out of the carry into the bit 7 position of the L register. Using these two instructions on the H and L registers performs a divide by 2 on the 16-bit register HL .

Once the half bit time delay finishes another check of the input level is done, if it is still low we have a valid start bit, if not then we have what is called a 'framing error', so we just go back and wait for the next falling edge of a start bit.

2. Receive the data byte

You can see in the diagram that we are now at time 0, halfway into the start bit, now we can simply read in the data byte delaying one full bit time before reading the middle of the next bit in the eight bit character.

Since we are reading our data in on bit 7of the input register, we do a RL, (Rotate Accumulator Left) on the input byte, shifting bit 7 into the carry bit and then doing a RR,(Rotate Right through carry) to move the received bit into the C register where our received byte will be assembled. After 8 bits have been received our character is in the C register.

3. Check the stop bits?

,We don't really care about checking our stop bits and there is probably something we want to do with our received byte, time is of the essence when receiving bit banged serial!

We don't bother checking the stop bits and just return the received character from the subroutine.

Conclusion

Bit Banging a serial port is a simple and effective way to connect a resource limited 'minimum system' like the TEC to a computer.

The ability to communicate this way opens up many possibilities for input and output, debugging, data logging, downloading/uploading code and data, communicating with serial peripherals; modems, radios etc.

Of course bit banged serial consumes all of the processor time during transmission and reception, which is why dedicated hardware devices such as the Z80 SIO and MC6850 ACIA were designed to offload the transmission and reception of serial data.

References:

Southern Cross Monitor TXDATA and RXDATA subroutines.

Jack Ganssle has a good description and some routines here;
http://www.ganssle.com/articles/auart.htm

Sparkfun have an extensive tutorial;
https://learn.sparkfun.com/tutorials/serial-communication

```
;----------------------
; receive a serial character
;----------------------
; wait for the start bit falling edge
STARTLOOP:
        IN      A,(KEYBUF)
        BIT     7,A
        JR      NZ,STARTLOOP
; delay for half the bit time
        LD      HL,(BAUD)
        SRL     H
        RR      L
        CALL    BITIME
; is the start bit still low?
        IN      A,(KEYBUF)
        BIT     7,A
        JR      NZ,STARTLOOP
; shift in the character
        LD      B,08H           ;8 bits to receive
SHIFTIN:
        LD      HL,(BAUD)
        CALL    BITIME          ;one bit time delay
        IN      A,(KEYBUF)
        RL      A
        RR      C
        DJNZ    SHIFTIN
        RET     ;character returned in c
;--------------
; BIT TIME DELAY
;--------------
BITIME:
        PUSH    HL              ;HL = delay time
        PUSH    DE
        LD      DE,0001H
BITIM1:
        SBC     HL,DE
        JP      NC,BITIM1
        POP     DE
        POP     HL
        RET
```

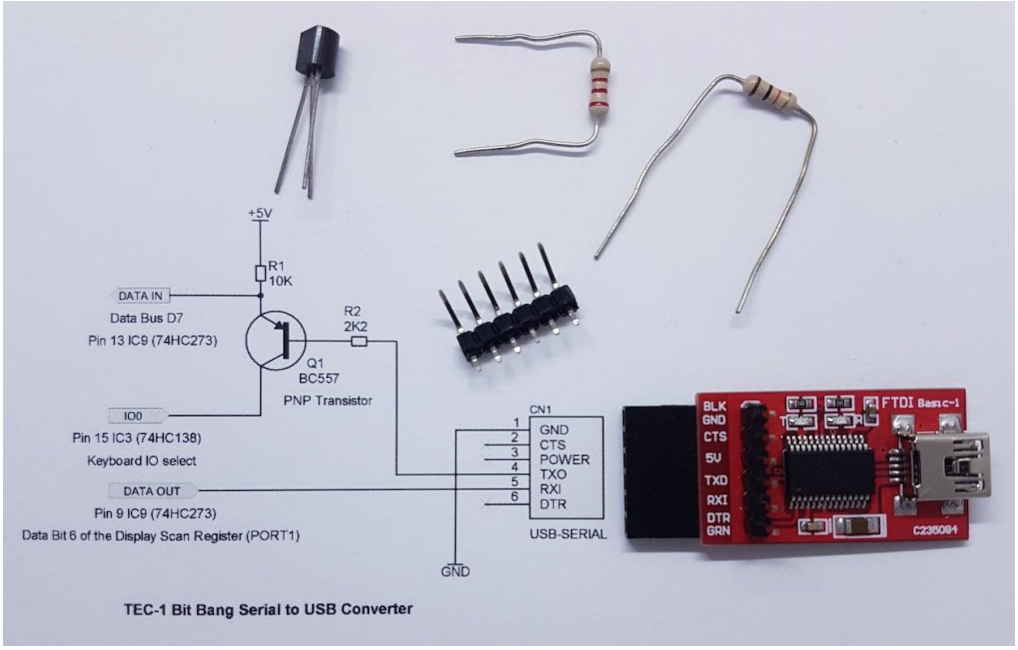
TEC Mods!

Welcome to the TEC Mods page! A place where upgrading has no limit....

If you like to tinker with your TEC and would like to share your work, send us your plans via GitHub and it will be shared here.

Add Serial to your TEC

EDITORS NOTE:
Craig Jones nicely posted this picture on Facebook. It came with no description, just the picture. It is labelled as "TEC-1 Bit Bang Serial to USB Converter". Its title says it all and the schematic is self-explanatory. It will work with the code that is explained in his article on the previous pages. Just set KEYBUF to 0 and PORT to 1.



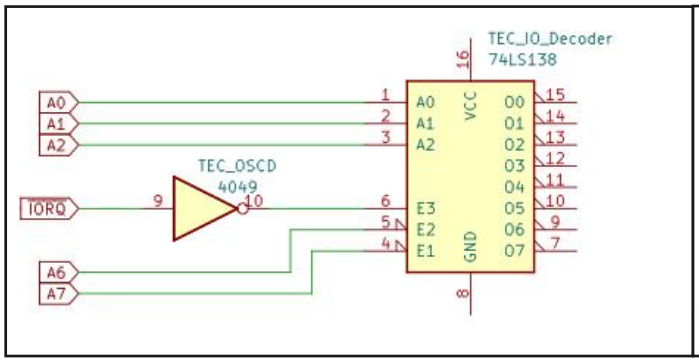
Credit: Craig Jones (Picture)

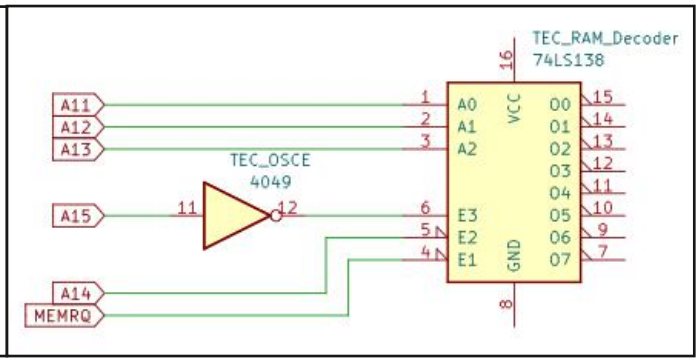
IO Decoder Exapander

The IO port decoder on the TEC only Decodes A0-A2 address lines. This means the TEC IO map 'wraps around' every 8 ports. This can be fixed with the first mod - which ensures the IO ports only occupy IO space 00h-3fh. This frees 40H onwards for other modules e.g. a 6850 at 80h for Grant Searle's Tiny Basic.

The Second mod is similar - the Memory decoder only goes up to A13, heance the memory wraps around every 16k - 0000H == 4000H = 8000H == C000H. If you want to add more than 16k of memory, you need this mod, which ensures the whole 64k addres space is fully decoded, avoiding any wrap-arounds.

The mods work on an original TEC with no additional chips required, by borrowing the two otherwise unused gates of the 4049 inverter. If you have a crystal oscillator, you can use an additional inverter chip - a 4049 or a 74xx14 will do nicely here as would almost any inverter gate. Credit: Craig Hart





A brief history of TEC-1 PCBs

The TEC-1 PCB has undegone several revisions over time. Although the designs are all very similar and generally software bacwards-compatible (Same memory map, IO port map, keyboard layout etc.), there are some subtle differences that may be of interest to collectors and the like.

By Craig Hart

PCB's were generally made in batches of 100 at TE and typically changed colour each batch (but not always). Between the basic design revision and the colour, the age of the machine can thus be roughly dated.

No precise record was kept of manufacturing dates and boards were not serial-numbered, so this information is being 'reverse engineered' and is mainly educated guesswork on the part of TEC owners and former TE staff. It is not to be taken as absolute or 100% accurate.

In short, the following major design revisions exist

TEC-1 (Issue 10, original, 1983). 7805 regulator above PCB bottom left corner. 8212 dispaly latch chips. no Shift key. MON-1 monitor ROM by John Hardy.

TEC-1A (Issue 12). Issue 12 page 14 states "The regulator is mounted under the PC board so that it cannot be bent over and broken off, the 2,200uf electrolytic has been changed to 1,000uf and the output latches have been changed to 74LS273 (or 74LS 374 or 74LS377). In all other respects, the boards are identical."

TEC-1B (Issue 13). Adds SHIFT key and MON-2 Monitor ROM by Ken Stone.

TEC-1B Rev.1 (Just before issue 15, 1990). Minor improvements by Craig Hart including improved labelleing of IO, memory port & other CPU pins.

TEC-1C (After Issue 15, 1991). CAD version by Craig Hart including mods for JMON and MON1/2 select switch.

TEC-1D Reproduction board by Ben Grimmit, 2018.

TEC-1E "MIT-Z" design by Ken Stone, 2020.

TEC-1F Reproduction of the TEC-1D board by Craig Jones, 2021. Major improvements to modernise the board.

Detailed cronology follows

First "Z-80 computer" prototype. This was not actually called a TEC-1, and was unlike any that are shown in the photos. It was casually refered to as Orac. It has not survived. Built by John Hardy.

First TEC-1 prototype. No silk screen. Incorrect track work. It was scrapped and is presumed lost, possibly sold as part of a grab-bag. Built by Ken Stone. Last seen by Craig Hart in the scrap pile at TE, late 1980's.

Second TEC-1 prototype. Bare fibreglass. No silk screen. Cut-out on top left corner. First to work without corrections to the PCB track work. Built by Ken Stone. This unit was mounted in a small rack case and displayed a pseudo error message to demonstrate one of the possibilities of computerised portable trafic lights. The board was notched to clear the power transformer. The keypad was paralleld by a touch keyboard on the front of the case, likewise the LED display. The ROM is labeled "DYCO"

* TEC-1 Batch 1. 1983. Red text white speaker on blue PCB. Issue 10 cover. Designed by Ken Stone and John Hardy. PCB Artwork by Ken Stone. Only 5 of this version exist, distributed to TE staff and John Hardy. The clock capacitor was marked as 220pf (Changed later to 100pf) and the speaker LED was hidden by the speaker and had to be moved.
* TEC-1 Batch ?. White text blue speaker on green PCB. Issue 11 cover.
* TEC-1 Batch ?. White text black speaker on green PCB.
* TEC-1 Batch ?. Blue text and white speaker on green PCB.
* TEC-1 Batch ?. Yellow text blue speaker on green PCB.

* TEC-1A prototype. Bare fibreglass. No silk screen overlay. New disply latch validation design. (There is a single 1A with the regulator mounted above the PCB, but at right angles to its former location, and provision is made for bolding it to the PCB. This board is the prototype 1A made to test the new latches, and has no solder mask or overlays.)

Issue 12 page 13 claims that over 1000 TECs were sold by this point.

* TEC-1A Issue 12 page 13 shows a 1A model in a monochrome photo featuring the diagonal stripes across the top left corner and the Retex case and with 7805 mounted under PCB.

* TEC-1A Batch ?. 2 x (dark) diagonal stripes, white text on green??? PCB. (B&W photo reference issue 12, p13. single proto???)

* TEC-1A Batch ?. White Text Yellow stripes on Blue PCB.

Cont....

* TEC-1B Issue 13 introduces the TEC-1B. Monochrome photo on page 9. It now features the SHIFT key and MON-2. Red with white speaker on green PCB with diagonal stripes. Ken stone and John Hardy names removed.

* TEC-1B Batch ?. Red diagonal stripes, white text on green PCB.
* TEC-1B Batch ?. Blue diagonal stripes, yellow text on green PCB.
* TEC-1B Batch ?. Blue diagonal stripes, white text on green PCB.

* TEC-1B Issue 14. No PCB mods but the idea of the MON 1/2 switch (mod by cutting a link on the PCB top side and adding a switch) was introduced here.

TEC-1B Rev.1. 1989/90 PCB design mods by Craig Hart. Released just prior issue 15. Red text white speaker on green PCB. Added labels to the memory and I/O port pins denoting the address range and port number. Returned Ken Stone and John Hardy's names to solder mask. I believe that a yellow on green 1B PCB exists but I can't locate photos just now.

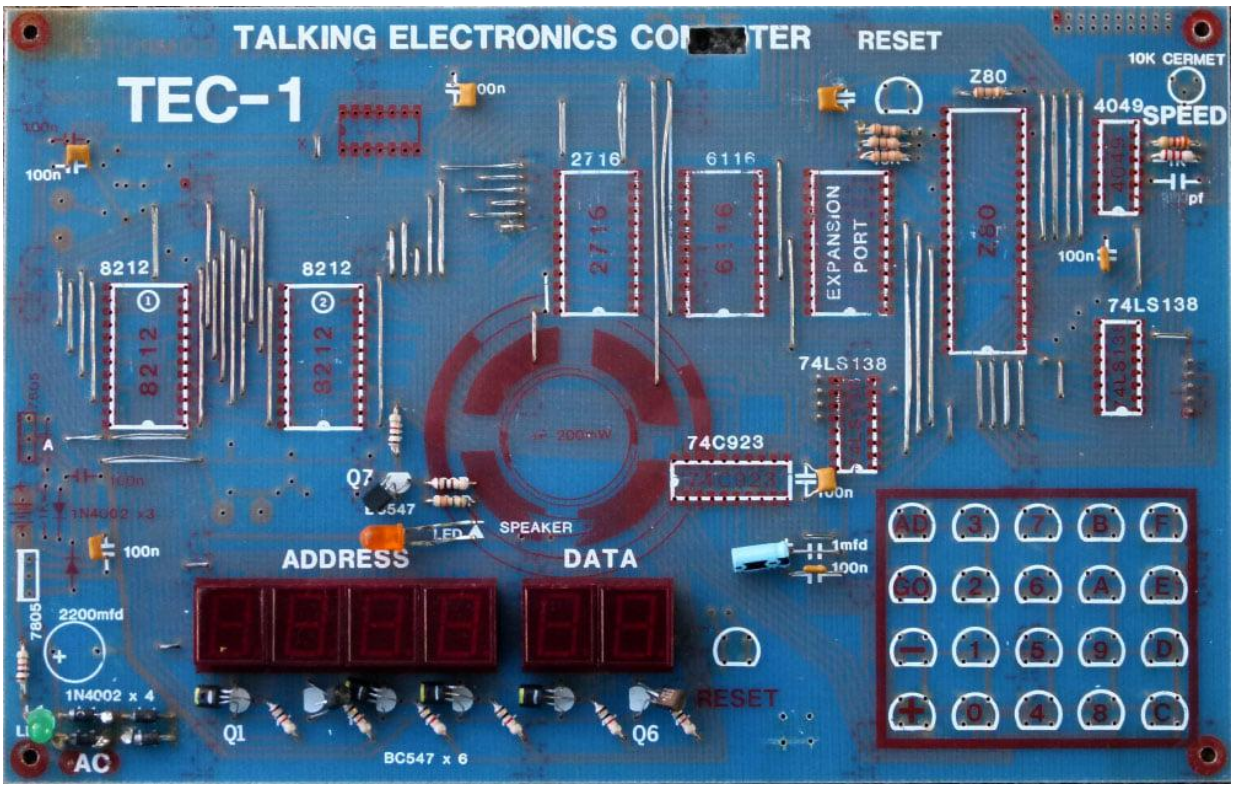
* TEC-1B CAD Prototype designed by Craig Hart. Possibly a prototype but not made in production (as I recall -Craig).

* TEC-1C 1991 PCB designed by Craig Hart. All white on green PCB; no stripe. TEC 1C annotation on solder mask side. Added some mods to incorporate JMON requirements (e.g. 4k7 keyboard mod resistor), MON select switch, Mini PCB speaker. Coverted to Protel Autotrax PCB CAD package. Previous PCBs were all done by hand with traditional tape and stencil on vellum. The 1C may have been the last run produced by TE as TE was all but out of business by the mid 90's. I believe that only one batch of 1C PCBs was ever produced. I did have this unit for a number of years but has been misplaced in various house moves over the years.

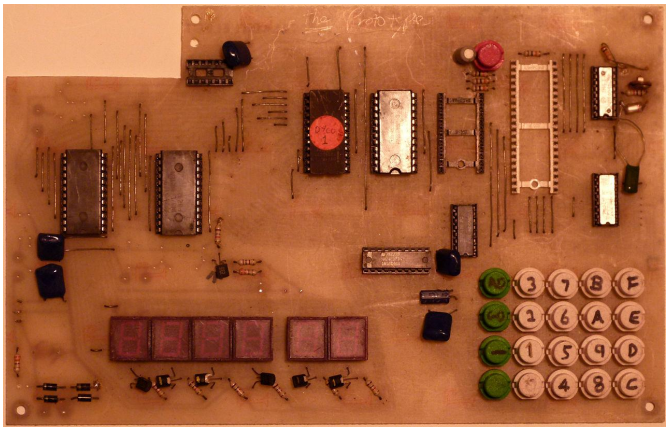
* TEC-1D 2018 Reproduction board by Ben Grimmit. White on green PCB. TEC 1D annotation on solder mask side. Revision information listed below the keyboard, componenet side. Made with Colin Mitchell's blessing.

* TEC-1E 2020 "MIT-Z" design by Ken Stone. Many new features. Currently at prototype stage. Not 100% TEC-1 compatible. (Ken coments: actually, it is, if you copy an original EPROM to a larger EPROM, but plans were to change the location of the stack)

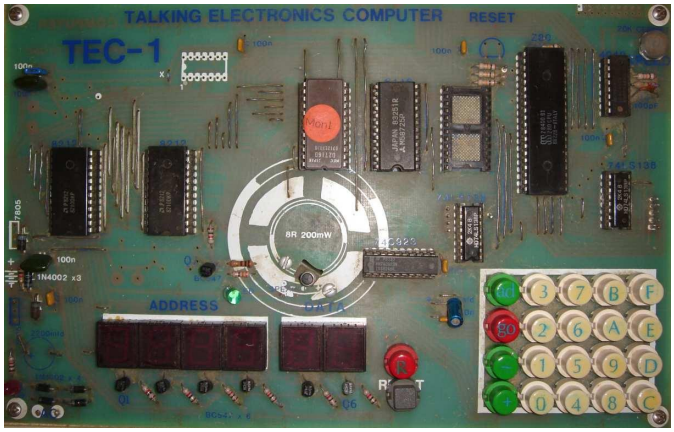
* TEC-1F 2021 Major improvements around address decoding from 2K to 8K blocks which are switchable. Incorporate a 4MHz clock along with the original CMOS clock. Extra lines to connect a UART receiver/ transmitter. Added a latch on Port 0 for easier input receiving. An On/Off switch! Aligned Z80 pins headers and labelled vias on top of the board.



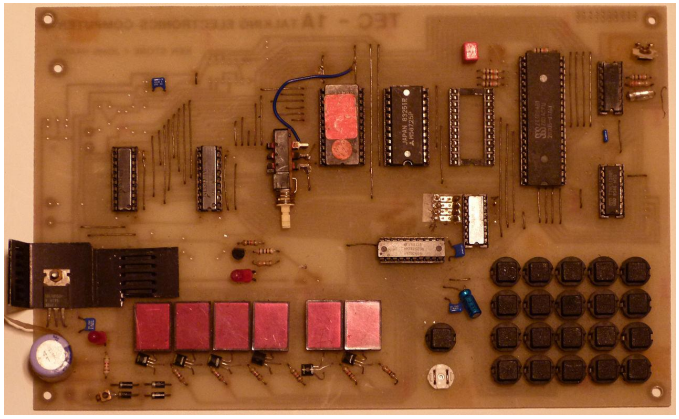
Original Issue 10 Cover Machine



TEC 1 Second Prototype

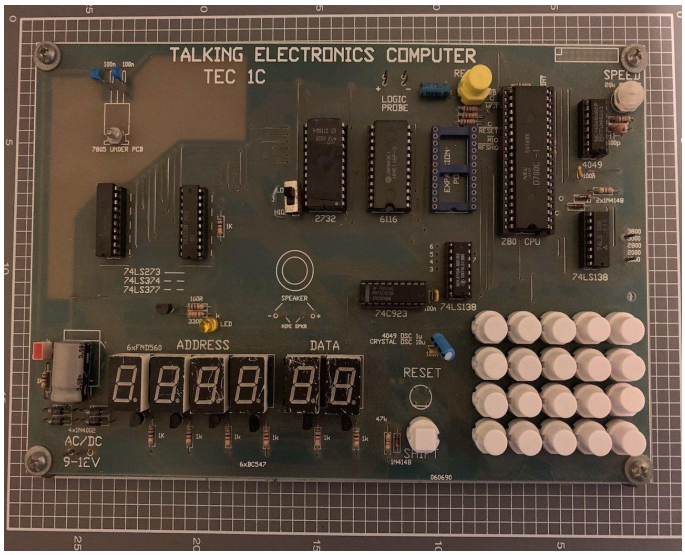


TEC 1 Blue / White / Green

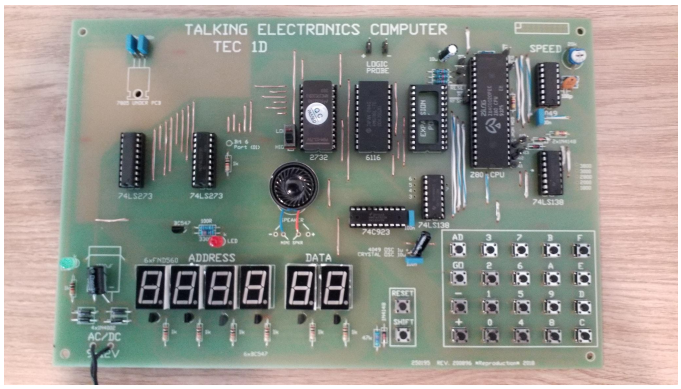


TEC 1A Prototype

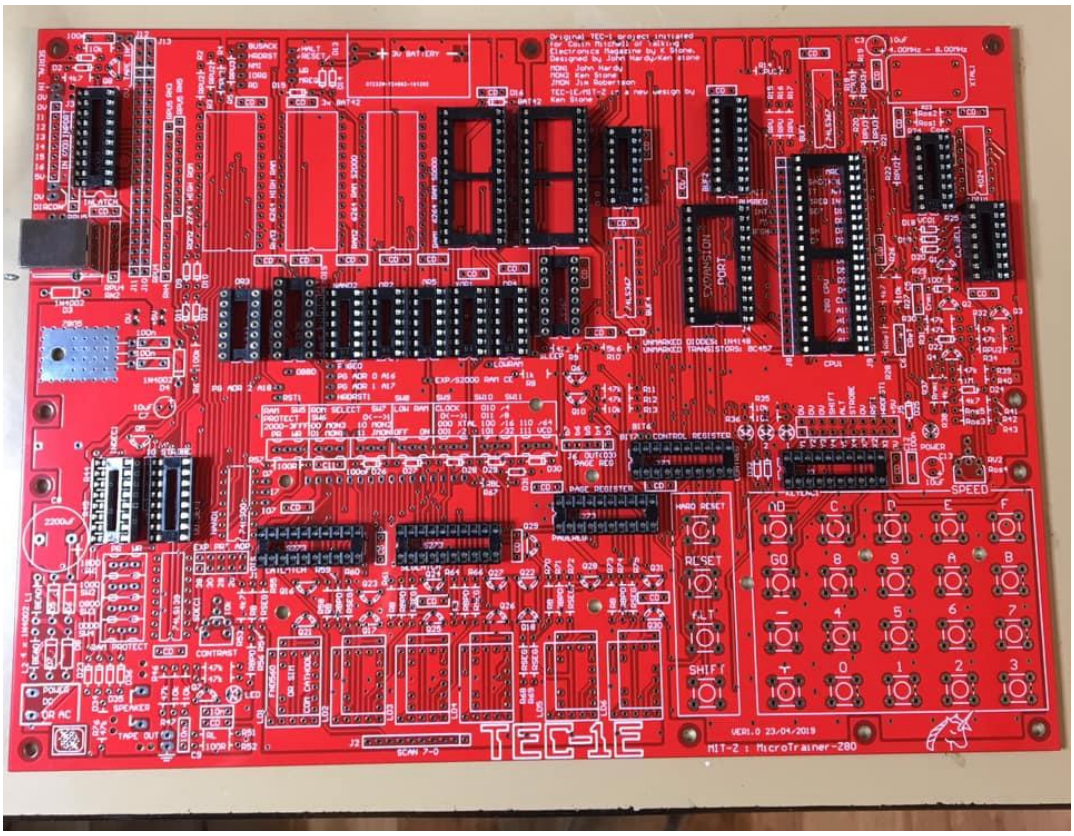
TEC 1A White / Yellow / Blue



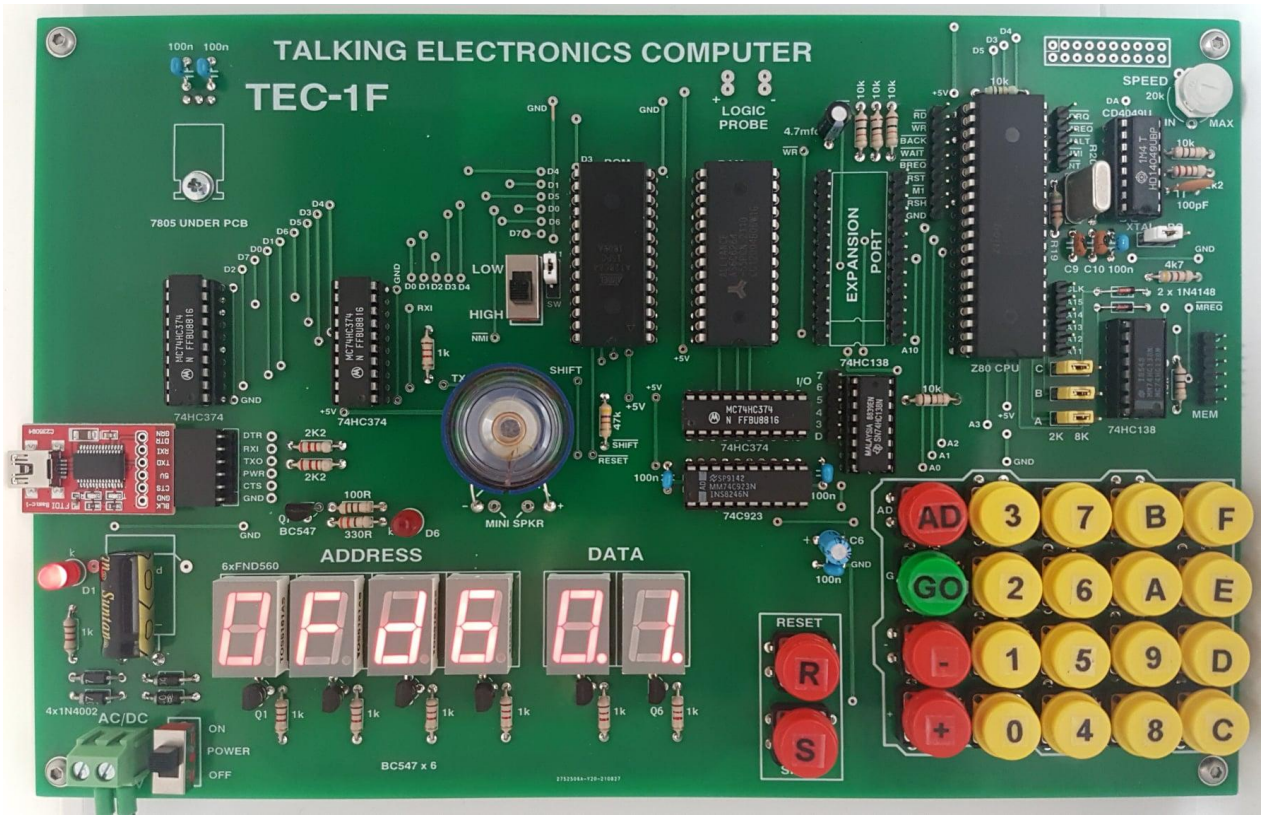
TEC 1C



TEC 1D


TEC-1E
SPEED
RESET
SHIFT
EXPANSION PORT

TEC 1E


TALKING ELECTRONICS COMPUTER
TEC-1F
LOGIC PROBE
SPEED
EXPANSION PORT
7805 UNDER PCB
LOW
HIGH
SHIFT
RESET
MINI SPKR
ADDRESS
DATA
6xFND560
AD
GO
RESET
AC/DC
ON
POWER
OFF
BC547 x 6
74HC374
74HC138
74C923
Z80 CPU
MEM

TEC 1F

Connect your TEC to the Zilog SIO

If you are a stickler for consistency and would like to use the Zilog chipsets for your serial communication then using the Zilog SIO (Serial Input / Output) chip is perfect for your data transmission needs. No need to send/receive a bit at a time, the SIO does all the hard work. This guide will go through the configuration, code and pin connections needed to get the SIO working on the TEC.

By Brian Chiha

The Zilog SIO is quite a powerful chip that can handle lots of different types of data transmission. Some of its features are:

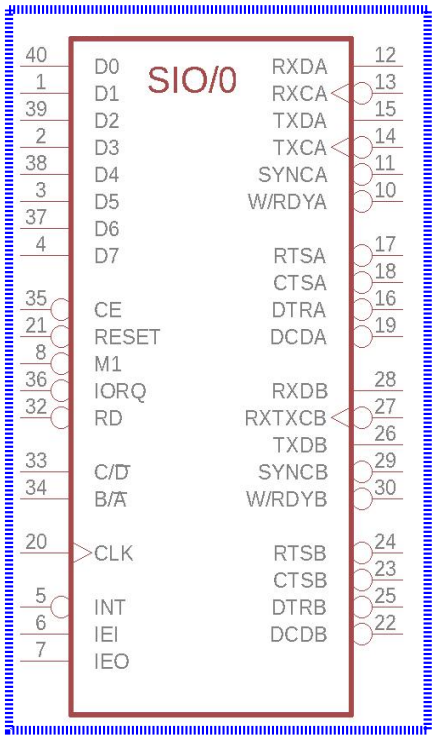
* Two independent full-duplex channels (A and B)
* Asynchronous and Synchronous transmission
* Buffered data for issues with lagging
* CRC and parity checking.

The SIO Chip

The chip comes in three different versions., SIO/0, SIO/1 and SIO/2. These versions only affect the 'B' channel and combine or drop some pins. Irrelevant for us, but just check the pinout before creating a PCB.

The pinout of the SIO isn't that complicated as most of the pins directly connect to the TEC. Looking at the pinout diagram, **D0-D7, RESET, M1, IORQ, RD, CLK** and **INT** are connected directly to the Z80 pins on the TEC. **IEO** and **IEI** (Interrupt Enable Input/Output) are for daisy-chaining and are not needed. **IEI** is to be tied high.

This comes to **CE, C/D** and **B/A. CE** is Chip Enable and this is connected to one of the TEC I/O Ports. In my setup, I use PORT 07. The SIO needs to know what Channel it is using **A** or **B** and if the data we are sending/receiving is a **Control Byte** or a **Data Byte.** Control bytes are used to set up the SIO and check the status of the SIO. Data bytes are the actual data being transmitted. When an IN or OUT command on the TEC is



performed, the chip is to be Enabled but <u>also</u> instruct the chip to which Channel is used and if it's a Control or Data byte. The TEC I/O decoder is connected to **A0-A2**. But we can also use **A3-A7**. I've connected **B/A** to **A4** and **C/D** to **A5**.

EG: If we want to send a Control Byte to Channel B, then we use **OUT (0x37), A.** A4 and A5 are high which gives us 3 and A0-A2 are high which gives us 7. Or receive a Data Byte from Channel A, then use **IN A,(0x07).** A4 and A5 are low, but still, need to activate Port 7.

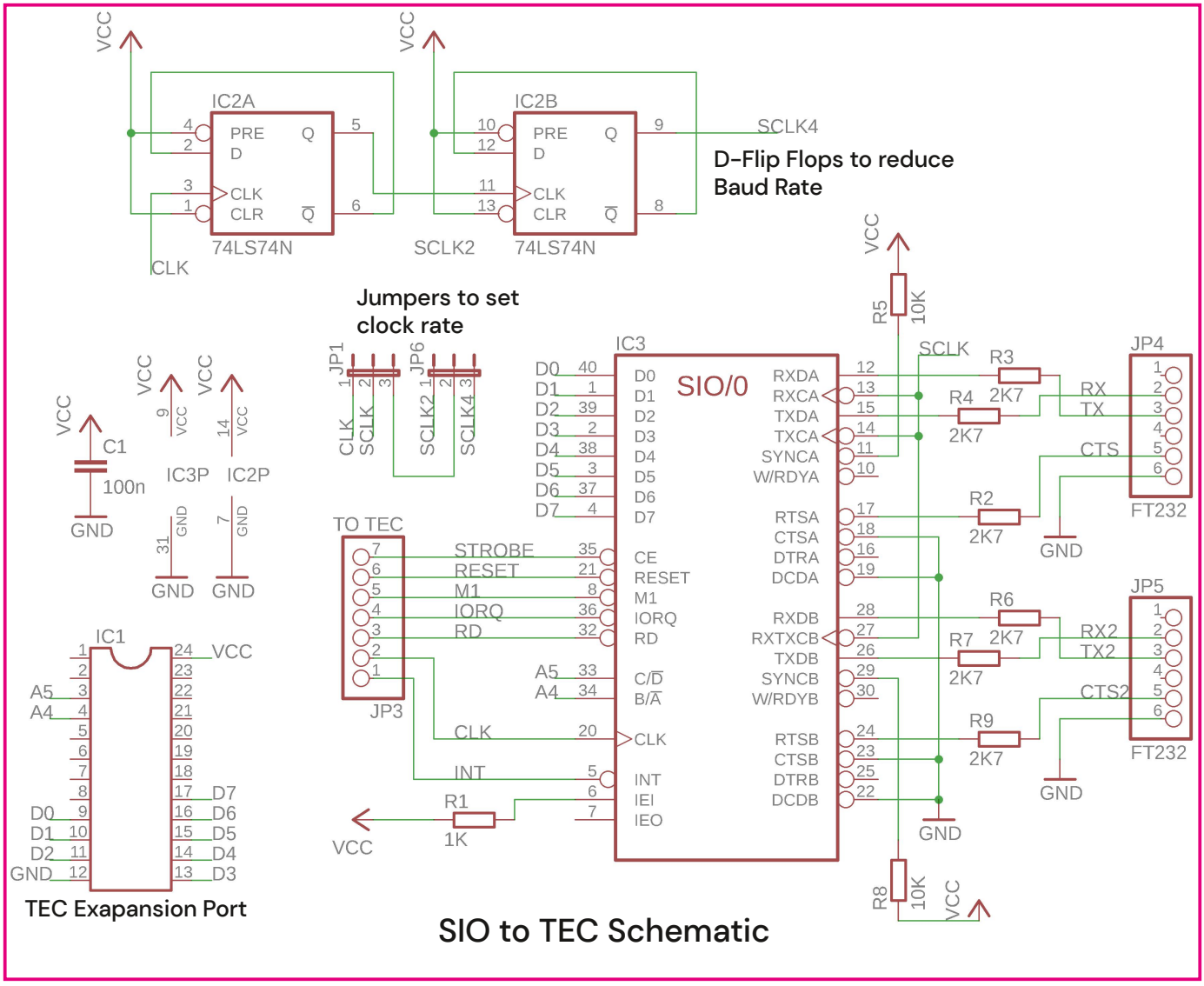
This is all the configuration pins connected. Now, lastly is the transmission Channel. We will only use Channel A. **RXCA** and **TXCA** is the BAUD rate for the transmission. They can be independently set but are usually connected together. Since I don't have a Zilog CTC (Clock) chip, I just connect these to the TEC clock. But wait!, doesn't the TEC run at 4MHz? Yes, it does, but the SIO can divide this clock rate down by 16, 32 or 64. Dividing by 64 gives us a manageable Baud Rate of 62500. Adding D-Flip Flops to the circuit can also bring down the Baud by one half and one quarter.

Lastly, the **RXDA** is the receive data pin, this is connected to the TX pin on the USB Serial adaptor, and TXDA is the transmit data pin, which is connected to the RX pin on the USB Serial adaptor. **SYNCA** is to be tied high as it's active low and isn't used. RTSA could be connected to the USB serial CTS pin and **CTSA/DCDA** can be tied to Ground as we don't want to block transmission. All other pins can be left as is.

Construction

Most of the pins needed to plug the SIO onto the TEC have access points on top of the board. These are the A4, A5 and Data pins on the Expansion port including GND and VCC. Port 07 on the I/O decoder and the rest from header pins. Except for IORQ, this is to be taken from underneath the board. Depending on what TEC you have, CLK might need to be taken from under the board too.

Connections to the USB Serial are to be protected using 2K7 Resistors. I recommend adding 2 D-Flip Flops to manage the Baud Rate. Where Q low, feeds back to D and Q high feeds into the next Flip Flop Clock. Then take the Q output you want.



Software Setup

To use the SIO, it must be configured to what type of serial transmission is required. To do this, SIO registers are to be set. This is where we use the Control Byte setting A5 high. There are Read-only and Write-only Registers. To communicate with a register, first select the Register, by setting bytes on WR0 and then send the desired configuration in the next output.

The Write Registers that are to be set are WR4, WR3, WR5 and WR1, and they are to be set in this order. On WR4 we need x64 clock divide, 1 stop bit and no parity. On WR3 we need 8-bits received, auto receive and receive enable. On WR5 we need 8-bits transmit and transmit enabled. Lastly, on WR1, we disable all interrupts. This might seem confusing (it was to me too), but look at the code to see which bits are set compared to the Register configuration.

Read register RR0 is also important as it lets us know when the transmission buffer is empty and if the SIO is ready to receive a byte. We check bits 2 and 0 for these.

In the code, setting the Write Registers is done by using the **OTIR** command. This command outputs data pointing to HL, B times to port C. Looking at the code, HL points to the CTLTBL. Then B is set as to count and C is set at the port. Once this is done HL points to the first Control Byte to send, and OTIR is called. The SIO is now ready to receive and transmit data.

The program requires a location to store or send the data (HL) and the length of the data (BC). The READ routine receives bytes from a Terminal to the TEC. It checks bit 0 (Rx Char available) in RR0, if it is set, then it reads one byte from the Terminal. Stores this value and repeats until all the data has been received.

The SEND routine transmits bytes from the TEC to the Terminal. It first sends a byte. Then it checks bit 2 (Tx buffer empty) in RR0. if it's still sending, it waits until the buffer is empty. Once empty, it sends the next byte and repeats until all bytes are sent.
The Z80 Assembly code and the SIO registers are on the next page. I use Coolterm as my Terminal and a USB to Serial adaptor to connect my computer to the TEC.

SIO Write and Read Register Configuration

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
WR0 Resetting/ Selecting	00 = Null Code 01 = Reset Rx CRC 10 = Reset Tx CRC 11 = Reset CRC/SYNCS		000 = Null Code 010 = Reset Interrupts 011 = Channel Reset (more options…)			000 = Register 0 001 = Register 1 …. 111 = Register 7	<<< Used to select the Register	
WR1 Interrupts	Wait/Ready Enable	Wait/Ready FN	Wait/Ready on R/T	00 = Rx Interrupt Disable (more options…)		Status Affecting Vector	Tx Interrupt Enable	External Interrupt Enable
WR3 Receiving	00 = Rx 5 Bits 01 = Rx 6 Bits 10 = Rx 7 Bits 11 = Rx 8 Bits		Auto Enable	Enter Hunt Mode	Rx CRC Enable	Address Search Mode	SYNC Char Load Inhibit	Rx Enable
WR4 General	00 = x1 Clock Mode 01 = x16 Clock Mode 10 = x32 Clock Mode 11 = x64 Clock Mode		00 = 8 Bit SYNC Character (more options…)		00 = SYNC Mode Enable 01 = 1 Stop Bit 10 = 1 1/2 Stop Bits 11 = 2 Stop Bits		Parity Even/ Odd	Parity Enable
WR5 Transmitting	DTR	00 = Tx 5 Bits 01 = Tx 7 Bits 10 = Tx 6 Bits 11 = Tx 8 Bits		Send Break	Tx Enable	SDLC/CRC-16	RTS	Tx CRC Enable
RR0 Status	Break/ Abort	Sending CRC/ SYNCS	CTS	SYNC/ Hunt	DCD	Tx Buffer Empty	Interrupt Pending	Rx Character Available

```
;SIO to TTY USB
;--------------
;Port 7 on TEC is connected to CE
;A5 is connected to  Control/Data, and A4 is connected
;to A/B
SIO_DA  EQU 00000111B ; A5 is C/D, and A4 is A/B
SIO_CA  EQU 00100111B
SIO_DB  EQU 00010111B
SIO_CB  EQU 00110111B

;Change these below to suit
ADDRESS EQU 0900H     ; Location of Data to receive/send
LENGTH  EQU 0050H     ; Length of Data to recieve/send

SETUP:
    CALL    INIT_SIO      ; Set up SIO for Async 8-N-1
    LD      HL, ADDRESS   ; Save start address in HL
    LD      BC, LENGTH    ; Save transmission length in BC

;;
;; Do either a JP READ or JP SEND Here
;;

;Receive data from USB,
;Stores data in HL address for BC bytes
READ:                     ; Read in data from USB
    XOR     A             ; Load A with RR0
    OUT     (SIO_CA),A    ; Select RR0
    IN      A,(SIO_CA)    ; Read RR0
    BIT     0,A           ; Check bit 0, RX Char. Available
    JR      Z,READ        ; If nothing, check again
    IN      A,(SIO_DA)    ; Read SIO for one byte!
    LD      (HL),A        ; Load A into HL
    INC     HL            ; Move HL to next location
    DEC     BC            ; Decrease BC
    LD      A,B           ; Load A with B
    OR      C             ; Or with C to check for zero
    JR      NZ,READ       ; If not zero then read again
    RST     00            ; All Done..Restart TEC
```

```
;Send data to USB
;Sends data at address HL for BC bytes
SEND:                     ; Send out data to USB
    LD      A,(HL)        ; Load HL to A
    OUT     (SIO_DA),a    ; Send out one byte!
CHECKSENT:
    XOR     A             ; Load A with RR0
    OUT     (SIO_CA),A    ; Select RR0
    IN      A,(SIO_CA)    ; Read RR0
    BIT     2,A           ; Check if TX buffer is empty
    JR      Z,CHECKSENT   ; Repeat until buffer is empty
    INC     HL            ; Move to next address
    DEC     BC            ; Decrease BC
    LD      A,B           ; Load A with B
    OR      C             ; Or with C to check for zero
    JR      NZ,SEND       ; If not zero then send again
    RST     00            ; All Done..Restart TEC

;Initialise the SIO for 8-Bit, No Parity, 1 Stop bit
INIT_SIO:
    LD      HL,CTLTBL     ; Point HL to Control Table
IPORTS:                   ; Get requires Bytes, Port and
                          ; Data
    LD      A,(HL)        ; Load Control Table (Bytes)
    OR      A             ; Test for zero
    RET     Z             ; Return if zero
    LD      B,A           ; Save bytes in B
    INC     HL            ; Go to Port Address
    LD      C,(HL)        ; Load C with port address
    INC     HL            ; Go to Data
    OTIR                  ; Output HL data, B times, to
                          ; port C
    JR      IPORTS        ; Jump to the next port if needed

CTLTBL:
    DB      09H       ; 9 Lines or Bytes
    DB      SIO_CA    ; Port Number
    DB      00011000B ; WR0: channel reset
    DB      00010100B ; WR0: select WR4 / Reset Int
    DB      11000100B ; WR4: presc. 64x, 1 stop bit
                      ; no parity
    DB      00010011B ; WR0: select WR3 / Reset Int
    DB      11100001B ; WR3: 8 bits/RX char, auto enable
                      ; RX enable
    DB      00010101B ; WR0: select WR5 / Reset Int
    DB      01101000B ; WR5: TX 8 bits, TX Enable
    DB      00010001B ; WR0: select WR1 / Reset Int
    DB      00000000B ; WR1: disable all interrupts
    DB      00H
```

The Register Table above is fully explained in the SIO Technical Manual. Only the Registers used in my code are displayed. Use the table to reference the bits set in the code.

References:

* SIO Technical Manual https://archive.org/details/Zilog_Z80-SIO_Technical_Manual
* Z80 Peripherals Manual http://z80.info/zip/um0081.pdf
* Circular Buffer on the TEC https://youtu.be/x_MR-b-TqW4

TECommunity Page

TEC Lovers from all around the world make up the TEC Community.

Craig Hart - Bio

Age: 51
Location: Adelaide
Profession: Senior IT Engineer

My first exposure to TE was as a teenager in the mid 1980's, with the 'cover projects' series. I didn't buy any, but did read them as much as I could in the newsagents after school. at the time I was interested in electronics and computers, and had some experience with an early 8 bit Z80 based computer called the Sega SC-3000, and Dick Smith Funway kits.

I first met the TEC around 1985, thanks to a TE magazine - possibly issue 13 or 14, which I saw in the local jewellers shop - who sold a few kits on consignment at the time. I saved my pennies and purchased a bare TEC-1B PCB (White on Red batch!) by mail order, then proceeded to buy parts as pocket money allowed, a little every few weeks. I eventually got the TEC going without keyboard buttons or 7-Seg displays, but the start-up beep was very rewarding - and it worked first go.

With time the TEC got finished and I religiously explored all the magazines had to offer. I purchased an SPO256 speech chip from Tandy in a clearance sale and built the interface, eventually succeeding to make a working design - my first ever scratch prototype.

By late 1986 I was looking for a job and TE was actually only one suburb away from my childhood home of Mentone, so I dutifully packed my precious TEC and veroboard speech module into the car, and talked dad into driving me up. I met Colin and showed him my work (after 45 minutes typing the opcodes in by hand); he was amazed and impressed at what I had created. He gifted me a speech module PCB which TE had developed independently; the designs were virtually identical, being basically a 'straight off the datasheet' design.

After a few regular visits I broached the subject of a job, and received 3 hours a Saturday working at 35 Rosewarne Ave. I was hooked. This evolved into a full time apprenticeship in 1988, doing just about everything from kit assembly, phone orders, mail runs, to design, testing, kit repair (from readers sending in things they couldn't get working), marking the digital electronics courses, and later, working at the TE shop.

TEC wise, I came to TE at a time after Ken Stone had left, so I volunteered to be the in-house computer guru of sorts. I burned ROMs by day and programmed by night. TE used an ancient Z80 development board called the MicroProfessor to burn all ROMs, as it was considered more trustworthy than the in-house design! Myself and TE staffer Paul set up a 286 with the desktop publishing system (Ventura Publisher) that issue 15 was produced on, and worked closely with Jim on finishing JMON and issue 15, for which I wrote the article to go with the speech module, did most of the artwork, diagrams and lots of other small jobs.

I also dabbled in the Microcomp and model railway computer, but the TEC was where all the focus was.

One TEC peripheral design never made it to market. It was called the MEGA-SIGN and was a 7-Seg display made of incandescent light bulbs - 4 or 5 to a segment. The PCB was laid out such that modules could be stacked horizontally or vertically with connectors at each edge that lined up. Hence, only one cable was needed. It loaded the Z80 bus so much that each board had a bus buffer chip on board - 74LS245 or similar. A PCB run was made but the kit never saw the light of day.

Another TEC project that never really got going, was a video adaptor for CRT displays/televisions. I never saw it, as it was worked on before my time/externally; it was effectively cancelled as it was too complex and costly - far more complex than the TEC itself - and had rather poor quality output.

Cont...

I was interested in CAD and helped TE move into 'modern' PCB design with a Roland plotter and Protel Autotrax, with external help from Ken who by then had returned to TE as a sort of contractor, to take photos and get out his second model railways book. A number of small, simple designs (The GNAT FM bug being perhaps the first in-house board) made it to CAD but I decided to teach myself Protel properly by converting the TEC to CAD. Hence, the TEC-1C was born, complete with JMON ready mod and other design revisions.

Issue 15 was really the pinnacle of my time at TE - one that was worked on quite hard by almost every staff member, for over 12 months. The move to a PC and CAD were big leaps and there were issues. Ventura was unstable and the computer was barely powerful enough to handle the page complexity of a TE magazine. Jim took up residence at TE for several months to see the DAT board and JMON articles through to completion. Spending time with Jim testing, debugging, documenting etc. was very enjoyable and I learned a great deal, especially some of the optimisations to fit JMON into 2k.

And then, there was Peter Crowcroft. Peter somehow found his way into TE around the time Issue 15 was reaching readiness for printing, supposedly as business development manager tasked with helping TE offshore some parts of the business. Kit production, magazine printing, component sourcing and the like was supposed to be moved to China allowing TE to get on with design work. In reality, Peter learned all he could about the business, then at short notice left, returned to his adopted Hong Kong, and started kits-r-us. Out of kits-r-us came the SC-1, an obvious evolution of the TEC sharing much commonality of design, but with some modernisation and restructuring of the hardware (e.g. 8k RAM/ROM) and just enough low level changes to potentially withstand a copyright suit. I will leave Craig Jones, SC-1 designer, to write more about this should he choose to do so.

I stayed at TE until roughly mid 1991. My last magazine was the Six BD-679 Projects book; a book created around the fact that Colin purchased 10,000 BD-679s at a distributor clearance sale for something like 5 cents each. The book was created solely to sell this inventory at a profit!! By 1991, Issue 15 was out and the initial sales surge been and gone, so there was preliminary talk of Issue 16 - Jim had ideas about releasing his utilities ROM, commented JMON disassembly and more in Issue 16 - however when it because obvious that TE was in decline and that Issue 16 was unlikely to ever start (much less get completed), Jim moved on to selling his 'Jim's Package' package instead.

By about 1990, Colin's focus had switched to FM bugs, phone tapping devices and bug detectors, and the TE shop. The TEC was lost by the wayside. Bugging was highly profitable (TE manufactured and sold them commercially, to private investigators and the public) but it was also illegal or at least legally dubious to do so. An article in the Age newspaper tipped of the Federal Police and TE was raided. I was at TAFE that day on my apprenticeship, so I was never involved as a defendant or called as a witness in the case. As far as I know, the ensuring legal outfall eventually ended the business, along with the general decline of electronics as a viable industry in Australia.

Having left TE, I moved into computers and away from electronics, although I did CRT monitor, PC power supply repairs and other electronics work for many years. My TECs got set aside and eventually lost in one or more house moves.

Blast forward to 2019, where I discovered the vintage computer forums on Facebook. Jim Robertson happened to contact me via the forums, and lo and behold he was looking for a password for a ZIP file of source code from 1990 that he created in order to keep it out of Peter Crowcroft's hands. By no small miracle, I got the password recovered and for the first time in 29 years, we viewed code Jim had written all those years earlier.

I was then introduced to the TEC-1 Facebook group and had soon reacquainted myself with all things TEC and SC-1.

Today I have two TECs and an SC-1 (both modified in various ways), and a bit of a passion for all that was, as well as all that could be. The TEC Facebook group has been an amazing resource, and I'm thrilled to see the little machine I had a hand in 30-something years ago, still a success today.

Baud Delay

EDITORS NOTE: To calculator the Baud delay, just use this formula: **T-states = Clock Speed / Baud Rate** For instance if the CPU clock is 4MHz and the desired Baud is 4800, the number of T-states is 4MHz/4800=833. The code that receives/sends one Bit must take 833 T-States to complete. Set the Baud delay to pad out the routine to achieve this. See http://www.overtakenbyevents.com/tstates/ to help with T-state calculations.

SPI and I2C Interface for the TEC-1 and SC-1

Presenting the SPI2C, a simple I/O interface and supporting software for interfacing with modern peripherals.
By Craig Hart

Introduction

One of the limitations of the TEC platform is a lack of 'usable' I/O to communicate with the outside world. TE Issue 14 brought us the I/O board, however this design was a very simple parallel style interface where each I/O pin was either a fixed input or output.

Many modern chips today tend to communicate using bidirectional, serial based signalling protocols – that is, one or two pins are used to transmit and receive data in a serial form. Until now, these devices were not compatible with the TEC-1 or SC-1.

SPI2C implements the two most common interfaces found across a large range of modern peripherals; firstly the SPI or Serial Peripheral Interface bus, and secondly the I2C (pronounced I squared C or simply I to C) bus.

Hardware Design

In the spirit of TE learning, we have not simply dumped a clever custom chip onto the Z80 bus. Instead, the SPI2C introduces a new design building block for the TEC –a bidirectional IO pin. That is a single pin that can both read and write data under software control.

We have built our interfaces using easy to understand 74xxx family TTL logic chips. In this way the entire operation of the design can be understood and followed.

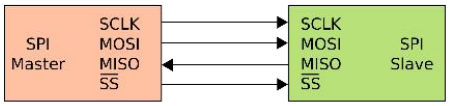
Of course, any hardware is only as good as the software it comes with, and so we are providing supporting software that implements the I2C and SPI bus protocols at the most basic level. Full source code is available via Github under an open source license, as are the PCB files.

This approach is far more educational than just 'talking to a smart controller chip' or calling a software library you don't have the source code to.

SPI - a simple bus

SPI is a very simple synchronous serial bus. It has a clock(SCLK) line, a chip select(SS) line, and two data lines read(MISO) and write(MOSI).

Don't be put off by the unfamiliar labels, just mentally substitute ('In' or 'read' for MISO, and 'Out' or 'write' for MOSI).



In the above image, the TEC plays the role of Master, and our peripheral device is the Slave.

When we say the bus is synchronous, we simply mean it has a clock(SCLK) line that dictates bus timing.

Data is considered valid when (and only when) the state of the SCLK line changes. Our code can be as fast or slow as we please and it does not matter what frequency the Z80 is running at, or if interrupts are occurring at the same time.

The SS line works just like the Chip Select (CS) line we are all familiar with – allowing multiple SPI slave devices to be connected to the bus, but only one device can be active at a time. Each SPI device has its own dedicated SS pin. The MOSI, MISO and SCLK lines are wired in parallel and shared by all devices.

This is where things start to deviate from the familiar TEC world.

The MISO pin is normally held in tri-state condition by all devices (nobody is outputting any logic level onto the bus). This is an input pun, so the SPI2C isn't outputting anything either; therefore a pull-up resistor is supplied to hold the bus at a logic 1 when idle.

Only when a slave device is actively sending data is the pin not tri-stated; the SPI standard says this pin is active LOW, so an SPI slave can pull this pin low to transmit data.

Theoretically, an unlimited number of SPI devices can share a single bus, however there are practical limits to the number of devices; our board offers up to 6 Chip Select lines; hence we can support up to 6 devices in the basic design.

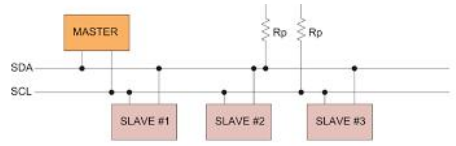
Some slightly non-standard SPI bus devices also have extra control pins for various tasks such as Reset, Backlight enable, mode selection or other custom functions. We can easily support these types of devices by simply adding extra control pins. The SPI2C provides the most common signals.

Duinotech XC-3714: an SPI bus based 8 Digit 7-seg display is available for around $10 from Jaycar etc.

I2C – something new

The I2C bus is a similar yet more advanced version of SPI. With I2C, the whole bus is reduced to just two wires – clock(SCL) and data(SDA).

The bus is also synchronous in nature just like SPI, however a very clever signalling protocol and some smart hardware design is



used to eliminate the other control wires completely. Both the clock and data lines use the same active-low-tristate hardware approach as SPI.

Chip select is integrated within the bus protocol (each I2C device has a unique 7-bit 'address' on the I2C bus), hence the SDA pin performs 3 functions - data in, data out and device select.

SPI offloads all the hard work to software, allowing the hardware to be as simple as possible.

Hardware Operation

The hardware design can be broken down into several sections, as follows.

I/O address decoder

The 74HC138 address decoder is functionally similar to the TECs own 74LS138s – except it decodes the IO address range 40h to 7Fh. Note that we ignore address line A0, so every odd port is a duplicate of the preceding even-numbered port.

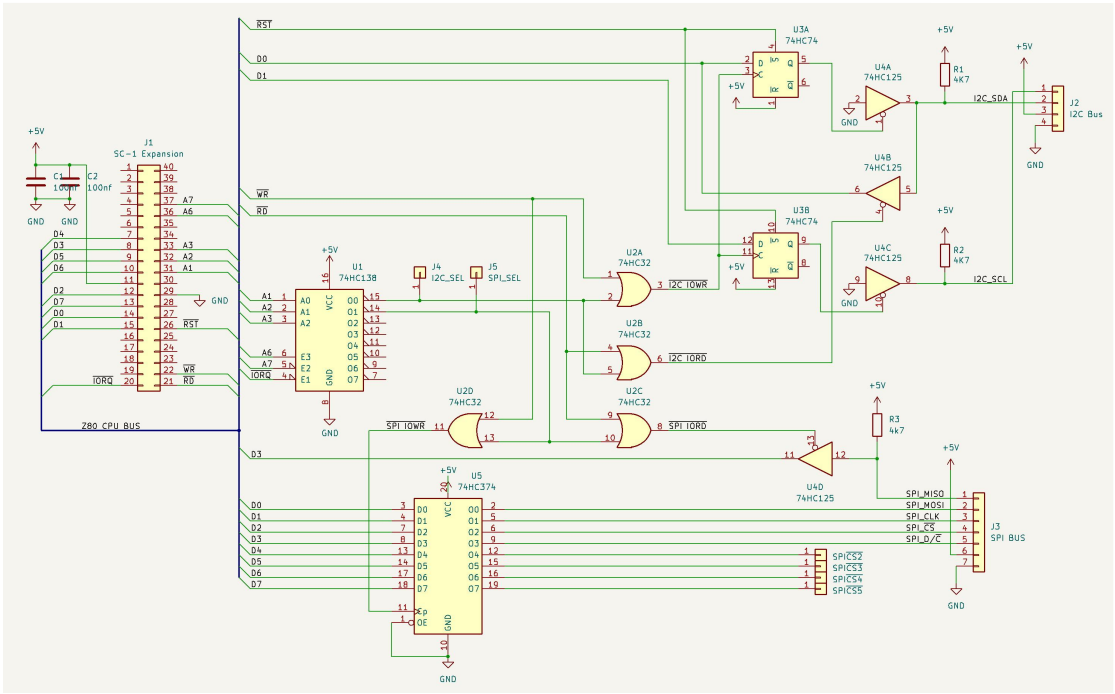
This means port 40h and 41h are an equivalent pair as far as software is concerned (...and port pairs 42h & 43h, 44h and 45h, etc.) so code can address either port and read or write the same device.

Also, since we are ignoring address lines A4 and A5, the I/O ports 'wrap around' every 16 addresses, so ports 40h, 50h, 60h and 70h all access the same device.

The reason for this, is that the 74HC138 just doesn't have enough pins to fully decode just one I/O port. The Z80 has plenty of IO port space so we don't really care about the overlap.

SC-1

The SPI2C is designed to 'plug in' to the SC1's Expansion connector with no further modifications.



SPI2C Circuit Diagram

TEC-1

The 74HC138 chip should be omitted entirely if you wish to use the TEC's built in IO ports e.g. ports 6 and 7.

Omit the 74HC138 and connect the I2C_SEL and SPI_SEL pins to the TEC's IO port select pins 6 and 7 instead.

I/O read and write cycle decoder

The purpose of the 74HC32 is to take our port select signals from the 74HC138 (or TEC) and combine them with the Z80's RD and WR signals, to create separate IORead and IOWrite signals.

This is important because different chips are used for the read and write functions, so they need to be selected individually based on whether the Z80 is executing an IN or OUT instruction (IOread or IOwrite cycle).

SPI controller

The 74HC374 is responsible for the write side of the SPI bus.

The SPI_IOWR signal from the 74HC32 enables this chip only when data is written to the SPI port – and all 8 bits are latched by this chip.

Reading from SPI devices is performed by Gate 4 of the 74HC125, and simply gates the MISO pin onto the Z80 data bus during a read cycle. The SPI bus's MISO pin is held in tri-state (high impedance) mode whenever data is not actively being sent by a slave device, hence this pin is shared by all SPI slave devices.

The 4k7 pull up resistor holds the MISO line high when idle, hence our software will read a '1' whenever the bus is idle. SPI devices only ever output a logic 0 to the MISO pin (*never* a logic 1!!)

Reads are gated onto the Z80 data bus to bit D3 by the 74HC125 in response to a decoded IORead cycle.

I2C controller

The I2C controller is a little more complex than SPI. I2C pins are active low; therefore held high by pull-up resistors (devices tri-state) whenever idle.

We need to create an SDA pin that supports the following abilities:
* Be at logic 1 when idle
* Output a logic 0 or 1 during our writes
* Read an input 0 or 1 state from a slave device (and not our own previous output) during reads
* Be independent of the SCL pin

We can't simply use a 74HC374 here as it is not bidirectional and other 74HC24x family chips won't suit either as we can't address pins individually (e.g. have one pin as an output at the same time another pin is set as an input).

To overcome these limitations, we have created two independent one-bit latches using a 74HC74 flip flop and 2 gates from the 74HC125 as our tri-state buffers.

SDA pin

Firstly, the 74HC74 latches our write of bit D0.

If our write is a logic 1, the I2C bus is placed into tri-state as the 7HC74's output latches this value and the logic 1 sets the HC125's output to tri-state mode. Since the slave devices are also idle/tri-state, and the 4k7 resistor pulls the bus up to logic 1.

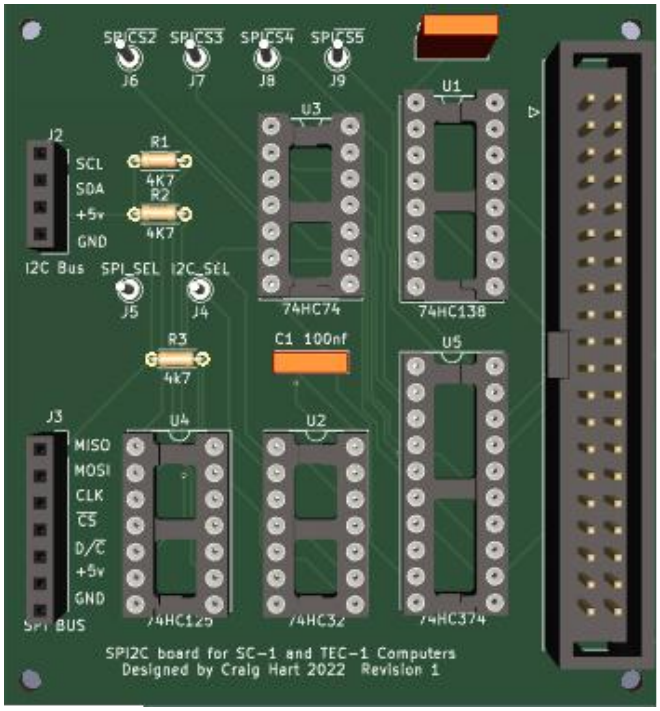
If we write a logic 0, this is latched by the 74HC74 and then a 0 is gated onto the bus by the 74HC125.

Note the input of the 74HC125 is wired to ground – hence we can only ever output a 0 to the I2C bus, never a 1.

Input is the same as for SPI – the other 74HC125 gate simply gates the SDA line directly onto the Z80 data bus during an IORead.

The observant amongst you would have noted – to read valid data from the SDA pin, we need to first output a '1' to the SDA pin, in order to set it to tri-state mode;
otherwise a bus short could occur.

The Z80 reset line sets the 74HC74 flipflop to a logic 1 so as to ensure the I2C bus is tri-stated at power up.



SPI2C PCB

Our software should also ensure a bus clash never occurs.

SCL pin

The SCL pin works the same way as the SDA pin, except it is only ever driven as an output, so we don't need a '125 gate on this pin, and any data read of this bit will be undefined.

Assembly

Any seasoned TEC builder should find assembly straight forward; simply load the parts onto the PCB as shown, the order isn't critical.

However, first decide if you will be building the SC1 version or the TEC version.

SC-1

The SC1 version is easy – fill all components as shown. The board is designed to plug directly to the expansion header on the SC-1.

TEC-1

Omit the 74HC138 and instead connect I2C_SEL and SPI_SEL pins to the TEC's IO port select pins for ports 6 and 7.

As the board uses a 40 pin Z80 bus header you will also need to pick up various signals from the TEC Expansion port and the Z80 CPU. Some soldering to the TEC PCB will be required to pick up all the required signals.

Alternatively, use Craig Jones' TEC CPU riser board to create an SC-1 style expansion bus connector.

Parts List

U1	74HC138	(SC-1 only)
U2	74HC32	
U3	74HC74	
U4	74HC125	
U5	74HC374	
C1	100nF	
C2	100nF	
R1	4.7k	
R2	4.7k	
R3	4.7k	
J1	40 pin IDC connector	
J2	4 pin SIL header	
J3	7 pin SIL header	
J4-J9	single pin header	

IC Sockets

3	14 pin
1	16 pin
1	20 pin
1	SPI2C PCB
1	40 pin IDC Cable

Software

The key to this project is the really the software; the hardware itself simply handles the electrical signals, and all the smart control is in the software alone.
The latest versions of all the software as well as schematic, PCB design, supporting datasheets etc. are available at my GitHub site
https://github.com/1971Merlin/SPI2C

A detailed line by line description of the software is beyond this article, however a quick review as follows:

spi_7segs

This program tests basic SPI bus functionality, using a MAX7219 8 digit 7-segment display module such as the Duinotech XC-3714 which is readily available from Jaycar for around $10.

The program simply scrolls a display buffer in RAM across the displays, but it demonstrates several code blocks in doing so – initializing and writing to the SPI bus, as well as managing a simple display buffer.

As a side note, it's mazing that you can buy EIGHT 7-seg displays, on a PCB, assembled, with a controller chip, pin headers etc. for just ten dollars. To build this yourself from discrete parts would be many times more expensive – hence the value in using these cheap modules and leveraging the wealth of products already out there flooding the Arduino market.

i2c test

This program builds on spi_7segs, adding support for an I2C bus based DS1307 Real Time Clock chip – again we are using the Duniotech XC4450 from Jaycar.

The program implements a simple clock reading from the DS1307 and displaying the time/date on the MAX7219 displays as well as on the internal 7-seg displays so you can check it is working correctly.

Obviously this program introduces the I2C bus control code, however it is a much more 'complete' program that includes several utility routines such as display scanning, keyboard polling. The program contains conditional assembly directives to assemble for either the SC-1 or TEC-1 hardware platforms.

Game of life 5110

This program uses a cheap 84x48 mono LCD display, also known as a 'Nokia 5110" display driven form the SPI bus. Once again this is a Jaycar part XC4616 and is less than $20.

John Conway's Game of Life is presented on the LCD, and is a good example of a practical use for the system as a whole.

Compiling the code

The code is written and assembled into the final binary using Telemark TASM – the table based assembler, which is available from

https://www.ticalc.org/archives/files/fileinfo/250/25051.html

To compile the code, simply run:

```
tasm -80 -g0 filename.asm
```

from a command prompt, and it will output an .obj file in intel hex, and a .lst file which is a text file showing the commented listing and the hex bytes to be keyed in.

I personally take TASM generated .obj file and use the the SC-1's serial upload feature (Fn 1) to upload the code from a PC at 4800 baud.

Of course you can type the HEX bytes in by hand and write your code with pen, paper and opcode lookup tables the way TE did, but you quickly realise this is not practical when developing software.

I strongly suggest and encourage use of a modern development environment, so you can focus on writing the software and not mundane tasks like looking up opcodes.

There are numerous development platforms such as Oshonsoft.com and asm80.com available online.

Where to From Here?

Having introduced some basic routines and the 'raw' code, well the sky is the limit in terms of possibilities.

Just about any SPI or I2C bus device can be used – and my goal is to build over time a library of subroutines to support numerous devices.

Any code will of course be published on Github as it is developed. And I encourage readers to become familiar with the various TEC Github resources available in the community.

Beyond this, I encourage and welcome you, dear reader, to start experimenting. Feel free to provide feedback, bug fixes and new features via Github and the TEC-1 Facebook group, and lets see what we can grow from here.

I look forward to receiving your input.

All source code and examples are available from my Github
https://github.com/1971Merlin/SPI2C
All my software is free, open source and GPL licensed

Serial Routines for the TEC

This article was published in the "TEC Times" March 1990. It is the last known publication for the TEC. The Bit Bang code can still work on the TEC and it shows another way to serialise data.
By Jim Robertson

RECEIVER

This is the routine I use when I wish to download a file from the IBM. It's a simple routine that converts a serial stream into bytes and stores them in RAM starting at the address provided at 0x0898. The routine also has an end address to allow a maximum file length. This is in case something goes wrong with the data transfer. Anything important can be protected by placing it above the end address.

No hand-shaking is needed as the TEC can cope with the speed of the data stream. It is up to you to ensure the TEC is ready before you send the data. The serial input is bit 0 of PORT 3. The DAT BOARD has provision for 2 diodes and a resistor at this input to clip an incoming RS232 signal. In the RS232 format, a logic 1 is represented by a negative voltage while a logic 0 is a positive voltage. The clipper on the DAT BOARD changes an RS232 logic 0 (positive voltage) into a digital logic 1 while an RS232 logic 1 is clipped to zero volts and becomes a digital logic 0.

This means that the inputted data must be inverted back into its true form. This is done with the CPL instruction at 0x092C. The format of the data is as follows:
* 2400 BAUD
* NO PARITY
* 8 BITS
* STOP BITS OPTIONAL
* TEC SPEED: 3.58 Mhz / 2

IBM SOFTWARE

The software I used for receiving the serial is PROCOMM. It is a public domain program and can be purchased from the Talking Electronics Shop. Cat S-449. The protocol to use is ASCII.

```
;Constants
START_ADR:  EQU     0898H           ;Start address
END_ADR:    EQU     089AH           ;End address
IN_PORT:    EQU     03H             ;Input port
OUT_PORT:   EQU     01H             ;Output port

            ORG     0900H

; --------------------
; Serial Input Routine
; --------------------
; Transmit bit is BIT 7 on the selected port
INPUT:
            ;Start of routine
            LD      HL,(START_ADR) ;Load HL with start addr.
RECLOOP:    ;Get a byte and increase HL until HL=BC
            CALL    GET_BYTE        ;Call Get Byte routine
            LD      BC,(END_ADR)    ;Load BC with end address
            OR      A               ;Clear carry flag
            PUSH    HL              ;Save start address
            SBC     HL,BC           ;Get length of transfer
            POP     HL              ;Restore HL
            JR      C,RECLOOP       ;More data to receive
            RST     00H             ;End

GET_BYTE:
            ;Receive a byte from serial port
            IN      A,(IN_PORT)     ;Check bit 7 on in port
            RLCA                    ;Move into carry
            JR      NC,GET_BYTE     ;Not sent, keep looking
            LD      B,40H           ;Load B with half baud
                                    ;delay
L1:         DJNZ    L1              ;Delay
            LD      E,00H           ;Load E with 0
                                    ;the receive byte
            LD      C,08H           ;Eight bits to receive
GET_BITS:
            IN      A,(IN_PORT)     ;Get bit 7 on input port
            RLCA                    ;Move into carry
            RR      E               ;Rotate Carry bit into E
                                    ;from the right
            LD      B,39H           ;Load B with baud delay
L2:         DJNZ    L2              ;Delay
            DEC     C               ;Decrease next bit count
            JR      NZ,GET_BITS     ;Still more bits
            LD      A,E             ;Save received byte in A
            CPL                     ;Invert bits
            LD      (HL),A          ;Store in memory
            INC     HL              ;Move HL to next address
            RET                     ;Exit
```

The sending software poses a few difficulties. One big problem is that some packages won't send the 1A character. Actually, I believe the problem is in the DOS serial interrupt and if the software uses it then it won't send the 1A character.

It is rare that I send anything back to the TEC and when I do, it's with a serial routine Craig Hart wrote and probably won't work with all computers as it directly manipulates the hardware; not a recommended practice. It is up to you to experiment around and find something that works.

I would like to hear from anyone who has found or written a good sending routine that doesn't have the 1A character problem.

Hardware wise, the CTS must be taken high before the IBM will send the data. This means that the IBM to TEC link consists of three wires: the ground, the serial data line and +5v.

Only ground and the serial data are required for the TEC to IBM link.

SERIAL OUTPUT ROUTINE

This is the complement routine of the serial receiver. It will send serial data through the TEC speaker bit. The data is taken from the latch side of the base resistor of the transistor inverter and inputted directly to an RS232 Rx input or the DAT BOARD serial input.

Strictly speaking the data stream is not RS232 compatible but in practice it works ok, although the occasional error may creep in.

Oh yes, before sending data, the key press beep must be turned off. To do this, place 0xFF at 0x0822 and put 0xAA at 0x08FF.

The serial sender uses the same start and end buffers as the receive described above with the same speed etc. Two stop bits are sent as this provides compatibility with all serial systems.

*This article is a reproduction from the 'TEC Times' published March 1990.

```
OUTPUT:
            ;Start of routine
            LD      HL,(START_ADR) ;Load HL with start addr.
SENDLOOP:   ;Send a byte and increase HL until HL=BC
            CALL    SEND_BYTE      ;Call Send Byte routine
            LD      BC,(END_ADR)   ;Load BC with end address
            OR      A              ;Clear carry flag
            PUSH    HL             ;Save start address
            SBC     HL,BC          ;Get length of transfer
            POP     HL             ;Restore HL
            JR      C,SENDLOOP     ;More data to receive
            RST     00H            ;End

SEND_BYTE:
            ;Send a byte to serial port
            LD      A,80H          ;Send start bit
            OUT     (OUT_PORT),A   ;Output to port
            CALL    DELAY          ;Delay for baud rate
            LD      A,(HL)         ;Get byte from memory
            INC     HL             ;Move to next byte
            LD      B,08H          ;Load B with bits to send
SEND_BITS:
            RRCA                   ;Put bit in bit 7
            XOR     80H            ;Flip bit
            OUT     (OUT_PORT),A   ;Output to port
            CALL    DELAY          ;Delay for baud rate
            DJNZ    SEND_BITS      ;Send next bit
            XOR     A              ;Zero A
            OUT     (OUT_PORT),A   ;Send Stop bit x 2
            CALL    DELAY          ;Call first delay
DELAY:
            ;Delay for baud rate
            PUSH    BC             ;Save BC
            LD      B,36H          ;Load B with delay
L3:         DJNZ    L3             ;Delay
            POP     BC             ;Restore BC
            RET                    ;Exit
```

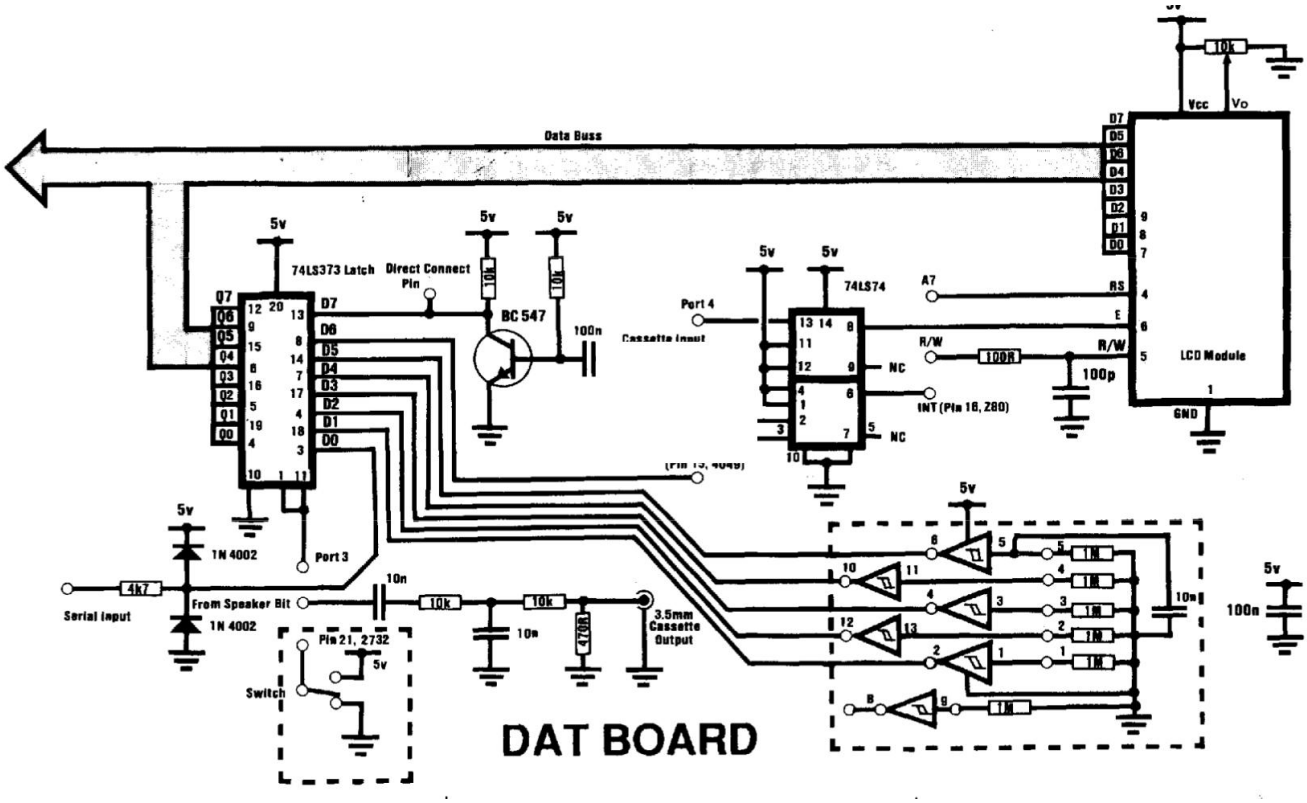


Image reproduced from TE Magazine Issue 15