Das BlinkenLights and Single Stepper

The TEC's biggest expansion board yet!

By Craig Hart

Introduction

Welcome to possibly the most complex board ever designed for the TEC & Southern Cross SC1!

Das BlinkenLights (henceforth DBL) is a multi-function add-on board designed for experimenters and programmers alike. DBL features the following main functions:

36 Realtime Z80 status LEDs

Visualize the state of the Z80 in real time – address, data and control busses all broken out into individual LEDs.

Independent RESET button

Reset the CPU any time without needing to reach across to the main machine keypad.

Single Clock

Step the CPU one T-State at a time.

Observe the operation of the Z80 in detail – both rising and falling clock edges can be held to verify the exact CPU timing and bus activity.

Slow Clock

Drive the CPU clock at low speeds from ~10Hz to ~10KHz). Allows programs to be slowed down so that individual instruction cycles are visible on the status LEDs as they execute.

MONSTEP Single Stepper

Supports JMON, SCMON and BMON software based single steppers triggered via the INT pin.

Independent Single Stepper

Step the CPU one machine cycle at a time, any time, without software modification; uses the CPU's WAIT pin to 'pause' operation.

How it Works

Each of these functions can be broken down into its own module and included or not, as you see fit.

Some of the functions are fairly complex and involve interaction with and/or mods to the TEC/SC1 to work, so please bear with me as we deep dive into the technical side of the design.

Have no fear however, you can enjoy and use the DBL board without needing to understand all the "hows" and "whys" it works.

Bus Status LEDs

The CPU bus signals are buffered by the 5 x 74HC245 chips and used to drive the various LEDs, which are grouped logically into Address Bus, Data Bus, and the various CPU control signal groups.

74HC245 may seem like a strange choice compared to the 74HC273 (etc) which TEC users would be familiar with, however I wanted to use a chip with a 'straight through' pinout, as well as introduce a new type of chip which I intend to use in future projects.

NB: We are not using the 'bidirectional' or 'tri-state' capabilities of the '245 in this design.

Reset Button

This is simply a momentary switch the grounds the RESET line. There is no need for pull-up resistor or cap, as the components on the main computer board still perform the reset timing actions.

Single Clock

The DBL board can optionally drive the CPU's CLK line instead of the TEC/SC1. This allows for single pulse clocking, in order to observe the CPU one T-state at a time.

NB: Single Clocking requires a CMOS Z80 CPU, as we are effectively halting the clock between pulses. NMOS CPU's don't support a stopped clock, and will corrupt the CPU state if no clock pulse occurs for more than a few hundred milliseconds.

Clock pulses are created by a debounced button (in fact the 3 key buttons are all debounced) using a simple RC network and a Schmitt trigger gate.

The CLK signal is then gated to the CLK line using a gate from a 74HC125 buffer chip to ensure a clean signal, and provide on/off control as required.

This option requires the original TEC/SC1 clock source to be DISABLED at the computer.

To disable computer CLK

Southern Cross

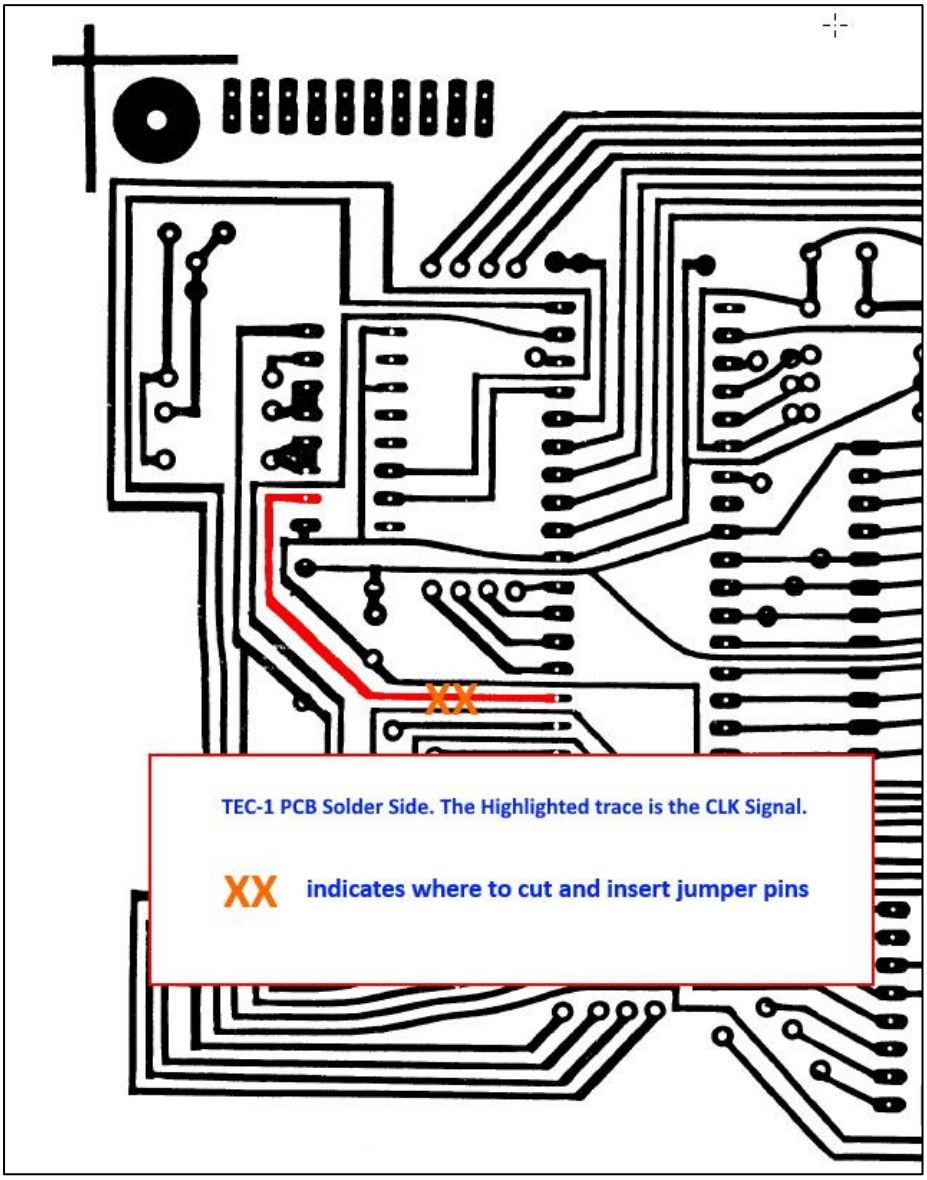
Remove the Fast/Slow switch and replace with a 3 pin jumper header. Use a jumper to select Fast or Slow, or remove the jumper entirely to allow DBL to provide the CLK source.

TEC-1F

Similar to the SC1, remove the XTAL/RC switch and replace with a 3 pin jumper header. Use a jumper to select XTAL or RC, or remove the jumper entirely to allow DBL to provide the CLK source.

TEC-1D and earlier

These machines don't have a CLK jumper option, so the best choice is to lift pin 2 of the 4049, or modify the PCB to add a jumper – see diagram.



Modifying a classic TEC to add jumper pins to the CLK line

To use the single clock, remove the CLK jumper from the TEC/SC1 and install it onto the CLK DRV board of the DBL instead. Transferring the same jumper from one board to the other helps avoid two CLK sources being active at the same time.

Set the Run/Pulse switch to PULSE and press the SINGLE CLK button to provide a single clock pulse. Hold the button down to work one CLK edge at a time and observe the CPU state on the bus status LEDs.

Slow Clock

Slow clocking allows the CPU to be clocked at a few instructions per second up to a thousand or so per second.

This allows software to "run" at a much reduced speed, in turn allowing the bus display LEDs to show what is happening in a pleasing display of flashing lights – our "Das BlinkenLights" for which the project is aptly named.

Use your best German accent when pronouncing please!

The slow clock is generated by the classic 555 timer chip running in astable mode, with a selection of timing capacitors and a 20k trimpot offering a wide range of selectable clock speeds.

To engage slow clock, set the CLK DRV jumper per single clock mode, and set the Run/PULSE switch to PULSE.

Use the Range jumper to select 'slow', 'medium' or 'fast' mode (10Hz, 100Hz or 1000Hz) and adjust the trimpot for best results.

Note: The MONitor will start up within about 10 seconds when clocked on the highest 'Fast' setting – at lower settings it may take several minutes before any audible sign of life starts to happen.

Lowering the CLK speed will also show the individual 7-segment displays being scanned and the speaker clicking on and off as it plays the startup tone.

MONSTEP Single Stepper

This is the 74HC74 based flip-flop circuit lifted directly from the TEC's DAT board/SC1 'Improved" stepper (both work identically).

MONSTEP requires JMON, SCMON or BMON – it won't work with MON-1 or MON-2 as these MONitors don't have any support for single stepping built in.

You will need to connect the CS pin from the TEC/SC1's ROM socket to the ROM CS input on the DBL board and install the INT jumper.

The circuit generates a hardware interrupt to the CPU INT pin each time a CPU M1 instruction fetch cycle occurs. The INTerrupt handler at `0038H` then launches the stepper display.

To prevent the stepper trying to step itself (Which would cause an infinite loop), the ROM CHIP SELECT line is picked up from the TEC/SC1 and connected to the ROM CS input pin on the DBL board. This disables the stepper hardware whilst ROM code is executing.

A limitation of MONSTEP is that the single stepper code has to be present in ROM and can't reside in RAM memory; it means you can't single step ANY of the MONitor ROM itself.

We have replaced the SC1's software stepper toggle (Fn 7) with a manual enable/disable push button and status LED, as the software toggled approach was deemed inflexible, as well as consuming an IO port. Fn 7 will not alter the stepper's state and is ignored.

Single Stepping is enabled whenever the MONSTEP LED is lit.

The stepper is always disabled at hardware RESET to avoid any potential for a conflict preventing proper monitor start up.

The single stepper can still be called from software by inserting a `RST 38H` opcode anywhere – even if MONSTEP is not enabled.

Stepping and the Program Counter Register

In the Z80, the PC register points to the NEXT instruction whenever an interrupt is generated. This means that by default, the PC value is incorrect.

The SC1 stepper simply accepts and displays this as-is, whereas JMON/BMON saves the previous value of PC from the last time the stepper was active and displays that. Therefore, JMON/BMON can display an incorrect PC each time MONSTEP is first turned on.

Advanced Notes – INT pin

The INT line from the stepper to the CPU is provided on a jumper. The jumper should normally be installed.

However, if you're working with interrupts from other sources such as a high speed serial chip or a Z80 CTC, you may wish to tap off the INT signal and route it via another path to manage your interrupts, in which case remove the jumper and pick up the Stepper's INT output from the square pad (left hand pad) on the PCB.

Note that this signal is driven active low only and relies on the TEC's built in pull-up resistor on this line. In this way the DBL board can share the INT pin with other hardware directly, however there is no way to poll the stepper in software to know that it was the source of the interrupt.

Independent Single Stepper

Sometimes you want to single step code you don't have the source code to; you want to single step the ROM and/or you're not using a MONitor with single stepping code support.

For these scenarios, we provide the Independent Single Stepper (henceforth ISS).

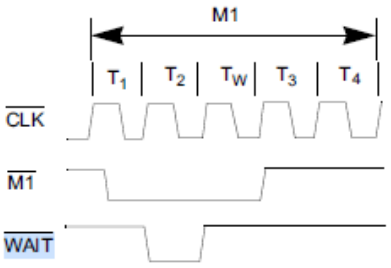
This circuit uses two flip flops. FF1's Q output is normally high when the CPU is not executing an M1 cycle.

FF1 detects the start of the M1 cycle (T2, rising CLK), and sets its Q output low accordingly. If enabled with Run/STEP, this sets WAIT and the CPU enters the wait condition at the end of T2 and after fetching the opcode. The CPU then insets continuous WAIT cycles Tw until WAIT is cleared.

FF2's idle state is with its Q output high. Once WAIT is set low by FF1, FF2 samples this when a pulse is received from the SINGLE STEP button, hence toggling it's Q output Low.

Once FF2's Q output goes low, this forces FF1 to immediately SET its Q output high, hence clearing WAIT and allowing the CPU to begin T3.

FF2 is itself also returned to the idle state on the rising edge of T3, hence allowing FF1 to resume normal operation.



To enable the ISS, set the Run/STEP switch to STEP mode. In RUN mode the ISS is disabled and the machine runs at full speed.

Pressing the SINGLE STEP button repeatedly is tedious, so you can also press the PULSE button to step at the rate set by the Slow Clock settings; you can also flip the Run/PULSE switch to PULSE mode to step automatically at the slow clock rate.

Note: You may flip the Run/STEP switch at any time without fear of crashing the system. Because the WAIT pin is used to hold the CPU, this mode works with NMOS Z80's.

Also note: using both ISS and Single Clock together at the same time will not work due to timing issues. No damage will occur; the system will most likely just crash. Feel free to experiment!!

Power Consumption

The DBL board is powered from the TEC/SC1 via the 40 pin cable.

We note that the board can draw a lot of power – with 38 LEDs drawing ~15mA each, the board could theoretically draw about 0.6 amp (Worst case).

However, the LEDs are usually toggling on and off rapidly, so the nett current draw is an average of around 200-250mA.

As power is drawn from the TEC's 7805 regulator, so expect it to get a bit warmer. You may wish to add a heatsink (which can be mounted on the top side of the PCB using the original 7805 mounting bolt).

The TEC with an NMOS Z80 at 4MHz draws about 200mA when running the MONitor (and about half that for a CMOS CPU), so the 7805 is still well under spec. for the total increased load of around 450mA.

If 7805 heat is a problem, try to keep the input voltage low - 9v AC is plenty. The higher the 7805's DC input voltage, the warmer it will run.

Hardware Debounce

One of the things we need to single step accurately, is a debounced button. Without debounce, we may accidentally allow multiple instructions or deliver multiple CLK cycles, leading to an erroneous interpretation of the CPU state.

In the DBL, debounce is performed by the 10k, 100R, 100n and the Schmitt trigger inverter gate associated with each button.

The secret is in the characteristic behaviour of the Schmitt Trigger itself, which has a characteristic called hysteresis. Hysteresis means the gate has two different threshold voltage levels which have to be crossed before the gate will change output state.

The thresholds are normally at 2/3 and 1/3 of supply voltage.

Lets say our NOT gate is currently outputting a 0 (therefore it's input is high). To set the output to 1, the input logic level must fall below 1/3 supply voltage, but the input can also rise back up to just under 2/3 of supply voltage WITHOUT altering the gate's state. This ability to ignore small changes in the input such as noise, is what we rely upon.

In our circuit, normally the gate's input is HIGH from the 10k & 100R charging the 100n up to +5v. Hence, the output is LOW.

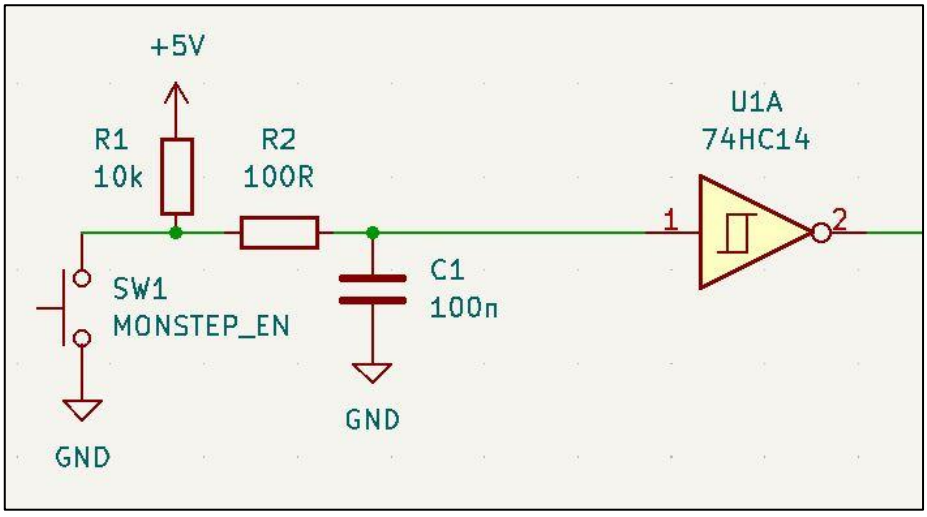
When the button is pressed, the 100n discharges rapidly via the 100R and the gate sees a LOW. The input cannot bounce high as the 100n will absorb any positive transients and prevent the input crossing the upper threshold..

Similarly, when the button is released the 100n charges and the gate switched to a high output once the upper threshold is crossed. The 100n absorbs any negative transients and prevents the input from crossing the lower threshold.

The 100R is primarily present to protect the switch contacts from arcing and pitting by limiting the discharge current of the 100n. It also protects the Schmitt Trigger's input from seeing extremely fast overshoot pulses, which could potentially glitch the gate itself.

The debounced button (or switch) is a very useful circuit module and one that is cheaply implemented with just a few components.

We will make further use of debounced buttons in future projects.



Hardware Debounce Components

Building the DBL

The DBL board can be considered modular in nature. You can build or omit the bus status LEDs, Reset, MONSTEP, Clock and/or Single Step sections as you see fit. We do hope you'll build the whole thing though!!

The parts list and following sections indicate the components required for each section. Assembly is merely a matter of fitting the parts for each section as required.

Both 6mm and 12mm push buttons have been allowed for in the design – fit either type as you please.

Two tricks for soldering in the very small Headers-

1. Put on a jumper cap first. This gives you something non-metallic to hold with a finger, and also provides better tactile feedback to tell when it's straight.

2. Initially solder one pin only – then check the jumper is straight and square. It's easy to re-melt one pin and adjust, then than try to fix the whole thing after soldering it fully.

Common Parts

Red Components

All versions of the PCB require the 40 pin IO connector, power LED and associated 330R resistor.

The Reset button should also be considered a common part, install if required.

Bus LEDs

Cyan Components

To assemble the bus LED's section, fit as follows:

- 4 x 74HC245 ICs
- 36 x 330R resistors (or your preferred value)
- 36 x LEDs
- 3 x 100n filter caps

A word on LEDs

The board was designed with 3mm LEDs in mind. 5mm LEDs don't quite fit unless you file the shoulders off to allow them to sit neatly side by side; also most 5mm LEDs have shoulders on the wires that prevent them being pushed down flush which makes them harder to line up neatly.

All LED's are orientated with the Cathode (Square PCB pad) facing toward the nearest PCB Edge.

I strongly suggest colour coding the LEDs by function – in my case I used Green for the address bus, Yellow for the data bus, Orange for the CPU control signals, and Red for power and Reset. Feel free to make your own choices here.

I have also left the value of the LED current limiting resistors open to user discretion; if in doubt use 330R which gives about 15mA per LED and is a good rule-of-thumb.

These resistors are shown with no value on the PCB.

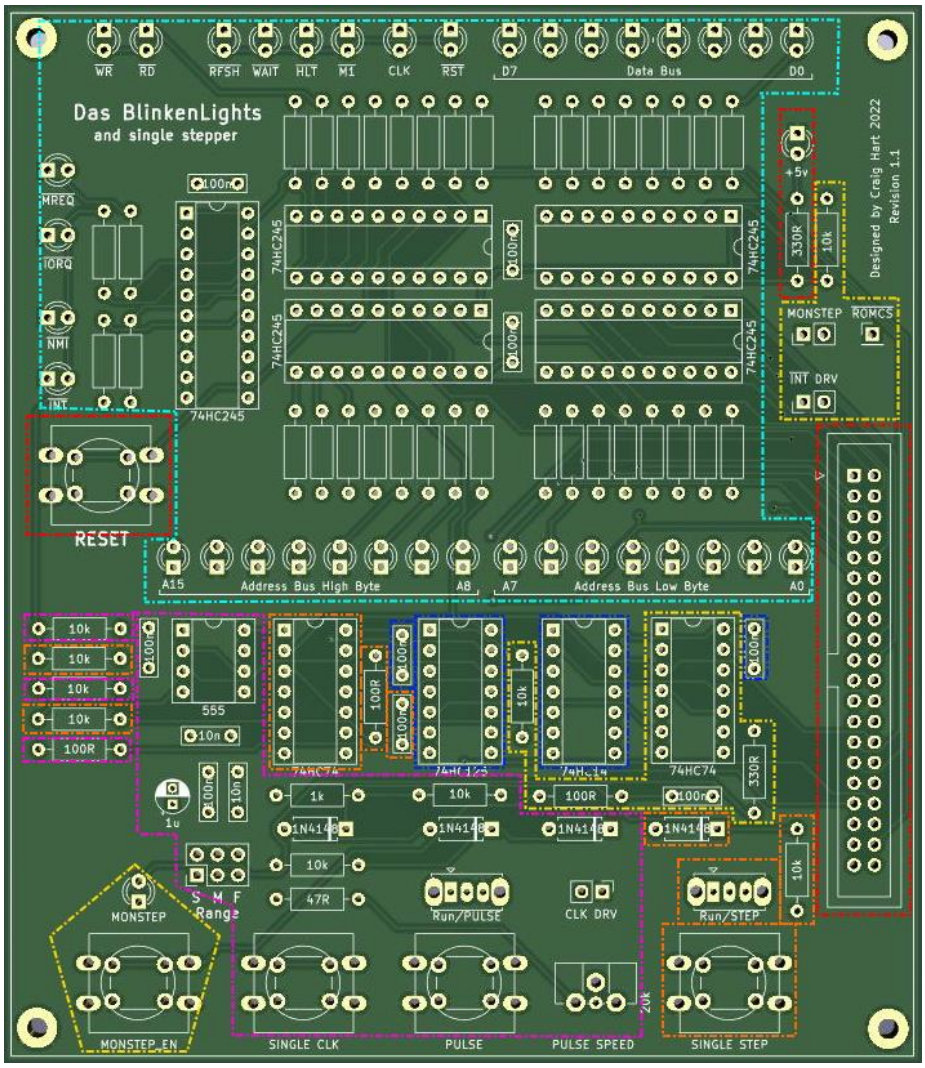
Some LEDs such as the White and Blue types may require different values, or you may wish to tweak for even intensity and overall current draw.

Shared Components

Blue Components

The 74HC125, 74HC14 and 2 x 100n filter caps are shared by the following three sections.

These parts are to be fitted whenever ANY of the following sections are built.



PCB layout showing the various component sections

MONSTEP

Yellow + Blue Components

To support MONSTEP, add the MONSTEP push button and LED., right hand 74HC74 and associated components.

Also install the MONSTEP and ROMCS headers.

Clock

Purple + Blue Components

To support Single Clock and Slow Clock, add the 555, SINGLE CLK and PULSE buttons, Run/PULSE switch, pulse speed trimpot and associated components.

Also install the Range and CLK DRV headers.

The 1u capacitor's NEGATVE side is shown in WHITE, positive side towards the Range header.

Independent Single Stepper

Orange + Blue Components

To support the ISS, add the left hand 74HC74, SINGLE STEP button, Run/STEP switch and associated components.

Parts List

Common Parts

Red Section

1 – 3mm LED
1 – 330R Resistor
1 – 40 pin IDC Connector
1 – Das BlinkenLights PCB
1 – 6 or 12mm SPST Push Button

Bus Status LEDs

Cyan Section
36 – 3mm LEDs
5 – 74HC245 IC
36 – 300R resistor
3 – 100n Mono Capacitor

Shared Components

Blue Section

1 – 74HC125
1 – 74HC14
2 – 100n Mono

MONSTEP Single Stepper

Yellow Section

1 – 3mm LED
1 – 74HC14 IC
1 – 100R Resistor
1 – 330R
2 – 10K
1 – 100n Mono Capacitor
1 – 1x1 Header Pin
2 – 1x2 Header 0.1" pitch
2 – Header Jumper caps
1 – 6 or 12mm SPST Push Button

Independent Single Stepper

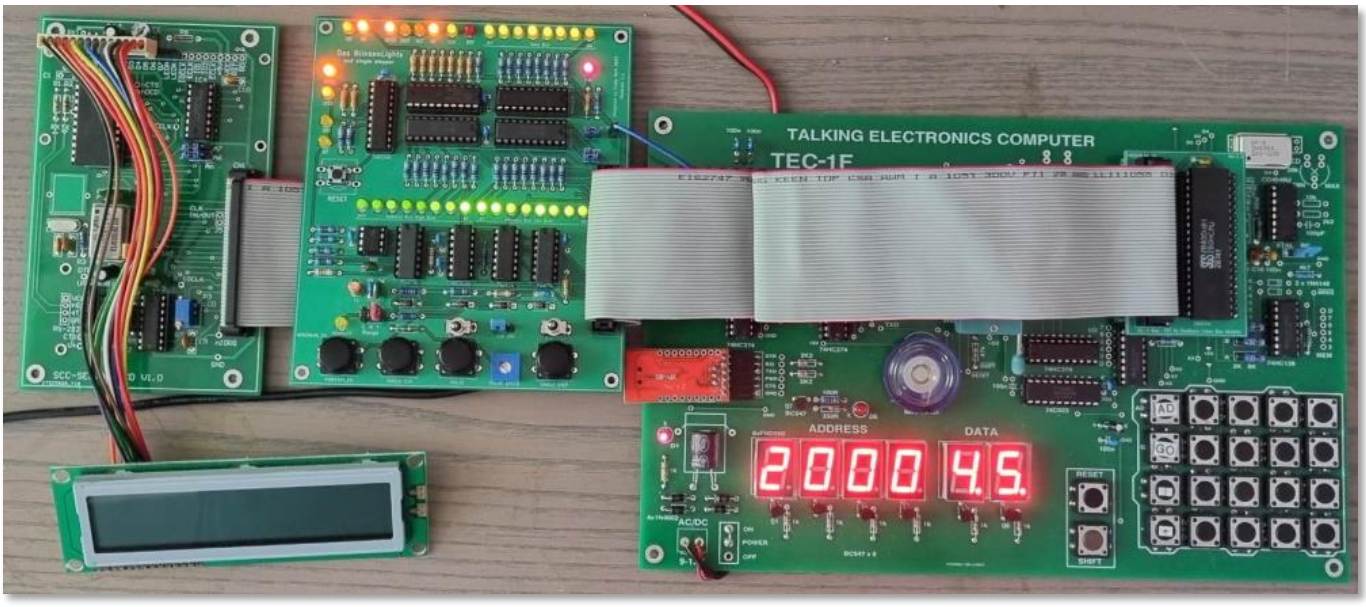
Orange Section

2 – 1N4148 Diode
1 – 74HC74 IC
1 – 100R
3 – 10k
1 – 100n Mono
1 – 6 or 12mm SPST Push Button
1 – SPDT Slide or Toggle switch

Slow & Single Clock

Purple Section

2 – 1N4148 Diode
1 – 555 Timer IC
1 – 74HC125
1 – 74HC14
1 – 47R Resistor
1 – 100R
1 – 1k
4 – 10k
1 – 20k trimpot
2 – 10n Ceramic Capacitor
4 – 100n Mono
1 – 1u 16v Electrolytic
1 – 2x3 Header 0.1" pitch
1 – 1x2 Header 0.1" pitch
2 – Header jumper caps
1 – 6 or 12mm SPST Push Button
1 – SPDT Slide or Toggle switch



Using the DBL Board

Firstly, set up the DBL board to its default configuration:

Set both switches to Run mode, ensure CLK DRV is not jumpered. INT DRV is jumpered, ROMCS wired to the TEC/SC's ROM Chip Select pin (if MONSTEP is to be used), Range Jumper and trimpot can be set to any setting.

Always return the DBL board to this configuration when starting afresh.

Experiment 1: Das BlinkenLights

Simply connect the DBL board in its default configuration (Use the TEC to SC1 adaptor from TEC Journal #2 if you're using a TEC). Power up your machine and look at the pretty lights!

Executing opcode 76h should cause the machine to stop and light the HLT LED. M1 and RFSH will also be lit (flashing, actually) indicating that the CPU is executing NOPs and continuing to perform memory refresh during this time.

Flipping the Run/STEP switch to STEP mode will show the address of the NEXT instruction on the Address bus LEDs

On the TEC, pressing a Key will light the NMI LED indicating a keypress. On the SC1 this does not work as the SC1 doesn't use interrupts for the keyboard.

Experiment 2: Slow Clock

Disable the onboard CLK source of the TEC (see How It Works section above) and enable the DBL Clock by installing the CLK DRV jumper.

Set the Run/PULSE switch to Run. Set the Range jumper to F and set the trimpot to maximum rotation.

Power up the computer and observe the boot process and LEDs as the machine starts up. You will need to wait a few seconds before the TEC starts to come to life.

The 7-seg displays will flicker as you adjust the trimpot, and you can slow the machine down to see the individual digits being scanned.

Experiment with the S, M and F ranges and see how the machine is affected. The Range jumper can be moved at any time and the machine will not crash.

Note: Starting the MONitor on the slowest speed takes quite a few minutes – be patient!!

Experiment 3: Single Clock

This experiment requires a CMOS Z80. Set the Run/PULSE switch to PULSE. Single-step the Z80 one T-state at a time by pressing the SINGLE CLK button.

Observe the instruction fetch cycle (M1 LED) refresh cycle (RFSH LED) and count button presses to see how many clock cycles an instruction takes to process. Try different instructions such as `NOP - 00h` – 4 clock cycles.

Compare the state of the various CPU bus lines with the M1 opcode fetch cycle.

Pressing and holding the SINGLE CLK button down allows you to observe the first half of a T state.

You can also press the PULSE button to feed slow CLKs to the CPU.

Note that if you Reset the CPU, the Z80 requires at least three clock cycles to fully reset, whilst RESET is active. So hold the rest button down and press the SINGLE CLK button at least 3 times.

Experiment 4: Independent Single Step (ISS)

Return the TEC to internal clock before proceeding: Remove the CLK DRV jumper from the DBL board.

At any time, flip the Run/STEP switch to STEP. Note the CPU stops (WAIT LED lit).

Press the SINGLE STEP key to single step the Z80; the Address LEDs indicate the memory address of the opcode being executed, and the Data LEDs indicate the first byte of the opcode.

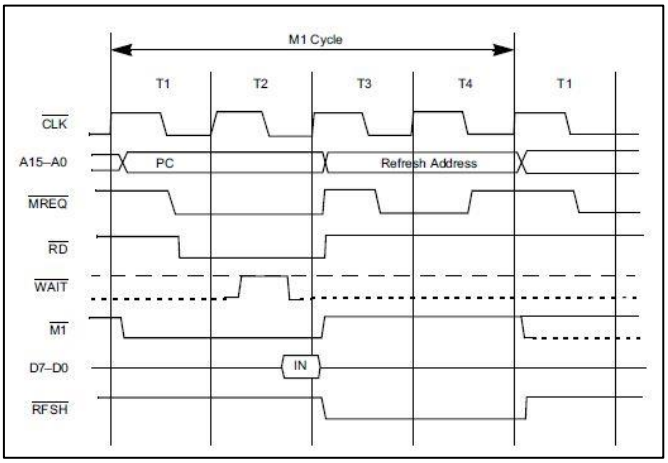
Return to Run mode, and enter the following code at 0900H or 2000H:

```
2000   00         NOP
2001   00         NOP
2002   00         NOP
2003   00         NOP
2004   00         NOP
2005   00         NOP
2006   00         NOP
2007   C3 00 20   JP 2000**
```

```
**Change the 20 to 09 if using
0900H in all examples
```

Start the program, then while running flip the Run/STEP switch to STEP mode.

Observe the Address on the Address LEDs, each time SINGLE



STEP is pressed, the address should increment, then return to the starting address as the JP instruction is executed..

Also, read the opcode on the data LEDs – all blank for NOP, and C3 for the first byte of the JP instruction. Note because the ISS works one INSTRUCTION at a time, you cannot step through the data bytes of the JP instruction itself.

Now, press the PULSE button (or switch run/Pulse to PULSE mode) to see the instructions executing at the speed set by the Range jumper and trimpot setting. Set the range to S and adjust the trimpot to a low position, and you should see the address bus incrementing one instruction at a time.

Experiment 5: Examine the HALT CPU state

HALT is a very special instruction. During HALT, the CPU executes NOPs and waits for an interrupt, in order to continue.

Enter and run the following code:

```
2000   76         HALT
2001   AA         XOR D
2002   C7         RST 00
```

Note that the opcode shown on the data bus LEDs is AA – this is the NEXT instruction to be executed AFTER HALT completes.

Note also that the address bus seems to show that A0-A6 are active. This is because the Z80 is doing refresh cycles across those addresses (recall the R register has valid values from 00-7Fh) and that the RFSH LED is lit.

Now Flip Run/STEP to STEP. Note that the Address bus now matches the next instruction to be executed (after HALT), but that RFSH is now not lit, since the CPU is both HALTed and WAITed!!

Set the Run/PUSE switch to PULSE. Not that this has no effect – the HALT condition wins over WAIT.

On the TEC only, press any key to generate an NMI Interrupt – the TEC will reboot as it executes the C7 instruction.

Experiment 6: Observe Memory and IO Access bus cycles

Enter and run the following code:

```
2000   DB 00      IN A,(00)
2002   C3 00 20   JP 2000
```

Observe IORQ and RD LEDs indicating the presence of IO Read cycles. Note that MREQ is also active as part of the instruction fetch.

Change DB to D3 and re-run.

```
2000   D3 00      OUT (00),A
```

Note that RD and WR LEDs are now both active as well as IORQ and MREQ, indicating Memory reads and IO writes taking place. You may use the slow clock to observe each instruction and verify this is what is taking place.

Experiment 7: Use MONSTEP Stepper

Recall that MONSTEP requires a stepper-capable MONitor and the ROM CS connection; see the notes in the How It Works section.

Enter the following code:

```
2000   FB         EI
2001   00         NOP
2002   00         NOP
2003   00         NOP
2004   00         NOP
2005   00         NOP
2006   00         NOP
2007   C3 01 20   JP 2001
```

Press the MONSTEP button to enable the stepper hardware THEN run the code.

The stepper will appear on the 7-seg displays and can be operated per the instructions relevant to that MONitor. Note with JMON/BMON the PC register is incorrect the first time the stepper is used.

Toggle MONSTEP off and on to see the code run and pause at whatever point it is at, in real time.

In JMON, replace EI with RST 28h

```
2000   EF         RST 28h
```

RESET the TEC and re-run. Note PC display is now correct on first step.

Experiment 8: Observe LDIR instruction

Enter and run the following code:

```
2000   21 00 21   LD HL,2100
2003   11 01 21   LD DE,2101
2006   01 FF 1D   LD BC,1DFF
2009   ED B0      LDIR
200B   C3 00 20   JP 2000
```

Set the DBL board for slow clock and observe the Address LEDs. Note how they increment (you'll need to ignore the repeated opcode fetches) as LDIR works its way through memory.

Also note there are TWO memory read cycles (1 x opcode fetch [M1 LED is active], 1 x LDIR) for every write. This means the LDIR opcode is re-fetched from memory on each loop.

Conclusion

The DBL board really is quite complex and feature rich. As such, it may take some time to come to terms with.

I hope you will find it more than just a "blinky lights" gimmick and use it both as a learning/demonstration tool, and as a debugging aid.

As always, you may find PCB Gerbers and other support articles via the authors GitHub webpage - https://github.com/1971Merlin

MONitor / Single Stepper reference table

MONitor	Single Stepper support	Default Interrupt Mode	Behaviour on INT	Enable Stepper
MON-1	No	Disabled	Executes `RST 00h` TEC Resets	-

MON-2	No	Disabled	Executes
			
			`LD HL,(08CCh)`
			`JP (HL)`
			
			Default value at `08CCh` is `0000h` so TEC Resets
				-

JMON and BMON	Yes	Disabled	Executes `JP xxxxh` `xxxx` is the start address of the stepper code in ROM, so starts stepper	• Execute `RST 28h` Enables interrupts and preloads the correct PC register value • Execute `RST 38h` (Breakpoint) Enters stepper immediately • Press Shift-2 Enables stepper, runs code from current MONitor address

SCMON	Yes	Enabled	Executes:
			
			`push hl`
			`LD HL,(RST38)`
			`JP (HL)`
			
			`RST38` is the start address of the stepper code in ROM, so starts stepper
				• Stepper is active as soon as MONSTEP is turned on
				
				• Execute `RST 38h` (Breakpoint)
				Enters stepper immediately
				
				** Fn 7 not used by DBL board

MONSTEP Stepper control functions reference

JMON and BMON	+, -	Scroll through CPU registers on 7-seg display
	GO	Step one instruction
	AD	Return to MONitor
	Any other key	Auto-step at about 2 instructions per second; any key cancels auto-step.

SCMON	+, -	Scroll through CPU registers on 7-seg display
	AD	Step one instruction
	Fn	Return to MONitor

Notes regarding stepper Program Counter (PC register) display

The PC register displayed on the stepper by SCMON is always the address of the *NEXT* instruction to be executed, not the address of the instruction just performed. This can be misleading.

JMON/BMON attempts to display an accurate PC by storing the value of PC each time it is launched; the stepper displays this "previously saved" value for PC.

This means an invalid value could be displayed the very first time the stepper runs, and also a stale value can be given if MONSTEP is turned off and then on again or `RST 38h` is called directly. To partially avoid this, call `RST 28h` to enable interrupts and preload PC. Note that the stepper does not work until Interrupts are enabled; however RST 28h is preferred over EI so that PC is displayed correctly.