DC-2000 (DC-2020) Reverse engineering notes

Doctor-Z ([z@dc-zone.com](mailto:z@dc-zone.com))

Note that all information in this document is gathered from reverse engineering in the aim of building a replacement CPU board to keep these beautiful consoles alive. I hope this information might be useful to anyone for future reference. Information in this document is not guaranteed to be correct and is continuously changing.

## 1. Goal:

Create an embedded computer system that is able to replace the original computer system. The hardware should be modular and created from widely available / generic & modern parts.

Current development set-up:

* MyStorm BlackIce MX – Ice40 FPGA used for (de)scrambling all the serial data
* PIC(16F877A) for Synchronious to Asynchronious serial translation (USART to UART)
* Raspberry Pi 4 model B – Running the new software (yet to be developed)
* Touchscreen (yet to be determined)

Toolchain:

* FPGAWars IceStudio (<https://www.icestudio.io>)

Project milestones:

* Create FPGA based implementation of the descrambler circuit.
* Capture the “magic” seed that feeds into the descrambled from the microprocessor (OSCRAM netlabel).
* Interface with USART from code
* Be able to read the HDLC packets
* Brute-force / capture the PALCE ic’s.
* Create FPGA based implementation of the (clock) scrambler circuits.
* Be able to write HDLC frames to the data bus
* Capture and analyze boot / initialization HDLC sequence between the original computer and the C3 chips. HDLC packet analysis.
* Recreate the initialization sequence in software and run it from the Raspberry.
* Reverse engineer the “received” HDLC packets from the cadence loop, parse data from the faders, knobs etc..
* Write software implementation of the new “OS”.

## 2. Hardware interfaces to implement on the new board:

* Cadence loop
* Meter Bridge
* MIDI / SMTPE

## 3. DC2000 Main Processor PCB – CN6 Power IN – Pinout

1. +5VDC
2. GND
3. +12VOK
4. -12VOK
5. 0VOK (Digital ground?)
6. +17VDC
7. -17VDC
8. AGND
9. VUNREG?
10. +5VDC (Display Processor?)
11. +48 VDC (Going to inverter for display EL-Foil)
12. AGND

## 4. Data signals

All data is running over scrambled RS-422 serial lines. It’s a clocked protocol with a separate clock wire (probably needed for the (de)scrambler to operate). Underlying the scrambled bits there is data protocol named HDLC (which is natively supported by the 68302). Note that the CRC of these packets may be different from the ones used in Ethernet/Xmodem/Ymodem and compares to a “magic” number. The Z8350 (which was used during development) documentation should cover these topics.

Todo:

* Find baud-rate (250K <-> 1MHz range?)
* Find out which CRC algorithm was used, otherwise try to detect it using: <https://reveng.sourceforge.io/>, we can capture enough CRC data from the original bus and feed it to reveng, so we should be able to calculate it.. otherwise brute force it.
* Find a way to get a transmit clock signal from a UART that triggers on each transmitted bit. Try to accomplish this with the built in UART from the Raspberry, Otherwise use the Z83C50 (successor of the Z83C50) which was used during development? Or find another microcontroller that has a UART with a transmit clock pin option. Hmm maybe we need a USART for this scenario? (never worked with USARTS before, so I need to investigate)

**Update:**

I’m starting to believe we need USART. But the Raspberry PI doesn’t have USART (only UART) this the serial module cannot handle the clock line / synchronous serial connections. In that case we could use a small micro controller (AVR Mega32?) that has USART which can act as a data register between the (de)scrambler (FPGA) and the Raspberry.

Maybe we can use a PIC:

<https://www.teachmemicro.com/serial-usart-pic16f877a/>

## 5. Data signals

The way each cadence sets up its address. Is that it sends out a broadcast “set address” packet which then as each cadence chip sets its address then sends another packet to next chip in line with the address incremented by one.

## 6. (De)Scrambler Circuits

The Scrambled and Descramber circuit are there to prevent harmonic interference in the audio domain. It’s scrambles a stream of bits (TTL) using a pseudo random sequence (OSCRAM input) that is known to both the scramber and descrambler.

The (de)scrambler circuit is documented here:

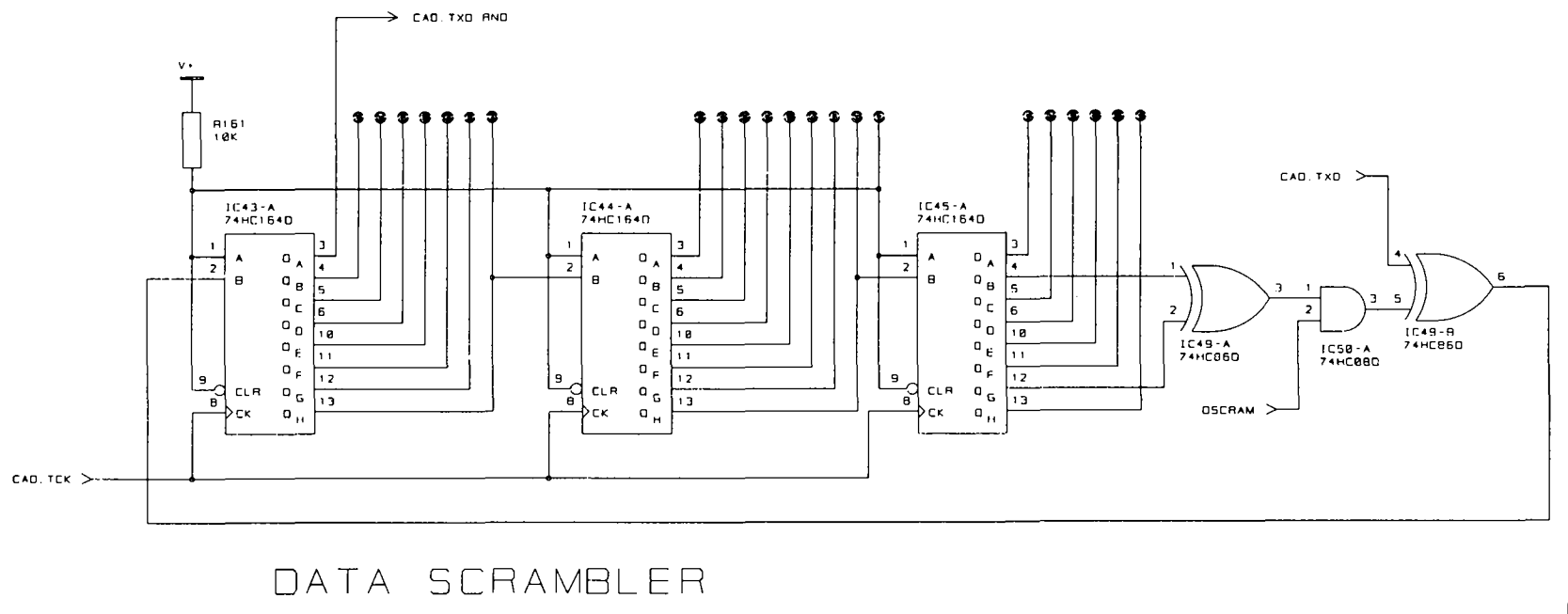
<https://patents.google.com/patent/US5530390>

https://patentimages.storage.googleapis.com/78/bd/7b/c68690c4fb010d/US4304962.pdf

From DC-2020 technical manual page 251

### 6.1 Scrambler

|  |  |  |
| --- | --- | --- |
| **Netlabel** | **Purpose** | **I/O** |
| V+ | Power 5VDC | Input |
| CAD.TCK | Transmit clock? | Input |
| CAD.TXD | Cadence transmit (signal to be scrambled) | Input |
| OSCRAM | Pseudo random input bit/TTL, feed by slave processor IC2 output pin 58 | Input |
| CAD.TXD RND | Cadence transmit Random? (scrambled signal) | Output |
|  |  |  |

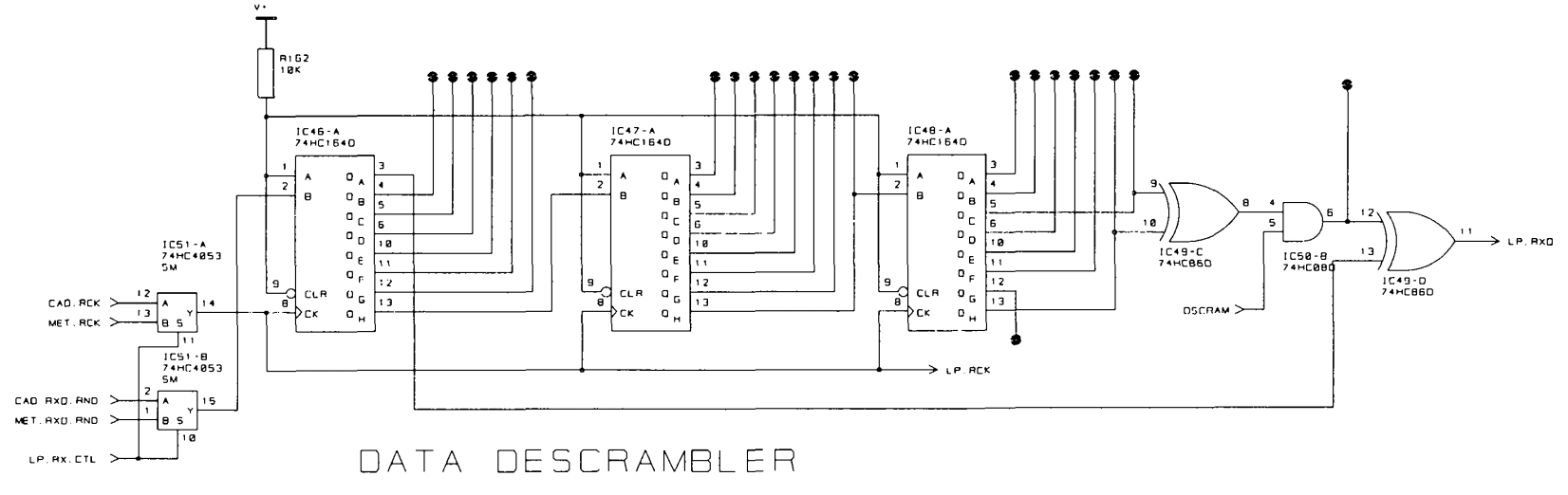


From DC-2020 technical manual page 251

### 6.2 Descrambler

|  |  |  |
| --- | --- | --- |
| **Netlabel** | **Purpose** | **I/O** |
| V+ | Power 5VDC | Input |
| CAD.RCK | Cadence receive clock? | Input |
| MET.RCK | Meter bridge receive clock? | Input |
| CAD.RXD RND | Cadence receive (scrambled signal to be descrambled) | Input |
| MET.RXD RND | Meter bridge receive (scrambled signal to be descrambled) | Input |
| LP.RX CTL | Unknown – switch between cadence / meter bridge as source? | Input |
| LP.RCK | ? Clock | Output |
| OSCRAM | Pseudo random input bit/TTL, feed by slave processor IC2 output pin 58 | Input |
| LP.RXD | Descrambled signal bit/TTL? | Output |

– What is the meaning of LP in this context? “Loop”?

  
It is unclear how the original computer, utilizes the descrambler circuit in terms of sharing it between the cadence loop and the meter bridge. As we can see in the schematic above there is a digital switching section in front of the input, which allows the microcontroller so assign the descrambled to one of the 2 sections. But the descrambler holds a 24-bit buffer, so I don’t quite understand how this switching takes place.. without mangling the data of the buffer.

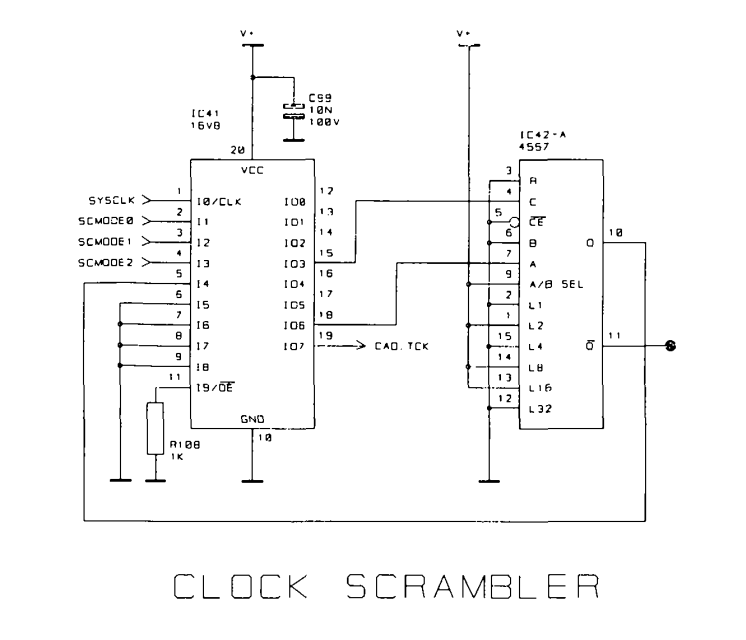
## 6.3 Clock scrambler

IC41 = RX1075 (PAL PALCE16V8Q/4) – Dump available in dc-zone.com download section.

IC42 = HEF 4557BT (1-to-64 bit variable length shift register)

Note that the dumps we have are probably faulty. These PAL ic’s have a protection fuse that makes them unreadable. As we can see in the schematic the input is only 4-bit parallel (SCMODE 0 > 3 and the returning value from the shift register) and only 3 outputs are connected. F.a.i.k. this IC is just a simple configurable logic gate array, so there is no state. Therefore, in respect to the 4-bit wide in input there can be only 16 logical input combinations. It should be easy to brute force by just feeding in the 16 different combinations and capturing the outputs. That will result in a thruth-table which can be converted into code or FPGA logic.

|  |  |  |
| --- | --- | --- |
| **Netlabel** | **Purpose** | **I/O** |
| V+ | Power 5VDC | Input |
| SYSCLK | System clock? | Input |
| SCMODE0 | ? | Input |
| SCMODE1 | ? | Input |
| SCMODE2 | ? | Input |
| CAD.TCK | Cadence transmit clock? | Output |
|  |  |  |



* Output 3 of PALCE feed into pin4 (Clock Input) of shift register.
* Output 6 of PALCE feed into pin7 (Data Input) of shift register.
* pin10 (buffered output) of feeds back into PALCE IC (pin 5)

ToDo:

* Find out what the exact meaning of the pins and the purpose of the “modes”..

# Scrambler power section: