Inside the Timex SCLD

A bit of history on the reverse engineering
of the Timex SCLD integrated circuit

Table of Contents

Introduction

This document briefly documents the reasoning, rationale and effort done to successfully understand how the Timex SCLD (chip name “TS 2068 PAL”) works internally.

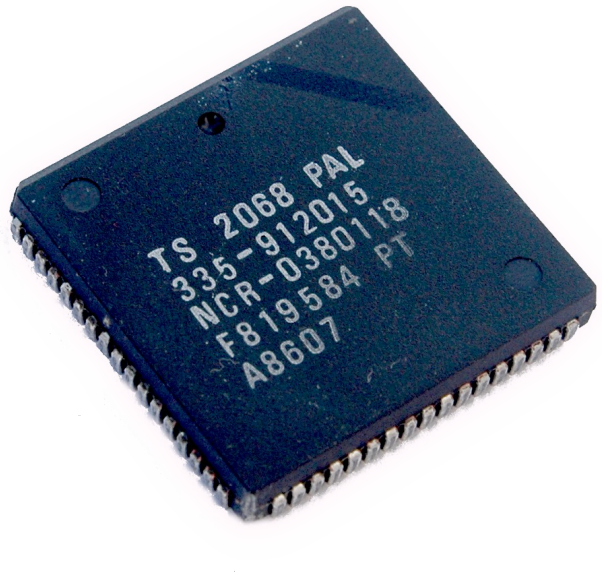
The intended audience is everyone who already has knowledge of the existence of the SCLD and basic notions of what the SCLD does in the Timex computers.

This document is based on a previous presentation I gave personally.

This document may include inaccuracies, simplifications and otherwise information that might not be entirely correct. Some of those are intentional and were chosen so to simplify the view for those who have less technical knowledge of the matters involved, some other are my own mistakes for which I apologise in advance.

What is the SCLD

The SCLD is an integrated circuit present in some of the Timex compuer models, such as the portuguese TC2048 and TC2068.



On the computer, the SCLD:

- Provides video signal (acts as the video card) and interface to the video memory
- Handles audio processing and keyboard
- Controls selection of ROMs and cartridges
- Generates CPU clock
- Generates “missing” signals for DRAM
- Adds IO ports to access AY-3-8910 external audio synthesizer IC.
- Handles both PAL and NTSC (via configuration pin and crystal selection -14MHz or 14.122MHz)
- Generates more video modes than ZX Spectrum ULA
- Although called TS2068, it is unrelated to the Timex model with same name.

Why we did this - the Problem and the Solution

Well, to simplify, the problem was:

- The SCLD is no longer manufactured, and there are no internal design documents available.
- Replacing an SCLD on a Timex computer requires cannibalizing another Timex.
- It’s the only component missing for a successful implementation of a Timex clone.

So, what is the solution?

Well, the solution we decided upon can be summarized like this:

- Reverse engineer the SCLD from a functional perspective
- Use a COTS CPLD (Complex Programmable Logic Device) with eventual additional electronics to serve as a drop-in replacement
- Achievable with a printed circuit board with similar footprint
- A joint project with LOAD ZX Spectrum museum in Cantanhede, Portugal

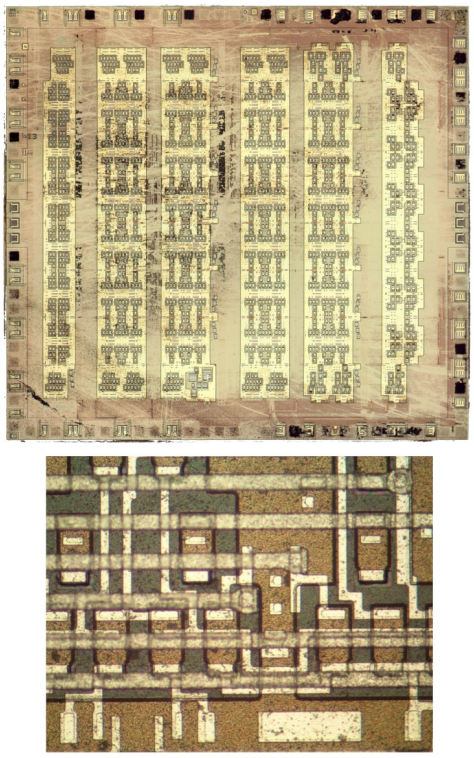
Reverse-engineering Integrated Circuits

There are several ways to reverse engineer ICs.

The most common are:

- Examination at transistor/wafer level
  - Requires etching the IC die with chemicals, one layer at a time, to expose metal layers first and then the substrate.
  - Requires complex imagery analysis.
  - Some cells might be non-standard, increasing complexity.
- Examination at functional level
  - Use a “testbench” to inject signals and observe outputs.
  - Analyse data to understand behavior, and then infer the internal design.

We decided to perform the examination at the funcional level. For We had no access to equipment for transistor level examination, and we did not have the knowledge to perform that examination efficiently.



A ULA-Like exposed die after removing the metal layers and a close-up (with metal layers) of some transistors.

Functional reverse engineering

Two main types of funcional reverse engineering can be employed:

- Fully electronic
  - Use signal generators (analogue and/or digital) to inject data on the device.
  - Use digital analysers, data loggers and oscilloscopes to observe and record device outputs.
- Mixed software and hardware
  - Microcontroller controls most inputs of the IC, and samples the IC outputs. Most suitable for digital devices, but analogue is also possible with some limitations.
  - Minimal hardware required to interface the IC to a microcontroller.

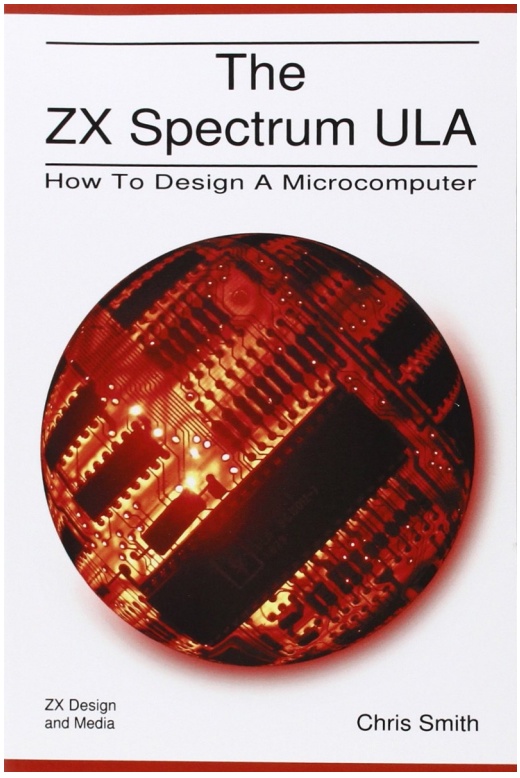
For simplicity, we decided to use a mixed software and hardware approach. This meaning that we would design a basic hardware to interface to the SCLD, and then use a microcontroller (on our case, an ATMEGA2560) to perform the analysis, all connected to a standard PC linked via serial-over-USB.

SCLD – where to start?

So, what do we know about the SCLD that allows us to start?

Well,

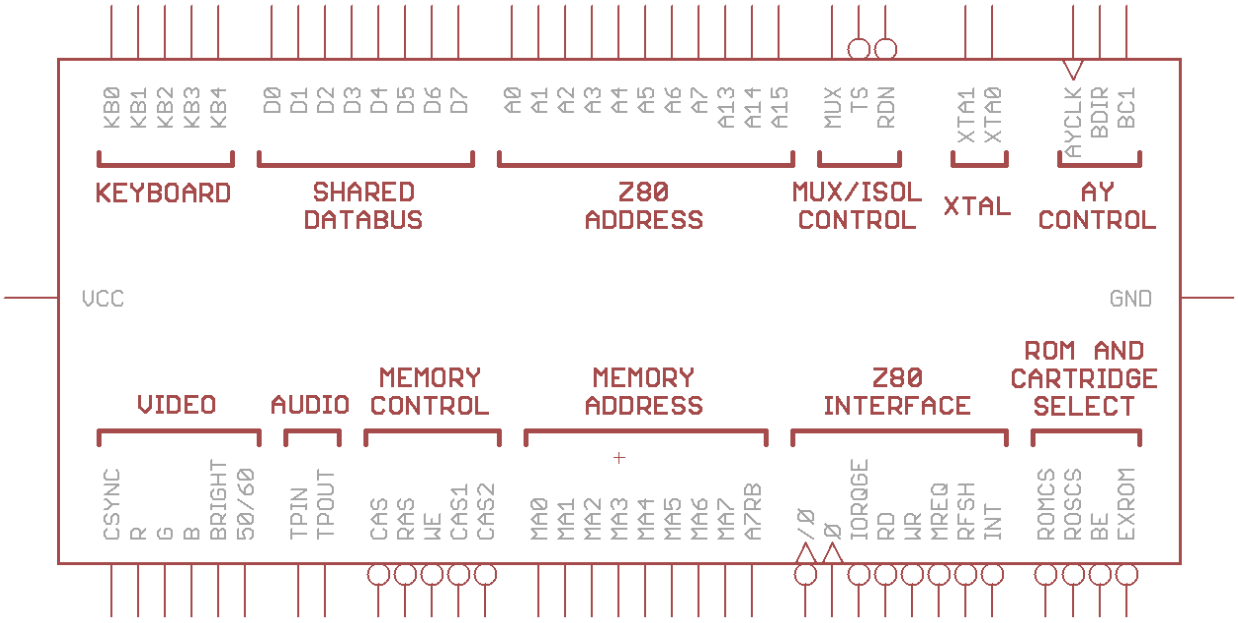
- SCLD is based (roughly) on ZX Spectrum ULA, which has been reverse engineered.
- The SCLD functionality is documented to some extent (games use the functionality)
- Mostly synchronous device, based on logic gates and storage elements (flip-flops and latches)
- Typically runs off 14MHz external crystal (PAL systems) or 14.112MHz (NTSC systems)



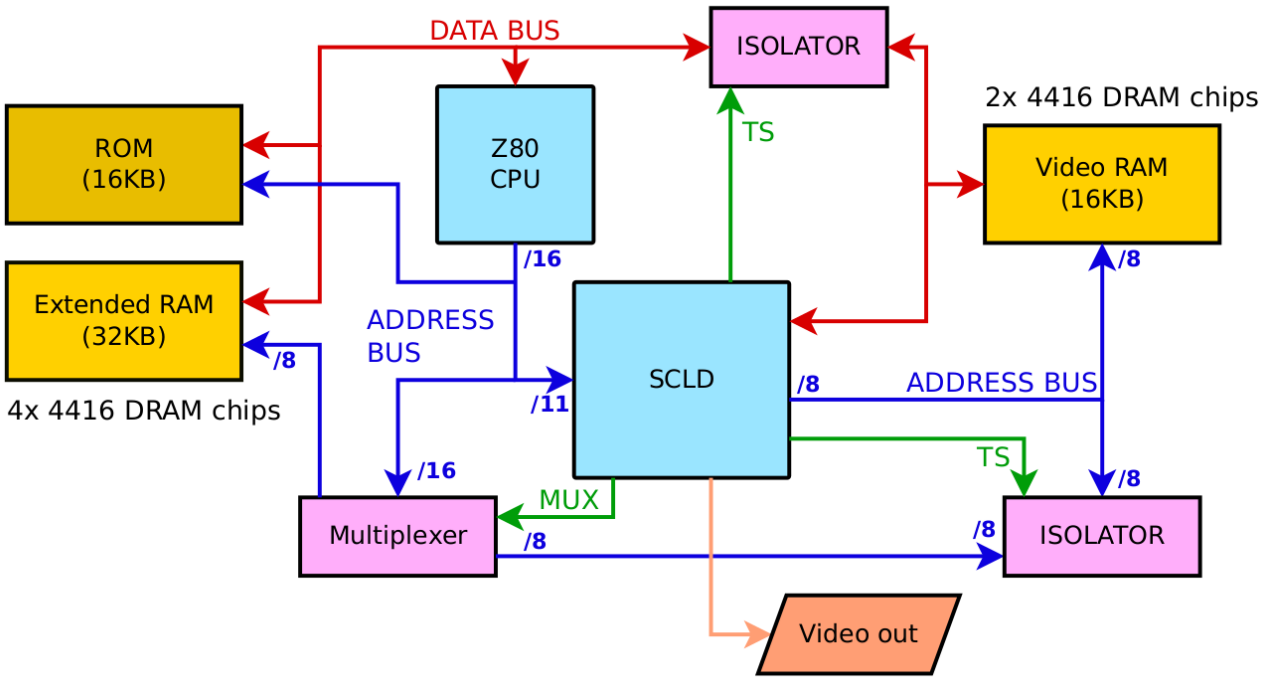
The ULA book from Chris Smith.

The SCLD interface and the computer

From schematics all across the web, we devised the SCLD interface. Without prior knowledge of the interface (or at least some of its signals) it would be near impossible to figure out what they do (like, is it an input, an output, how can we exercise those interfaces?)



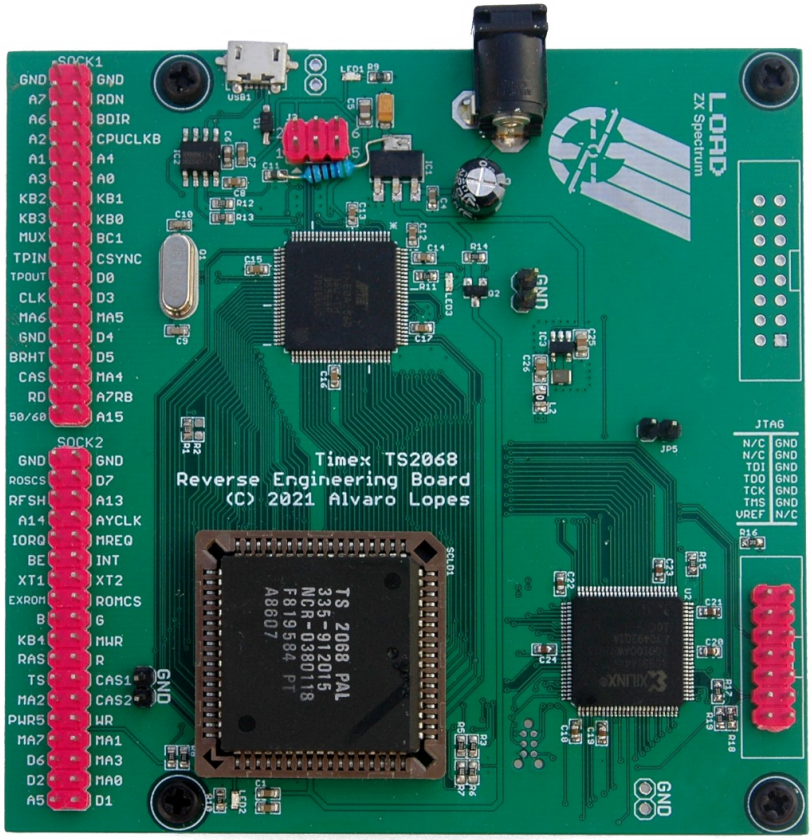
Searching online (around the Web, thanks João Encarnado!) we also learned how one of the computer that uses the SCLD is laid out internally. A block diagram of the Portuguese TC2048 is as follows:



The approach

So, after analysing the SCLD interfaces and the TC2048 schematic, we decided on the following approach:

- Software-based test bench with custom board based on ATMEGA2560
- Run stimulus, capture output for analysis
- Design replacement using CPLD
- Run tests using the same framework targeting CPLD
- Board with SCLD socket – can target the SCLD itself or the CPLD (mutually exclusive)
- Header with all SCLD signals for oscilloscope inspection
- Cost-effective solution – less expensive than official Arduino Mega.



A picture of the “reverse engineering” board, with the SCLD attached on a PLCC socket. In addition to the ATMEGA2560, a CPLD intended to hold the redesigned CPLD has also been added (but never actually used).

What can that system do?

The system, as simple as it might look, and with software even simpler than the hardware, can:

- Modulate SCLD clock (at a very low speed – not 14Mhz!)
- Simulate the Z80, RAM, keyboard and board discrete components (muxers, etc)
- Capture data from all digital SCLD pins .
- Pseudo event-triggered flow.
- Wait for signals to change – edge, level.
- Execute test cases in C++.
- Outputs industry standard VCD file (ASCII)

The software can be resumed like this:

- Simple approach, reads and writes individual SCLD signals.
- Has bus models to simulate nominal devices.
- Can emulate a Z80 CPU.
- Emulates RAM.
- Emulates Keyboard.
- No “video” generation at this point (for image inspection).
- Pseudo event-triggered flow
- Wait for signals to change – edge, level.
- Test cases can be linked in and executed after uploading to the board.
- The execution can be saved locally on a VCD file and can even be used to be “replayed” on a VHDL simulation on the replacement SCLD.

Test cases

Well, not necessarly test cases - the system was designed to allow us to inject data and then we can observe the outputs on a waveform viewer, although it is possible to perform “checks” in run time while the test runs.

But before we start designing a new test case, we need to answer a few questions:

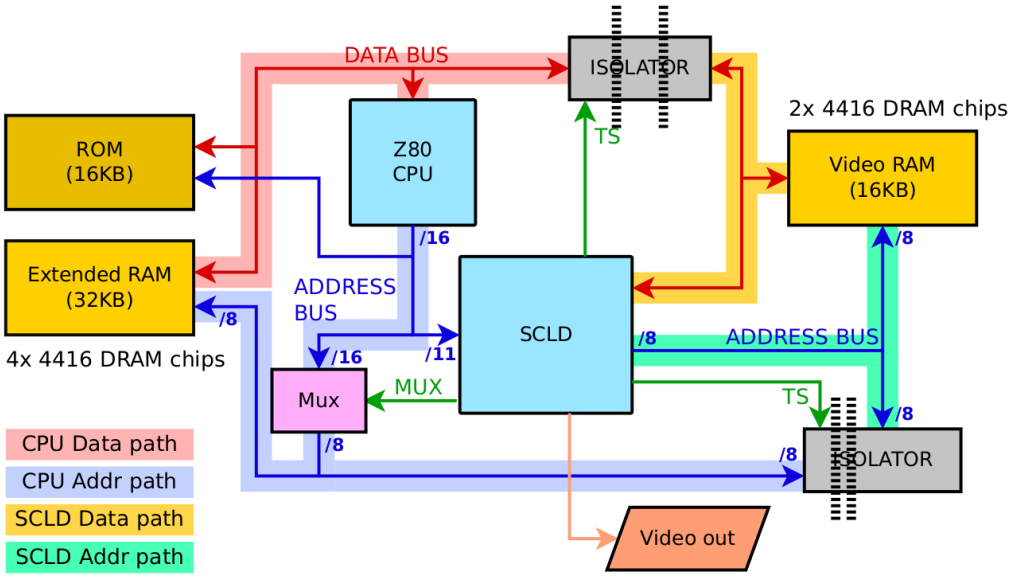
- What do you want to inspect or observe?
- What stimulus you need to provide to the SCLD
- How are those stimulus synchronized to the SCLD behavior
- How to implement the test case with the framework

Use case - Design a test case to understand when CPU contention occurs

One of the hardest parts of understanding the behaviour of the SCLD is to understand how contention is implemented. Contention happens when both CPU and SCLD need to access the Lower RAM, and when this happens the CPU needs to be “stopped” and only resumed after the SCLD is finished.

Simultaneous access to different resources

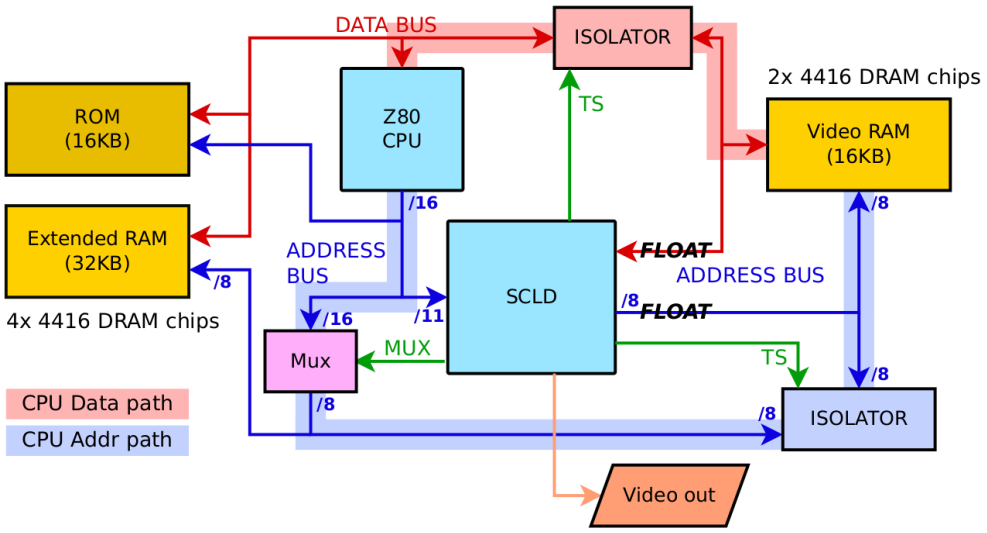
When the CPU is accessing the extended RAM and the SCLD is accessing the Video (Lower) RAM, the isolators prevent mixing data and address [not all models have a physical isolator, sometimes only resistors].



The isolation is controlled by the TS line.

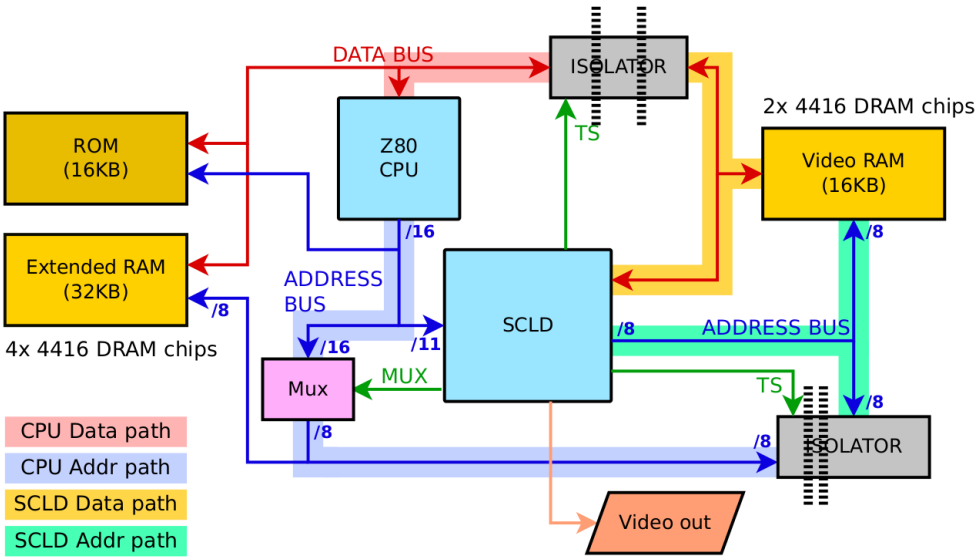
Exclusive access to shared resource

When the CPU accessing the Video (Lower) RAM and the SCLD is not generating video, it floats the data and address bus to allow CPU access. The isolators move data between CPU and Video RAM.



Simultaneous access to shared resource

When the CPU tries to access Video RAM and the SCLD is accessing Video RAM, the CPU needs to be “stopped” to prevent video being disturbed. This means that Isolators need to be active. Changes in video timing would cause artifacts to show on screen, such as horizontally displaced pixels and even invalid pixel data. CPU access can only resume/be processed after the video access from the SCLD.



Examining contention

We want to observe when contention occurs. Suspicion is that if access is performed near the actual instant the SCLD needs to access video memory, then the CPU is stopped.

For this, we need to trigger Z80 memory accesses at different times, and observe how the relative time between these accesses and the CPLD internal “workload” affects the contention.

We know (from the schematics) that TS line needs to be high for the SCLD to access the video RAM, so we schedule our reads in relation to the TS line.

Let’s delay from TS line going up a variable number of cycles and perform reads, then observe the output.

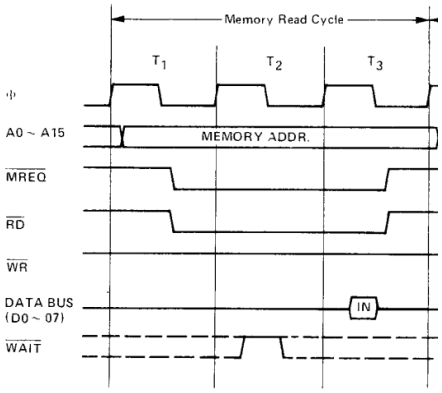
Implementing the test case

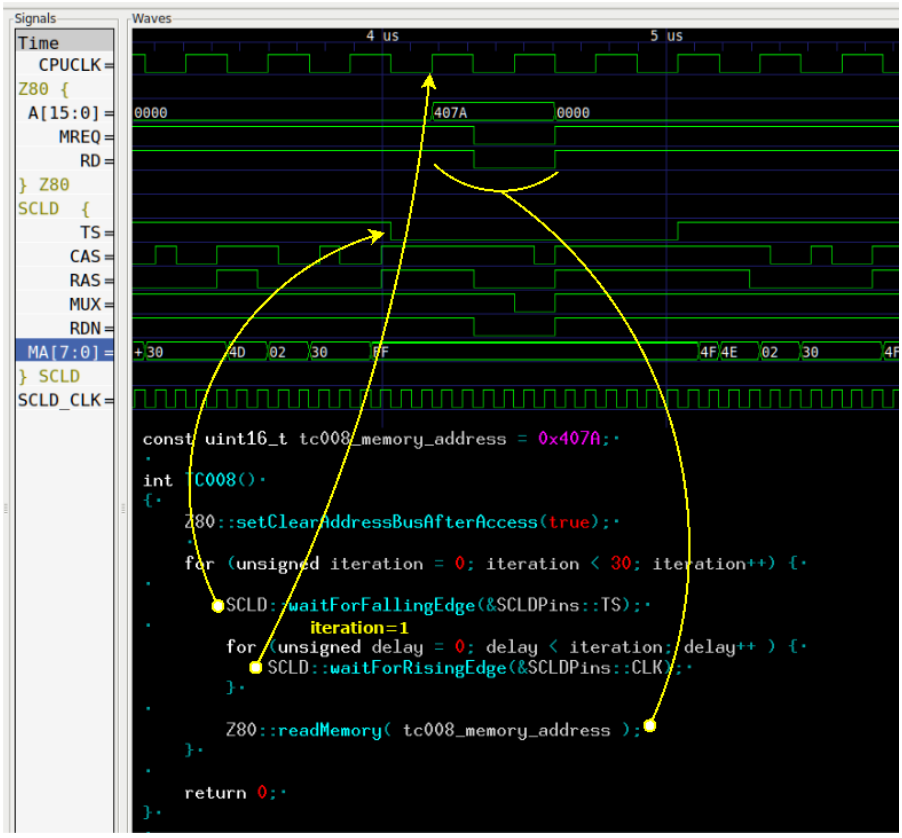
- Simple test with 30 iterations
- Clear address when no bus access is performed to be better visualised in VCD.
- Syncronise to TS line.
- Wait for a variable number of clock cycles.
- Perform a memory read to a fixed video RAM address (0x407A was arbitrarily chosen)

```
#include "testcase.h"
#include "pinout.h"
#include "Z80.h"

const uint16_t tc008_memory_address = 0x407A;

int TC008()
{
    Z80::setClearAddressBusAfterAccess(true);

    for (unsigned iteration = 0; iteration < 30; iteration++) {

        SCLD::waitForFallingEdge(&SCLDPins::TS);

        for (unsigned delay = 0; delay < iteration; delay++ ) {
            SCLD::waitForRisingEdge(&SCLDPins::CLK);
        }

        Z80::readMemory( tc008_memory_address );
    }

    return 0;
}
```

Initial run 1

- Wait TS for falling edge
- On iteration one, wait for one CLK (CPUCLK) rising edge
- Perform a typical Z80 Memory Read.

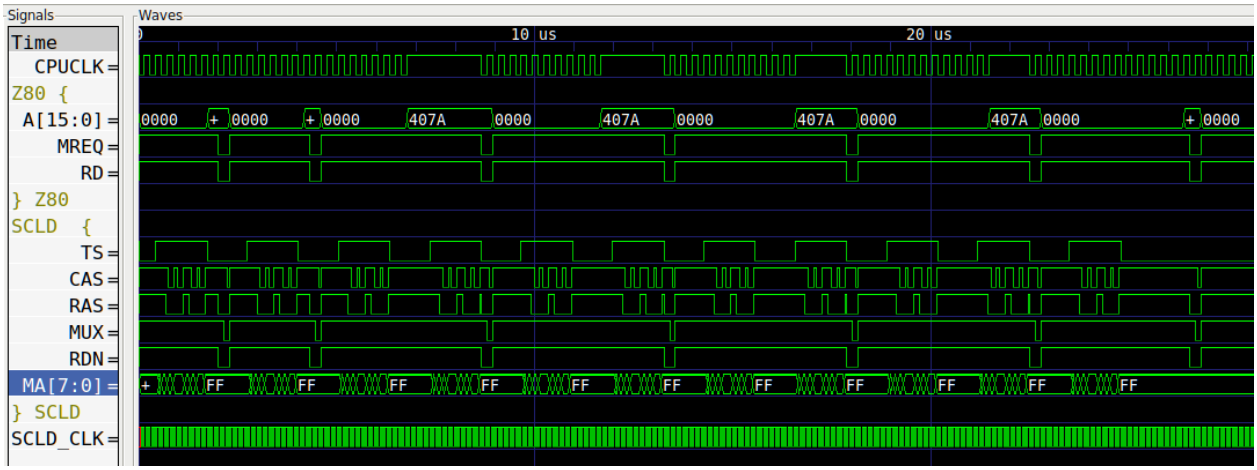




Zooming out

Contention clearly visible on CPU clock.

(Who’d imagine ?)



Result analysis

So, after we run the test, we can reach a few conclusions about the contention:

- Contention is achieved by holding the CLK line at high level
- As per (1), the contention starts before the TS line is activated
- This is easily explained: the CPU read/write cycle needs to complete before TS line goes up, so contention starts early
- As per (2), the MREQ/RD lines are not used for contention purposes, only the address
- This means that addresses (such as IO) can contend the CPU, even if memory is not to be addressed.