1 Introduction

1.1 The computer being simulated

The simulator models a small mainframe computer called "Sirius".

Sirius was designed by Ferranti in the late 1950s, a time when computing was still in its infancy. It was marketed as: "a truly general purpose machine suitable for a wide variety of uses in industry, commerce, science and technical education"[1]. Computers of the time were challenging to use, especially for anyone unfamiliar with their inner workings. Sirius took a few small steps in the right direction: It had decimal displays, it could be slowed down for demonstrations and educational purposes, and it could be programmed in "autocode", a (slightly) higher-level language than the more usual machine code.

1.2 How the simulator came to be written

In 2002, an original copy of the "Sirius II Logical Diagram" came to light. The simulator was developed to see if it was feasible to build a working software model of the logic. It was an interesting exercise to fill the dark winter evenings and, as it turned out, it *was* feasible: It worked!

The simulator models the level of design expressed on that diagram - a network of logic gates running continuously. Signals propagate through the modelled network and words cycle around modelled delay lines just as they did in the real hardware.

Rather unusually for a simulator, and reflecting its circuit-level origins, it has a software oscilloscope built in. This was a useful tool during development of the model (no wiring list had been found at the time and there were some obvious errors on the schematic diagram which needed to be corrected). The oscilloscope can still be used to probe the logic.

One thing led to another: Having got the schematic design to work, a user interface was added to see what Sirius looked like in action.

1.3 Simulation performance

Simulating the design at gate level, and running continuously, means that the simulator runs much more slowly than the real thing (of the order of 20 to 40 times more slowly depending on the host PC). Although it now emulates the appearance and controls of Sirius, there is still that detailed gate-level simulation running below the surface, and that takes time.

1.4 Installing and uninstalling the software

The software requires a PC running Microsoft Windows. The simulator makes heavy demands on PC resources so the higher specification machine you have, the better.

It is stand-alone software so does not truly 'install'. (It doesn't update the Windows Registry nor does it appear in the Windows Start Menu.) To use it, download the software and associated files into a single folder of your choosing. Run the programs by double-clicking on the executable files: 'SiriusSimulator.exe' and 'Teleprinter.exe'.

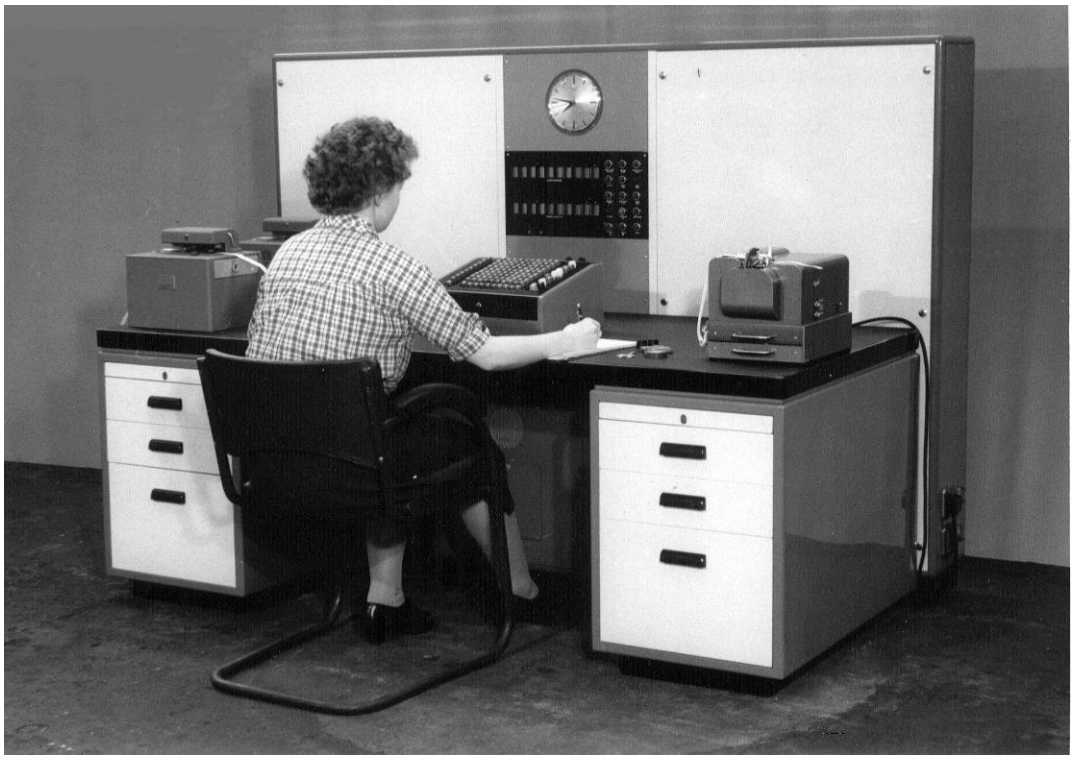
The software was developed more than ten years ago. It has been recompiled recently using the same (old) compiler and still seems to run properly on Windows XP, 7 and 10.

When you wish to uninstall the software, simply delete the folder containing it.

2 A Ferranti Sirius computer installation

2.1 What it looked like

Sirius was Ferranti's first computer to use transistors rather than thermionic valves (vacuum tubes). This allowed its use in normal office environments without special power supplies or air-conditioning. It was compact enough to stand behind an office desk with its control unit and peripherals placed on the desk or nearby.



In this photograph the clock and display panel can be seen set into the mainframe cabinet.

Two Ferranti paper tape readers are positioned on the left of the desk with a paper tape punch on the right. Input and output was solely by punched paper tape. (There was provision in the hardware for interfacing to other devices and some were added later but we won't go into them here.)

The computer operator is sitting in front of the electro-mechanical control unit.

The power supply cabinet, attached to the mainframe, can be seen under the desk protruding into the rear of the footwell.

The two 10-digit decimal displays on the display panel, with accompanying buttons and lights, were used for monitoring the progress of programs and to help when debugging programs and hardware faults.

2.2 Use of paper tapes

What you *don't* see in the above photograph is a teleprinter which was necessary for preparing and editing input paper tapes and for printing output tapes. The name, teleprinter, might suggest an electrical connection existed to the computer but this was not the case. (It would have been far too slow.) Paper tape was the only input/output medium. Program preparation, and the printing of results, always occurred offline.

The photograph below shows what Ferranti called its "simplified tape editing equipment": A paper tape reader is to the left of the teleprinter with a paper tape punch ("reperforator attachment") fixed to its right-hand side.

Programmers prepared programs and data by writing them on paper. Then they sat at a teleprinter desk like this one to type them in, thus getting a printed hard copy and a simultaneously-punched paper tape for input to the Sirius computer. The printed version would be checked for typing errors.

The paper tape could be edited if necessary by placing it in the tape reader of the editing equipment and stepping it through, character by character, copying or deleting characters or typing new inserts, to get a corrected version punched out for sending to the computer. (Alternatively, small tape edits might be done using a hand punch, scissors and masking tape.) Later modification of programs would be done in the same way.

The results of computation, punched on paper tape by the computer, would be read and printed by this same equipment.

2.3 Extra memory

Also nearby may have been extra store cabinets. The basic machine had 1000 words of store within the mainframe. Up to three 3000-word store units could be added, each one housed in a tall 2-door metal cabinet the size of a wardrobe.

2.4 The chosen configuration to be simulated

The simulator models a particular Sirius installation with two paper tape readers, one paper tape punch and one extra store cabinet. This configuration was a popular choice for customers in the early 1960s. It had 4000 words of store, the minimum required to run autocode programs.

3 Modelling the system: An overview

To model the system in the absence of paper-tape handling equipment and a teleprinter some changes are necessary:

- Two programs are used: the simulator itself, and a substitute for the teleprinter.
- Intermediate files are used instead of paper tapes.

In the 1960s	Now
The programmer writes the program on paper.	The programmer types the program into a text file. (e.g. 'my program.**txt**).
The programmer types the program on a teleprinter to get a printed copy and a simultaneously-punched paper tape.	The "teleprinter" program converts the ASCII text into the Sirius character set and writes 5-hole punching codes into an intermediate file. (e.g. 'my program.**ptr**')
The punched paper tape is taken from the teleprinter to the Sirius computer and placed in a tape reader.	The intermediate file from the "teleprinter" program is opened in the simulator.
The computer (with Initial Orders or the Autocode Compiler already loaded) reads the paper tape and runs the program. The results are punched out on paper tape.	The simulator (with Initial Orders or the Autocode Compiler already loaded) reads the file and runs the program. The results are written into another intermediate file. (e.g. 'my program.**ptp**')
The paper tape is taken from the computer's tape punch to the teleprinter and placed in its tape reader to be printed.	The file from the simulator is decoded by the "teleprinter" program and displayed on screen (and can be saved and/or printed).

As can be seen in the table, intermediate 'paper tape' files are named after the program source file. The file extension varies to distinguish the different stages. e.g.:

- 'my program.ptr' is a paper tape file to be **r**ead by the simulator,
- 'my program.ptp has been **p**unched out by the simulator'.

Both represent punched paper tapes so both use the same internal coding. A third variant, '.ptw' is used for Primary Input "woosh" files. These are pre-compiled system programs like Initial Orders or the Autocode Compiler, or released application software.

There is another file type introduced later in this document: The simulator has an unofficial (non-authentic) 'back door' through which store images can be loaded instantly from file. These files have the extension '.sto'.

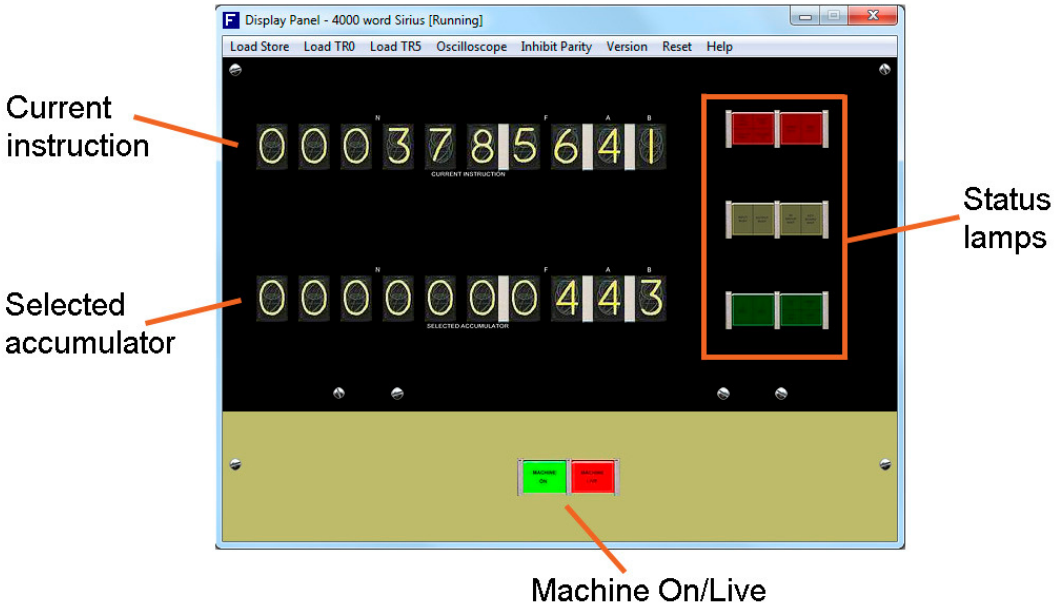
4 The Simulator

4.1 The User Interface

The user interface is based on photographs of a Ferranti Sirius II computer which survives as a superb, though non-working, exhibit at Monash University, Melbourne, Australia [3]. There are two windows displayed when the simulator loads, the Display Panel and the Control Unit.

4.2 The Display Panel

Mouse-clicks are used to "press" the "Machine On/Live" on/off buttons.



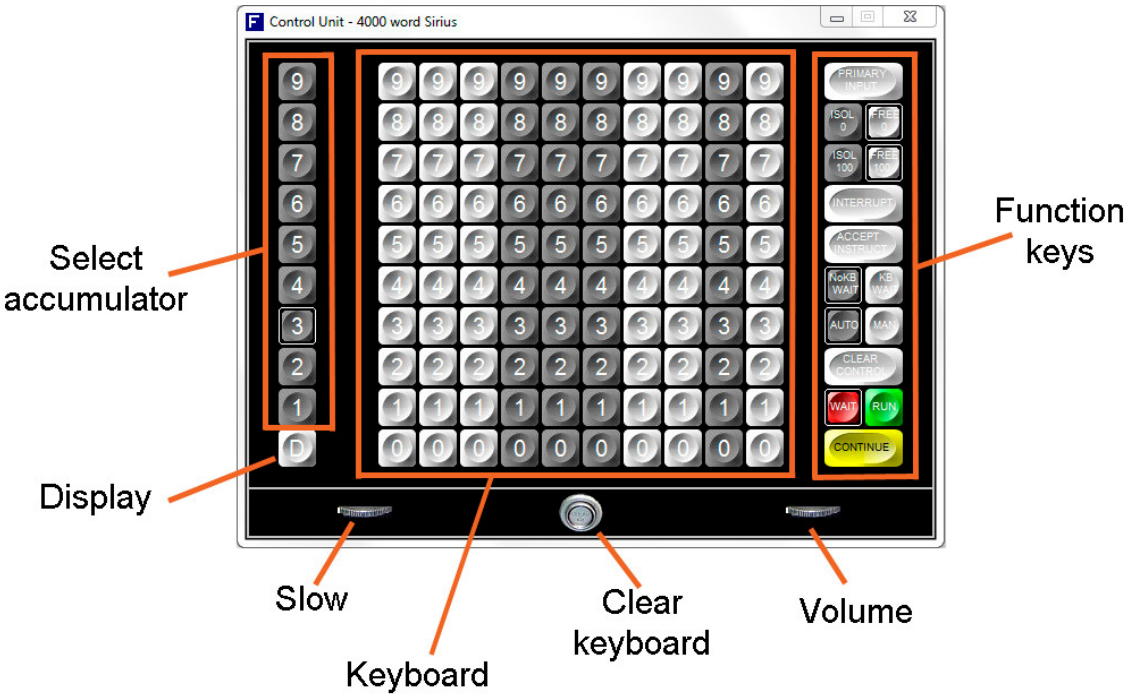
Menu bar detail:

- Load Store: A very fast but non-authentic way to load a store image (see paragraph 4.5). It's an optional 'back-door' facility which avoids performance limitations of the simulator, useful for repetitive tasks like loading the autocode compiler.
- Load TR0/Load TR5: To make a paper tape file (*.ptr) available for reading. Equivalent to positioning a real paper tape in a Ferranti paper tape reader.
- Oscilloscope: Used for the built-in oscilloscope feature (see paragraph 7).
- Inhibit Parity: Substitutes for an engineer's control of that name which was inside the back of the mainframe. If this is used, the red 'Parity Check Inhibited' status lamp will illuminate as a warning. It works but you shouldn't need to use it.
- Version: Early versions of Sirius had individual, round, status lamps instead of the rectangular multi-element lamps which appeared on later versions. You can choose whichever version you prefer by clicking on "Version" in the menu bar to swap between them.
- Reset: Resets the simulator to the state just after loading.
- Help: Displays help information. Context-sensitive help is available by right-clicking over displayed items as well.

The display panel window is drawn to a scale which makes labels too small to read on a PC monitor. Hovering the cursor over labels or status lamps will pop-up the text in a larger font.

4.3 The Control Unit

The simulator's Control Unit behaves like that of a real Sirius. Mouse-clicks are used to press the keys.



As on a real electro-mechanical Sirius Control Unit:

- The central bank of 10 columns of keys, called the "keyboard", is used for inputting a single 10-digit word, one digit per column. The keys stay down when pressed (clicked with the mouse) but are interlocked so that only one key can be down in a column at once.
- The separate left-hand column of numbered keys selects the accumulator whose contents are to appear in the lower 10-digit numeric display of the Display Panel.

  Depressed keys are highlighted by a surrounding white rectangle, as though you can see the edge of the panel cut-out. (In the image above, key 3 is depressed.)
- The function keys on the right perform various functions as labelled. The wider ones do not stay down when pressed. The smaller ones are arranged in pairs, side by side. Only one of the pair can be down at a time so the pairs act like toggle switches. See paragraphs 6.6 to 6.8 in the Ferranti Sirius Programming Manual [4] for full details.
- Slow: This thumbwheel allowed the operator to slow the processor for demonstration purposes. Click to the right or left of the white line to move the thumbwheel on the model. Note that the red 'Not full speed' status lamp illuminates. Return the thumbwheel line to the left to restore full speed.
- Volume: The right-hand thumbwheel controls the volume of the loudspeaker. Click to the right or left of the white line to adjust the volume.
- Clear KB: Between the thumbwheels is a silver-coloured button. Pressing this (clicking it) unlatches and releases any keys which are down on the keyboard.

4.4 Notes and limitations

- Ferranti TR5 paper tape readers (max 300 characters per second) and Teletype punches (max 110 cps) delayed the real Sirius during input and output. “Input Busy” and “Output Busy” lights illuminated on the Display Panel at these times. The simulator is already slow enough (see paragraph 1.3) so it has been decided not to model the speed of these peripherals at the simulated model speed. A delay has been included to make the lamps flicker at the right times but not for proportionally as long as on the real machine. During Primary Input, the chosen delay is even smaller.
- Sirius probably had random numbers in the store at switch-on. Small pieces of code called interludes were available on paper tape to clear the store and run checksums on system programs. The interludes can still be run but are not essential on the simulated model. The store is initialised automatically.
- The simulation model is derived from a ‘Sirius II’ logic diagram which describes the complete mainframe logic. The logic diagram for the separate store cabinets has not been found so modelling of the 3000-word store extension is an educated guess. 4000 words are necessary to be able to run autocode programs.
- In “order code”, orders which have A=0 used as a destination (paragraph 8.10 in [4]) cause a “pip on the hooter". Coding errors detected by Initial Orders or the Autocode Compiler jump to a 6901 loop stop (paragraph 7.13.1 in [4]) and cause "a continuous note on the hooter" (a rapid sequence of ‘pips’) which continues until the WAIT key is pressed. The ‘pip’ has been implemented by a procedure call. Repeated procedure calls do not operate fast enough to produce an audio frequency so it is not possible to play music on this simulator.
- Mains power was controlled by a switch in the power supply unit under the desk and at the wall socket. We are unsure how the Display Panel’s rectangular power on/off illuminated buttons were wired so have assumed that:
    - Pressing the green button (labelled “MACHINE ON” on the Monash University Sirius [3]) switches the logic circuit on and is illuminated while the logic circuit is running (i.e. While the simulation engine is running).
    - Pressing the red button (labelled “MACHINE LIVE” on the Monash University Sirius) switches the logic off but the lamp remains illuminated all the time that mains power is connected (i.e. All the time that the simulation software is loaded).

4.5 An unofficial ‘back door’

The simulator is quite slow for the reasons set out in paragraph 1.3. A fast alternative facility (a ‘back door’) has been provided for rapidly loading pre-assembled, or pre-compiled, programs into store (see ‘Load Store’ in 4.1 above). This is non-authentic and only intended for routine tasks which become tedious from repetition. For example, loading Initial Orders or the Autocode Compiler. They still run authentically, only the loading of them is speeded up.

Real Sirius computer users loaded such system programs in ‘Primary Input’ format. They loaded so quickly that the paper tapes “wooshed in”. They don’t “woosh in” on the simulator! Hence this option. Both methods, Primary Input and Load Store, achieve the same result: they both reload saved images of the store.

Store images of Initial Orders and the Autocode Compiler are provided with the worked examples, but so too are Primary Input versions. You can “woosh them in”, if you wish.

5 The Teleprinter

5.1 The Teleprinter program

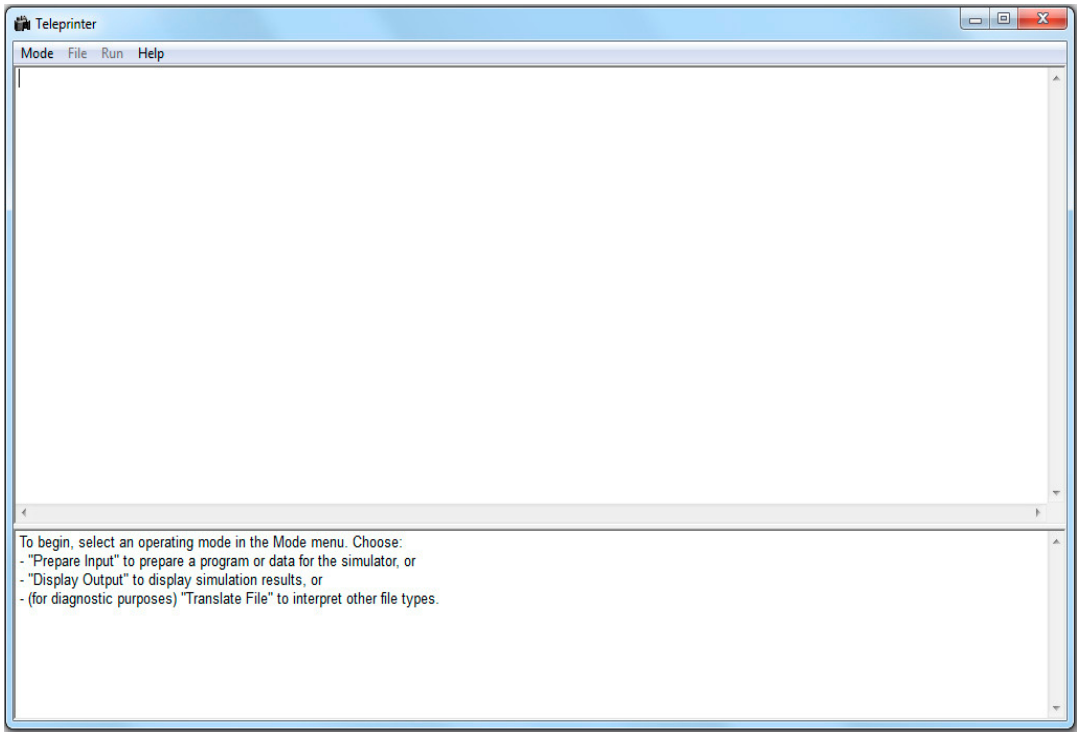
The teleprinter program makes no attempt to look like a real teleprinter. It exists only to support the simulator. The name “teleprinter” has been retained because it has some functionality in common with a real teleprinter and is needed at the same stages of program development.

- Although paper tapes are not used on the PC, the simulated hardware still expects to read and write characters in the 5-hole punching code used on paper tape. The punching codes go into a file instead of on paper tape.
- The Sirius character set is different from (and smaller than) the ASCII character set and some characters which were available on teleprinter keyboards don't occur on normal PC keyboards. For example ≠ (not equal to). Character conversion and the insertion of shift characters is necessary. (See paragraph 6 if you are writing a new Sirius program or editing an existing one.)
- And there are some formatting and ‘paper-tapey’ things like ‘run out’ to deal with.

So a substitute “teleprinter” facility is necessary. Since it exists, a few useful extensions have been included.

To load the teleprinter program, double-click on the file “Teleprinter.exe”.

The window is divided into two panes:



The upper pane can display program code, data or results. The lower pane gives prompts and messages.

5.2 Creating files for input to the simulator

1. Specify the operating mode on the menu bar (Mode > Prepare Input).
2. Either load an existing program (or data) from a text file (File > Open...), or type a new one in the upper pane (after File > New).
3. The contents of the upper pane can be edited on-screen, saved to disc (File > Save/SaveAs...) or printed (File > Print). If a file is saved, the previous version will be renamed to *.bak, first.
4. Run the teleprinter to create a paper tape file for the simulator (Run > Punch Paper Tape). It finishes very quickly. In the lower pane, you will see either "++++++ FINISHED OK ++++++" or a list of errors or warnings.

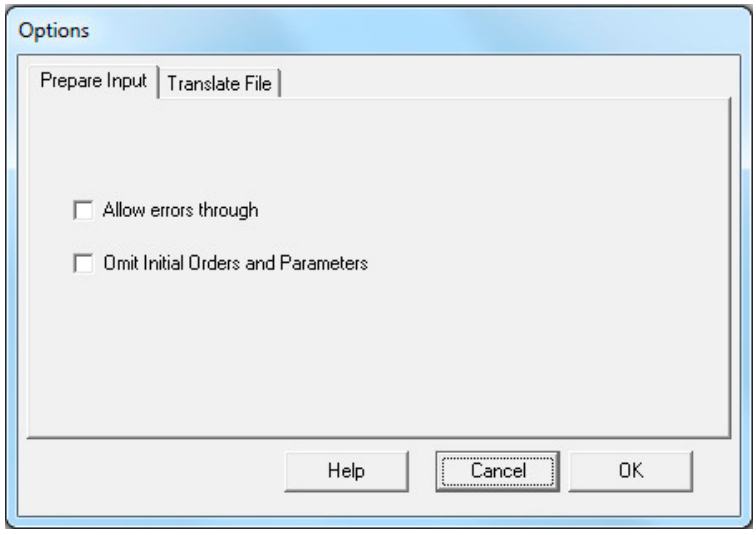
Real teleprinters were electro-mechanical devices so could not report errors. The teleprinter program may detect some errors but do not expect it to be as rigorous as a modern compiler. It is simply a teleprinter replacement tool.

Clicking on an individual error/warning message will highlight its position in the program displayed in the upper pane. Errors can be corrected by editing and saving the program in the upper pane (or in a separate text editor).

Note: The 'Run' menu (in step 4 above) has some extra output formats. These were needed by the developers to create some of the system files which had been lost. They aren't needed for normal use of the software.

Options

Some variation is possible: After selecting "Prepare Input" in the Mode menu, click "Options...", also in the Mode menu, to access the following:



[/] Allow Errors Through

The task of error checking is the responsibility of Initial Orders or the Autocode Compiler running in the simulator. However, the teleprinter program still has to parse the source in order to inject shift characters when converting to the Sirius character set. In so doing, it may report errors and delete its output file. If the errors are believed to be spurious, the output file can be saved by ticking the "Allow Errors Through" box and running again.

[/] Omit Initial Orders and Parameters

Used by the system developers and not needed for normal use of the software.

5.3 Displaying results after simulation

The simulator outputs to a file with the extension ".ptp", instead of punching a real paper tape. To read this file:

1. Load the teleprinter and specify the operating mode
   (Mode menu > Display Output).
2. Open the simulation results file, "*.ptp" (File menu > Open...)

The listing will be displayed and can be printed (File menu > Print) or saved to disc (File menu > SaveAs...)

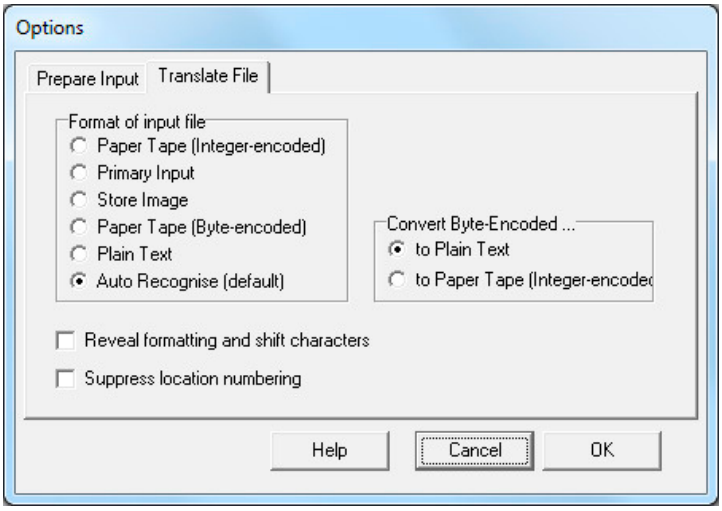
5.4 Displaying other types of file

The teleprinter can interpret other types of file associated with the simulator (Mode > 'Translate File'). It will usually recognise the file type and display it in an appropriate way but there are options to steer the choice if necessary:

- Binary files from a real paper tape reader
  Files '*.bin' and '*.tap' from a real Ferranti paper tape reader are in this format. The files contain blocks of bytes where each byte represents one punched paper tape character in binary. The teleprinter displays them as readable text. There are options (see below) to reveal the position of formatting and shift characters, or to convert the bytes directly to integers (the format required by the simulator) .
- The gate-level simulator's substitute for paper tapes
  Files '*.ptr' (program and data input) and '*.ptp' (simulator output) contain punched paper tape codes. They are text files containing lines of space-separated integers. The binary equivalents of these decimal integers give the punching patterns (0 = blank, 1 = hole) in Sirius paper tape code. The teleprinter translates them into readable text, just as a real teleprinter would do when reading real paper tape.
- Primary Input ("woosh") files
  Files 'Initial Orders.ptw', 'Autocode Compiler.ptw', and other '*.ptw' (some application programs) are in this format. They are, in effect, store dumps coded as for Paper Tape files above. Readable lines of text (10-digit words) are recovered by the teleprinter and prefixed by their store location number. This numbering can be suppressed - see options.
- Store Image files
  Files with the extension '*.sto' are in this format. They are text files containing a list of 10-digit words for direct loading into the simulator's store.

Options

After selecting "Translate File" in the Mode menu, select "Options...", also in the Mode menu, to access these options. Choose the required options, then open (or re-open) the file via the File menu:



Format of input file

You may wish to specify the input file format if the software does not recognise it correctly, or to force display in a different format. (For example, specifying a file as "Plain Text" will display the text actually in the file, rather than its translation.)

Convert Byte-Encoded

You may wish to specify the output format when reading binary files: The default is to translate them to plain text but they can be converted directly to the integer paper tape code understood by the gate-level simulator.

[/] Reveal formatting and shift characters

Ticking this box will reveal the position of control and formatting characters amongst the visible text: <CR> carriage return, <LF> line feed, <ER> erase, <FS> figure shift, <LS> letter shift.

[/] Suppress location numbering

If the input file is in Primary Input (“woosh”) format, it will be displayed in Store Image format with each line prefixed by its store location number. If this box is ticked, the prefix will be omitted and woosh files will be displayed (in effect) as Store Image files. The display can be saved to a new file which could, in turn, be interpreted (by reading it with the teleprinter again) or loaded instantly into the simulator (Load Store) instead of waiting for a slower ‘Primary Input’ load.

6 Writing a new Sirius program

Programs (called “programmes” at the time) can be written as machine-level “order code” or in autocode.

New programs can be typed directly into the teleprinter window, or prepared in a separate text file. They can be named to represent their function e.g. “myprog.txt”. The “myprog” part of the name is carried through to later intermediate file names.

6.1 Differences from the programming manuals

The Ferranti Sirius Programming Manual [4] and Description of the Autocode [5] describe how Sirius programs were to be written in the 1960. There are some differences and special cases to be aware of when writing a Sirius program now: Teleprinters had some keys which don't occur on a normal PC keyboard and the Sirius character set is smaller than the ASCII character set.

The teleprinter program will handle the translation if you make these changes:

The bracketed numbers below, e.g. (34/85), refer to the 'value inside computer' column of Table 6.1 in the Programming Manual [4].

- **LF/CR** (34/85)
  Use ‘Enter’ or ‘Return’ at the end of each line of text.
- **Letter Shift** (79)
  Omit altogether.
- **Figure Shift** (50)
  Omit *except when:*

  … using figure shift for terminating text or titles
  Program name ‘N’ directives and Autocode ‘TEXT’ orders are accompanied by text (names, titles, headings, etc) which can include spaces and spread over several lines. A single figure-shift character is essential for terminating the text. Use a vertical bar | to represent figure-shift in these cases. Examples:

```
NMY PROGRAM|

N
MY PROGRAM
|

TEXT
SALES SUMMARY
Q2 1959
|
```

  … using figure shift for run out
  ‘Run out’ (blank tape produced by repeated figure-shifts) is allowed between sections of code (in the form of repeated l’s) but is unnecessary. It will be ignored in the same way that leading spaces are ignored. The teleprinter automatically adds some ‘run out’ at the beginning and end of paper tape files
- G**reater than or equal to** (55)
  Instead of a ≥, type a } or the two characters >=
- **Not equal to** (74)
  Instead of a ≠, type a # or the two characters /=
- A**rrow** (60)

Instead of a →, type a ^ or the two characters −>

- **Multiply** (70)
  Instead of a ×, type a lower-case x.
- **Erase** (39) is not available and not necessary since errors can easily be corrected. (On real paper tape, mistakes had to be erased by backspacing and over-punching.)
- The teleprinter software allows **comments** and **tabs** to be included in the source file. These are a non-authentic extensions to the language. They do not get passed through to the simulator. Anything on a line from an @ sign onwards will be ignored. e.g.:

  0450 29 @ Set message length in X5

  The teleprinter replaces any tabs by single space characters.

6.2 Use of subroutines

The Ferranti Sirius subroutine library has not been found. Fortunately the S1602 Autocode Compiler, supplied with the installed software, contains a number of useful subroutines which can be called by autocode programs:

MOD, INT, FRAC,
SIN, COS, TAN,
SECANT, COSEC, COTAN,
LOG, EXP, EXPM,
SQRT,
ARCTAN, ARCCOS, ARCSIN.

6.3 Debugging programs

Initial Orders and the Autocode Compiler are rather cryptic and unhelpful in explaining coding errors. We have been unable to find a list of the displayed autocode compiler's error code numbers and their meanings.

(Note: Some compilition errors result in a "6901 loop stop" which causes a continuous sequence of 'pips' on the hooter. It was probably fast enough to be a continuous audio tone on a real Sirius. To stop the sound, click the 'WAIT' key on the displayed Control Unit.)

It pays to be extremely careful in designing a new Sirius program and in typing the source code.

6.4 File naming

In the 1960s, input and output paper tapes had names written on them in pen or pencil. With the simulator, the input filename (e.g. "myprog.txt") is used to name the output file (e.g. "myprog.ptp"). This simple rule can get confused if a separate file of data is loaded on TR0 after the program itself. To prevent program output being incorrectly named after the data file, please ensure that data filenames include the word "data" (e.g. "myprog data.txt"). Otherwise, it will still work but the results may end up in a different file from the one you were expecting.

7 Using the Simulator's built-in Oscilloscope

A storage oscilloscope is built into the simulator. It can be used to probe the gate-level hardware design while it is running.

7.1 A brief introduction to Sirius logic

The Sirius logic design style was invented by Ferranti engineers in the late 1950s.

In this design style, logic gates are formed from circuit elements called "neurons". The neuron design is fixed, regardless of the type of logical function required.

Logic signals propagate as pulses of *current*. Each input pin of a neuron connects to a winding on a small transformer. Signals, arriving at a neuron, 'drive' or 'inhibit' depending on which end of the winding they connect to. When two or more inputs are used, driving signals (current flowing in one direction) and inhibiting signals (current flowing in the opposite direction) are, in effect, added up in the magnetic field resulting in the transformer's core. Since the majority direction "wins" this was known as "ballot-box logic".

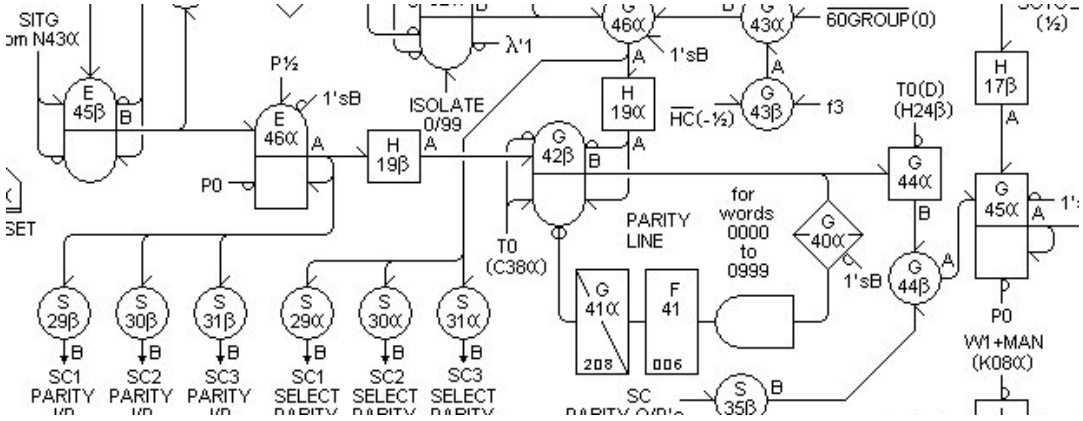
For example, with two driving inputs (and other inputs unconnected) a 2-input logical OR function occurs. With two driving inputs and a third 'tied off' as an 'inhibit', the result is a 2-input logical AND.

The resulting pulse in the output winding of the neuron's transformer gets reshaped to be 1µs wide and is output at the correct time by a small circuit involving two transistors. All neurons are 'clocked', the output occurring either 1µs or 2µs after the inputs. The delay depends entirely on the timing waveforms connected. There is no difference in the design of the neuron.

7.2 The schematic diagram

The complete logic design [6] was drawn on one very large sheet of paper. The diagram is divided into boxed areas labelled "Box A", Box B", etc, corresponding to boxes of plug-in packages on shelves in the mainframe.

A small part of the diagram is shown here:



Most of the symbols represent neurons. Each neuron symbol has a number and a Greek letter which, together with the box label, creates a unique identifier. Neurons with a 1µs delay, called "half neurons", are repesented by circular symbols. Neurons with a 2µs delay, "full neurons", are represented by square symbols. (Remember, it's the external timing signals which determine the delay. There is no physical difference in the neurons.)

Although the neuron design doesn't change, there are two different sizes. A plug-in package can either hold two separate but identical neurons (α and β on the diagram) each with four available inputs and one output, or it can hold one with two input stages combined to form a "double-entry neuron" having eight inputs and one output. Double-entry neurons are represented by double-height symbols. They can be rectangular (if both halves are clocked as

full neurons) or lozenge-shaped (if both halves are clocked as half-neurons) or with one round end and one square end (for a mixture of half and full).

Inputs and outputs are drawn on any side of the symbol or top or bottom, as convenient. Inputs are always identified by a tick mark or semicircle. A tick mark indicates a "drive", a semicircle indicates an "inhibit". The output is the one without a mark. (Inputs with two tick marks or two semicircles indicate that two windings are connected.)

The double height symbol near the right hand edge of the diagram above (G 45α) is used as a staticizer. Its output (2μs after its inputs) loops back as a driving input for the next clock beat. The diagram also includes part of the delay-line store via power driver G 40α.

"Ones generators" create signals (e.g. "1'sB" in the diagram) for tieing-off inputs. They generate clocked pulses of current because, of course, it's the *change* of current in the transformer winding which creates the magnetic field.

There are two main timing waveforms, T1A and T2A, plus their antiphases, T1B and T2B. A letter near each neuron symbol ('A' or 'B') indicates the clock phase. Neurons with only one input used are delays (used so that inputs to a later neuron arrive 'in sync').

The schematic diagram may look as though loads (inputs) are connected in parallel but they are, in fact, wired in series. The pulse of current passes through each winding in turn, terminated after the final load.

There are no 'wired-ors' in the design. Where outputs appear to be connected, timing signals ensure that no more than one output drives at once.

7.3 Probing the Sirius logic design in the simulator

The complete logic diagram is supplied in a file [6].

Signals are 'probed' on the outputs of specified neurons (gates). A list of neuron identifiers, *starting with the one which will act as trigger*, is typed in a text file. Up to 40 can be probed at once.

The start of file "Probes1.txt" (supplied for one of the worked examples) is shown here:

The first line of the file is used as a title for the display. Neurons to be monitored are then listed, one per line.

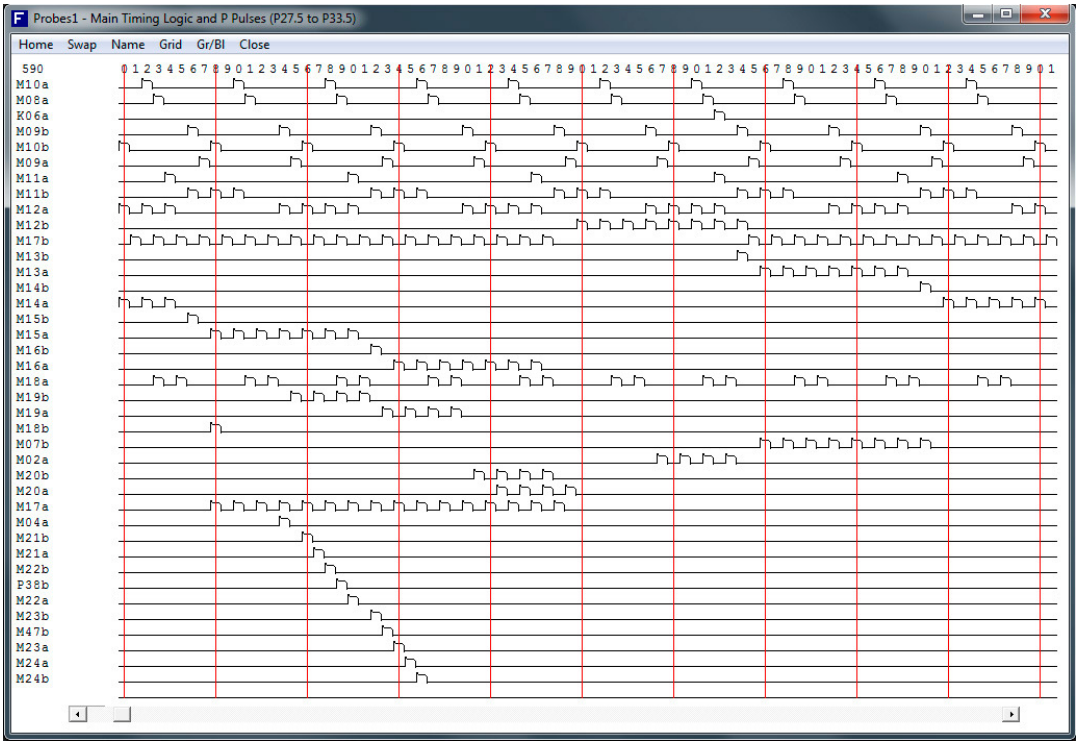
```
Probes1 - Main Timing Logic and P Pulses (P27.5 to P33.5)
M08b  ph3
M09b  ph0
M10b  ph1
...
```

Here, "M08b" identifies the output of the neuron marked "08β " in Box M of the logic diagram. The second field "ph3" is a comment, separated from the neuron identifier by one or more spaces.

Having prepared a 'probes file', and with the simulator running, click 'Set Probes' in the "Oscilloscope" menu of the simulator's Display Panel to select and open the file.

If the first neuron in the list has been triggered, the outputs of all listed neurons will be displayed every microsecond for 20,000 microseconds, long enough for the longest multiply and divide instructions (16ms).

The display resulting from the supplied "Probes1" file is shown below:



Clicking ‘Gr/Bl’ on the menu bar swaps between the traditional ‘green on black’ oscilloscope display and the black & white version shown above, which is better for printing.

Neuron identifiers are listed down the left hand margin in their order of occurrence in the probes file. Either the identifier or its associated comment can be displayed by clicking ‘Name’ on the menu bar. A click on any identifier will highlight its waveform in red.

Time is plotted along the top horizontal axis in micro-seconds (µs) after the trigger pulse. The number displayed in the top left hand corner (‘590’ in the example above) is the time in micro-seconds from the trigger pulse to the start of the current screen. The scroll bar allows movement along the trace. Clicking the scroll bar arrows causes a 1µs step. Clicking elsewhere on the scroll bar gives a 40µs step. The position of each pulse is accurate but the pulse shape, whilst resembling the real thing, is artificial.

Clicking ‘Grid’ on the menu bar will overlay a pattern of grid lines spaced one ‘digit time’ (8µs) apart. Clicking it again removes the lines.

A click at any point on the diagram will produce a vertical red grid line to aid visual alignment.

Clicking ‘Home’ on the menu bar will return to the beginning of the trace, and will remove any grid lines or highlights which have been added.

Clicking ‘Swap’ on the menu bar will flip the display about the vertical axis. (It is sometimes easier to decode the pulses when reading from left to right.).

The waveforms are stored in a file named after the input file. For example, traces resulting from file “Probes1.txt” will be stored in file “Probes1.wve”. This can be re-displayed by clicking ‘Show Waves’ in the ‘Oscilloscope’ menu of the simulator’s Display Panel.

References

1. The Advantages of the Ferranti Sirius Computer, DC43, May 1960.
   (Search for it online.)
2. Computer Conservation Society
   (http://www.computerconservationsociety.org/)
3. Monash Museum of Computing History
   (http://www.infotech.monash.edu.au/about/museum/)
4. Ferranti Sirius Computer - Programming Manual, LD11, November 1960.
   (Transcription supplied.)
5. Ferranti Sirius Computer - Description of the Autocode, CS302, November 1961.
   (Transcription supplied.)
6. Sirius II Logical Diagram. (Re-drawn and supplied as a .tif file)