The Nicolet 1080

A Personal Perspective and Some Nostalgic Reflections

Peter McClintock
Department of Physics, Lancaster University

24 August 2015

The Nicolet 1080 minicomputer/dataprocessor facilitates the acquisition (measurement and ensemble-averaging) of a signal from an experiment as well as for its subsequent analysis. Applied to a diversity of research purposes in Lancaster between 1976 and 1996, it led to more than 150 scientific publications involving 66 researchers in 10 countries – ranging from measurements of the Landau critical velocity for roton creation in superfluid ^{4}He to the first observations of optimal fluctuational paths in nonlinear systems subject to noise. These personal reflections provide summaries of the NIC-80's unusual hardware and of how NIC-80 systems evolved in Lancaster, my introduction to the NIC-80, and descriptions of its use in practice. References are provided giving examples of Lancaster scientific publications reporting on research facilitated by the NIC-80. Descriptions of stochastics programmes and utilities that we wrote for the NIC-80 are provided in appendices.

1. Summary of the Nicolet 1080 Series Hardware

NIC-80 variants. The Nicolet 1080 series consisted of several very similar machines of which about 270 were manufactured in the 1970s. In what follows, I will focus on the LAB-80, MED-80, and BNC-80 which I will refer to collectively as the NIC-80 family. Their immediate predecessor was the 1080 itself and, in computing terms, they were all practically identical to it. The difference was that the 1080 was absolutely covered in hardware switches and adjusters, whereas in the NIC-80s nearly all the functions in question were performed by software. Hence the NIC-80s looked tidier and friendlier – and were less expensive to manufacture.

The LAB-80 and BNC-80 were identical (Nicolet sold the latter via the Bruker Data Corporation, and allowed them to put their name and logo on the front panel). The MED-80 had a different internal digitiser with more channels, better-suited to work in physiology and medicine, but otherwise was the same as the LAB-80 and BNC-80. There was also an NMR-80, which I understand was the same as the latter two machines.

Dual operation. Each of these machines was capable both of making measurements from an experiment, using an internal 100 kHz digitiser or an external transient recorder, and of analysing the results.[1] Thus in one mode it behaved as a hard-wired signal-averager, ensemble-averaging sequential repetitions of a signal into a block of memory selected with the buttons on the front panel: this averaging enabled the extraction of a weak signal buried in electrical noise because the noise, being random, tended to average away to nothing, thereby exposing the signal. Once the acquisition process was complete, the NIC-80 switched to its minicomputer mode, facilitating immediate analysis of the acquired/enhanced signal. Operations in the two modes shared the same 20-bit memory and used the same 20-bit arithmetic logic unit (ALU.) This configuration was a vast improvement over earlier arrangements where there had to be a time-wasting transfer of the averaged signal to a separate minicomputer for analysis.

Memory. The NIC-80 used ferrite core memory. The unique 20-bit word length was a characteristic Nicolet feature said to result from a compromise[2] between precision and depth for averaging, on the one hand, and cost on the other: ferrite core was exceedingly expensive.

The NIC-80s normally operated with either 8K or 12K of memory, corresponding to either two or three 4K "stacks" of ferrite core. The 20-bit memory conferred huge advantages in comparison with the 8-bit and 16-bit competitors of that era. It also provided for very simple and efficient operation in which most computer instructions, including the destination address, could be contained within a single 20-bit word.

Number conventions. The NIC-80 used octal, with bit-19 (the leftmost bit) treated as the sign bit. Thus the largest positive integer was 1777777 (524,287 in decimal); the largest negative integer was 2000000 (–524,288); and 3777777 (octal) represented -1 (decimal).

Decimal numbers. As Jack Kisslinger comments,[2] "The 1080 was completely unaware of decimal arithmetic." So this had to be accomplished in software using the Floating Point Package.[3]

Programming. Versions of BASIC were available,[4,5] but the real power and computational capabilities of the NIC-80 required assembly language – which, fortunately, was relatively easy.

Hardware multiply/divide. A particular strength of the NIC-80 lay in its hardware multiply and divide facilities. These utilised an additional 20-bit register, the MQ, along with the Accumulator. They enabled two 20-bit octal numbers to be multiplied with 40-bit precision, or a 40-bit octal number to be divided by a 20-bit octal number, in a single instruction. In the latter case the result was the quotient and the remainder, each to 20-bit precision. The hardware multiply/divide hugely speeded up arithmetic processing and made the NIC-80 an effective machine for all scientific applications, including those requiring the computation of Fourier transforms, e.g. nuclear magnetic resonance (NMR) spectroscopy.

Buttons. The upper horizontal row of buttons on the front panel selects the pages of "data memory" to be used for the signal during data recording. The NIC-80 regarded octal addresses 0-7777 as programme memory and 100000-117777 as data memory; the address range 10000-100000 did not exist. So selecting START = 0, SIZE = 1K would set 100000-101777 as the block for signal acquisition/averaging. In reality, all the memory was identical and many of our programmes ran partly in the address range of so-called data memory. The buttons also allow for using extended memory, beyond the 12K that could be fitted internally.

Knobs. Two knobs on the front panel serve as a general-purpose facility for making manual adjustment, usually related to the CRT display, e.g. control of expansion and offset, or positioning of cursors. They are in fact just multi-turn potentiometers whose voltage output can be digitised and read into software, so they can be used for almost any purpose where fine adjustments are to be made by hand while monitoring the results, e.g. for level-setting routines in the Stochastics Library (see below).

Internal ADC. The sampling rate of up to 100 kHz was set via software, and the resolution up to 12 bits was adjustable with the front panel buttons or under software control. In practice we left the buttons set permanently to "Program Control".

Vertical display scale shift. The switch is used to bring the displayed data within the vertical range seen on the CRT. (The VDSH instruction left-shifts the Accumulator one bit for each click that the knob is set back from the 131K position.)

LEDs and switch register. The 3 rows of LEDs on the front panel show the contents of the Accumulator, Instruction Register and Program Counter, respectively. Using the switch register (one switch per bit) and buttons enabled the contents of any address to be read, or to be set to a chosen value. (To inspect the contents of the MQ required that they be transferred to the Accumulator.) It was possible to start the machine running at a chosen address, or to step through the instructions one-by-one noting the change in the Accumulator at each step, which could be very useful in tracking down "bugs" in programmes. When the machine ran, all the LEDs twinkled – an amazing sight!

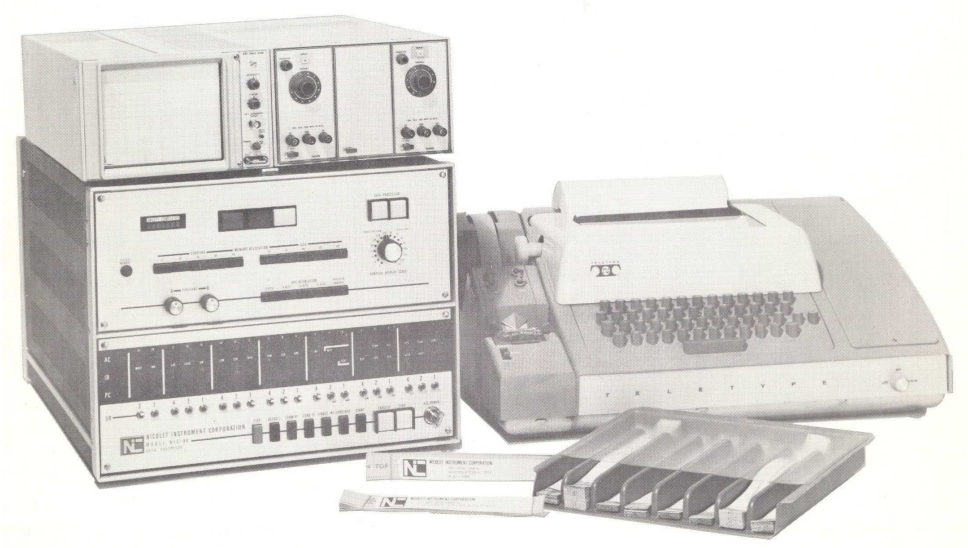

Figure 1: The minimal NIC-80 configuration[6] as purchased by Lancaster. It consists of the NIC-80 itself with 8K of internal memory, plus a high-resolution Tektronix CRT display for data (on top), a set of paper tapes with Nicolet programmes (front), and an ASR 33 Teletype (right). Later, we added a third 4K stack of ferrite core memory and NIC-298 (floppy) and NIC-294 (Diablo) disk drives. We also replaced the Teletype with a VDU together with an optical tape reader and fast punch.

Input/output. The standard minimal configuration, as we initially purchased in Lancaster was a shown in figure 1. The Teletype provided the sole input/output facility for the NIC-80. It was used for reading in paper tapes with Nicolet programmes, punching paper tapes with data or new programmes, and for printing out results or listings of programmes, as well as for keyboard entry of commands and programmes. Being purely mechanical it was noisy and very slow.

However, the NIC-80 also had an RS232 connector that we later used for a visual display unit (VDU) with dot-matrix printer. In addition, there was provision for connecting both hard and floppy disk drives, as well as fast (optical) paper tape readers and a fast tape punch. Using these devices in place of the Teletype transformed the NIC-80's capabilities, provided the "feel" of a much more modern computer, and enabled overlay-programming with hugely enhanced computational capabilities (see below). There was also a second channel of RS232. In the NIC-80 as delivered, the RS232 B-channel was inactive. When we eventually discovered it and connected it up, it allowed us to transfer data to our Unix mainframe and to write software (e.g. a Kermit) that facilitated the transfer of programmes and data between NIC-80s, as well as for general communication between our NIC-80s and the outside world.

2. Using the NIC-80 for Research in Low Temperature Physics

Why did we buy a NIC-80? It was to support our research on liquid ^{4}He – an exotic liquid that exhibits the strange property of *superfluidity* when cooled below 2 K, i.e. close to the absolute zero of temperature. That is, it loses all its viscosity so that a moving object experiences no drag at all and behaves almost as though it is moving in a vacuum. However, there was a long-standing prediction by the famous Soviet physicist Lev Landau that the superfluidity would be lost once the object exceeded a critical velocity. We wanted to test this theory. We used negative ions as the moving objects, and their arrival after traversing the experimental cell could be registered as a tiny pulse of current – a very weak signal buried in random electrical noise mostly generated by the electronics.

Our initial experiments enhanced the signal/noise ratio by use of a Hewlett-Packard signal-

averager. It did the job very well, but there was no easy way of getting at the averaged signal other than to plot it out on paper and measure it with a ruler to find out how fast the ions had been going. So we planned to use instead a NIC-80 where we could both average the signal and then analyse it in a precise and quantitative way after averaging had been completed.

Arrival of our first NIC-80. We took delivery of our minimal configuration NIC-80 (as in figure 1) in 1977. Rather than the internal digitiser, which would have been too slow, we used an external Biomation 8100 transient recorder to acquire the signals, which were then passed to the NIC-80 for ensemble-averaging. Fortunately for me, I had a very talented PhD student at that time, David Allum, who took charge of the NIC-80, worked out how to operate it, wrote the required analysis software to complement Nicolet's Lab80 signal averaging programme,[6] and then instructed me in how to use the system. I found it quite daunting when I first tried it on my own, after David had finished his PhD and left to take up a job in computing.

Operating the NIC-80. As the Nicolet programming manual[1] puts it, "When a computer is first manufactured, it 'knows' nothing. It does not even know how to read in program tapes." The full procedure for loading a programme was therefore as follows –

1. Toggle in 14 instructions from the switch register – the so-called Nico-Loadeon bootstrap – which occupy a particular set of octal addresses (7736-7753).

2. Insert the Nico-Loadeon paper tape into Teletype tape reader, and start the processor at address 7740. If all goes well, this deposits the Self-Checking Binary Loader in addresses 7632-7777. It was supposed to be retained there... but in practice the bootstrappping procedure was needed quite frequently (so frequently, in fact, that David tells that he could reproduce the 14 octal instructions from memory!).

3. Insert the programme tape into the Teletype, and start the processor at address 7777. It takes typically 40 minutes or an hour to read in. At the end, or sometimes at intermediate stages, the tape would halt and the Teletype bell would ring, indicating a checksum error. It was then necessary to repeat the process from the beginning.

Once the programme tape had been successfully loaded, the NIC-80 could be started at the appropriate address (usually 0) to run the programme.

Upgrading the NIC-80. We realised almost immediately that a disk drive was going to be essential – running the NIC-80 via the Teletype from paper tape was altogether too slow and inefficient and limiting. Fortunately SERC agreed to fund a NIC-298 8" floppy disk system.[7–9] It absolutely transformed the capabilites of our NIC-80. At the same time, we upgraded the memory to 12K by adding a third 4K stack (for about £3500, if I remember correctly). And we replaced the Teletype with a VDU – a standard ASCII terminal – which was much faster as well as far quieter.

Later, when we embarked on research in stochastic nonlinear dynamics (see below) we added further hardware and enhancements, including many items bought cheaply from (or given to us by) other people who were upgrading e.g. to NIC-1180 or NIC-1280. These included NIC294 Diablo 30 hard disk units,[10,11] optical paper tape readers,[12] and a Remex paper tape reader/punch (a bargain from Nicolet).[13] In this way we obtained 5 additional NIC-80s, all complete working systems (after minor repairs and renovation) with disk drives and peripherals.

Breakdown of superfluidity and beyond. In our low temperature experiments, we quickly found[14] that the averaged ion signals took very interesting exponential shapes, rather than being trapezoidal as expected. These effects were attributable to a second mechanism by which the superfluidity could break down – through individual ions creating *quantised vortex rings* (like microscopic smoke rings). We were able to programme the NIC-80 to extract the

exponents and hence measure the vortex nucleation rates, which could then be compared with theoretical expectations.

A lot of this work was done in collaboration with Frank Moss from the University of Missouri St Louis (UMSL), who was a well-known low temperature physicist with research interests similar to my own. He paid two sabbatical visits to Lancaster while on leave from UMSL. The combination of Frank and the NIC-80 changed my life and my career.

Isotopically pure superfluid ^{4}He. Frank's first visit was in 1979. Being a vivid and somewhat larger-than-life character he had somehow persuaded UMSL to give him an 18-month sabbatical instead of a normal 12-month sabbatical. His idea was to split his time between Lancaster and Exeter. On arrival he was able to "hit the ground running" and contributed to the research immediately because he was already closely familiar with the NIC-80 – he had an identical system in his own laboratory in UMSL. We had just developed a helium isotopic separation cryostat, and we were thus able to carry out the first investigation of ion motion and vortex ring creation in isotopically pure superfluid ^{4}He. We discovered that even the tiny traces of the rare ^{3}He isotope in natural helium (about 2×10^{-7}) can exert a profound influence[15] on vortex creation in liquid ^{4}He, and that the creation rate is a rapid function of temperature.[16] Frank then left for Exeter. Shortly afterwards we found that the ion-vortex-ring transition could be quenched (stopped from happening) by sufficiently strong electric fields.[17]

A change of direction. On his return to Lancaster several months later for the final part of his sabbatical, I found that Frank had developed a new enthusiasm that was totally unconnected with liquid helium or low temperature physics. He had spent a lot of his time in the Exeter University Library and had become enthralled by a controversy that was raging in stochastic nonlinear dynamics. The point at issue was quite technical, but of very wide applicability – to any system whose fluctuations could be approximated by Gaussian white noise, that is, totally random noise. Hence resolution of the matter at issue would relate to problems not only in physics, but also in engineering, mathematics, environmental science, economics, finance, biology and so on. The question was: what is the appropriate calculus for integrating the Langevin equation to find the Fokker-Planck equation describing the fluctuating experimental system? There were two basic choices: the Ito or Stratonovich calculi. Many authors gave all their results for both calculi because they were uncertain which one to use.

What Frank realised instantly was that, by using the NIC-80, we could do an experiment to settle the question definitively (see below). His enthusiasm was highly infectious and, in the end, I agreed to collaborate. It led me off into an entirely fresh direction, with a whole new world of fascinating research problems to be solved, numerous new collaborators and friends, and a vastly enlarged perspective on science.

I remained a low temperature physicist too. Although I supported the low temperature research with newer Nicolet products, including a NIC-1280 (strictly an 1180E) with an internal 20 MHz digitiser and the *phenomenally* fast[2] 24-bit Fourier array processor,[18] I continued to use and develop the NIC-80 for experiments in stochastic nonlinear dynamics as described below. It was ideally suited for this application.

3. The NIC-80 and Stochastic Nonlinear Dynamics

Ito v. Stratonovich. To perform Frank's initial experiment, it was necessary to measure a probability density. This was at a time when many theorists believed that probability densities were not even physical observables – so it was an ambitious project. After very extensive discussions, we agreed that Frank would construct the nonlinear analogue experiment in UMSL, and that I would write the required NIC-80 acquisition and analysis software in Lancaster. I did so, and mailed him the result (DENSFN) on an 8" floppy disk. When he did the experiment, it provided the first observation of a noise-induced phase transition and clearly demonstrated[19,20]

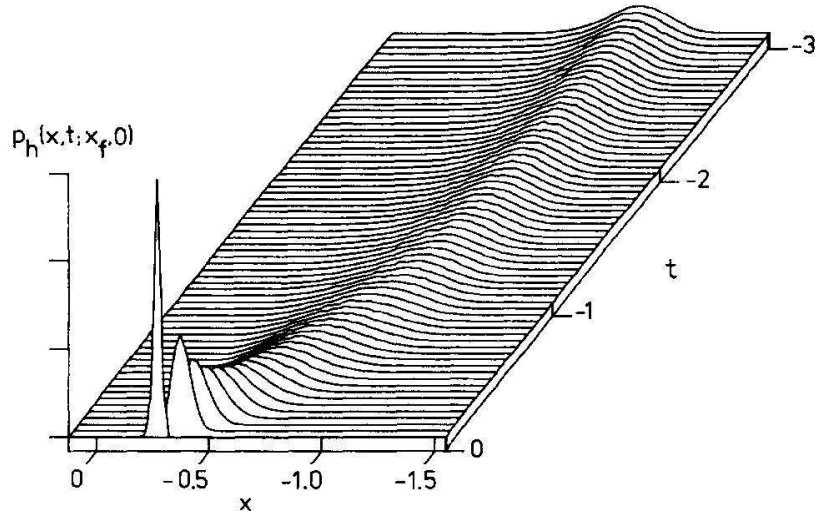

Figure 2: Pioneering measurement of the prehistory probability density[36] $p_h(x, t; x_f, 0)$ of a rare large fluctuation. The system is subject to noisy forcing under equilibrium conditions. In the remote past (time $t = -3$ at the top), the system is fluctuating about its stable state at $x = -1$. The large fluctuation brings it to the chosen final value of $x_f = -0.7$ at the present time $t = 0$. The optimal path is the ridge (locus of maxima) along the top of the distribution. The distribution was created and plotted by a NIC-80, as were also the axes and tick-marks; the labels and numbers were added by hand using a pen with Indian ink and a stencil, as was standard practice at the time.

that a continuous physical system is described by the Stratonovich calculus*.

Lancaster Nonlinear Group. At this point, Frank and I both realised the potential we had unleashed for solving other problems in stochastic nonlinear dynamics. We continued to work closely together for several more years, and Frank paid another sabbatical visit to Lancaster (1986–87) – this time, not as a low temperature physicist, but as a nonlinear dynamicist. We had a joint NATO grant that aided collaboration within Europe, and co-edited the 3-volume book[21] *Noise in Nonlinear Dynamical Systems*. After that, we gradually drifted apart and mostly worked on different nonlinear problems, with Frank eventually moving into neurodynamics. By this time, I had a flourishing Lancaster Nonlinear Group, mostly funded by SERC/EPSRC, and characterised by numerous international research collaborations and frequent publication in high impact scientific journals. I retired formally in 2008, but the group is burgeoning, and I remain an active member. It has now been renamed the Nonlinear Biomedical Physics Group to reflect a modified emphasis under its new leader, my colleague Professor Aneta Stefanovska, who took up her permanent position in Lancaster in 2006.

The 1991 Stochastics Library[22] brought together DENSFN and the many stochastic analysis facilities that had been created subsequently – at one stage it felt as though every new project involved the measurement of a new stochastic quantity. Details are given in Appendix A but, basically, the Library includes provision for the measurement and ensemble-averaging of: single-variable probability distributions;[19,20] two-variable probability distributions;[23,25] first passage times, crossing-times after a trigger, sojourn times, Akito jump distributions or distributions of level-crossing counts;[26,27] correlation functions;[28,29] power spectral densities;[31–34] two-dimensional distributions in a variable and time;[24,35] the time evolution of moments;[30] phase portraits; the pre-history probability of level-crossing transitions;[36–38] and two-dimensional distributions in amplitude and phase, $P(A, \phi)$. The citations refer to a few examples of papers where some of the routines in question were used.

*It is now well understood – indeed obvious – that, because the noise in a real physical system cannot be truly white, the Stratonovich calculus must the right one to use because it takes the evaluation point in the middle of the infinitesimal interval, unlike the Ito calculus which takes it at the beginning. But it was far from obvious to many people at the time.

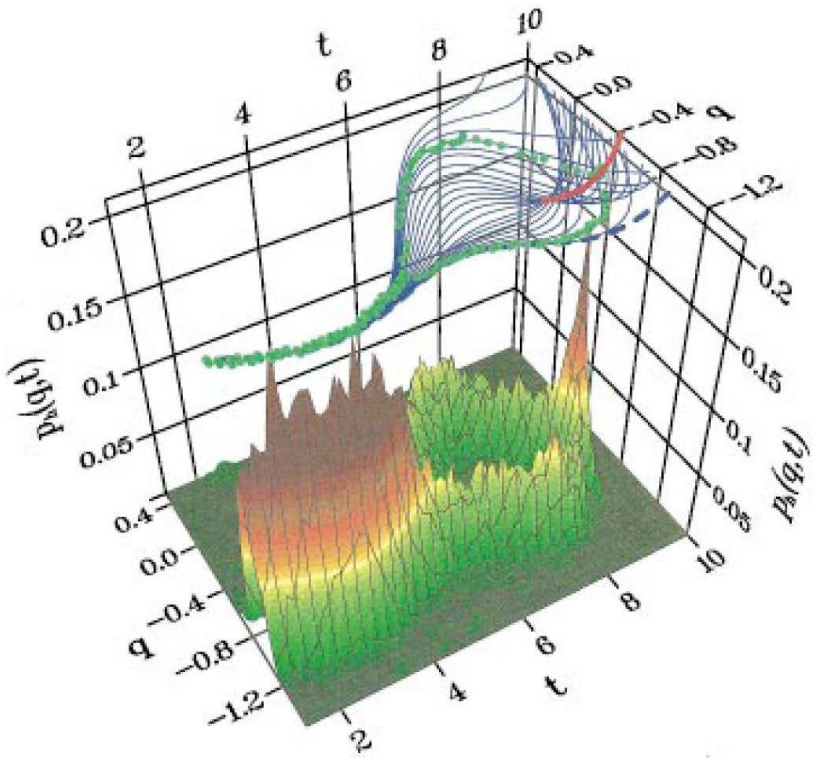

Figure 3: A prehistory probability density $p_h(q, t)$ measured by a NIC-80 under non-equilibrium conditions[37] when the nonlinear system was being driven by a periodic force (with q rather than x being used for the coordinate, and the direction of t reversed, compared to figure 1). The top-plane compares the measured optimal paths (green points corresponding to the ridges of the probability density) with theoretical predictions (thin continuous lines). The final point reached by the large fluctuatons is on the system's *switching line* (red). Only under these conditions is a *corral* of optimal paths created, as illustrated. Here the data were again acquired and ensemble-averaged by the NIC-80, but its Kermit was used to transfer the resultant prehistory probability distribution to an external system for plotting and comparison with theory.

Two examples of probability densities measured with a NIC-80 are shown in figures 2 and 3. They illustrate the *optimal paths* followed by (rare) large fluctuations in noise-driven nonlinear systems, identified by measurement of the relevant *prehistory probability density*. Note that they are simply illustrative, chosen from among the enormous number of interesting and scientifically important examples that have been published.

The theory of large fluctuations in nonlinear systems predicts that, though rare, they should all tend to take the *same* least-energy path in any given system. The importance of such processes arises because nearly all of the important "events" that occur in noisy systems – e.g. first order phase transitions or chemical reactions – come about through large rare fluctuations. There was already a well-developed, but almost untested, body of theory – whence the importance of analogue experiments. Electrical circuits were used used as generic models of the systems of interest, and driven by white noise. The NIC-80 would then track the fluctuations $x(t)$ of the system. On rare occasions, an extra-large fluctuation might take it to a pre-selected value x_f far from the system's stable state. The NIC-80 was programmed so that, whenever this happened, it would "grab" the immediately preceding path $x(t)$ and add it to an ensemble average representing the prehistory probability of arrivals at x_f. From the theory, it was expected that the prehistory probability would be a clearly-defined and relatively narrow tube of paths: the so-called *optimal path* would be the ridge along the top of the prehistory density. Because large fluctuations are by definition rare, the acquisition of their prehistory distributions is slow and laborious, often extending over a day or even a few days to build up a single distribution.

Figure 2 illustrates a measurement of the prehistory density for (rare) arrivals of the system at x_f, under equilibrium[36] conditions, thereby demonstrating the physical reality of optimal paths. Not immediately obvious in the figure, but clear on careful inspection, the density unexpectedly broadens before converging to the final point at $t = 0$, a feature that was quickly accounted for theoretically. Figure 3 illustrates measurements under non-equilibrium conditions[37] when the system is being driven by a periodic force. These results demonstrated the physical reality of *cusps* and *switching lines* together with *critical broadening* of the distribution near a cusp point and many subtle and interesting physical details as discussed in the original paper.[37]

4. Lancaster Utilities for the NIC-80

Over the years, I wrote several utility programmes for the NIC-80 to add additional facilities and to make the system more convenient to use. The more important of these are listed here and described in greater detail in Appendices B – E.

LINKER overcomes the NIC-80's "memory gap" between 7777 and 100000. A single command can run a composite programme consisting of any number of files, or can move it between disks. LINKER also provides for overlay generation. Details are given in Appendix C.

CLEAR zeroes the entire memory, apart from the disk monitor head 7600-7777.

CHECK tests the validity of data and programmes stored on floppy (NIC-298) or Diablo (NIC-294) Nicolet NIC-80 disk systems and reports any checksum errors that are detected.

DIMON links together and runs automatically all of the NIC-80 diagnostics programmes.[39]

PNED is a supplementary text editor that provides facilities for extraction of text as disk files, or for the insertion of text from files, during editing.

FBAKUP enables the copying of whole disks on single-floppy-disk[7] NIC-298 systems.

DSKGEN provides for automatic initialisation of floppy disks by linking together the sequence of routines provided by Nicolet. It saved us a lot of time.

DIABLO, FLOPPY and BOOTS are for use on dual floppy/Diablo systems. A *RUN DIABLO to the floppy monitor transfers attention to the Diablo; and *RUN FLOPPY to the Diablo monitor does the opposite. BOOTS provides spare bootstraps for emergencies.

FDMOVE is a utility for moving files between NIC-298 (floppy[7]) and NIC-294 (Diablo[10]) disk drives on hybrid 12K NIC-80 systems.

PNICL implements a communications protocol that enables files to be passed between Nicolets. It is fully compatible with FILMOV on the NIC-1180 and NIC-1280.

KERMIT implements a communications protocol that enables files to be passed, universally, between all kinds of computers. See Appendix D.

5. Epilogue: After the NIC-80

Capabilities of the NIC-80. Frank Moss remarked in the mid-1980s "You can do *anything* with a Nicolet" (meaning a NIC-80), and it was almost true. For example, the DENSFN programme to measure probability densities must have been quite different to anything the NIC-80 designers had considered but, given the extraordinary flexibility of the system, it was nonetheless straightforward to implement. (The only unexpected problem arose from the design of the internal digitiser, for which certain levels were favoured over others, giving rise to strange patterns in measured densities; we overcame this difficulty by ensemble-averaging the probability density of a triangle wave (which should have given rise to a constant level, but did not), using the result as a "calibration" by which we could divide subsequent density measurements.) Likewise, when we first measured probability densities in two variables $P(x, y)$ we had no means of

plotting these 3-dimensional objects; but I was able to write a contour-following routine for the NIC-80 and the corresponding (x, y) values could be taken to the NIC-80 digital-to-analogue convertors and applied to an $x - y$ plotter to draw the contours. Later, I also wrote a routine for making perspective plots of the density on the $x - y$ plotter. And the Kermit opened up new vistas for the NIC-80 by connecting it to the outside world.

Longevity of the NIC-80. My collaborators and I used our NIC-80s on a daily basis for 20 years, and they did everything that we asked of them. This is an *extraordinarily* long lifetime for anything in the computer area, given the exponential rate of progress and development, and it was made possible by a number of factors. First, the visionary original design by Nicolet... Secondly, the excellent and friendly service provided by the local (Warwick) Nicolet people, especially in the earlier stages when I constantly needed advice. Thirdly, as the years rolled by, I became steadily more knowledgeable and expert in NIC-80 programming, and more familiar with the hardware, so that I was almost always able to find a way of doing what was needed. In addition, with the extra utilities (see above), the system became really easy and simple to use.

Moving on. However, all good things come to an end. For the NIC-80, the end relative to its beginning came later than for any other computer that I know of, but it came. The NIC-80 itself could have continued for a while longer – its speed, and the speed of the internal digitiser, were both well-matched to the characteristics of the analogue electronic circuits that we used for nonlinear modelling. Of course it eventually became harder to obtain spare parts, e.g. the special diodes needed for the digitisers, but we always managed in the end. The real problem was the disk drives. The gigantic Diablo 30s stored relatively little data. Although that did not matter except æsthetically, they were prone to crashing occasionally unless serviced regularly, and sometimes crashed despite regular servicing. Such crashes were often extremely destructive: the head hit the disk and scraped off the oxide layer, sometimes right down to bare metal. Consequently, both the head and the disk were destroyed and all data on the disk were lost. The man who serviced our Diablos in the latter stages came from far away (Essex, I think), so the drives were unavoidably expensive to run. Furthermore, he was already at retirement age, and there was no reason for younger people to learn to service such ancient equipment. The 8" floppies worked fine, but of course were slower, and they started to look slightly ridiculous (as $5\frac{1}{4}$"s, and then “stiffies”, came and went). They were also becoming harder to get hold of. No doubt it would have been possible to write new software for interfacing the NIC-80s to modern recording media... but, putting everything together, we concluded that the time had come to move on. I did so with profound regret and a considerable sense of loss.

The replacement Stochastics Library was created by Igor Kaufman[40] in collaboration with Dmitri Luchinsky. It was initially intended to run on PCs to which Microstar digitisers[41] had been fitted, and it is extremely flexible in every respect. The system is Matlab-based and multi-platform (Linux as well as Windows). In principle, it does everything the NIC-80 Library did, and it has been a triumphant success.

The end of the NIC-80 saga came for me in 1997. It had been an immensely satisfying and productive 20 years. The tally of the Lancaster Nonlinear Group and ^{4}He Group researchers and collaborators during this period shows that this remarkable instrument had supported the work of 66 researchers, not only in the UK, but also in Belarus, China, Germany, Italy, Japan, Russia, Spain, Ukraine, and USA, leading to more than 150 scientific publications in total.

Questions and comments can be sent to `p.v.e.mcclintock@lancaster.ac.uk` and are welcomed. The scientific papers cited as examples below are all available on request, and many of them can be downloaded from
`http://www.physics.lancs.ac.uk/people/peter-mcclintock`

APPENDICES

APPENDIX A
The 1991 NIC-80 Stochastics Library

The Stochastics Library[22] is worth discussing in reasonable detail because it gives one a good idea of what could be done with a NIC-80. The Library consists of a suite of stochastic dynamics analysis programmes. Each programme is designed for a specific purpose, and is self-contained, but shares many commands and facilities in common with the other programmes in the Library. The programmes include –

DENSFN – ensemble-averages single-variable probability distributions,[19,20] $P(x)$

XYDENS – ensemble-averages two-variable probability distributions,[23,25] $P(x, y)$

FPTIME – ensemble-averages either: first passage times, crossing-times after a trigger, or sojourn times $P(t)$; or Akito jump distributions; or distributions of level-crossing counts, $P(n)$[26,27]

CORAVG – ensemble-averages correlation functions,[28,29] $C(s)$

POWSPC – ensemble-averages power spectral densities,[31,32] $Q(\nu)$

XTDENS – ensemble-averages two-dimensional distributions in a variable and time,[24,35] $P(x, t)$

MOMAVG – ensemble-averages the time evolution of moments,[30] $\langle x^n(t)\rangle$

PHASEP – ensemble-averages and analyses phase portraits $y(x)$

PREHST – ensemble-averages the pre-history probability of level-crossing transitions,[36–38] $P_h(x, t; x_f, 0)$

APDENS – ensemble-averages two-dimensional distributions in amplitude and phase, $P(A, \phi)$

Structure of the Stochastics Library. All of the programmes use overlays, many of which are shared in common. The programmes are bi-diskual in that they will run both with floppy[7] (NIC-298) and with hard[10] (Diablo-30, NIC-294) disk drives, and they are all constructed on the same basic principles. A sequence of $x(t)$ (and, sometimes, $y(t)$ simultaneously) is digitized in blocks of 1-4 K, and each of them is then analysed to produce the selected probability distribution, which is ensemble-averaged in a separate block of memory. When the chosen number of realisations has been processed, the resultant distribution is displayed on the CRT and a variety of commands becomes available for analysing it and examining it in more detail.

Memory page-layout of the Stochastics Library. The twelve 1K pages of the NIC-80's memory are used as follows –

0 - 1777	LAB-80 routines, main command block, data acquisition
2000 - 3777	LAB-80 routines, expansion facilities
4000 - 5777	Data analysis for $P(x)$, $P(x, t)$ etc, and stochastic command block
6000 - 7777	FPP72, disk monitor overlays, disk monitor head
100000 - 101777	Acquired $x(t)$, overlay loading
102000 - 103777	Acquired $x(t)$
104000 - 105777	Acquired $x(t)$
106000 - 107777	Acquired $x(t)$, file-header construction
110000 - 111777	Distribution ensemble-averaging e.g. $P(x)$
112000 - 113777	Overlay generation, replica construction
114000 - 115777	Loading linker, NICBUG, BOOTS, overlay loader
116000 - 117777	Absolute addresses, command initiation

where all the the addresses are in octal. The Nicolet LAB-80 programme,[6] which occupies 0-3777, has been extensively modified so as to incorporate extra commands and to interface it to the larger system of which it becomes one part.

Page 114000 is identical for all programmes in the Library. The standard Nicolet octal debugging programme[1] NICBUG is incorporated (114632 - 115365) just above the loading linker (114000 - 114631), and can be used to examine or modify the contents of particular addresses and generally monitor the operation of the programme. The disk[7,10] and NICOLODEON[1] HSR bootstraps, which occupy 115366-115476, are for emergency use. The entry points, using the switch register (or NICBUG), are: 115367 to read a floppy monitor head; 115414 to read a Diablo head; and 115436 to read in the NICOLODEON paper tape[1] on the high speed reader (HSR). Page 114000 also includes the system overlay loader, and an overlay linker that enables one overlay to call another.

Page 116000 has first and last sections that are shared by all programmes in the library. The first part (116000 - 116577) contains standard routines for printing integers or floating point numbers, displaying data blocks of arbitrary size, displaying horizontal or vertical lines, and for double-precision acquisition. The last part (117660 - 117776) consists of absolute addresses for expansion, least-squares fitting of data, plotting data, etc, which are used by the various overlays (see below). Addresses 116600 - 117657 are available for normal programme use; and 117777 is reserved as an indirect jump address for NICBUG.

Overlay loading. The overlays are all loaded at 100000 (i.e. at START = 0, as set on the buttons). Some of them are patched together during assembly of the programme, but the majority are free-standing. There is provision for up to 36 overlays: up to 26 standard ones named STLAYA, STLAYB, STLAYB ... ; and up to 10 programme-specific overlays, which are given names formed from the first two letters of the programme name, suffixed by “LAY" and an alphabetical letter. The overlays for FPTIME would therefore be: FPLAYA, FPLAYB, etc. The overlay name is written onto the start and end of the overlay itself, in each case, and subsequently checked by the loader in order to prevent errors. The overlay generators STDGEN (or STDOVR for generation on the same disk) and e.g. FPTGEN (or FPTOVR), usually on pages 0 and 112000 respectively, create the overlays from a series of feedstock files loaded at 100000. The latter are called STCOD1, STCOD2 ... for the standard overlay feedstock, or are given names made up of the first two letters of the programme name suffixed by “-COD" and a number (e.g. FPCOD1, FPCOD2, etc for FPTIME). They are constructed by putting together a selection of standard 1K modules, including those listed below.

Standard overlays. The following modules are used in common by most of the programmes in the Library –

- PFILE : contains the routines for file storage and loading, auto-incrementation of file-names, construction of file-headers, and so on. It requires the presence of a loading linker on page 114000. (Usually generated as STLAYA).
- CALADC : carries the routines for calibration of the digitizer, as called by the CD command. It also holds the double (40-bit) and triple-precision (60-bit) routines used for fast averaging over the abscissa and ordinate under the AH and AV commands. (Usually STLAYB).
- LABCOM : incorporates: the re-structured LAB-80 commands (that combine two or more of the original ones, e.g. TM replacing PT, NT, AR); the code to implement initial entry to the programme (types the programme name, plus an HE help message, and asks for the date); and the text of the command summary of the standard LAB-80 commands. (Usually STLAYC).
- PBOX : is mainly concerned with standard plotter facilities. It contains routines for: plotting axes; drawing a box around the plotter area and adding ticks to it; identifying the plotter

ranges; and calibrating the plotter and CRT display. The latter facility has been moved from page 0, where Nicolet placed it, in order to make space for the EM emergency recovery routine. (Usually STLAYD).

- PCRASH : carries the collection of routines accessed by the DR data-recovery command. It facilitates the recovery of data from a corrupted disk following a bad software “crash". It also allows for a file that has been deleted in error to be restored to health again. (Usually STLAYE).
- SETLEV : provides the facilities called by the SL level-setting command. It also includes an externally callable arrow display for the identification of levels or distributions. (Usually STLAYF).
- HEDPRN : contains the routines for interpreting and printing out the information in the first (standard) section of a data file-header, and the code for implementing the O command to reset the data-processor parameters to their original values. (Usually STLAYG).
- HEDFIN : finishes off the standard header printing. It must follow in sequence immediately after HEDPRN. (Usually STLAYH).
- LSFITS : holds the routines needed for least-squares fittings (straight lines, exponentials, Gaussians, quadratics, Lorentzians). The various summations, and all of the final results, are placed in absolute addresses on page 116000. (Usually STLAYI).
- XYDISP : carries the routines for displaying 2-D cross-sections through 3-D distributions, and for displaying 1-D slices with cursors. (Usually STLAYJ).
- XYPLOT : holds the contour-plotting routines for 3-D distributions, both for closed and open (terminating at a boundary) contours. A 64×64 matrix is assumed. (Usually STLAYK).
- PHASE : is used in particular by the PHASEP programme. It computes phases relative to a trigger and phase differences, for sinusoidal signals at the ADC inputs.
- XYZPLT : is a 3K self-contained overlay system for three-dimensional plots of data from XYDENS, XTDENS, PREHST and APDENS. (Usually STLAYL, M, N).
- DPAVG : contains: the routines for double-precision data acquisition; the comparison routines for ADC inputs A and B, computing the amplitude-equalising factor and offsets; and the routine for finding the number of points per quasimonochromatic noise (QMN) oscillation. The module loads initially at 100000 as a standard overlay, but immediately relocates itself to 2000 so that data memory can be used for output. (Usually STLAYO).
- COMCOM : contains a variety of code moved out of the LAB80 root. It loads initially at 100000 as a standard overlay, but relocates and runs on page 2000. All initially unrecognised commands entered in the LAB80 main command block come here for checking, which is why there is a slight pause before the “??" expostulation when a mistake is made. (Usually STLAYP).
- PPLOT : carries routines for plotting the contents of one $x(t)$ block against the contents of the adjacent one; and for a CRT display of the same kind. (Usually STLAYQ).
- EVENT : provides the facility for capturing and ensemble-averaging rare events, such as noise-induced explosions, under the XP command. It loads as a standard overlay, but runs on page 2000. (Usually STLAYR).

There are also two pieces of code that are used during assembly for the creation of programme-specific overlay feedstock files -

- LSCALC : only occupies part of an overlay (100000 - 100377). It provides the comparison displays for fitted curves, data, and cursors. The code for numerical readout of the physical quantities corresponding to cursor positions and contents, positions and height of maxima, peak-widths etc is patched in separately by each individual programme to complete the overlay. (Usually e.g. FPLAYB).
- PLOTS : likewise only fills part of an overlay. It provides the plotter command block and interface to PBOX, XYPLOT and XYZPLT (see above). The code for calculation of

numerical quantities, and for plotting the actual distribution, is patched in by each of the individual programmes. (Usually e.g. FPLAYD).

APPENDIX B
General Utilities for the NIC-80

Some general utility programmes for the NIC-80 are described briefly below. LINKER and KERMIT are described separately, in Appendices C and D respectively

CLEAR may be used to zero the entire memory, apart from the disk monitor head 7600-7777. It enables code to be patched and loaded onto areas that are free of extraneous material, during programme construction. The storage limits of the programme are: CLEAR 0-27; 0. It may be used on both floppy[7] (NIC-298) and Diablo[10] (NIC-294) disk systems. Following a *RUN CLEAR command, the programme first re-locates part of itself to the scratch loading area (7760-7774) of the monitor head. The latter section of the programme then clears 0-7577 and 100000-117777 and exits to the disk monitor.

CHECK is used to test the validity of data and programmes stored on floppy (NIC-298) or Diablo (NIC-294) Nicolet NIC-80 disk systems. To run the test, enter *RUN CHECK (return). The programme allows an optional disk change; following a return, it then loads in sequence all of the files stored on the disk, printing their names as it does so. If any checksum errors are detected, these are reported. If there are no errors on any of the files, CHECK will print “ALL OK" before finally exiting to the disk monitor. The programme occupies the bottom 1K of the NIC-80’s memory, with storage limits: CHECK 0-1777; 0. It incorporates within itself both NICBUG[1] and the standard set of spare bootstraps (see below instructions for DIABLO, FLOPPY and BOOTS) re-located from page 114000. In operation, it uses all available data memory (100000-117777 in a 12K system), and so will obliterate the standard bootstraps that may have been left by a previously run programme on page 114000. For emergencies (e.g. failure to raise the monitor by starting at 7600, due to corruption of the monitor head), there is a hidden command block after the initial “INSERT DISK FOR VERIFICATION MESSAGE", which gets printed when the system is started at address zero (e.g. by pressing the data-processor start button). The following commands are then (secretly) available:

B	–	call NICBUG, the standard octal debugging programme
I	–	identify version of CHECK
M	–	read a new monitor head from the disk in use
N	–	read in the NICOLODEON paper tape on the HSR
↑Q	–	CTRL/Q quit to the disk monitor

If the initial message is not printed, it may still be possible to access the bootstraps by use of the switch register. Start the processor at 1367, 1414 or 1436 to read a floppy or Diablo monitor head, or read in the NICOLODEON paper tape respectively.

DIMON links together and runs automatically all of the diagnostics programmes listed in the NIC-80 service manual.[39] That is, it runs it turn without operator intervention: the Shift Test, Jump Test, Operations Test, Worst Pattern Generator Memory Test, and the relevant (floppy[7] or Diablo[10]) Disk System Exerciser Test. When, as often happens, the operator is uncertain whether there is a really problem or not, but just seeks reassurance, Dimon saves a huge amount of time. If any errors are found, they are reported, and the operator can intervene at that point.

PNED is a supplementary text editor that provides facilities for extraction of text as disk files, or for the insertion of text from files, during editing. These possibilities are not provided by the standard Nicolet text editor[42] NED in either its floppy (FNED) or Diablo (DNED) versions.

On first entry, the programme calls the Disk Command Interpreter (DCI),[7,10] which prints a “@", and awaits instructions. The main command line should then either be of the form
@FILE6.A/FILE7.A:I
for insert mode; or of the form
@FILE6.A:E
for extract mode. In the former case, FILE6.A is to be modified by the insertion of additional lines of text in the form of one or more disk files; the edited result will be called FILE7.A. In the latter case, FILE6.A is not edited, but selected lines of text are to be extracted and stored as separate disk files.

In insert mode, PNED prints the text line that is currently being pointed at (line 1, on first entry), for information. It then asks for the line number above which text is to be inserted; and for the filename of the insertion, which should be entered after the usual DCI prompt @. The process then repeats, for as many text insertions as are required. When editing is complete, a CTRL/Q should be entered, at which point the edited text will be stored on disk under the output filename specified in the main command line. Note, first, that the line numbers of the text insertion points should have been identified prior to entry by use of DNED or FNED in the usual way. Secondly, insertions must be made sequentially, in line order; PNED always moves forwards through the text, and cannot go backwards. All line numbers refer to the *original* input file (FILE6.A above) and take no account of insertions that have been made.

In extract mode, PNED will again print the current text line and await instructions, in this case for details of the text to be extracted: for the initial and final line numbers; and for the filename under which the extracted text is to be stored on disk. The latter is an output file, so that its name must be preceded by a slash, in accord with the usual DCI convention (e.g. “/EXTRCT.A"). The process then repeats for as many text extractions as are needed. Finally, entry of a CTRL/Q will exit PNED and restore control to the disk monitor. Note that, as with insert mode, extractions must be made sequentially in line order and the relevant line numbers must have been identified using DNED or FNED prior to entry.

FBAKUP enables the copying of whole disks on single-floppy-disk[7] NIC-298 systems for 12K NIC-80s. The storage limits are FBAKUP 0-1074; 0.

Following a *RUN FBAKUP command, the programme instructs the operator to transfer disks as required. All disk swaps are completed with a carriage return. If the source disk is completely filled, seventeen disk swap cycles will be required, but proportionally fewer cycles will be needed for partly filled disks. If the wrong disk is inadvertently inserted at any stage, an error message is given.

When the destination disk is first inserted, its directory is immediately killed in order to avoid misleading results occurring if the transfer is terminated prior to completion. For the same reason, the directory of the source disk is the last item to be transferred, after all the data and other files have been written to the destination disk. Any checksum errors detected when reading the source disk are reported, but the transfer is nonetheless be completed. On completion of all the transfers, the programme asks whether or not the destination disk is to be verified. If the operator answers with a Y, each file will be loaded in turn, and all checksum errors will be reported. If there are none, the programme will print “ALL OK" before control returns to the disk monitor. Files that were already corrupt from the source disk, which have already been reported as such during their transfer to the destination disk, will not be reported again; the verify option simply tests the validity of the storage process on the destination disk. If the operator enters an N, the disk will not be verified and, instead, control will be returned to the disk monitor immediately.

DSKGEN provides for automatic initialisation of floppy disks by linking together the sequence of routines provided by Nicolet. It saved us a lot of time, as the original sequence of operations (see pp 3-4 and 60 of the NIC298 manual[7]) was quite onerous and time-consuming for the user.

A sequence of floppy disks (e.g. a new box of 10) can be initialised with a *RUN DSKGEN, after which the programme prompts for disk changes, as needed. If errors are detected when a disk is being tested, they are reported and the user is given the choice of re-testing that disk immediately, or proceeding to initialise the next one. At the end of the procedure, each disk has stored upon it SYSGEN, WARLK2, CHECK and DIABLO; WARLK2 will already have been run, so that there will also be a backup directory ///DIR at the end of the data area, stored on the final track.

Because DSKGEN has to incorporate within itself: FORMAT,[7] DEXER,[7] NICBUG,[1] SYSGEN,[7] WARLK2,[43] and CHECK (see above), it uses the whole 12K of the NIC-80's memory, with WARLK2 initially having to be stored in two separated sections. The code is swapped around as necessary in memory as initialisation proceeds. Initially, the programme page layout is -

0-577	run area available for FORMAT and DEXER
600-7577	upper section of WARLK2
7600-7777	disk monitor head
100000-101777	DSKGEN loading linker, DEXER work area
102000-103777	DGNGEN generating linker, DEXER work area
104000-104577	lower section of WARLK2
104632-105365	NICBUG
105366-105376	spare bootstraps for NIC-298, NIC-294, HSR NICOLODEON
106000-107777	DSKGEN programme code
110000-117577	SYSGEN, incorporating DEXER and FORMAT

There are two programme-linkers. DSKGEN is used to run the programme from a library disk; DGNGEN is used to move the whole system from one library disk to another. The linkers in question are created by RLINK, an earlier version of LINKER. The filenames and storage limits of the DSKGEN system are:

DSKGN1 – 600-7577; 7600
DSKGEN – 100000-100617; 100000
DGNGEN – 102000-102617; 102000
DSKGN2 – 104000-117577; 106000

The DSKGEN sequence contains a number of pauses, during which the processor *appears* to be "dead" for several seconds at a time. These are unnerving for an operator using the programme for the first time, but are necessary to allow time for the head load pad to disengage.

DIABLO, FLOPPY and BOOTS. The programmes DIABLO and FLOPPY are for use on dual floppy/Diablo systems – a configuration that seems not to have been envisaged by Nicolet, but which we found very convenient and efficient: running acquisition/analysis programmes from the Diablo resulted in nearly-instantaneous overlay loading, so was ideal for the heavily-overlaid Stochasics Library programmes; but storage of results and data was better done on floppies which could be stored much more conveniently and which were less prone to crash. DIABLO and FLOPPY enable the disk being addressed to be changed. Thus, a *RUN DIABLO to the floppy monitor will transfer attention to the Diablo; and *RUN FLOPPY to the Diablo monitor will have the reverse effect.

Both programmes are manifestations of a short piece of code called BOOTS (115366 - 115476) which is built for crash recovery purposes into all of the stochastics library software, and into many of the other Lancaster programmes for the NIC-80. They are identical except for having different starting addresses, the storage limits being:

FLOPPY: 115366 - 115476; 115367

DIABLO: 115366 - 115476; 115414

Also included in BOOTS, and hence in DIABLO and FLOPPY as well, is the NICOLODEON bootstrap[1] for the high speed paper tape reader. Where BOOTS is incorporated into a larger programme, the bootstraps may be accessed by subroutine calls to addresses:

115366 – to read a floppy monitor head

115413 – to read a Diablo monitor head

115435 – to read in the NICOLODEON tape on the HSR

Alternatively, the processor may be started at address 115367, 115414 or 115436, respectively, by use of the switch register. Because the bootstraps (unlike the abbreviated versions in the Nicolet manuals) are unmodified by use, they can be accessed as many times as necessary, on a succession of different disks if need be.

In a few cases, BOOTS was re-assembled on a different page of memory for particular programmes, but using the same page-relative address: for example, in DSKGEN BOOTS starts at 105366; and in CHECK at 1366. The start addresses of the individual bootstraps are similarly modified.

FDMOVE is a utility for moving files between NIC-298 (floppy[7]) and NIC-294 (Diablo[10]) disk drives on hybrid 12K NIC-80 systems. The storage limits are 110000-117777; 110000. The programme can be started with a *RUN FDMOVE command from either the floppy or the Diablo disk monitor. Outline operating instructions are contained within the programme itself and can be printed with a CTRL/H.

After the “@" prompt, the name of the file to be transferred is entered, immediately followed by an “.F" or “.D" option, and closed by a carriage return. The option character is used to select the *source disk* , either floppy or Diablo respectively, from which the file is to be read, it will, of course, then be transferred to the other disk. (The FDMOVE syntax necessarily deviates from the standard Nicolet convention of “-D1", “-D2", etc, because “-D1" would refer ambiguously both to the default floppy and to the default Diablo disks). After the first file has been transferred, the system will prompt for the filename of the next one, and so on.

Transfers can be of single, named, files; or of groups of files whose names match a mask. In the latter case, a “$*$" can be used to represent any first or last triad of characters within the (6 character) filename, and a “?" can be used to represent any individual character. To transfer only the core image files, a “$**$", or a “??????", may be entered; and so on. If the destination disk runs out of space, the system will say so and wait for another disk to be inserted.

Exit from the programme may be effected with a “CTRL/D", “CTRL/F" or “CTRL/Q", which cause jumps to the Diablo, floppy or original disk monitors respectively. These three control characters are read and acted on at any of the points where the programme is awaiting keyboard entry.

FDMOVE occupies 110000-117777 and starts at 110000. On first entry, it attempts to read (160 word) monitor heads from both disks, and it places these at 117400 and 117600 respectively. These heads can then be swapped very rapidly into their run positions at 7600, whenever required. Only if both heads have been read successfully, without checksum errors, does the system proceed to the main body of the programme; otherwise it will either “hang", or it will exit immediately to the monitor with an appropriate error message.

The directory of the source disk is read and held at 0-1777 (floppy), or at 0-2777 (Diablo); the usual 4000-7577 (floppy) or 3000-7577 (Diablo) memory blocks are used for DIRFUN and for the directory of the destination disk. The loading area for the files to be read to and written from is 100000-105777 corresponding to 3 floppy disk records (of 2000 words) or 2 Diablo disk records (3000 words). Where files are longer than the loading area, they are normally transferred in two or more separate sections of up to 6000 words and stored on sequential records

on the destination disk. Core image or assembler binary files are treated in this straightforward manner; ASCII files, however, require special treatment because of their record-related structure.

Source files prepared by the disk editors (e.g. by FNED or DNED) consist of continuous strings of 8-bit ASCII characters, packed at five characters per 20-bit word (PACASC). The character strings end at the end of a line (i.e. at a CR), a few words before the end of each record, and are terminated with a series of form-feed (214) characters, so as to complete the current pair of words. Because of the mismatch in size between the Diablo (3000 word) and floppy disk (2000 word) records, an ASCII file cannot simply be transferred to a space of the same size on the other disk: the record terminators would obviously end up in the wrong places. In transferring ASCII text files, therefore, the PACASC is unpacked to standard 8-bit characters, the existing record terminators are removed, and the text is re-packed at the required buffer size with new terminators being added at the end of each record. The total number of records occupied will, of course, be different on floppy and Diablo disks.

PNICL is the NIC-80 version of a communications protocol that enables files to be passed between Nicolet data-processors. It is fully compatible with FILMOV on the NIC-1180 and NIC-1280, and will configure itself for either floppy (NIC-298) or Diablo (NIC-294) disk drives used with the NIC-80.

This utility was kindly provided (sourcecode) by Don Parker of Nicolet in 1985 as a “development programme" (i.e. an unfinished version), NICL. After the correction of a couple of bugs, it was found to work well; the corrected version is called PNICL to distinguish it from the original.

The data link passes between the RS232 B-channels of the two data-processors. These two sockets should therefore be joined, *crossing over the transmit and receive wires*, and making sure that the B-channel baud rate is the same on both machines. Following a *RUN PNICL command, the programme will announce itself and ask whether the user wishes to receive (R) or transmit (T) a file.

If receiving a file, PNICL will ask for a filename or an immediate return: in the latter case, the received file will retain its original name; in the former case, it will be stored on disk under the new name provided by the user. Following a return, the system will wait to receive the promised file. While waiting fruitlessly, a “!" will be printed periodically; whenever a packet (of 255 bytes) is received successfully a “." is printed. A “?" indicates a corrupted packet and a “#" a re-transmission.

When transmitting a file, PNICL will ask for the file or group name. Either one file may be named, or a mask may be entered with a “∗" to represent the first or last (if a 6 character name) triad of characters, or a “?" to replace any individual character. All files matching the mask will then be transmitted. A “." will be printed each time a transmitted packet is acknowledged as having been successfully received. A “∗" indicates that the last packet was corrupted, and a “#" a retransmission.

When the file transmission or reception is complete, PNICL will return to its initial “RECEIVE (R) or TRANSMIT (T)" query and wait for further work. More file transfers may be implemented, or a CTRL/Q may be entered to exit PNICL and restore control to the disk monitor. One or more CTRL/Qs may also be used to abort a file transfer session at any time.

APPENDIX C
LINKER for NIC-80 programmes

LINKER is a programme-linker for NIC-80s with either hard or floppy disk systems. It enables groups of programmes to be stored, loaded and run as though they consisted of single files. There is also an option to facilitate the generation of overlay programmes.

Principle of operation. LINKER functions by generating a short linking programme which is automatically stored on disk under a name provided by the operator. Whenever this linker is subsequently called into core and started, it immediately loads all the separate sections of the programme which is being consolidated. Then: either the programme itself is started; or, alternatively, the programme (plus linker) is first stored automatically on a new disk followed by overlay-generation, if required, and then started. Because the linker is always called from disk immediately prior to being run, no disadvantage need arise from its being positioned in a region of memory where it will subsequently be overlaid by data when the consolidated programme is in use (but see also applications note 4, below).

The programme will adjust itself automatically at configuration time for either NIC-294 (Diablo)[10] or NIC-298 (floppy)[7] disk systems. It can be run on any pair of pages of memory outside the monitor swap area provided only that it is always loaded and started on a page boundary.

Loading and initial storage of LINKER from paper tape. LINKER should be loaded into core from the binary paper tape in the usual way, using the *BIN command. After raising the monitor the programme may then be stored by entering *STO LINKER 100000-103077; 100000.

Starting LINKER. If the linker is subsequently to be operated on page 100000, LINKER may be started immediately by means of a *GO 100000 or *RUN LINKER command. If, however, the linker is required on a different page of memory, it will be convenient to commence by relocating LINKER to the page in question. For example, a linker could be created on page 112000 by entering the commands *LOA LINKER 112000 followed by *GO 112000.

Configuration procedure. Immediately on being started, LINKER will call in the Disk Command Interpreter (DCI)[7,10] to allow configuration with the names of up to five files which are to be linked. Following the @ prompt, the user should enter: a slash; followed by a name chosen to represent the composite programme; followed by the names of the files to be linked; followed by an option character (D, G, L, 0 or T: see below). The filenames should all be separated by commas without any embedded spaces. For example, if three files ROTON1, ROTON2 and ROTON3 are to be loaded together, effectively forming a composite file to be called ROTON, the command line would be

@/ROTON,ROTON1,ROTON2,ROTON3:L [return]

The configured front section of LINKER (632 octal words for the D, L and T options; or 1716 words for the G and O options) will then be stored on disk under the selected name, i.e. ROTON in the above example. At any time subsequently, the DEMON command *RUN ROTON will cause the three linked files (which must, of course, already be resident on the disk) to be loaded into their usual positions in core. What happens next is dependent on the option that has been chosen.

The loader (L) transfer (T) and dual (D) options. The linkers made with these options are designed to be short enough to coexist on the same page of memory as NICBUG (which starts at relative address 632). If the L (loader) option was chosen, the processor will be started at the start address of ROTON1. If the T (transfer) option was chosen the programme will type SWAP DISKS. A carriage return will then cause the composite programme to be stored automatically on the new disk (as separate core-image files ROTON, ROTON1, ROTON2 AND ROTON3). Any pre-existing files with the same names will be deleted. The processor will be started as before at the starting address of ROTON1. If the D (dual role) option was chosen,

the programme will print the query TRANSFER OR RUN (T or R): and the user can respond with a T or R according to choice.

The G (overlay generation) option. The G option enables the linker to generate and store a series of overlays from the code placed in one or more overlay feedstock files. When this option is used, the programme assumes that the second filename configured by the DCI is the overlay feedstock file and, furthermore, that any additional feedstock files will have the same name with its final (numeric) character incremented. (With the G option, because of this incrementation feature, the second filename must have exactly six characters; the other filenames can have fewer characters). A suitable command line for the G option would therefore be
@/ROTGEN,ROCOD1,ROTON2,ROTON1:G [return]
where: ROTOGEN is the name of the linker to be created; ROCOD1, ROCOD2, ROCOD3 ... are the names of the feedstock files; ROTON1 and ROTON2 are the root sections of the programme.

In the above example, the ROTGEN linker will first look up and store the directory data of all the filenames configured (including the data for ROCOD2, ROCOD3 ... which have been implicitly specified by choice of ROCOD1 as the second name entered). The files ROCOD1, ROTON1 and ROTON2 are then loaded and a disk swap requested. When a new disk has been inserted and a carriage return entered, its directory is examined and entries made for all of the files and overlays that are to be stored. The explicitly specified files (ROTGEN, ROCOD1, ROTON1, ROTON2) are then written to the disk; ROCOD1 is broken up into separate 1 K overlays and these, too, are written into the spaces that have been prepared for them. A disk swap is requested, and ROCOD2 is dealt with in a similar manner; and so on, until all of the feedstock files have been both stored and broken into overlays .

In selecting names for the overlay sequence, LINKER uses the first two characters of the first configured filename sufficed by -LAYA, -LAYB etc. In the above example, the overlays would therefore be named ROLAYA, ROLAYB, ROLAYC... Up to a maximum of 26 overlays can be created in this way. Before being stored, each overlay is “tagged” with its name and a 770000 UNPCK terminator deposited on both its first and its last three words.

After the files and overlays have all been stored, the processor is started at the starting address given to the second file (ROCOD1 is the example above) to be configured.

The storage data for the files in the above example, with the ROTGEN generator-linker configured on page 112000 with
*LOA LINKER 112000
*GO 112000
followed by the DCI command line as already specified, might therefore have been:
ROTGEN (automatically stored by LINKER as 112000-113716; 112000)
ROCOD1 100000-111777; 0
ROCOD2 100000-111777; 0
ROTON1 0-7577; 0
ROTON2 114632-117777; 0
ROTON (automatically stored by LINKER as 114000-114631; 114000)

With these directory data, there would be a total of 10 overlays called ROLAYA-ROLAYJ, and the processor would be started at 0; ROTON would be a loader-linker previously configured on page 114000 with @/ROTON,ROTON1,ROTON2:L and used to RUN the programme on subsequent occasions from the disk in question.

The O (overlay generation only) option. The O option creates a linker that will load the files and create a sequence of overlays on the same disk. Its effect is identical with that of the G option except that no disk swap(s) is (are) requested, and no files are stored on the disk except

the overlays. The processor is then started at the starting address of the second filename, as before. Linkers made with the O option are intended for use on dual drives where all of the main files and feedstock files can be transferred quickly (without needing to do disk swaps), but where it is still necessary to generate the overlays before the programme in question can be run.

General application notes. Note that –

1. LINKER operates in conjunction with single-disk systems, or with Unit I in dual-disk systems.

2. The linkers created by LINKER are themselves also fully relocatable to run at the start of any page outside the monitor swap area.

3. Because LINKER links *only the names* of the files in question, and does not modify them in any way, they can still be loaded individually by use of DEMON, if desired. The necessary directory data is always read at run time. Thus, the files themselves can be modified at will without any consequent necessity to create new linkers.

4. With the D, L and T options, and where there is sufficient memory available to retain the linker in core after use, while the consolidated programme is in action, it can sometimes be convenient to add a “self-storing" transfer command to the command lists of the relevant applications software, arranged so as to cause a jump to 100045 (or equivalent address if the linker is operating on a different page), thus entering at the “SWAP DISKS” stage and moving the (configured) programme to a new disk. If a virgin copy of the programme is to be transferred then the jump should, of course, be to 100000 (or equivalent address). (In the case of the G option, it will usually be necessary to reload and start the linker/generator, in order to move the overlay feedstock files etc to the new disk.)

APPENDIX D
KERMIT for the NIC-80

Introduction. The NIC-80 KERMIT provides a basic implementation of the packet-exchange protocol described by Frank da Cruz.[44] It is applicable to NIC-80s with 12K of memory and either a floppy[7] (NIC-298, NIC-299) or hard[10] (NIC-294/Diablo 30) disk drive. KERMIT can be used to transfer programmes and data between NIC80s and external computers, using the RS232 B-channel. It uses standard sized packets; and innovations such as sliding windows, repeat-prefixing, or 8th-bit-prefixing have not been implemented.

Command summary Once the programme has been installed and started with a *RUN KERMIT, the following commands are available –

B - call NIGBUG (octal debugging programme)
C - connect to external system (dumb terminal mode)
    ↑G - return to command block
D - do a directory listing
E - set external echo (for a Unix machine)
H - help: print command summary
I - toggle indicator symbol printing and packet archiving on/off
K - kill (switch off) external echo
L - login on remote system
M - return to DEMON monitor (same as Q)

N - print notes on operation
P - set KERMIT parameters (future implementation)
Q - quit, return to DEMON monitor (same as M)
R - receive programmes or data from external system and file them on disk
    ↑G - unconditional return to command block
    ↑X - abort transfer of current file
    ↑Z - abort transfer of file series
    CR - retransmit last packet, and continue
S - send programmes or data to external system and
    file it on disk (with ↑G, ↑X, ↑Z and CR as for R command)
T - toggle RS232 B-input off/on
U - configure username and password (for L command)

Details of individual commands are given below.

Data formats. Files sent from the Nicolet are always sent as text files to the external system, but in two different formats, depending on whether they start off as text files or core-image files.

Text files, which have an .A extension to the filename, are held on the NIC-80 in the form of packed 8-bit ASCII, with five characters per pair of 20-bit Nicolet words. Before transmission to the external system they are unscrambled, and then sent as a sequence of 7-bit ASCII characters (i.e. with Nicolet's bit-7 removed) either in the usual Nicolet upper-case, or in lower-case, as appropriate. The file will therefore appear on the external system in readable form and, if necessary, can be edited there before being returned to the same (or a different) Nicolet.

Non-text (core-image) files are sent to the external system in the form of decimal integers, one per 20-bit word, separated by commas, with a carriage-return/line-feed after every eighth number. This is a convenient format for FORTRAN, and is readily converted to whatever other format may be needed by any particular plotting package. Thus, core-image files such as specta and other forms of averaged data can be readily be analysed, manipulated, edited and plotted on the external system. Nicolet programmes are also core-image files, but obviously mean nothing outside the Nicolet. Nonetheless, they can be transmitted, held on an external system, and sent to a different Nicolet or returned to the original one, where they are reproduced in identical form as 20-bit integers.

All files sent to an external system, either text or integer, have a first line added consisting of three decimal integers. These carry relevant directory information, and represent respectively: the length of the file, in 20-bit words; the load address in the Nicolet; and the programme starting address. One or more of these numbers will be used by the receiving Nicolet to indicate progress and to make an appropriate directory entry once the file has been stored. Where, for some reason, a non-Nicolet file (or one not previously transferred by KERMIT) is to be sent to a Nicolet, a first line containing three appropriate numbers separated by commas *should be edited into the start of the file before it is sent*.

Conversion of filenames. Filenames on the NIC-80 are held as packed 6-bit ASCII, with a maximum of six characters before the extension. The names are converted and transmitted to the external system as 7-bit ASCII, in lower case to conform with unix conventions. When a file is received by the NIC-80, its name is converted to the packed 6-bit format. Characters after the sixth are ignored. So also are any points (periods) unless these are followed by an upper-case or lower-case “A", “B" or “C"; which are treated as filename extensions and converted to upper-case if they are not that already. An “A" extension implies that the incoming file will be treated as a text file. It will be converted and stored in standard form as packed 8-bit ASCII, which can subsequently be read or edited using one of the NIC-80 editors (e.g. NED or FNED).

Dumb terminal mode (C, ↑G) Following the C (connect) command, the system enters its dumb terminal mode. Any characters entered at the keyboard are transmitted over the RS232 B-channel to the external system; any incoming characters received on the B-channel are sent to the terminal and printed or displayed on the VDU screen. Thus it is straightforward to login to the remote system, in the usual way, if it is e.g. a mainframe or workstation. Once the remote system has been prepared for receiving or sending a file (see sections 5 and 6 below), control can be returned to the KERMIT command block by entry of a ↑G.

In dumb terminal mode, conventional (XON/XOFF) flow-control is operative. Thus the incoming data stream can be stopped by entry of a ↑S (assuming that the remote system follows the same protocol), and restarted again with a ↑Q. Likewise, the remote system can interrupt and restart the data stream from the NIC-80.

In practice, large quantities of data (including login messages etc) often arrive very fast immediately after logging in to a mainframe. To avoid loss of characters caused by buffer-overflow in the Nicolet under such conditions, it is important that the baud-rate setting of channel-A should be equal to or higher than that of channel-B: see also Appendix E.

Sending files to the external system (S). To send a file to an external system the usual procedure is as follows. First login, using dumb terminal mode as described above. Activate the remote Kermit and set it into receive mode, e.g. by typing “receive". Return to the NIC-80 KERMIT with a ↑G, and enter the S send command. The system will ask for the name of the file to be sent, which should be entered at the @ prompt. If the user does not respond to the prompt, the system will eventually time-out and jump back to the command block. The same is true at all points where user-input is awaited *except* of course in the command block itself.

Once the two Kermits have made contact, and transfer of the file has started, progress can be monitored by watching the arrow on the CRT display as it advances from left to right across the screen. In the case of text files, which are always filed as complete disk records but usually only occupy part of them, progress of the arrow will tend to underestimate the proportion of the file that has been transferred; the transfer progress of core-image files will be indicated precisely.

On completion of the file transfer, the system will ask whether any more files are to be sent. If the user answers with a Y, the @ prompt will appear for entry of the next filename; and if he or she answers with an N, control will return to the KERMIT command block. Multiple file-transfers can often be effected conveniently by means of a joker/wildcard convention. One or more “?"s can be used to replace one or more characters when entering the filename and/or extension (and the usual Nicolet “∗" can also be used in place of the first or second triad of characters). All files whose names match the corresponding mask, where the “?" is assumed to represent any character, will then be sent to the external system.

If necessary, transfer of the current file can be aborted with a ↑X, and transfer of a file series can be aborted with a ↑Z, provided of course that the external system supports these facilities. Alternatively, a ↑G, can be entered to force an unconditional return to the command block. If the packet exchange somehow becomes stuck during file transfer, entry of a carriage return will cause the last outgoing packet to be re-sent, and may possibly get the transfer started again.

Receiving files from the external system (R). To receive a file, after one has logged in and activated the remote Kermit on, the normal procedure is as follows. Place the remote Kermit in send mode, e.g. with command “send filename" where filename is the name of the file that is to be sent to the Nicolet, return to the local KERMIT with a ↑G, and enter the R receive command. Once the transfer has started, the progress arrow will be seen advancing across the screen as described above for the S command; the arrow will be fainter, however, because it is displayed only during sending packets which, for receive mode, are relatively short acknowledgements of data packets received.

If a file with the same name as the incoming one already exists on the Nicolet's disk, the usual "DELETE:" query will be printed and must be answered with a Y if the original file is to be overlaid. Multiple transfers of incoming files can be performed provided that the remote Kermit supports this facility.

The ↑Z, ↑G and CR commands are available for aborting or attempting restart a file transfer, as described in the previous section.

File transfer between two Nicolets To effect direct file transfer by KERMIT between two NIC-80s, a slightly different procedure is used. KERMIT is run on both machines, giving the commands locally in each case. One Nicolet is put into receive mode with an R, and the S command is then used to send files from the other one. Alternatively, it is equally easy, and somewhat faster, to transfer files between any pair of Nicolet systems (NIC-80, NIC-1180 or NIC1180E) by means of PNICL (see Appendix B).

Diagnostic facilities (B, I). If KERMIT seems not to be functioning properly in relation to a particular external system, some simple facilities are available to help identify the problem. Entry of a B invokes the standard NICBUG NIC-80 octal debugging programme, with its usual facilities for placing breakpoints and for examining and modifying the contents of specified addresses. Entry of an I toggles on (or off) the packet-indicator printing facility and also causes the last few packets sent/received to be preserved for examination using NICBUG. The significances of the packet indicator symbols are as follows -

ℓ	–	the packet received has the wrong length specified
s	–	a packet has been sent
S	–	the same packet has been re-sent
r	–	a good (uncorrupted) packet has been received
w	–	the packet received has the wrong sequence number
ℓ	–	the packet received has the wrong length specified
c	–	a checksum error in the received packet
n	–	a NAK (negative acknowledgement) has arrived
N	–	a NAK to the *next* packet has arrived
t	–	receiver has timed-out while awaiting a packet
T	–	a time-out occurred during reception of a packet

Because of the order in which KERMIT performs validity checks of an incoming packet, not all relevant symbols will be printed: only one error is noted for each invalid/corrupted packet. Thus, if the same packet arrives a second time, only a w will be printed showing that the sequence number is wrong, regardless of whether or not the length and checksum of the packet are correct; this will normally be followed by an S to indicate retransmission of the last packet.

With the I facility toggled on, the previous four outgoing packets are preserved at octal 200 intervals in addresses 102000-102777. The previous four incoming packets are preserved within 103000-103777. They can be inspected by invocation of NICBUG under the B command as described above, and it is then usually fairly obvious where the problem lies.

Help commands (H, N). A command summary can be obtained by entry of an H, and some succinct notes on operation are printed after an N. The latter are intended mainly to help anybody who has acquired the NIC-80 KERMIT programme, but does not have a copy of these instructions.

Convenience commands (L, U, E, K). There are four commands that simply reproduce, automatically, what the user could otherwise do manually in dumb terminal mode (see above). They are particular to the system at Lancaster University, but could readily be modified for use under other conditions. The L command effects an automatic login, once KERMIT has been configured with the username and password by means of the U command. Note that the U command

should be entered for a *newly loaded* copy of KERMIT. This ensures that, when the configured version is re-stored (automatically) on the disk, the copy of FPP72 held initially at 102000 will be intact. There are three stages to the login process. First, KERMIT makes contact with a PAD, watching out for the "$>$" PAD prompt. Secondly, it calls up the Lancaster Sequent Symmetry, coded as lancs.cent1, and looks out for the ":" prompts corresponding to the standard Unix "Username:" and "Password:" queries, which it answers appropriately. Thirdly, it watches out for a "$>$" prompt from operating system showing that the login has been completed. The K and E commands can be used to turn off (kill) or restore the echo from the remote system, assuming that this is operating a full-duplex protocol, and that it is running under Unix.

Miscellaneous commands (M, Q, D, T). Exit to the Nicolet Demon monitor can be effected by entry of either a Q or an M. One can return to the programme again by typing GO 110000 to the DEMON $*$ prompt. Entry of a D will cause the disk directory to be printed, without needing to leave the programme. The T command is a toggle that activates or de-activates the RS232 B-input.

Hardware considerations. For optimal operation of KERMIT, a few minor changes may be needed to the NIC-80, as described in Appendix E. In particular –

- The RS232 B-channel may need to be connected up and activated.
- The NIC-80's RS232 UART chip can be operated at much higher speeds.
- The NIC-80's $\sim$ 100 ms delay on detection of carriage returns (during which all RS232 circuitry is paralysed) should be eliminated.

With these small modifications, the Lancaster NIC-80s are normally operated with 19200 baud set on their RS232 channels-A, and 9600 baud on their channels-B, and seem to be entirely reliable at these speeds.

Emergencies. The usual BOOTS (see above) are held on page 114000 so, in emergency, the processor can usually be stared at 115367 (NIC-298/299), 115414 (NIC-294/Diablo hard disk), or 115436 (NICOLODEON paper tape on HSR). Alternatively, NICBUG may still be available at 114700 – a quicker way of entering code to test the system than by using the switches.

Construction of the executable programme from the sourcecode. There are two sections of code to be assembled. The first, which provides the main part of the programme, is KERM**.A, where the ** are two numeric digits representing the development number of the version in use. The second, very short, piece of code is called BOOTS.A and provides the spare bootstraps (for the disk-drives and high-speed paper tape reader) for use in emergencies. On a NIC294 Diablo system, both KERM**.A and BOOTS.A are assembled with ASM, creating binaries KERM**.C and BOOTS.C which can then be loaded with LOADER. The sequence of commands for constructing NIC-80 KERMIT is then –

```
*RUN LOADER
@BOOTS.C:M
*STO BOOTS 115366 - 115476; 7600
```

(Once BOOTS has been assembled and stored, it of course need not be re-assembled when a future version of KERMIT is being constructed).

```
*RUN LOADER
@KERM**.C:M
*LOA FPP72 102000
*LOA NICBUG 114632
*LOA BOOTS
*STO KERMIT 102000-117777;110000
```

where FPP72 is the standard Nicolet NIC-80 floating point package[3] and NICBUG is the octal debugging programme.[1] To construct the programme on a NIC-298/299 floppy system, the procedure is identical except that the code is assembled with FASM and loaded with FLOAD.

The programme layout when KERMIT is in use as follows –

0 - 2777 I/O disk buffer (0-1777 for floppy)
3000 - 5777 disk directory (4000 - 5777 for floppy)
6000 - 7777 FPP72, DIRFUN overlays, DEMON monitor head
100000 - 100777 input buffer for dumb terminal mode
101000 - 101777 output buffer for dumb terminal mode
102000 - 102777 output packet construction
103000 - 103777 input packet construction
104000 - 105777 text for command summary, notes, etc
106000 - 107777 work routines for receive mode
110000 - 117777 command block, controlling routines
112000 - 113777 work routines for send, packet construction and deconstruction
114000 - 115777 filename parsing, NICBUG, BOOTS
116000 - 117777 disk routines, general utilities, arrow display

When first loaded, FPP72 occupies 102000 - 103577 but it is immediately moved to its operating position in 6000 - 7577 as soon as the programme starts; this arrangement allows KERMIT to be stored conveniently as a single file.

Principles of operation. The heart of the dumb terminal emulator mode lies in a routine called RSDUTY on page 110000. In effect, this simulates an interrupt system in software. It inspects, in continuous rotation, each of the four RS232 flags and takes action accordingly: there are no separate wait loops for the individual devices. The advantage of this approach is that it is almost impossible to take the NIC-80 unawares, e.g. for it to miss an incoming B-channel character while in a wait loop servicing the keyboard or printing a character. Note that, even at 9600 baud, the time taken for a character to arrive or be sent is $\sim$ 1 ms, which is a lot longer than the NIC-80 instruction time of $\sim$ 4-6 μs. Thus, rather than waiting while a character is being received/transmitted/entered/printed, it is much better for the NIC-80 to be doing other things.

The packet send/receive modes also use RSDUTY. However, when receiving, the system watches out for the arrival of a $\uparrow$A signifying the start of a packet, and then diverts the incoming data stream to the packet input buffer at 103000. The output packets are constructed and sent from the buffer at 102000. The current output packet is preserved there until it has been acknowledged, and so it can easily be re-sent if the first copy gets corrupted or lost.

Note that, although the appearance of the “@" prompt for filename entry seems at first sight to imply use of the usual DCI routine, the routine actually employed is a more sophisticated one held on page 114000, enabling the use of jokers/wildcards. It is a slightly modified version of the keyboard input routine extracted from PNICL or NICL (see above).

APPENDIX E
Minor RS232 hardware modifications to NIC-80

Activating the the RS232 B-channel Many NIC-80s were shipped with only the A-channel RS232 link operative. To activate the B channel, just complete the wiring in accordance with the existing schematics –

(a) Add the extra UART and associated chips and small components to the RS232 PCB from slot 11. If they are not already fitted, it will be found that empty positions are ready waiting for them, and it is simply a matter of soldering them in.

(b) Add the additional RS232 “D-socket” to the back of the NIC-80, a few inches above the existing A-channel socket.

(c) Connect the socket to appropriate pins on the underside of slot 11, simply paralleling the wiring of the A-channel in accordance with the schematics.

Speed-up for the NIC-80 RS232 board. Items (i)–(iv) in the information reproduced below was kindly provided at no cost by Nicolet. The upgrade described was carried out on the six Lancaster NIC-80s with complete success. thereby raised the maximum data rate from 2400 baud to 19200 baud, on both channels A and B.

(i) Modify Oscillator

1. Replace 422.4 KHz crystal with 6.7584 MHz.
2. Replace T16, T17, T18 (2N3302) with 2N5224.
3. Replace 100K resistor with 22K.
4. Replace 22K resistor with 3.3K.
5. Replace 2.2K resistor with 4.7K (this resistor is connected to T18).
6. Replace 0.1 micF capacitor with 500pF.
7. Replace 1000pF capacitor with 100pF.
8. Replace 47pF capacitor with 100pF.

(ii) Modification for 19.2K baud

1. Cut run between IC 18 pin 14 and IC 17 pin 3.
2. Piggy back new 7493 onto IC 18, soldering pins 2, 3, 5, 10 of new chip to the same pins of IC 18. Leave other pins unconnected.
3. Jumper pins 1 and 12 of new chip.
4. Jumper IC 17 pin 3 to pin 14 of new chip.
5. Jumper IC 18 pin 14 to pin 11 of new chip.
6. Jumper unconnected switch positions.
7. Connect new baud rates to switches.

– a. Pin 1 of new chip has 19.2K.
– b. Pin 9 of new chip has 9.6K.
– c. Pin 8 of new chip has 4.8K.

(iii) Optional Modification for 110 baud

1. Cut run between IC 19 pin 14 and IC 16 pin 13.
2. Piggy back new 7493 onto IC 19, soldering pin 2, 3, 5, 10 of new chip to the same pins of IC 19. Leave other pins unconnected.
3. Jumper pins 1 and 12 of new chip.
4. Jumper IC 16 pin 13 to pin 14 of new chip.
5. Jumper IC 19 pin 14 to pin 11 of new chip.
6. If this modification is not made the 110 switch position will no longer be correct.

(iv) Parts list for (i)–(iii)

1. 6.7584 MHz crystal (one).
2. 2N5224 (three).
3. Resistors: 22K, 3.3K, 4.7K (one each).
4. Capacitors: 500pF (one), 100pF (two).
5. IC 7493 (two).
6. Jumper wire.

(v) Disablement of the Teletype delay. To disable the Teletype delay (see above), proceed as follows –

(a) Cut the connection coming from pin 8 of IC 30.
(b) Connect pin 3 of IC 31 to earth.

This is not a Nicolet-approved option – but it works.

References

[1] *Programming the Nicolet 1080 Stored Program Computer: A Course in Programming for the Beginner* (revised), Nicolet Instrument Corporation, Madison, 1974.

[2] See Jack Kisslinger's web page: `http://www.versci.com/index.html`

[3] *Nicolet 1080 Series Floating Point Package – 1972*, Nicolet Instrument Corporation, Madison, 1972.

[4] *An Introduction to Nicolet BASIC* Nicolet Instrument Corporation, Madison, 1975.

[5] *Nicolet MED-80 SBASIC Software System* Nicolet Instrument Corporation, Madison, 1977.

[6] *Instruction Manual for LAB-80 General Signal Averaging Package NIC-80/S-7307*, Nicolet Instrument Corporation Madison, revised 1974.

[7] *NIC-298 Disk Programming System* (for 8" floppies), Nicolet Instrument Corporation Madison, 1975.

[8] *Modification and Maintenance Manual for NIC-298/NIC-299 Diskette Storage Unit*, Nicolet Instrument Corporation, Madison.

[9] *Shugart SA800/801 Diskette Storage Drive Service Manual*, Shugart.

[10] *DEMON/II Disk Executive Monitor for the Nicolet 294 Disk System* (for the Diablo-30 hard drive), Nicolet Instrument Corporation, Madison.

[11] *Diablo Maintenance Manual for Series-30 Disk Drive*.

[12] *Decitek 270 Series Paper Tape Reader Installation, Operation and Maintenance Manual*.

[13] *Remex Technical Manual for Tape Reader/Perforator System*.

[14] P C E Stamp, P V E McClintock and W M Fairbairn, "Possible influence of thermal rotons on vortex nucleation by negative ions in pressurized He II below 1K"; *J. Phys. C: Solid State Phys.* **12**, L589–93 (1979).

[15] R M Bowley, P V E McClintock, F E Moss and P C E Stamp “Vortex nucleation in isotopically pure superfluid He-4"; *Phys. Rev. Lett.* **44**, 161–4 (1980).

[16] G G Nancolas, P V E McClintock, F E Moss and R M Bowley, “Temperature-dependent vortex nucleation in isotopically pure superfluid He-4"; *J. Phys. C: Solid State Phys.* **14**, L681–6 (1981).

[17] G G Nancolas and P V E McClintock, “Quenching of the ion/vortex ring transition in He II by intense electric fields"; *Phys. Rev. Lett.* **48**, 1190–2 (1982).

[18] *Co-Processor Manual*, Nicolet Analytical Instruments, Madison, 1984.

[19] J Smythe, F E Moss and P V E McClintock, “Observation of a noise-induced phase transition with an analog simulator"; *Phys. Rev. Lett.* **51**, 1062–64 (1983).

[20] F E Moss, P V E McClintock and W Horsthemke, “Moss et al respond"; *Phys. Rev. Lett.* **54**, 606 (1985).

[21] *Noise in Nonlinear Dynamical Systems:* vol 1 *Theory of Continuous Fokker-Planck Systems*, pp 384; vol 2 *Theory of Noise-Induced Processes in Special Applications*, pp 416; vol 3 *Experiments and Simulations*, pp 296; ed. Frank Moss and P V E McClintock, Cambridge University Press, 1989.

[22] P V E McClintock, *1991 Stochastics Library*, Lancaster, 1991.

[23] F E Moss and P V E McClintock, “Measurements of two-dimensional densities for a bistable device driven by coloured noise"; *Z. Phys. B: Condensed Matter* **61**, 381–86 (1985).

[24] R Mannella, F Moss and P V E McClintock, “Postponed bifurcations of a ring-laser model with a swept parameter and additive coloured noise"; *Phys. Rev. A* **35**, 2560–66 (1987).

[25] F Moss, P Hanggi, R Mannella and P V E McClintock, “Stochastic phase portraits of a damped bistable oscillator driven by coloured noise"; *Phys. Rev. A* **33**, 4459–61 (1986).

[26] N G Stocks, R Mannella and P V E McClintock, “Influence of random fluctuations on delayed bifurcations: The case of additive white noise"; *Phys. Rev. A* **40**, 5361-5369 (1989).

[27] N G Stocks, R Mannella and P V E McClintock, “Influence of random fluctuations on delayed bifurcations. II The cases of white and coloured additive and multiplicative noise"; *Phys. Rev. A* **42**, 3356-3362 (1990).

[28] J M Sancho, R Mannella P V E McClintock and F Moss, “Relaxation times in a bistable system with parametric, white noise: theory and experiment"; *Phys. Rev. A* **32**, 3639–46 (1985).

[29] R Mannella, S Faetti, P Grigolini and P V E McClintock, “On the relaxation of fluctuations in the steady state of the Stratonovich model"; *J. Phys. A: Math. Gen.* **21**, 1239-52 (1988).

[30] J Casademunt, J I Jimenez-Aquino, J M Sancho, C J Lambert, R Mannella, P Martano, P V E McClintock and N G Stocks, “Decay of unstable states in the presence of coloured noise and random initial conditions. II. Analogue experiments and digital simulations"; *Phys. Rev. A* **40**, 5915-5921 (1989).

[31] M I Dykman, R Mannella, P V E McClintock, F Moss and S M Soskin, “Spectral density of fluctuations of a double-well Duffing oscillator driven by white noise"; *Phys. Rev. A* **37**, 1303-13 (1988).

[32] M I Dykman, R Mannella, P V E McClintock and N G Stocks, “Fluctuation-induced transitions between periodic attractors: Observation of supernarrow spectral peaks near a kinetic phase transition"; *Phys. Rev. Lett.* **65**, 48-51 (1990).

[33] M I Dykman, R Mannella, P V E McClintock, S M Soskin and N G Stocks, “Noise-induced narrowing of peaks in the power spectra of underdamped nonlinear oscillators"; *Phys. Rev. A* **42**, 7041-7049 (1990). [But note that the data as finally published were acquired with a NIC-1180.]

[34] N G Stocks, P V E McClintock and S M Soskin, “Observation of zero-dispersion peaks in the fluctuation spectrum of an underdampe single-well oscillator"; *Europhys. Lett.* **21**, 395-400 (1993). [But note that the data as finally published were acquired with a NIC-1280.]

[35] F Moss, D K Kondepudi and P V E McClintock, “Branch selectivity at the birfurcation of a bistable system with external noise"; *Phys. Lett. A* **112**, 293–96 (1985).

[36] M I Dykman, P V E McClintock, V N Smelyanski, N D Stein and N G Stocks, “Optimal paths and the prehistory problem for large fluctuations in noise-driven systems"; *Phys. Rev. Lett.* **68**, 2718-2721 (1992).

[37] M I Dykman, D G Luchinsky, P V E McClintock and V N Smelyanskiy, “Corrals and critical behaviour of the distribution of fluctuational paths”, *Phys. Rev. Lett.* **77**, 5229–5232 (1996).

[38] D G Luchinsky and P V E McClintock, “Irreversibility of classical fluctuations studied in electrical circuits”; *Nature* **389**, 463–466 (1997).

[39] *Diagnostic Tests and Maintenance Instructions for Nicolet Data Processors* Nicolet Instrument Corporation, Madison, 1976.

[40] I Kh Kaufman, *LancsView Data Acquisition System. Technical Documentation*, Lancaster University, 1997.

[41] Microstar Laboratories, Inc., Bellevue, WA 98004, U.S.A.

[42] *The NED Text Editor for the Nicolet 1080 data system*, Nicolet Instrument Corporation, Madison, 1975.

[43] *The WARLK2 Disk Utility for the MED-80*, Nicolet Instrument Corporation, Madison.

[44] Frank da Cruz, *Kermit: a File Transfer Protocol*, Digital Press, Bedford, Massachusetts, 1987.