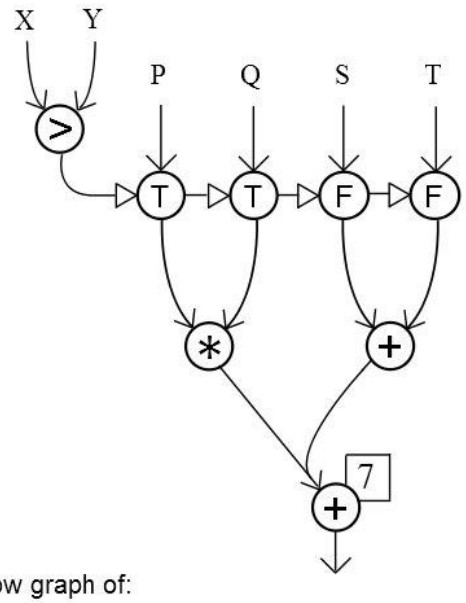
THE STATIC DATA FLOW INSTRUCTION CELL CHIP

William Ackerman

In the mid 1970's, a general design for a static data flow processor took shape within the MIT Computation Structures Group of Jack Dennis.

In a data flow computer, the program is expressed directly as a data flow graph, with nodes for the instructions and arcs carrying results from one instruction to the next. When all the required operands have arrived, an instruction "fires", performing the required operation and sending the results to successor nodes. Conditionals and loops are handled with special "T" and "F" nodes, that take a data operand and a boolean operand, and pass the data operand only if the boolean operand has the required value.

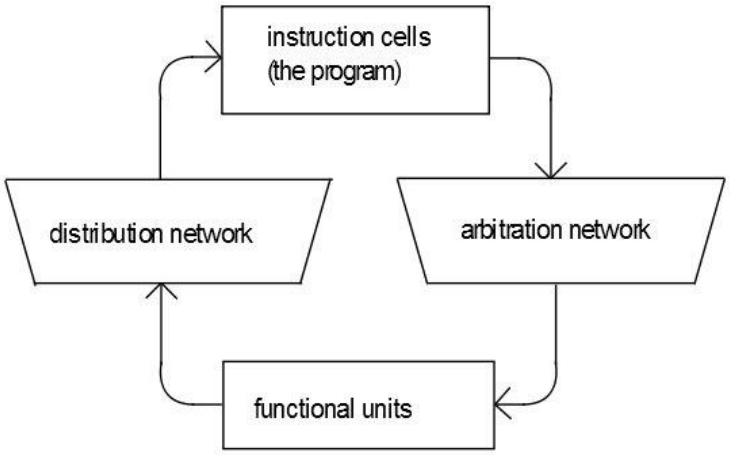


A data flow graph of:

```
(( x > y) ? P*Q : S+T) + 7
```

Some looping constructs could lead to an "overrun"; data might be sent to an operand register before the previous operand was consumed---this is an aspect of the "safety" problem in Petri nets. When this is a possibility (it isn't common), additional backward-pointing "acknowledgement arcs" are added to the graph, carrying no data but nevertheless controlling an instruction's readiness to fire.

The program, in the form of nodes of the data flow graph, is loaded into a hardware structure called the "instruction memory", which consists of a large number of "instruction cells". Each node is loaded into a fixed cell, and the cells have fixed addresses within the network mechanism. (This is what makes the design a "static" data flow machine.)



Each cell contains an opcode and the destination addresses of those "downstream" cells that are to receive its result; these items do not change. It also contains operand registers and "operand present" bits, which do change.

As the "upstream" cells complete their operation, their results arrive in the operand registers, and the "present" bits are set. When all the required "present" bits are on, the cell hardware recognizes this and makes the instruction "fire". An *instruction packet* is created, containing the opcode, operand values, and all the destination cell addresses to which results are to be sent. In principle, the hardware looks at all instructions at once, and can "fire" large numbers of them independently and asynchronously.

The instruction packets go through an *arbitration network*, which sends them to an available functional unit. There are a large number of asynchronously operating functional units. They contain no persistent data. They do not have assigned addresses; all functional units for a given opcode are identical.

The arbitration network finds an available functional unit that supports the required opcode, and delivers the instruction packet to it. The functional unit performs the indicated action, and then, for each destination in the list, forms a *result packet* containing the result.

The result packets go through a *distribution network*, which uses the address to send it to the appropriate cell. That operand register's "present" bit is set, (they get cleared when an instruction fires), and the process continues.

Beginning in the late 1970's, as the abstract design became more concrete and plans were formed for an actual realization of a machine, the design was modified to simplify the firing rules, and to take advantage of locality in the network. Instead of "operand present" bits for each operand, there is an "enable count" that is initialized to the correct value, and counts downward as packets arrive. When it reaches zero, the instruction is enabled for firing. When it fires, the enable count is set back to the correct value. Constant ("immediate") operands, such as the "7" in the figure above, are treated simply by initializing the operand register as required, and adjusting the enable count so that no incoming

packet is ever expected for that operand. Acknowledge signals are simply packets that contain no data and do nothing but decrement the count.

Also, instead of separate hardware structures for each individual instruction node and each individual functional unit, the machine would be divided into "cell blocks", containing a few tens, hundreds, or thousands of instruction cells, along with the necessary functional units. The overall program graph would be mapped onto cell blocks in such a way as to maximize the number of "local" or intra-block arcs. The "enabled" or "ready to fire" status of each cell (that is, the fact of its enable count being zero), would be kept in a flip-flop; the collection of these for all the cells in the block comprised the "enable memory". The control mechanism of the cell block would watch all of these bits and fire the instructions, executing them on one of the local functional units. Those destination arcs that were local would send the results directly to the local operand registers, manipulating the enable counts of the the various cells, and perhaps setting their bits in the enable memory. Nonlocal results would still be sent out, through a packet switching network, to other cell blocks.

During this time the Computation Structures Group developed a number of necessary tools. The VAL data flow programming language was developed, and various compilers and simulators were written. An "engineering model" of a cell block was designed and constructed, using commercial bit-slice integrated circuits for the control and ALU mechanisms. This was run as a peripheral device on a Unix computer.

Also beginning in the late 1970's, as the Mead-Conway design methodology made actual fabrication of chips accessible to ambitious students and academic researchers, "VLSI Fever" hit the Electrical Engineering and Computer Science department at MIT and many other universities. Lynn Conway taught a pilot course in the methodology, which was then picked up by other instructors and teaching assistants. A few members of the Computation Structures Group took the pilot course, though their projects were not related to data flow.

The course was extremely popular across the entire department, and, in subsequent semesters, more members of the Computation Structures Group took it. Beginning in the early 1980's, ideas about VLSI and ideas about data flow computer design began to merge, and some projects were undertaken that related to data flow. There were chips for small prototype routing networks, chips for arbiters, and chips to explore self-timed logic methodology.

Members of the Computation Structures Group also contributed to the development of the simulators and other tools that were required for class projects.

While this was going on, there were also investigations of various theoretical issues of program structure, semantics, compiler design, and hardware description languages. There were also analyses of various "number crunching" algorithms from such places as NASA, the Goddard Institute for Space Studies, and an extremely sanitized weapons code from Lawrence Livermore Laboratory. These analyses showed that such programs would be well suited for execution on a static data flow computer.

In 1983, an extremely ambitious class project was undertaken by William Ackerman, Nena Bauman, and Barnes Woodhall---a complete instruction cell processor. This would be capable of direct execution of compiled VAL programs. Local data arcs would be handled completely on the chip, and nonlocal packet transmission and reception would be handled through a parallel I/O port. It had an 8 bit data path, the usual integer instruction set, and an enable memory to handle 64 cells. The enable memory was connected to a priority encoder that would choose the lowest-numbered enabled cell for next execution. The cell contents themselves---opcodes, enable counts, enable reset values, operand registers, destination addresses, and flag bits---were stored in an external 2K by 8 bit RAM chip. Each cell used 32 bytes in that chip. Only the enable bits themselves were actually stored in the cell block. The chip was implemented in 5 micron CMOS, with fabrication supervised by MOSIS, the organization that had taken over the earlier supervision by Xerox PARC.

This project turned out to be too ambitious to bring to successful completion in a one-semester course. In addition to the complexity of the problem itself, the node-extraction, design rule checking, and simulation tools were not sufficiently robust for a project of this size. The recent conversion from NMOS to CMOS added to the problems. The project was completed and submitted for fabrication, but without the expectation that it had a significant likelihood of working.

Over the following two years, away from serious time pressure, Ackerman undertook a deliberate and methodical research program into making chips that could be design-rule-checked, node-extracted, simulated, and successfully fabricated. The investigation was done in CMOS, using the new 3 micron rules. The overall effort was a collaboration with several other people in the MIT VLSI group. The design rule checker was developed for the 3 micron rules, the node extractor was improved, and the "RNL" simulator was improved to get it to simulate CMOS chips reliably. People designed CMOS pad libraries and gate libraries, and a teaching assistant wrote a "regular structure generator" that could be used to make repetitive structures such as programmed logic arrays (PLA's.)

Prior to submitting the improved data flow cell block, several test submissions to MOSIS were made, testing the robustness of the tools, testing the robustness of a new precharged PLA design, and characterizing various gate libraries, pad libraries, and other circuits. These libraries, and a paper about designing large projects that can be successfully simulated and fabricated, were given to the MIT VLSI group.

<<put in ref>>

Trial fabrications of the gate libraries, PLA tests, pad tests, and the cell block itself were conducted throughout 1984 and early 1985. Several test projects were sometimes submitted at the same time, and redundantly fabricated on several fabrication runs. MOSIS gave the runs codes consisting of the low digit of the year, the month in hexadecimal, and a letter identifying the fabrication run in that month.

They were also given fanciful names like Mona, Ophra, Ethel, Amadeus, and Clyde. So, for example, run M53X, called "Xanadu", was fabricated in March 1985.

A cell block that very nearly worked was on run M45H (Hansel), in June 1984. That run also saw a test of a very large instance of the new precharged PLA design.

The final cell block was submitted in June 1985, on fabrication run M56L (Linda.) Thirteen bonded devices (along with several unbonded "costume jewelry" chips) were received in August. One of them worked perfectly.

A test rig was built for the working cell block chip. It has a UART serial RS232 connection for downloading and communication with a host computer. Outgoing ("foreign") packets are sent out through the serial port, and incoming packets from that port are sent to the cell block. Also, the data in the last outgoing packet is stored in a register connected to some display LED's.

The crystal clock for the UART requires that the cell block operate at a microcycle speed of much less than 1 MHz. When disconnected from the crystal clock, it has functioned properly at about 1 MHz. The chip is 6.7 x 6.8 millimeters in size, uses 3 micron design rules, makes sparing use of second metal (this technology arrived late in the project), has about 7300 transistors, and is in a 40 pin package. It operates a commercial 2K by 8 bit commercial RAM chip (a 6116) for storage of the instruction cells.