

L23.1 Altera's FLEX 10K Devices

Altera's FLEX 10K devices are the industry's first embedded PLDs.

They are based on reconfigurable CMOS SRAM elements.
This enables 100% testing prior to shipment.

They can (must) be configured on a pc board which facilitates field changes and special test modes.

The next slide shows performance achieved for some designs.
All performance values were obtained with Synopsis DesignWare or Library of Parameterized Functions (LPM) functions.

The designer simply inferred or instantiated a function in Verilog, VHDL, AHDL, or a schematic design file. Actually, one can mix design techniques in a single design.

L23.2 FLEX 10K and 10KA Performance

Table 6. FLEX 10K & FLEX 10KA Performance

Application	Resources Used		Performance				Units
	LEs	EABs	-1 Speed Grade	-2 Speed Grade	-3 Speed Grade	-4 Speed Grade	
16-bit loadable counter (1)	16	0	204	166	125	95	MHz
16-bit accumulator (1)	16	0	204	166	125	95	MHz
16-to-1 multiplexer (2)	10	0	4.2	5.8	6.0	7.0	ns
256 x 8 RAM read cycle speed (3)	0	1	172	145	108	84	MHz
256 x 8 RAM write cycle speed (3)	0	1	106	88	66	60	MHz

Notes:
(1) The speed grade of this application is limited because of clock high and low specifications.
(2) This application uses combinatorial inputs and outputs.
(3) This application uses registered inputs and outputs.

L23.3 FLEX 10K Devices

Each FLEX 10K device contains two arrays, embedded and logic.

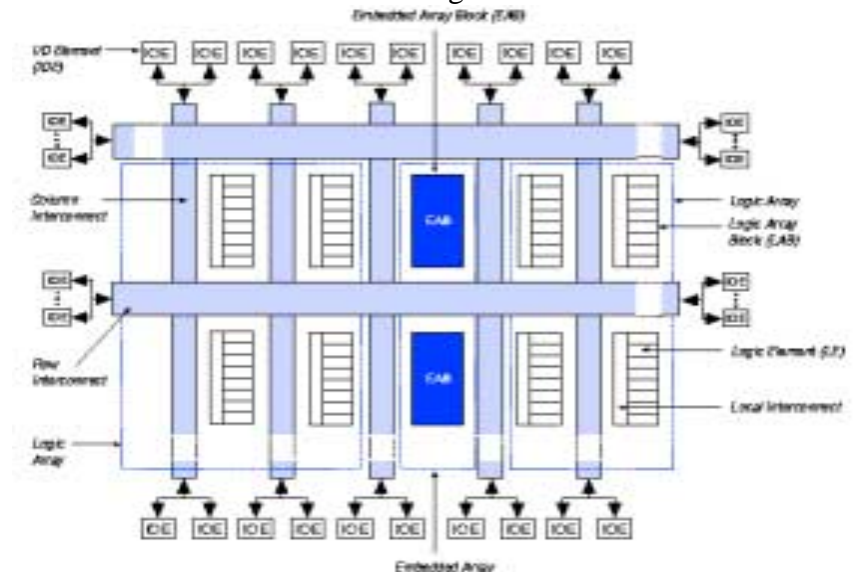
The embedded array is used to implement a variety of memory or complex logic functions. E.g., a DSP function.

The embedded array consists of a series of EAB, each of which provides 2,048 bits. The EAB can be used to create RAM, ROM, dual-port RAM, or a FIFO. EABs can be used independently or they can be combined to implement larger logic functions.

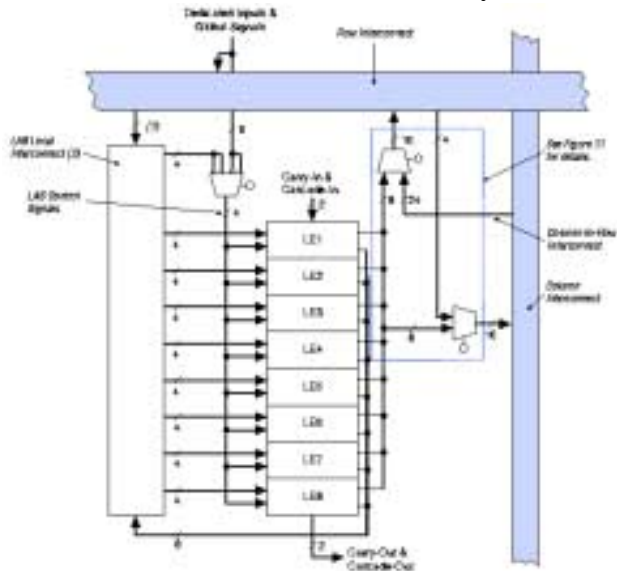
The logic array, which consists of LABs, performs the same function as the sea-of-gates in a gate array. It is used for general logic such as counters, adders, state machines, etc.

Each LAB has eight LEs and local interconnect. An LE consists of a 4-input LUT, a programmable FF and dedicated signal paths for carry and cascade paths.

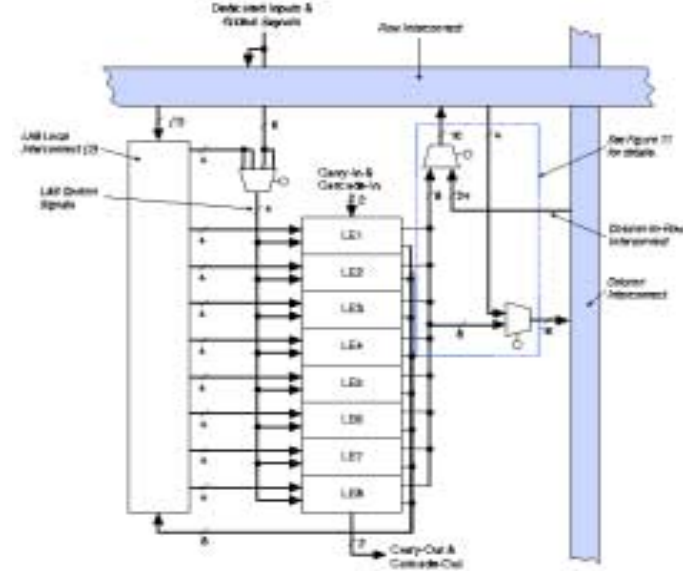
L23.4 FLEX 10K Block Diagram



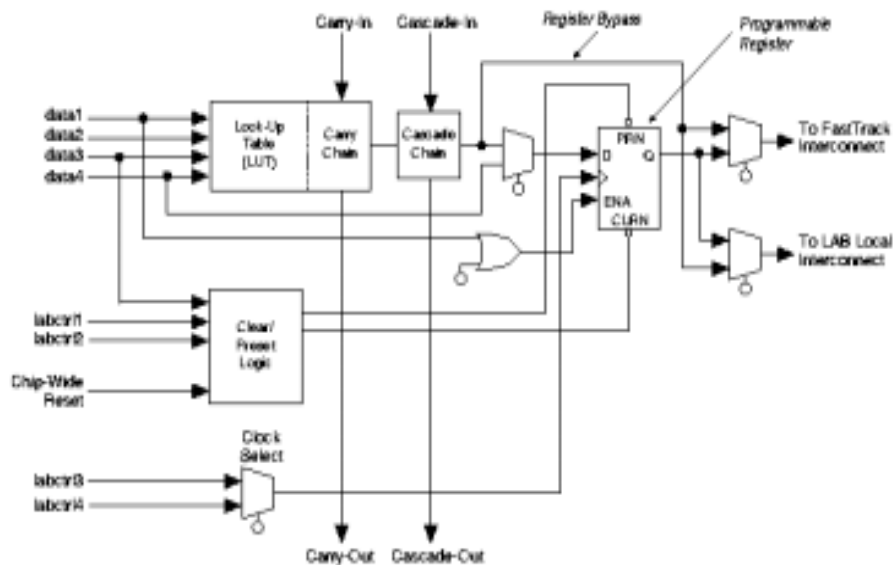
L23.5 FLEX 10K Embedded Array Block



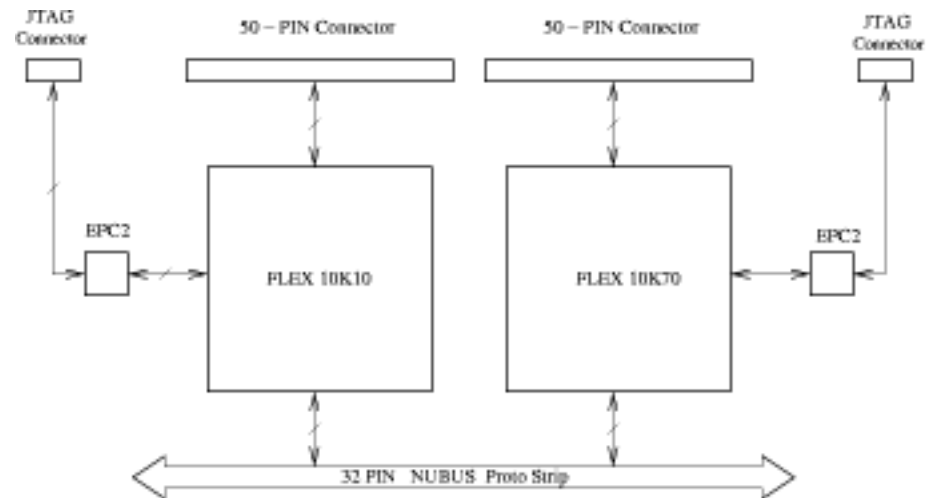
L23.6 FLEX 10K Logic Array Block



L23.7 FLEX 10K Logic Element



L23.8 FPGA Module



L23.9 FPGA Module

The PC board was developed by Brian Perrin as his MEng thesis which was completed last Thursday, April 4, 2002.

Details of the connections to the NuBus proto strip are in Table 1 of the documentation.

You can connect a 3M cable to both of the FPGAs or from an FPGA to K1 or K2 on the kit. Details of the connections to K1 and K2 are given in Table 2 of the documentation. You can also figure out the connections between the FPGAs if you use that mode.

BEWARE - if you use both a CPLD board and an FPGA board, they are connected to each other via the NuBus proto strip.

L23.10 Programming the FPGA Module

The FPGAs are SRAM based, so need to be configured on power on.

This is done automatically as each FPGA is wired to a flash prom.

Altera's software, MAX+plusII, is used to program the EPC2 flash prom with <project_name>.pof via a JTAG interface.

There is no simple (quick) way to use MAX+plusII to "erase" the EPC2s. One must program them with a VHDL derived file to tri-state all I/O pins connected to either the NuBus or the 50-pin connector (should you use it). This should be done on a new (to you) pc board as you do not know what teh EPC2s contain. Then, all you need do is program the FPGAs you use taking care that you tri-state any pins to be driven by other chips.

Of course, you should ensure that there is no bus contention on any of the NuBus pins.

L23.11 MAX+plusII

MAX+plusII software runs on any Athena Sun computer.

```
setup 6.111  
max2win&
```

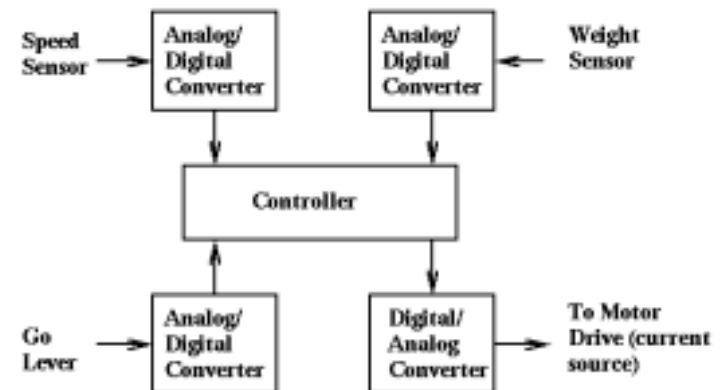
The first time on an X server not used by you before, the software will load fonts into a cache file. Unfortunately, this takes some time. If you ran it once on the console of an Ultra 5 then it should not load fonts another time you run it from the console of an Ultra 5, etc.

This software does a lot of different things and, thus, it is harder to use than software which doesn't do as much, e.g., galaxy.

See "A Beginner's Guide to MAX+plusII" on the 6.111 web page.

L23.12 L13.4 Block Diagram Fri. March 8, 2002

The first thing we do (as digital engineers) is to convert all analog signals to their digital representation and to convert the computed force command (motor drive current) to the required analog voltage.



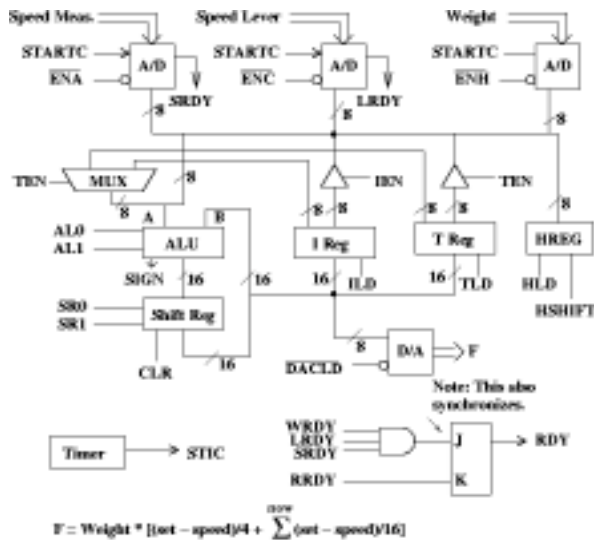
L23.13 Trolley Car Datapaths

AL0 A11 Operation

0	0	Add	A+B
0	1	Add1	A+1
1	0	Inv	/A
1	1	Sub	A-B

SR0 SR1 Operation

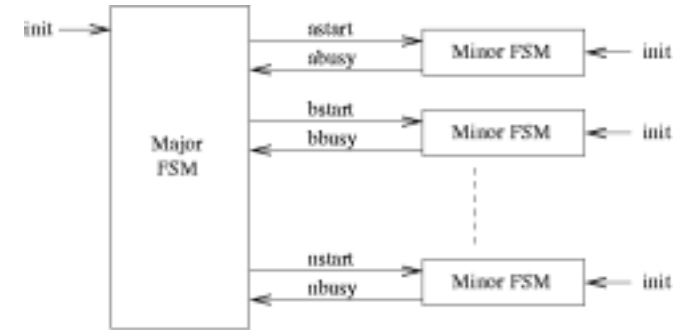
0	0	Hold
0	1	Load
1	0	Rotate Left
1	1	Shift Right Arithmetic



L23.14 L20.1 FSM Hierarchy

Problem Statement:

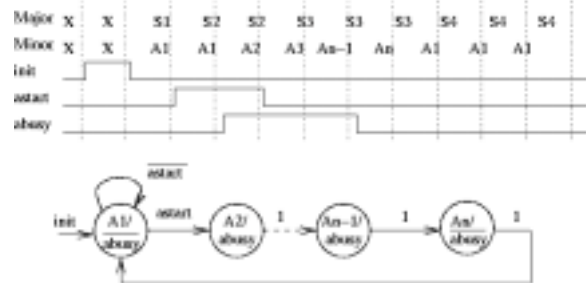
We want to coordinate (control) a sequence of operations performed by other FSMs. To do this we divide the problem into multiple FSMs which may take multiple (sometimes an unknown number) clock cycles. These minor FSMs are controlled by a major FSM. Minor FSMs may also be decomposed into multiple FSMs. All FSMs operate on the rising edge of the same clock.



L23.15 L20.2 Minor FSMs

Minor FSMs are started by a control signal (start) generated by the major FSM which controls this particular minor FSM. All minor FSMs are initialized by a signal (init) which is synchronized to the clock and asserted for a single clock period.

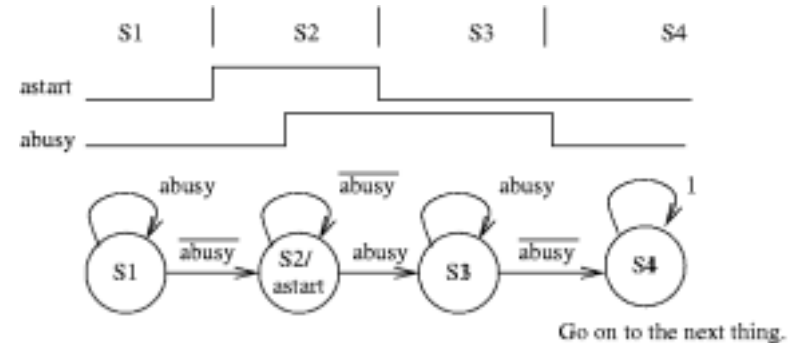
The makeup of minor FSMs is arbitrary, but with some constraints. The shortest path back to the starting state (reached by init) must be at least two clock cycles.



L23.16 L20.3 Major FSMs

The major FSM is initialized by the same init signal supplied to the minor FSMs.

Major FSMs generate a control signal (start) for the minor FSMs. They test the busy signal generated by the minor FSM to determine both when the minor FSM is available and when it is done.



L23.17 Major Control in Words

Wait for STIC to be asserted.

If `/(rdy` and `/bcompute)` start over (something took too long).

Load DAC with previous computation. Note that one uses the eight high order bits. Load `hreg` (`enh`).
Clear the DAC and the `rdy` flip-flop.

Add speed measurement (`ena`).

Sub speed lever (`command`) (`enc`).

Compute, then clear the shift register.

If sign multiplyn else multiplyp.

Go to start.

L23.18 Compute Control in Words

Shift the shift register right (arithmetically) twice to divide by four.

Save the result in the Treg.

Shift the shift register right (arithmetically) twice to divide by four again.

Add in the Ireg.

Store in the Ireg.

Add in the Treg.

Save in the Treg.

L23.19 Multipln Control in Words

Invert the Treg.

Add one to that result.

Save the shift register into the Treg.

Multiplyp

Save the shift register into the Treg.

Invert the Treg.

L23.20 Multiplp Control in Words

Do the following eight times.

Assert `hshift` to shift the `Hreg` right.

Assert `rot` to rotate the shift register right.

If the LSB of `Hreg` is one, then add `Treg` to `sr`.

Then do eight rotates to get the high order bits back in the right place.