INTRODUCTION TO THE LOGIC DESIGN TOOL

Problem:
It's a digital world.

More and more microprocessors are found in more and more devices, and more of those devices are taking on safety critical roles. An error in those devices can cause great damage or even cause loss of life. A prime example of a critical digital device is an aircraft computer used to fly an airplane. The wrong error could cause loss of a plane, the pilot, the crew, and the passengers.

The proper operation of the aircraft computer depends upon proper specification of the digital logic which directs that operation. Logic is an operation that results in a true or a false assertion – do this or do that. Logic in software specifies whether an operation will occur and when it will occur. For example, airplane flight control logic might check that before the landing gear is retracted up (do this), the aircraft weight must be off the landing gear, the aircraft must be at a certain height, speed must be above a certain level, and no ice is on the wings. If any of those conditions are not true, then the landing gear must remain down (do that).

If a digital system has only a small number of input conditions, the specification of the logic is easy. With two inputs that are either true or false, the number of final decisions based on those two inputs is four. With three inputs, the number of decisions is eight. With four inputs, the number is 16, five is 32, and so on. When the operation of these inputs changes based on other conditions, like whether the plane is upside down, right side up, taking off, landing or in a tailspin, then the conditions become even more complex. Complexity increases rapidly with an increase of input conditions and different environments.

Logic equations may contain language such as “retract the landing gear when there is no ice on the wings, no weight on the landing gear and speed is greater than 120 knots **OR** the aircraft is in level flight, there is no ice on the wings, the speed is greater than 220 knots, and the altitude is above 1300 feet **OR** the speed is greater than 300 knots, the aircraft is in a tail spin and the altitude is greater than 3000 feet **OR** … and so on”. Making all this logic consistent so that all conditions are covered and no conditions are specified more than once (complete and unambiguous) can be daunting. But consistency is imperative. If an error is introduced, and the landing gear inadvertently retracts before takeoff, or comes down during high speed flight, the aircraft and crew could be lost.

As the aircraft conditions become more complex, the chance that an unintentional and possibly hazardous action will occur becomes greater.

Solution:

The Logic Design Tool (LDT) is a graphical aid for developing and analyzing digital logic. LDT is essentially Computer Aided Design (CAD/CAM) for the logical specification of software or hardware. With LDT, all logical conditions and all actions are displayed in a manner where they may be viewed, considered, specified, executed and analyzed before the logic is automatically generated in software or hardware source code. This is done by representing any ‘black box’ transform of inputs to outputs as a ‘logic space’ hierarchy of Karnaugh combination maps (Figures 1 and 2). LDT also displays the logic with multiple views and actions, so the designer has a greater chance to find a design error or a logical oversight. It allows the design work to be done up front and as a whole - not piecemeal - and as such can save significant integration, test, debug and rework time. Logic is strongly typed to increase rigor, yet LDT makes navigation through even a large logic space manageable.

FIGURE 1
TRANSFORM FIELD DEFINITION

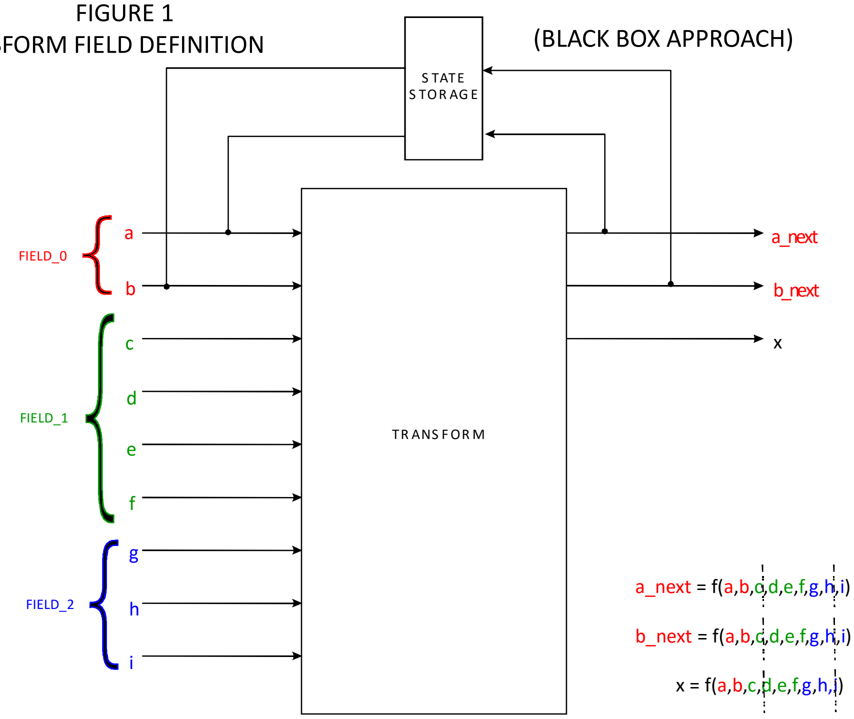
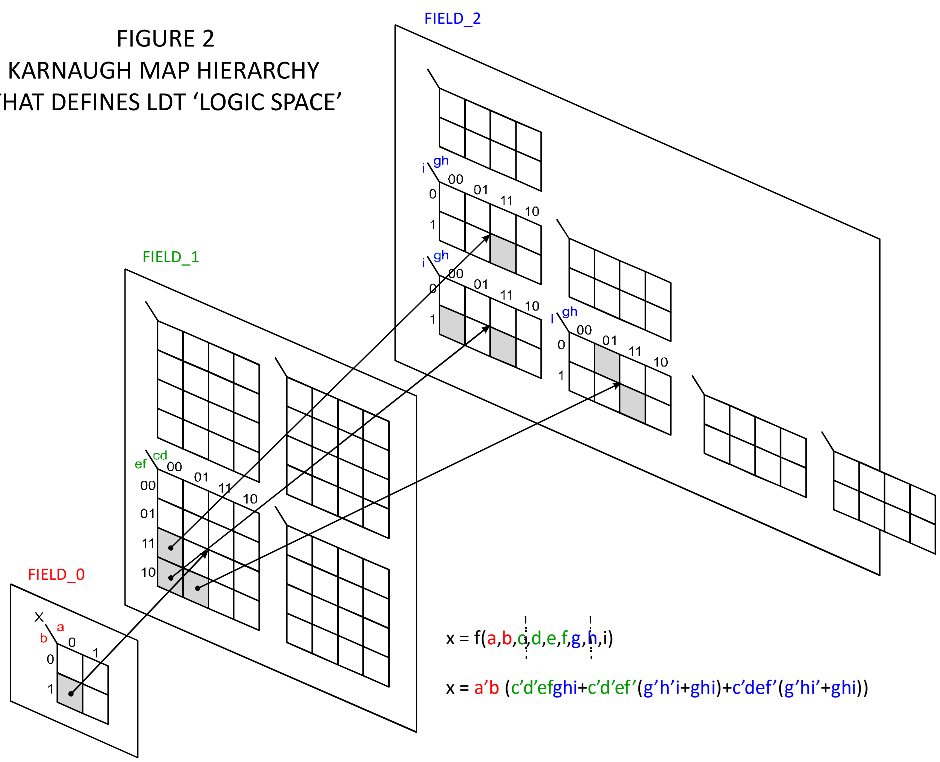


FIGURE 2
KARNAUGH MAP HIERARCHY
THAT DEFINES LDT 'LOGIC SPACE'



LDT applies to more than just aircraft flight control. It also has application to other areas where the operation must be proven correct, such as medical instrumentation, data encryption, network integrity, key financial transactions or large volume consumer electronics. As digital systems become even more common, complex high assurance applications, and the need to guarantee their proper operation, will continue to grow. Multiple studies show that logical errors account for 30 to 40 percent of requirement, implementation and testing errors. And latent errors in the high level requirements may not be discovered until the system is fielded. The cost of finding and fixing the error in the initial design phase is orders of magnitude smaller than finding and fixing the error after the system is fielded (or the plane crashed).

A side benefit of specifying all conditions is that analysis can be made of the whole system. One example of analysis is: "is it possible to get from a safe condition (landing gear up in level low speed flight) to a hazardous condition (landing gear down during high speed flight)"? LDT can find worst and best case execution times (what is the worst case execution time from speed less than 140 knots to gear down).

Another significant benefit of having all conditions specified is that exhaustive test software, which typically takes 30 percent of development cost, can be automatically generated and reduced (via Quine-McClusky reduction) to the minimum size and execution speed.

Also, because the documentation (which is again automatically generated) has all cases specified, that documentation becomes 'lawyer-proof' after being signed by the customer.

We believe that because LDT can expose most logic errors, it will become the standard for developing all digital systems of any significant complexity.

A digital world needs a logic tool to ensure digital device safety.