COUNTING GATES

William Bricken
December 2001

2001 Xilinx FPGA product listings assign 12 "logic gates" per logic cell, with 9 logic cells per CLB. So "maximum logic gates" is 108 times the number of CLBs.

"System gates" refer to logic plus memory. This metric is fudgable, since each logic cell (look-up table) can be used as 12 logic gates or as approximately 50 memory "gates". The percentage of logic cells used as memory generates the "typical system gate range". "System gates" is also greatly distorted in architectures which add bulk RAM (eg Virtex-EM vs. Virtex-E), since bulk RAM adds to system gates but not to logic functionality.

Recommend: use "logic gates", not system gates. Use 50% of maximum logic gates as a conservative capabilities estimate.

Maximum logic gates must be decreased due to at least three factors:

1. Assumes perfect logic mapping onto arbitrary 4LUT structures. Experience suggests this mapping is about 50-70%.
2. Assumes unlimited routing resources. Not! Limitations in routing reduce possible mappings to about 50%, worse for smaller gate parts and for inexpensive Spartan parts.
3. Assumes perfect integration between memory and logic use. This is perhaps the biggest problem, since logic/memory integration is an unsolved problem. Percentage logic to memory vary widely across applications.

As well, all Xilinx FPGAs are flip-flop limited, so sequential logic structure as well as combinational logic structure limits mappability.

One major benefit of BTC architectures is that we don't distinguish logic from memory, and like Xilinx, can swap off logic for memory (for us it is 4-to-1 logic gates per FF, not 1-to-4 as in LUTs). The main point is that we do not have a flip-flop ceiling, we are flexible there.

And as a reminder: FPGA architecture wastes huge amounts of territory in routing resources which are *not used* once the functionality is mapped. Similarly, the available logic/memory on a Xilinx part is *fixed*. The only option available for designs "close to the maximum" is to buy a larger chip. (BTC has excellent tools for time/space design trade-off to solve this problem.)

How Should BTC Count Gates?

We have used 4 gates per row, with no consideration of flip-flops or block RAM memory. We should probably use close to the same memory/logic ratios as Xilinx. They have 4 flip-flops per CLB (i.e. per 108 logic gates), 1 FF per 28 logic gates. Given a logic usage of 50%, this is 1 FF per 14 effective gates.

Block RAM is in addition to FFs, and is a different issue. To be realistic in the FPGA market, we will have to address dual-usage Occlusion Arrays, part memory, part logic. Not so for the CPLD market, which is even more FF limited, and has no concept of block RAM.

Rounding, I'd suggest 1 FF per 3 rows (1 FF per 12 gates), i.e. 75% logic usage, 25% FF. We get 1 FF per row. So per 1000 rows (~ 1 mm^2 chip area), we'd have 3K logic gates and 250 FFs. In truth, we can mix in any proportion: 1 row gives either 4 gates or 1 FF.

With the compiler, it is a bit more complex, since the placement and use of FFs is tailored for ASIC architectures, not BTC architecture. I estimate we can eliminate (compile out) 50% of the FFs in a naive FSM design. Another main use of FFs is pipelining. We get pipelining from the compiler, and don't need to use FFs. In summary: the BTC architectures use FFs in a fundamentally different way which largely eliminates the FF bottleneck in other FPGA and CPLD architectures.

Note: the above has a few very important selling points which we are neglecting to mention:

1. flexible logic/FF usage
2. far less FF dependency
3. compile time logic/FF trade-off