# Hardware Multiplier Design
# - One Day from the Idea to the Implementation -

**Dan NICULA, Florin SANDU, Carmen GERIGAN**
*Transilvania* University, Department of Electronics & Computers
29 Eroilor, 2200 Braşov, ROMÂNIA
Tel.: +40-68-41.30.00 ext. 116, Email: nicula@vega.unitbv.ro

**Abstract:** *This paper relates to the successful experience of implementing a two complement 4 bit multiplier, based on Booth's algorithm. There was necessary only one day to convert the idea into a hardware implementation. The designer must had the following pre-requested knowledge: high level digital design, VHDL modelling for synthesis, medium experience of using Electronic Design Automation (EDA) software tools like: V-System, Cadence, XACT.*

**Keywords**: High level design, VHDL synthesis, EDA tools, programmable logic devices.

## 1. Problem and Solution

The design cycle of complex hardware is getting shorter and shorter. Nowadays, the major objectives in VLSI design are: *the design quality* and *the designer's productivity*. How could these objectives be reached in the same time?

Using modern EDA software tools, it is possible to design and to test a medium complexity system during only one day. To do this, a *top-down methodology* has to be used. A key point of this approach is a Hardware Description Language (HDL) capable to offer support at all levels of description (behavioural, structural, physical).

Figure 1 illustrates this integrated design flow that reduces the amount of code that has to be maintained and the risk of inconsistencies between different models.

## 2. Block Schematic

The Booth's multiplication algorithm is described in [3]. This algorithm is suitable for multiplication of numbers coded in two's complement and has the advantage of its simplicity. Figure 2 illustrates a data-path that implements multiplication algorithm together with a control-path (implemented like a synchronous Finite State Machine - FSM).

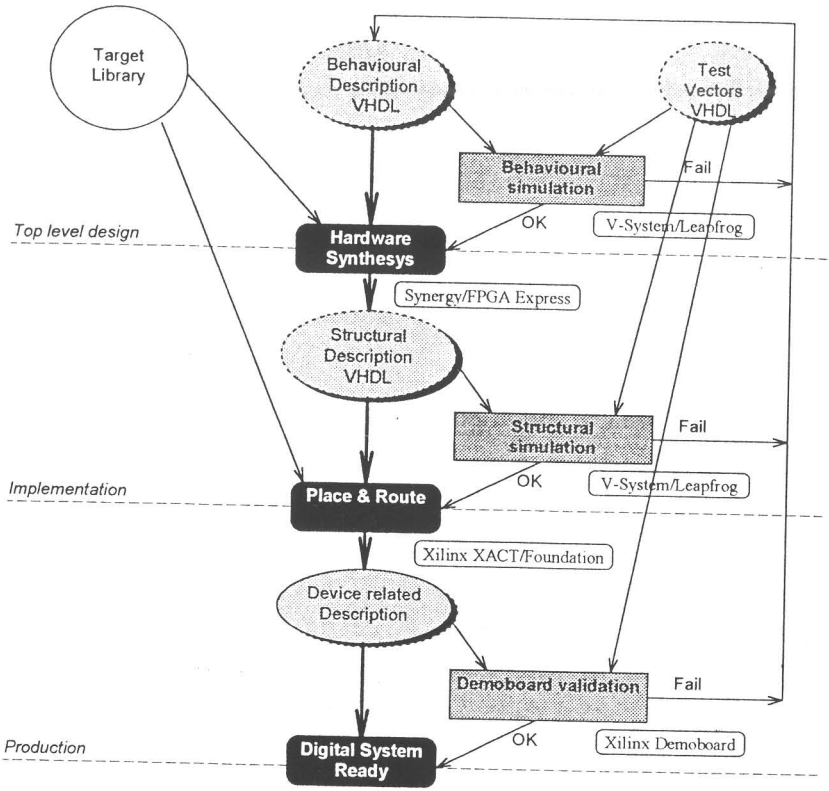


**Figure 1**. Top-down design flow.

## 3. VHDL Modelling and Simulation

The quality of the design depends of the designer's modelling style. The synthesised logic is directly inferred from the structure of the hardware description.

The behavioural VHDL code was developed started from an algorithmic description of multiplication. The VHDL code has generic dimensions for operands, providing a very convenient way for extending the data-path dimensions.

The multiplier is decomposed in two parts: *data path* and *control path*. Data path is modelled using many processes. All registers receive the global asynchronous reset and the global clock. Control path is modelled like an explicit FSM as stated in [4].

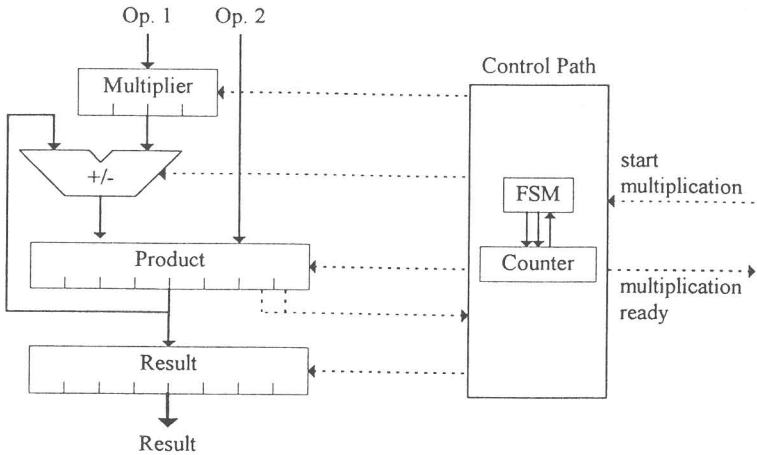


**Figure 2**. Data Path and Control Path of Booth's multiplier.

Modelling strategy follows the rules in [4]. Data type *std_logic*, declared in the *IEEE package std_logic_1164* was used. All units used only arithmetic operators declared in *IEEE package std_logic_arith*.

The behavioural simulation has been carried out on a PC, using V-System/Windows. The structural simulation (post synthesis) has been carried out on a workstation, using Leapfrog from Cadence.

## 4. SYNTHESIS

The high level architect's goal is to write a sufficiently detailed behavioural model of the system so as to be able to use a software tool for synthesis. Before synthesis, the model is technology independent. The synthesis process is very strong technology dependent. After synthesis, for programmable devices, an automated "place and route" process will follow.

The behavioural VHDL model of multiplier has been synthesised using Synergy from Cadence on a XILINX/FPGA target library.

Making a constrain regarding preserving the boundaries allows a better control of logic structure generated for a specific block.

The cost report generated by software shows that the area devoted to *Control Path* is only the half from the area devoted to *Data Path*. It must be mentioned that the *Control Path* is not dependent by the operand dimension.

The following page presents the cost report as it was generated after synthesis.

MODULE: Work.MUL.STRUCTURAL_SYN

| Cell | Count | Area each | Area total |
|---|---|---|---|
| Subtotal | 0 | | 0.00 |

| Sub-modules | Count | Area each | Area total |
|---|---|---|---|
| Work.DATA_PATH.BEHAVE_SYN | 1 | 70.00 | 70.00 |
| Work.CONTROL_PATH.BEHAVE_SYN | 1 | 32.00 | 32.00 |
| Subtotal | 2 | | 102.00 |
| Total | 139 | | 102.00 |

Block Summary:
No block

MODULE: Work.CONTROL_PATH.BEHAVE_SYN

| Cell | Count | Area each | Area total |
|---|---|---|---|
| xvcc | 1 | ** | ** |
| inv | 14 | 0.00 | 0.00 |
| fdpe | 1 | 1.00 | 1.00 |
| or4 | 1 | 1.00 | 1.00 |
| or3 | 2 | 1.00 | 2.00 |
| and3 | 2 | 1.00 | 2.00 |
| nor2 | 3 | 1.00 | 3.00 |
| and2 | 3 | 1.00 | 3.00 |
| or2 | 9 | 1.00 | 9.00 |
| fdce | 11 | 1.00 | 11.00 |
| Total | 47 | | 32.00 |

MODULE: Work.DATA_PATH.BEHAVE_SYN

| Cell | Count | Area each | Area total |
|---|---|---|---|
| xgnd | 1 | ** | ** |
| buff | 1 | 0.00 | 0.00 |
| inv | 28 | 0.00 | 0.00 |
| or2 | 2 | 1.00 | 2.00 |
| nand2 | 3 | 1.00 | 3.00 |
| m2_1e | 3 | 1.00 | 3.00 |
| or4 | 3 | 1.00 | 3.00 |
| fdce | 4 | 1.00 | 4.00 |
| or3 | 9 | 1.00 | 9.00 |
| fdc | 9 | 1.00 | 9.00 |
| add4 | 2 | 5.00 | 10.00 |
| and2 | 11 | 1.00 | 11.00 |
| nor2 | 16 | 1.00 | 16.00 |
| Total | 92 | | 70.00 |

## 5. ROUTING FPGA

Started from the structural description of the multiplier (gates and flip-flops), the next step is to implement the design into a programmable device.

The design flow has three stages.

- Design Entry. In this case, the design was imported directly from Cadence to XNF file format.
- Design Implementation. During this stage the logic is mapped onto the target device architecture. The designer controls the implementation process through a constrains file. During the place and route process, several report files are generated. The reports contain information about the number of CLB used for implementation and timing information.

```
PLACEMENT RESULTS FOR DESIGN MULTIPLIER
Partitioned Design Utilisation Using Part 4003APC84-6
```

| | No. Used | Max. Available | %Used |
|---|---|---|---|
| Occupied CLBs | 37 | 100 | 37% |
| Bonded I/O Pins | 20 | 61 | 32% |
| F and G Function Generators(*) | 55 | 200 | 27% |
| H Function Generators | 12 | 100 | 12% |
| CLB Flip Flops | 20 | 200 | 10% |
| IOB Input Flip Flops | 0 | 80 | 0% |
| IOB Output Flip Flops | 0 | 80 | 0% |
| 3-State Buffers | 0 | 240 | 0% |
| 3-State Half Long Lines | 0 | 40 | 0% |
| Edge Decode Inputs | 0 | 120 | 0% |
| Edge Decode Half Long Lines | 0 | 16 | 0% |
| CLB Fast Carry Logic | 6 | 100 | 6% |

```
CPU Times
CPU time taken for Partition:    0 hrs  0 mins 17 secs
CPU time taken for Placement:    0 hrs  3 mins 54 secs
========== End of Report ==========
ROUTING RESULTS FOR DESIGN MULTIPLIER
Routing Summary
---------------
Number of unrouted connections :  0
CPU time taken for Routing  0 hrs 1 mins 31 secs
========== End of Report ==========
```

It is to be noticed that the entire process (partition, place and route) takes only 5 minutes and 42 seconds.

- Design Verification. The FPGA can be verified immediately in the target application board, after it is programmed in circuit using the download cable.

## 6. Implementing a 8 x 8 bit Multiplier

The VHDL code was written with reusability in mind. This means that VHDL code for this project has several properties:

- process and technology independent: in order to be implemented in FPGA or different silicon technologies;
- software tool independent: in order to be used by different CAD environments;
- user independent: it is easy to understand and modify.

The following table presents the necessary time for another multiplier development (8 × 8 bit) using the same VHDL code.

| Modification | Code | Verification | User time | Execution time |
|---|---|---|---|---|
| VHDL | 30 s | 2 min | 10 min | 1 min |
| Synthesis | - | 20 min | 10 min | 15 min |
| Netlisting & Constraints | - | - | 10 min | 1 min |
| Place & Route | - | - | - | 5 min |
| Total Time | 30 s | 22 min | 30 min | 22 min |

It means that 75 minutes are enough to implement a N×N bit multiplier. This time includes verification (simulation), user time (setting constraints for synthesis and netlist, opening CAE tools, import and export project), and execution time (computer time).

## 7. Conclusions

For an experienced designer, using EDA tools, it is possible to drastically shorten the design cycle of hardware design. We proved that a medium experienced graduated student is able to fulfil an assignment during only one working day. This very short design and implementing cycle it is possible due to making use of top-down design methodology, having support of EDA tools and implementing the hardware on a pre-manufactured demonstration board.

## 8. References


[1] Toac°e G., Nicula D.: *Electronicã Digitalã (Digital Electronics)*, Teora, Bucharest, Romania, 1996

[2] *** *The Programmable Logic Data Book*, XILINX 1994

[3] Patterson D.A., Hennessy J.L., *Computer Organization and Design The Hardware/Software Interface*, Morgan Kaufmann, 1994

[4] * * *: *Synergy VHDL Synthesizer and Optimizer Modeling Style Guide 1.2.1*, Cadence on-line documentation