AN EXAMPLE OF ILOC ABSTRACTION (VECTORIZATION AND MODULARIZATION)

William Bricken
March 2003

The ILOC tools for abstraction of netlists into arbitrary bit-width vectors and into repeated functional modules are briefly described. Vectorization and modularization are combined to reduce a substantive circuit design, including the use of higher rank vectors in matrix abstraction.

Vectorization

The ILOC vectorization tools permit wires of varying bit-widths to be abstracted into a single N-bit wire. The modularization tools permit functional modules (library components and macros) to be automatically defined and constructed bottom-up from recurrent logic patterns within a netlist. The function generation tools permit modules to be defined and automatically constructed top-down from functional and behavioral specifications

Combining the modularization tools, the vectorization tools and the function generation tools creates a set of design tools with which a designer can compose, analyze, and explore a complete circuit abstractly, without regard to specific bit-widths or functional replication.

One important aspect of the ILOC abstraction tools is that all boundary logic deletion-reduction and rearrangement transformations can be applied directly to the entire abstract form, thus changing all instances of a module or a vector at one time. This permits efficient transformation of groups of signals over a given functionality, as well as automated generation of functionality to any bit-width.

Function modularization and vectorization are similar tools that achieve similar results. Fundamentally they differ in the way abstracted forms are stored and accessed. In modularization, the function template (i.e. library component or macro) serves as an abstracted module that can be referenced with particular bindings any number of times. Thus functional abstraction is useful when a fixed functional structure is mapped during technology mapping, and when a stable modular component has been identified. Functional modularization reifies a functional form while keeping i/o bindings flexible.

In contrast, vectorization applies to a particular set of matching patterns and usually is not extended when other pattern instances are identified. Thus vectorization applies to a fixed number of instances. In vector abstraction, an input/output template serves as the abstracted component that is referenced by a given functional structure a specific number of times. Thus vector abstraction is useful when the same functionality is applied to a

collection of inputs and results in an aligned collection of outputs. It is useful when a stable i/o bundle can be identified. Instances of a vector output can be referenced independently, in effect decomposing the vector after a function is applied. Vector abstraction reifies an i/o structure while keeping functional form flexible.

Example, the I7 Design

Vectorization is of particular value whenever the same vector occurs in different functional relations throughout the circuit. Figures I through V show a substantive design, I7 from the MCNC benchmarks, that has 199 inputs and 67 outputs. I7 is composed of 331 conventional gates, most of them having 4 and 5 inputs, an equivalent of just over 900 conventional two-input gates.

The ILOC internal form condenses to 31 vectors of size 2, 4, and 28. Figure I shows the internal ILOC netlist in vector form. It clearly shows that the design consists of six separable subcircuits. The first subcircuit is a one-bit combination of various enable signals, and fourth subcircuit is a two-bit combination of enables. The other four subcircuits are quite similar, differing primarily in bit-widths of 4 and 28 bits. Figure II shows the set of vectors that constitute the entire design, including i/o and all internal wires.

Vector Modularization

One important aspect of management of circuit size and complexity is to be able to express the circuit functionality succinctly. I7 requires over 100 *pages* to describe in EDIF 2 0 0 format. ILOC achieves a factor of 100 reduction over conventional formats, requiring a single page to describe the functionality of the I7 circuit. Additionally, substantive replication of structure is still observable in Figure I. Figure III shows the application of functional modularization to the ILOC vector body for I7. Use of a single vector module reduces the functional body to six lines, one line for each independent output. In the vector modularization, vectors are bound to the module input labels, and to the four module binding cells. These vectors are all of the same size for a particular module cell, but do not have to be of a particular size across module cells. In the example, both 4-bit and 28-bit vectors bind to the same module structure. Thus modularization not only applies to vectors as well as to cells, it also is insensitive to the size of the vectors bound to the module input and output. Transformation can be applied to the module body, even though different bindings may be to different vector lengths.

Technical Details of Functionality

Figure III shows the application of cell abstraction to the vector cells with the vector module. In particular, one XNOR structure and two MUX structures are identified and abstracted into patterns. Since these are vector patterns, they require care in interpretation. For example, **=2** refers to four different sets of XOR gates, one for each index in the vector **Va**. The size of the vector **Va** is indeterminate when localized in the module, so that its size is defined by the binding cells **V91-** through **V94-**. In the case of binding cell **V91-**, **Va** is bound to **Vi11**. **Vi11** is a vector that has 28 indices. Thus, **=2** in the vector-module expansion of **V91-** represents 28 separate XNOR gates, each receiving the same **ac** input, and one other input from **Vi11**. Succinctly, **=2** tests 28 inputs signals for equality to **ac**. The rest of the module is also easy to read. For binding cell **V91-**, cell **>3** is 28 MUX gates, all using the **ac** input as a selector between 28 pairs of input signals. Cell **>1** uses the **ad** input to steer to output either the vector of equality comparisons, or the vector of selected inputs from cell **>3**.

Summarizing, the module selects one of two inputs, tests a different input against the same selector, and then selects one of those results using a second selector. This is replicated for 28 inputs sets of three. Note that the MUX cell structures that were lost when the circuit was expanded to vector format in Figure I are regained in the modularization in Figure III.

Matrix Abstraction

Since vector cells can be converted into modules, they can also be converted into higher rank vectors, in effect constructing a matrix abstraction of the design. Figure IV shows the matrix abstraction of I7. Each new vector consists of other lower rank vectors. Here the vector components of a matrix cell are not the same size, they are merely an ordered list of ordered lists. Consistency of size is maintained by each cell in the ILOC internal netlist form. Compatibility of vector sizes is assured during the construction phases of vectorization. In the matrix abstraction of I7 shown in Figure IV, the ILOC netlist reduces to seven cells, each consisting of a one line parens cell-form. Six cells provide output in the form of enable signals combined with the central functionality of I7, while one single matrix parens form consisting of 11 containers and 5 different atoms describes the entire central functionality of the circuit. This matrix cell can be transformed using any applicable ILOC deletion-reduction rules, parens rearrangement rules, or restructuring rules.

Similar to Figure III, the matrix body of Figure IV is further abstracted using cell abstraction on matrix cells. The two MUX structures and the one XNOR structure are again identified, this time referring to a 4x28 matrix of single MUX and XNOR gates.

Advantages for Layout

These abstraction capabilities do more that make large circuits tractable for human designers, they also specify quite exactly how placement and routing can be optimized, since the abstractions show exactly how much fan-out and distance there is between signal bundles, as well identifying the places that the circuit can be most economically partitioned for resource mapping.

To illustrate the placement and routing information, Figure V shows an abstract matrix schematic of the I7 circuit. The gates in this schematic cannot be interpreted as conventional gates since some of their inputs are vectors and vectors of vectors. When a wide bit-width wire enters a gate in this schematic, the gate would be replicated a number of times to match the bit-width of the input. Also, within the schematic, the M4 matrix is decomposed into 4 vectors, indicating that the bit-line bundles split at that point. This succinct schematic represent the entire logic functionality of the I7 circuit, while suppressing the repetitive details of hundreds of conventional gates. Thus the schematic is a hybrid of conventional logic functionality and abstract functionality, while never loosing a precise mapping to the physical circuit it represents. The vector and matrix tools provide top-down block-like abstraction without enforcing blocks that obscure the actual logical transformations of the particular circuit. These capabilities also apply to automated partitioning, and to placing and routing of logical functionality, addressing what is considered to be one of the most difficult issues in design actualization.

The EDIF specification of I7 requires 5577 lines. The ILOC matrix specification requires 27 lines, for a compression factor over 200 to 1. The size and complexity of the software programs required to expand each representation into a form that is suitable for general transformation and display is approximately the same for EDIF and ILOC formats.

An important feature of matrix abstraction is that is illustrates the general capabilities of ILOC for hierarchical bottom-up abstraction. Should repetitive structure show up in the matrix format, it too could be vectorized to form a rank three matrix. Similarly, during top-down design, each abstraction level can be articulated as an abstract vector slice, or as a module component. Thus ILOC contributes powerful tools for the abstraction and abstract transformation of circuit designs, tools that can be applied to accelerate synthesis and transformation of circuit structures, enhance design capabilities, provide hierarchical top-down and bottom-up design capabilities, and in general provide management of large, complex, and difficult to understand circuit structures.

Figure I: MCNC Benchmark Circuit I7, Vector Expansion to ILOC Vector Format

ILOC vector body for the I7 benchmark circuit, expanded with vectors

Outputs

```
((oa 1)
 (Vout2 Vtop2)
 (Vout3 Vtop3)
 (Vout4 Vtop4)
 (Vout5 Vtop5)
 (Vout6 Vtop6))
```

Vector Body	**Vector Sizes**
((1	
(ad (ab)(ae)))	-- 1
(Vtop2	-- 28
(((ad (ac Vi11) ((ac)(Vi11)))	
((ad) (ac Vi41) ((ac) Vi42)))))	
(Vtop3	-- 28
(((aa (ac))	
((aa) ((ad (ac Vi10) ((ac)(Vi10)))	
((ad) (ac Vi12) ((ac) Vi13)))))))	
(Vtop4	
((ac)(ad (ab)(Vi29))))	-- 2
(Vtop5	-- 4
(((ab) (ad (ac Vi9) ((ac)(Vi9)))	
((ad) (ac Vi7) ((ac) Vi8)))))	
(Vtop6	-- 4
(((aa) (ad (ac Vi43) ((ac)(Vi43)))	
((ad) (ac Vi39) ((ac) Vi40))))))	

Figure II: MCNC Benchmark Circuit I7, a Large XOR/MUX Circuit in ILOC Vector Format, Primary and Internal Vector Components

Vector IDs		Signal IDs	Vector Sizes
(Vo2-	Vout2	<pn..qo>)	-- 28
(Vo3-	Vout3	<oh..pi>)	-- 28
(Vo4-	Vout4	<ob oc>)	-- 2
(Vo5-	Vout5	<od..og>)	-- 4
(Vo6-	Vout6	<pj..pm>)	-- 4
(V2-	Vtop2	<1..28>)	-- 28
(V32-	>V32	<>214..>241>)	-- 28
(V33-	=V33	<=113.. =140>)	-- 28
(V34-	>V34	<>173..>200>)	-- 28
(V41-	Vi41	<gp..hq>)	-- 28
(V42-	Vi42	<fj..gk>)	-- 28
(V11-	Vi11	<dz..ec>)	-- 28
(V3-	>Vtop3	<>33..>41 >46..>64>)	-- 28
(V31-	>V31	<>201..>209 >246..>264>)	-- 28
(V19-	=V19	<=81..=108>)	-- 28
(V23-	>V23	<>141..>168>)	-- 28
(V10-	Vi10	<al..bm>)	-- 28
(V12-	Vi12	<cx..dy>)	-- 28
(V13-	Vi13	<br..c2>)	-- 28
(V4-	Vtop4	<65 66>)	-- 2
(V17-	=V17	<=75 =76>)	-- 2
(V29-	Vi29	<af ag>)	-- 2
(V5-	Vtop5	<67..70>)	-- 4
(V30-	>V30	<>242..>245>)	-- 4
(V16-	^V16	<^71..^74>)	-- 4
(V18-	=V18	<=77..=80>)	-- 4
(V7-	Vi7	<ct..cw>)	-- 4
(V8-	Vi8	<bn..bq>)	-- 4
(V9-	Vi9	<ah..ak>)	-- 4
(V6-	>Vtop6	<>42..>45>)	-- 4
(V28-	>V28	<>210..>213>)	-- 4
(V35	=V35	<=109..=112>)	-- 4
(V25-	>V25	<>169..>172>)	-- 4
(V39-	Vi39	<gl..go>)	-- 4
(V40-	Vi40	<ff..fi>)	-- 4
(V43-	Vi43	<ed..fe>)	-- 4

Figure III: MCNC Benchmark Circuit I7, Vector Modularization

Preparation

```
((oa 1)(Vout2 Vtop2)(Vout3 Vtop3)
 (Vout4 Vtop4)(Vout5 Vtop5)(Vout6 Vtop6))

((1        (ad (ab)(ae))   )                                  --  1
 (Vtop4    ((ac)(ad (ab)(Vi29)))     )                        --  2
 (Vtop2    (V91)   )                                          -- 28
 (Vtop3    (((aa (ac)) ((aa) V92)))   )                       -- 28
 (Vtop5    (((ab)(V93)))   )                                  --  4
 (Vtop6    (((aa)(V94)))   )                                  --  4

 (V91  ((ad (ac Vi11) ((ac)(Vi11))) ((ad) (ac Vi41) ((ac) Vi42)))   )
 (V92  ((ad (ac Vi10) ((ac)(Vi10))) ((ad) (ac Vi12) ((ac) Vi13)))   )
 (V93  ((ad (ac  Vi9) ((ac)( Vi9))) ((ad) (ac  Vi7) ((ac)  Vi8)))   )
 (V94  ((ad (ac Vi43) ((ac)(Vi43))) ((ad) (ac Vi39) ((ac) Vi40)))   )  )
```

Modularization

```
  ((oa 1)(Vout2 Vtop2)(Vout3 Vtop3)
   (Vout4 Vtop4)(Vout5 Vtop5)(Vout6 Vtop6))

  ((1       (ad (ab)(ae))   )                                  --  1
   (Vtop2   (V91-0)   )                                        -- 28
   (Vtop3   (((aa (ac)) ((aa) V92-0)))   )                     -- 28
   (Vtop4   ((ac)(ad (ab)(Vi29)))     )                        --  2
   (Vtop5   (((ab)(V93-0)))   )                                --  4
   (Vtop6   (((aa)(V94-0)))   )                                --  4

   (V91-  vector-module ((Va Vi11)(Vb Vi41)(Vc Vi42)) ((Vo V91-0))   )
   (V92-  vector-module ((Va Vi10)(Vb Vi12)(Vc Vi13)) ((Vo V92-0))   )
   (V93-  vector-module ((Va Vi9) (Vb Vi7) (Vc Vi8))  ((Vo V93-0))   )
   (V94-  vector-module ((Va Vi43)(Vb Vi39)(Vc Vi40)) ((Vo V94-0))   )  )

   ((vector-module)
    ((Va unk)(Vb unk)(Vc unk))
    ((Vo 1))
    ((1   ((ad (ac Va) ((ac)(Va))) ((ad) (ac Vb) ((ac) Vc)))   )  ))
```

Cell abstraction of the vector module body

```
        ((1   ((ad (ac Va) ((ac)(Va))) ((ad) (ac Vb) ((ac) Vc)))   )  )

  ==>   ((>1   ((ad =2) ((ad)(>3)))   )
         (=2   (((ac Va) ((ac)(Va))))   )
         (>3   ((ac Vb) ((ac) Vc))   )  )
```

Figure IV: MCNC Benchmark Circuit I7, Matrix Abstraction

Preparation

```
((oa 1)(Vout2 Vtop2)(Vout3 Vtop3)
 (Vout4 Vtop4)(Vout5 Vtop5)(Vout6 Vtop6))

((1        (ad (ab)(ae))   )                                  --  1
 (Vtop4    ((ac)(ad (ab)(Vi29)))     )                        --  2
 (Vtop2    (V91)   )                                          -- 28
 (Vtop3    (((aa (ac)) ((aa) V92)))   )                       -- 28
 (Vtop5    (((ab)(V93)))   )                                  --  4
 (Vtop6    (((aa)(V94)))   )                                  --  4

 (V91  ((ad (ac Vi11) ((ac)(Vi11))) ((ad) (ac Vi41) ((ac) Vi42)))   )
 (V92  ((ad (ac Vi10) ((ac)(Vi10))) ((ad) (ac Vi12) ((ac) Vi13)))   )
 (V93  ((ad (ac  Vi9) ((ac)( Vi9))) ((ad) (ac  Vi7) ((ac)  Vi8)))   )
 (V94  ((ad (ac Vi43) ((ac)(Vi43))) ((ad) (ac Vi39) ((ac) Vi40)))   )  )
```

Matrix abstraction

```
((M1-    M1  <Vi11 Vi10 Vi9 Vi43>   )                         -- 28
 (M2-    M2  <Vi41 Vi12 Vi7 Vi39>   )                         -- 28
 (M3-    M3  <Vi42 Vi13 Vi8 Vi40>   )                         -- 28
 (M4-    M4  <V91 V92 V93 V94>   )                            -- 28

((31       (ad (ab)(ae))   )                                          --  1
 (Vtop4    ((ac)(ad (ab)(Vi29)))     )                               --  2
 (Vtop2    (M4-1)   )                                                 -- 28
 (Vtop3    (((aa (ac)) ((aa) M4-2)))   )                              -- 28
 (Vtop5    (((ab)(M4-3)))   )                                         --  4
 (Vtop6    (((aa)(M4-4)))   )                                         --  4
 (M4       ((ad (ac M1) ((ac)(M1))) ((ad) (ac M2) ((ac) M3)))   )  )  -- 4x28
```

Cell abstraction of the ILOC matrix body

```
((31       (ad (ab)(ae))   )                                  --  1
 (Vtop4    ((ac)(ad (ab)(Vi29)))     )                        --  2
 (Vtop2    (M4-1)   )                                         -- 28
 (Vtop5    (((ab)(M4-3)))   )                                 --  4
 (Vtop6    (((aa)(M4-4)))   )                                 --  4
 (>Vtop3   (((aa (ac)) ((aa) M4-2)))   )                      -- 28
 (>M4      ((ad =M5) ((ad)(>M6))   )                          -- 4x28
 (=M5      (((ac M1) ((ac)(M1))))   )                         -- 4x28
 (>M6      ((ac M2) ((ac) M3)))   )  )                        -- 4x28
```

Figure V: MCNC Benchmark Circuit I7, Matrix Abstraction Schematic

Matrix schematic for the I7 benchmark circuit,