Data Path Placement with Regularity

Terry Tao Ye and Giovanni De Micheli
Department of Electrical Engineering, Stanford University, Stanford, CA 94305
Email: {taoye, nanni}@stanford.edu

Abstract

As more data processing functions are integrated into systems-on-chip, data path is becoming a critical part of the whole VLSI design. However, traditional physical design methodology can not satisfy the data path performance requirement because it has no knowledge of the data path bit-sliced structure. In this paper, an Abstract Physical Model (APM) is proposed to extract bit-slice regularity information from Data Flow Graph (DFG) and it is used for interconnect and congestion planning. A two step heuristic algorithm is introduced to optimize the linear placement of APM to satisfy both the wire length and routing track budget.

1 Introduction

As today's VLSI design moves to the deep sub-micron domain, more and more functions are embedded into a single chip. The emerging of DSP application and digital communication in recent years also brings more complicated data processing operations onto silicon. Data path is becoming the most critical part of the system. However, high performance data-path design is still very time consuming, especially in the mostly hand-crafted physical design stage. Traditional ASIC design tools often generate inferior results on data path circuits compared to those designed manually, because they can not exploit the unique structure of data path: the *regular bit-sliced structure*.

Data path circuits placed by traditional physical design tools are likely to be congested during routing, especially when the data path is getting wider (64 bits and more). As the interconnect delay dominates the timing in the deep sub-micron domain, signal and clock skews vary randomly among different bits and the wire delay is hard to estimate accurately. The timing closure between synthesis and placement has become a serious problem. This makes the regular placement critical for data-path design. A regularly placed data path circuit will eliminate the clock/signal skews among different bits. It will create well planned routing for interconnects to reduce congestion. More important, it will generate a predictable interconnect scheme. Accurate interconnect delay estimation can be predicated in synthesis stage. The so-called synthesis-placement gap could be under control.

Data path related research could be categorized into four areas:

(1) Data path Synthesis: It takes behavioral level HDL or Data-path Description Language (DDL) as inputs. The design is synthesized for resource sharing, scheduling, and operator optimization (e.g. CSA/CPA merging in MAC) [4]. However, it only generates circuit netlist, with no or little information to guide the placement.

(2) Module Generation: It is focused on the layout generation at the transistor level. Askar et al. [1], Serdar et al. [2], and Kim et al. [3] proposed different techniques for transistor folding and merging. These techniques are mainly used for customized designs and they can not fit into ASIC design flows, where standard cells are used as building blocks.

(3) Regularity Extraction: Different extraction techniques were proposed by Arikati et al. [6], Nijssen et al. [5], and Hassoun et al. [7], which use template matching methods to find common structures inside a netlist. However, in ASIC design flow, when netlist is generated, the data path structural information is already lost. Even if some regular pattern exists, the functional information is difficult to retrieve.

(4) Data path module placement: The research carried on this subject either focused on the reduction of track density [8] [9] [10], or congestion and total wire length minimization [11] [12]. It was assumed that numbers of available routing tracks are the same for all modules. While total wire length is minimized, there is no control on the length of each interconnect.

Here we propose a new method to generate an optimized data path placement with the regularity information extracted directly from Data Flow Graph (DFG). It takes both the interconnect delay and bus congestion constraints into account and creates a bit-sliced placement to satisfy the timing and routing track budget. It defines a global placement for circuit blocks, which can be directly used by detailed placement steps that follow. This proposed methodology could well fit into current ASIC design flows. There are two contributions in this work. (1) Abstract Physical Model (APM). Abstraction of the regularity information of one bit slice of data path, which includes both the functional connectivity and interconnects congestion estimation during routing. (2) A two step heuristic method to find an optimal linear relative placement of data path building blocks.

The paper is organized as follows: Section 2 discusses data path placement styles and why regularity is necessary for data-path design. Section 3 analyzes why the traditional data-path design flow can not satisfy the performance requirement. Section 4 gives details on how to model a data path into its Abstract Physical Model. In Section 5, a two step heuristic method is used to solve the linear ordering problem of APM circuit blocks. The results of some benchmark data-path designs are shown in Section 6 and conclusions will be drawn in Section 7.

2 Regularity of Data Path

Data path is the circuit performing bit-wise data operations in parallel on multiple bits. Each operation is corresponding to a dedicated functional block, such as adder, register, buffer, multiplexer, multiplier, etc. Inter-bit connections might exist among the bit wise operations (e.g. carries in an adder). There are two groups of interconnects flowing in perpendicular directions, as shown in Figure 1. One is the data flow, which runs parallel horizontally. The other is control flow, which goes vertically. Control flow can either be global control signals which operate on every bit simultaneously (e.g. CLOCK signal, SEL of a MUX array), or local control signals which operate on adjacent bits (e.g. CARRY-IN/OUT).

Data path regularity has some unique features:

a) Data path circuit is best placed in a bit-sliced structure

0-7803-6445-7/00/$10.00 © 2000 IEEE

[13]. The cells operating on one bit are placed in one row abutted next to each other, assuming they are placed horizontally. This row structure is repeated for different bits vertically. The benefits of such a bit-sliced structure are:

- Different bit slice rows have the same height and width.
- Control signals (both locally and globally) can be aligned vertically

b) Interconnects of one bit data flow are routed horizontally inside the corresponding bit slice row. Even if there are loops existing in the data flow, the routing is still confined within that bit slice, as shown in Figure 1.

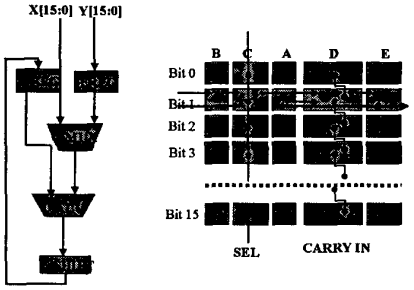

Figure 1: **Regularity Placement and Routing of Data path Circuit**

Regular placement and routing are necessary for data path performance [14]. As the wire load on interconnects becomes more and more significant, non-regular interconnects could create long wire delays as well as big timing variations among different bits. Both will greatly deteriorate the circuit performance. By placing the data path in a bit-sliced pattern, data flow schemes among different bits become identical. Not only will the signal and clock skews be eliminated, the interconnects and buses can also be better planned in advance to control their length and congestion. It becomes possible for data path synthesis tools to make an early estimation on the interconnect delays for each bit. This will in turn greatly improve the accuracy of synthesis result.

Regular structured placement is not only area efficient, with a recognizable structure, it is also easy to modularize as an IP block for design reuse.

3 Traditional ASIC data-path design flow

In today's ASIC design flow, the data path circuit is generated separately from other parts of the design. The data path is first described in a special Data-path Description Language (DDL) [4] or in behavioral HDL. The user is required to specify explicitly what kind of implementation he/she needs for each operation. For example, the user needs to specify a Ripple-Carry-Adder (RPA) or a Carry-Lookahead-Adder (CLA) for an adding operation, a Booth Multiplier or a CSA-Array-Multiplier for a multiplying operation. The target implementation is normally in-lined as pragmas in the HDL or DDL. The data path compiling tools will then generate the circuit according to the pragmas and create circuit netlist as output. (Figure 2)

The data path netlist generated is then labeled with a don't-touch attribute and merged with netlists of other circuits in the design. The whole netlist is fed into physical design tools to perform placing and routing. Because the placement tool has no knowledge of the data path structure, it has little control of the exact location where a cell might be placed. The bit-sliced structure and regularity information will be lost.

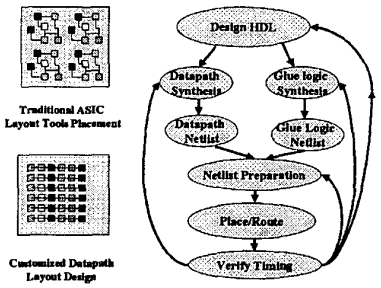

Figure 2: **Traditional Data-path Design Flow**

The placement has no guarantee of interconnect length and signal/clock skew. The routing is easy to get congested especially in the data path area. If performance requirement can not be met, these steps have to be iterated several times until timing closure is achieved. (Figure 2)

The problem in the above mentioned design flow lies in the gap between placement and synthesis of data path circuits. The synthesis has no control of placement and the placement has no knowledge of the data path structure.

The analysis above shows a placement tool needs to exploit the bit-sliced information from the Data Flow Graph (DFG) before it is lost in merging with other circuits. In the next section, a new modeling method will be introduced to extract the bit-sliced information directly from DFG. Both wire length and congestion could be planned in this model for data path placement.

4 Abstract Physical Model (APM)

4.1 Model of one bit slice

Abstract Physical Model (APM) is the bit-sliced abstraction of a data-path circuit. It consists of circuit blocks (CB) corresponding to one bit operations in the data path. The circuit block is the basic building unit of APM. It can either be a standard cell from the library, e.g. AND, OR, XOR, DFF, Full Adder, or it can be a data-path leaf cell or group of standard cells, e.g. 1-bit multiplier, Booth MUX. In Abstract Physical Model, each circuit block is represented as a rectangular box, with wires indicating its input and output interconnections. The height of the box is the height of corresponding standard cell or data path leaf cell, and the width is the cell width or the sum of widths of the cells in the block.

Some examples of the circuit blocks are shown in Figure 3. They could have different configurations.

1) 1:1 In-and-Out Like a D-Flip-Flop, the data flow is just a simple in/out format.

2) Interlacing (2:1 or N:1) For data operations like bit-wise AND/OR or multiplexing, the circuit is an array of cells for one bit operation with interlacing inputs. Its circuit block model is represented as two inputs of the same bit, and one output. This configuration can also be used for bit-wise operations on multiple inputs, where the circuit block should be modeled as N:1 interlacing.

3) N:M The Carry-Save-Adder (CSA) or full adder (FA) take three inputs and generate two outputs, A 4:2 adder (composed of two FAs) takes four inputs and generates two outputs. They can be modeled as the N:M configuration.

4) Orthogonal For a multiplication circuit, the two data flows for multiplier and multiplicand are best routed perpendicular to each other. This not only eliminates the routing congestion, but also yields uniformed propagation delays among

different bits. The circuit block of one-bit operation is modeled as a box with two inputs from orthogonal directions. One is assigned as the main branch data flow, which is the data flow running throughout the whole data path and lies in the same direction as the output data flow. The other will be the side branch data flow, which is normally the constant, or the coefficients fetched from memory units.

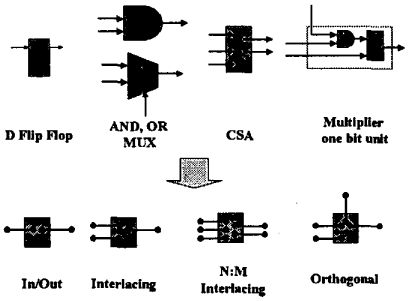

Figure 3: **Circuit blocks configuration for APM**

The number of wires coming in and out of a circuit block is the average number of data flow per bit slice. A portion of CSA array multiplier is shown in Figure 4. For the CSA block of bit slice **n**, one input is from the AND cell in the same stage, one input is the SUM output from CSA of previous stage of the same bit slice, another input is from the CARRY output of previous staged CSA, but of the bit slice **n-1**. One output of this block is going to the CSA input of the same bit slice in the next stage, but the other output is going to the CSA input of the next bit slice **n+1**. Considering average data flow per bit slice, this CSA block should be modeled as 3-input 2-output block.

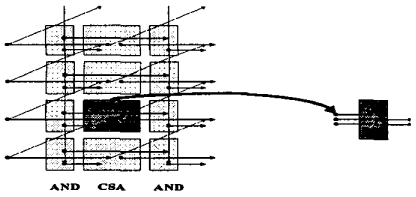

Figure 4: **Interconnect Modeling of a circuit block in APM**

Here should be noted that for the bit slice of LSB and MSB of a data path, there are always fewer interconnects compared to the bit slice in the center. The Abstract Physical Model might over estimate the interconnect numbers on LSB and MSB. This will not be a problem since over estimation in this case will yield a more conservative result.

4.2 Some Abstract Physical Model Examples of data path circuit module

1)Array Structure A register array can be simply modeled as a 1:1 in/out block, the AND/MUX array can also be modeled as a 2:1 interlacing block. (Figure 5)

2)CSA Array Multiplier A portion of CSA Array multiplier is shown in Figure 6. As in most applications, only the MSBs are going to output. This can keep the data path width uniform throughout the data flow. The APM extraction of a CSA array multiplier is a chain of AND blocks and CSA blocks. As mentioned earlier, the interconnect number between the blocks is the average number of interconnects per

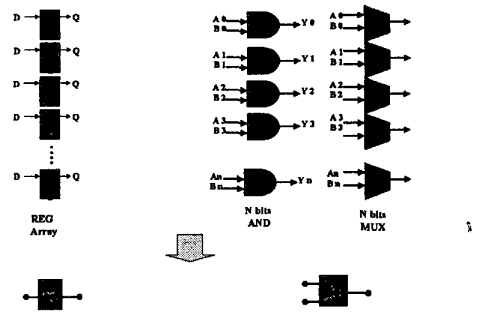

Figure 5: **Abstract Physical Model for an Array Structure**

bit slice. As shown in Figure 6, two adjacent CSA have two interconnects connected. The CSA on the first stage is omitted because it is not needed for the first bit multiplication.

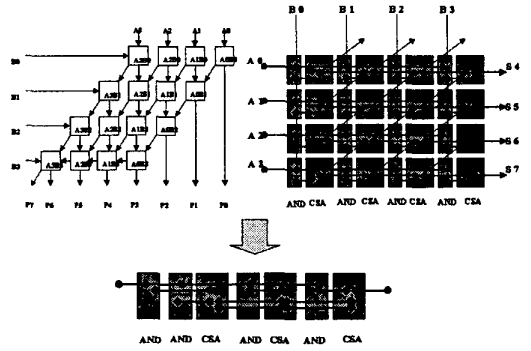

Figure 6: **Abstract Physical Model for a CSA-Array-Multiplier**

3)Booth Multiplier Booth multiplier uses the Booth encoding to reduce the partial products by half. The Booth-MUX will generate the partial products of +2, +1, 0, -2, or -1 times the multiplicand according to the encoded bits. The sum of partial products is added up by the CSA array. The first stage CSA takes the first three partial products and generates SUM and CARRY as outputs. Each CSA in the following stages will take one more partial product plus one SUM from previous CSA of the same bit and CARRY from CSA on the one-less-significant bit slice. In the last stage, a carry-propagation-adder (CPA) is used as the vector merger to merge the carries and sums. The APM extraction for one bit slice will be a chain of Booth-MUX and CSA, and one CPA in the final stage. (Figure 7)

4)Carry Look Ahead Adder (CLA) Carry look-ahead adder normally consists of three stages. 1) The PG generation block. 2) The carry look-ahead logic, and 3) The final SUM generation, which is normally a XOR gate of Carry and P. While it is fairly straightforward to extract one bit slice for PG generation and SUM generation blocks, it is hard to decompose a CLA logic generation circuit into bit-wise blocks, because the carries in CLA logic are calculated in an incremental manner. However, since the number of cells in CLA logic can be pre-calculated, a place holder block for the CLA logic can be reserved with the area estimated to hold all the CLA logic cells. The number of rows covered by the place

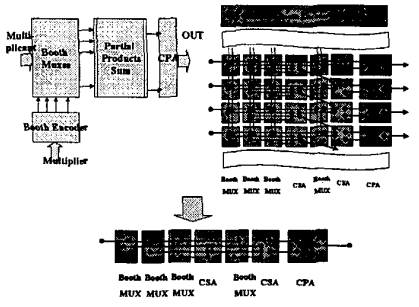

Figure 7: **Abstract Physical Model for a Booth Multiplier**

holding block is the bit width of the data path to align with other circuit blocks. In this way, we can use the width of the CLA place holder block as the average circuit block width per bit slice in the APM, as shown in Figure 8.

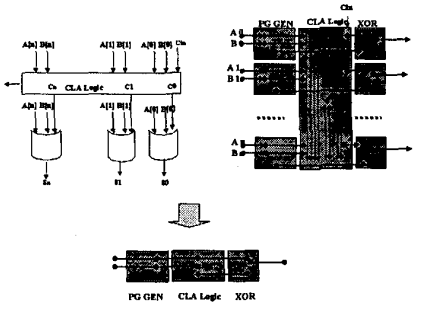

Figure 8: **Abstract Physical Model for a Carry-Lookahead Adder**

4.3 Wire Length Budget and Routing Track Budget

Wire Length Budget Wire length budget is the maximum wire length allowed for each net; therefore its interconnect delay will not exceed the timing slack of the corresponding path. Consider the logic network of Figure 9. Each node is corresponding to a data operation, or circuit block inside one bit slice. During data path synthesis, a timing slack has already been calculated for each node (e.g. $S_h = 110ps, S_k = 40ps$). Without loss of generality, we assume that the slacks are the same on all bit slices. If this assumption does not hold, then the worst timing slack on the corresponding nodes of all the bits should be considered. This slack could be used as the interconnect delay allowed for the net connecting them (net between V_h and V_m, V_k and V_l). In order to calculate the wire length budget for each net, an RC delay-length look-up table is pre-calculated for the interconnect based on the process and design rules. Wire length budget could then be estimated directly from this look-up table.

Routing Track Budget Horizontal wires are contributed in two ways; 1) the IN/OUT connections between circuit blocks, and 2) the wires feeding through a circuit block. The total count of interconnects is the sum of the two, affected by the relative placement or the ordering of the blocks. An example is given in Figure 10, there are three inputs and two outputs plus three feeding-through wires on circled block.

As discussed in Section 2, all wires connecting circuit blocks of one bit should be routed over the cells within the bit slice.

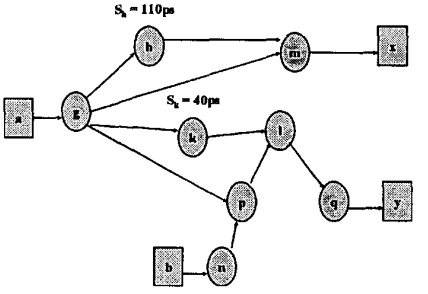

Figure 9: **Wire length budget can be calculated from timing slack**

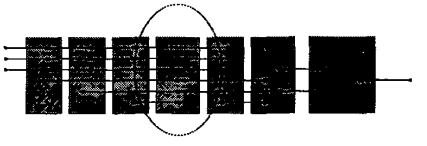

Figure 10: **Congestion estimation of APM**

However, there is a limit on the routing tracks available vertically and horizontally. Normally, vertical interconnects used by control signals are less than the horizontal ones used by data flow. When the number of horizontal interconnects exceeds the track capacity within the cell height, extra tracks have to be allocated between bit slice rows. This situation is called congestion overflow. The height of the bit slice row is determined by the cell height plus the overflow tracks. It can be written as:

$$Bitslice\_Height = Height_{cell} + N_{overflow} * Pitch_{track} \quad (1)$$

Where $N_{overflow}$ is number of overflow routing tracks. Since the width of the data path is fixed by the abutment of circuit blocks, the height of bit slice row will then determine the area of the data path module.

The maximum number of horizontal interconnects allowed to route on each circuit block is called routing track budget. It is the total number of interconnects for IN/OUT pin connection, plus the number of tracks available for feed-through. If the height of data path module is fixed, the routing track budget is then determined by: 1)The physical structure of each blocks, e.g. the internal wires, and pin locations. 2) The relative locations of other blocks connected to this block. As shown in Figure 11, we assume the circuit block allows a maximum of 12 horizontal routing tracks. In the first case, both inputs and outputs are from the same side, four tracks are used by IN/OUT interconnects, the available feed-through tracks are 8, the routing track budget is 12. In the second case, inputs and outputs are from opposite sides, only two tracks are used for IN/OUT connection, the routing track budget will thus be 14. To simplify the APM modeling, we like the routing track budget to be determined only by the internal structure of the circuit block itself, which is also the worst case estimation (12 in this example).

4.4 Create APM from HDL

With the analysis above, here we propose a new data-path design flow (Figure 12). The data-path design is first described in HDL (or DDL). Data path circuit is then compiled according to the pragmas in the HDL/DDL module, along with the

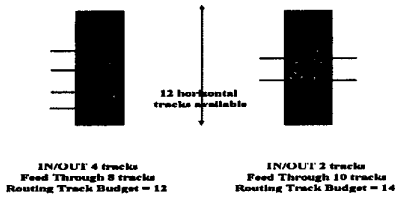

Figure 11: **Congestion estimation of one circuit block**

cell logic and physical information from the library. Also based on the pragma specification, APM of the data path is created. Timing slack is calculated for each node by the synthesis tool. This slack will be translated into wire-length budget for each net connecting the circuit blocks in APM. The physical information can also be retrieved from the cell library and a routing track budget could be estimated for each circuit block. Using the two budgets as constraints, optimization is performed on APM to find an optimal placement solution.

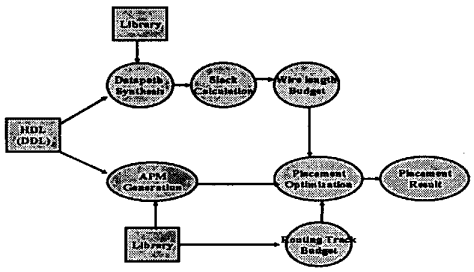

Figure 12: **Proposed data-path design flow**

A MAC (multiplication accumulation) design is given in Figure 13 as an example. The APM for each data operation is generated based on the pragma specified in HDL (or DDL). The data flow between circuit blocks is extracted in DFG. The APM for the whole data path is then created.

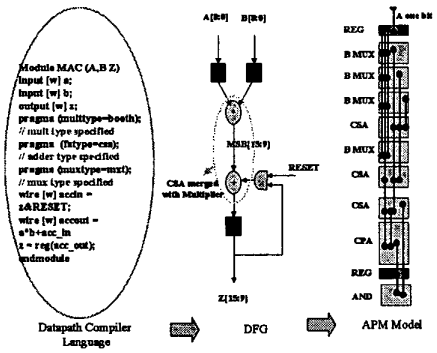

Figure 13: **Modeling of a MAC circuit**

After the data path is abstracted into its APM, data path placement problems become the linear placement problems of the circuit blocks. As shown in Figure 14, both wire length violation and routing congestion violation exist in Scheme (A). In placement Scheme (B), the long wire was shortened. Scheme (C) further eliminates the congestion over cell A.

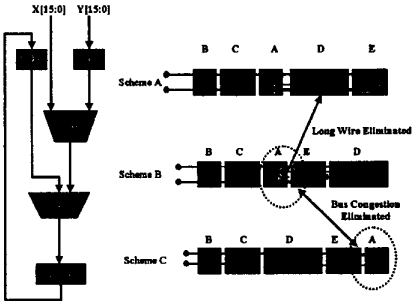

Figure 14: **Different placement plannings may eliminate wire length violations and congestion violations**

Based on the above analysis, an APM circuit block placement should satisfy the following constrains:

1. Place the circuit blocks in one row, where the wire length of each interconnect should not exceed the wire length budget.
2. Plan the interconnects routing horizontally, where the wire congestion should not exceed the routing track budget of each block.

5 Placement of circuit blocks in APM

5.1 Problem Formation

We are given a set of m circuit blocks $V = \{v_1, v_2, ..., v_m\}$ and a set of n interconnects $N = \{n_1, n_2, ..., n_n\}$. Each circuit block $v_i \in V$ has its own routing track budget r_i and is connected by a subset of interconnects $N_{v_i} \subset N$. Each interconnect $n_j \in N$ has its own wire length budget l_j and is connecting a subset of circuit blocks $V_{n_j} \subset V$. The problem is to find a linear placement of the V such that all the interconnects satisfy the wire length budgets, and all the blocks satisfy the routing track budgets.

$$Track(v_i) < r_i; \forall v_j \in V; 1 \leq i \leq m \quad (2)$$

$$Length(n_j) < l_j; \forall n_j \in N; 1 \leq j \leq n \quad (3)$$

In data-path design, the circuit blocks are placed abutted to each other. This makes the linear placement problem equivalent to the linear ordering problem, which is known to be *NP-hard* [15]. Many heuristic methods have been proposed on this issue. Kang [16] proposed a method to start the ordering process from the most lightly connected seed. Hur et al. [17] introduced a relaxation and clustering technique for linear placement. Yim et al. [11] used genetic algorithm for the initial global ordering and then use the simulated annealing for further optimization. All these algorithms were focused on the global interconnect planning. They assumed all the locations (blocks) have the same routing track budget. However, in reality, some blocks might be more routable than others, because they have different internal structures. The above-mentioned heuristic methods also focused on the total wire length minimization, and they had no guarantee on each wire to be routed within wire length budget.

Here we propose a two step heuristic method to solve the linear ordering problem. A quadratic objective function is first minimized to find the initial ordering of the circuit blocks. The relative locations of the blocks will be based on the timing budget of each interconnect. Then a sliding window optimization is performed on the initial ordering to solve the wire length violations on each interconnect as well as congestion violations on each circuit block.

5.2 Quadratic Placement

Giving n blocks $v_i \in V, 1 \leq i \leq n$,with locations on x_1, x_2, x_n, the total weighted square wire length objective function can be expressed as

$$\Phi(x) = \sum_{i,j=1}^{n} w_{ij}(x_i - x_j)^2 = \mathbf{x}^{\mathbf{T}}\mathbf{Q}\mathbf{x} \qquad (4)$$

where the x_i, x_j are the locations for block v_i and $v_j \in V$, and $\mathbf{x}$ is the linear vector for the location [18]. $\mathbf{Q}$ is the quadratic matrix for the weight factor w_{ij}, where w_{ij} is 0 if there is no connection between block v_i and v_j. The diagonal elements w_{11}, w_{22}, w_{nn}, etc. of the quadratic matrix are the negative sums of all the elements on the same row. When block v_i and v_j are connected, the value of w_{ij} should be a function of the wire length budget of the interconnect between v_i and v_j. Here we call this function $force(i,j)$.,

$$w_{ij} = \begin{cases} 0 & i \neq j \text{ and no interconnect between } v_i \text{ and } v_j \\ force(i,j) & i \neq j \text{ and } v_i, v_j \text{ are connected} \\ -\sum_{k=1,k\neq i}^{n} w_{ik} & i = j \text{ (The sum of } w_{ik} \text{ in the row i)} \end{cases}$$

The function *force(i,j)* is used to adjust the relative locations of v_i and v_j, and it should be a function to decrease as the wire length budget between v_i and v_j increases. In our experiment, we choose the simple reciprocal function

$$force(i,j) = \frac{c}{l(i,j)} \qquad (5)$$

where *l(i,j)* is the wire length budget between v_i and v_j, c is a constant to be determined by experiment.

Normally, the positions of some circuit blocks are pre-fixed before the placement, like the IO cells or the input/output register arrays. These blocks are denoted as $V_f \subset V$, and their corresponding location vector is denoted as $\mathbf{x}_f \subset \mathbf{x}$. Similarly, the locations of all the movable blocks are denoted as vector $\mathbf{x}_c \subset \mathbf{x}$. The objective function can then be re-written as:

$$\Phi(x) = (\mathbf{x_c} \quad \mathbf{x_f}) \begin{pmatrix} Q_{cc} & Q_{cf} \\ Q_{fc} & Q_{ff} \end{pmatrix} (\mathbf{x_c} \quad \mathbf{x_f})^{\mathbf{T}} \qquad (6)$$

Solving the zeros of the derivative of the objective function

$$\mathbf{Q_{cc}x_c} + \mathbf{Q_{cf}x_f} = \mathbf{0} \qquad (7)$$

which can be rewritten as ;

$$\mathbf{Q_{cc}x_c} = -\mathbf{Q_{cf}x_f} \qquad (8)$$

By solving this linear equation, we can get the solution for $\mathbf{x_c}$, which is the linear placement vector for all the movable circuit blocks. However, the solution for the linear equation is a real numbered value. The APM circuit blocks are placed by abutment. The solution in $\mathbf{x_c}$ can not be used directly as the actual locations for the blocks; instead, they should be interpreted as relative ordering of the blocks. By combining the movable vector $\mathbf{x_c}$ with the pre-fixed vector $\mathbf{x}_f$, an initial order of all the circuit blocks can be acquired.

Different algorithms could be used for the initial global ordering. Using quadratic objective function, the relative ordering of the circuit blocks could be easily adjusted by selecting different $force(i,j)$ functions. Quadratic objective function is also sensitive to long wires, which could be another benefit when long interconnect is a concern.

5.3 Optimization

The initial ordering solved by the quadratic objective function only indicates relative locations for the circuit blocks, which is constrained by the wire length budget of each interconnect. For example, when two blocks are connected by an interconnect having a long wire budget, they will be placed farther apart than the ones with tighter wire budget. However, the local ordering of the circuit blocks might not be optimal because quadratic placement has no knowledge of circuit block size and routing track limit, even though it can generate a good global placement ordering. A non-optimal local ordering might also occur when several blocks have similar values in the vector solution of the linear equation. A computation error could misplace them relative to each other.

Local optimization steps are needed after the initial ordering. A local optimum is determined when the following conditions are met: 1) By re-adjusting the order of cells, the interconnects violating the wire length budget are eliminated or their number is reduced. 2) The blocks violating the routing track budget are eliminated or their number is reduced.

Here we present a Sliding Window (SW) heuristic method for this optimization (Figure 15). At each time, n consecutive blocks are considered for optimization. These n blocks are covered by the Sliding Window of size n. The window is sliding from the first block. At each sliding stage, a search is performed within the window range to find the best local ordering, then the window is moved to the next block until the last block is covered. This process could repeat several times until no further improvement can be found. (Figure 15)

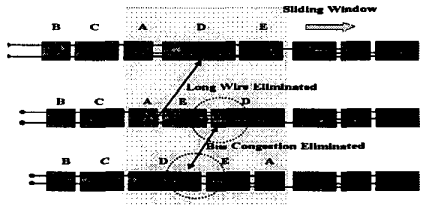

Figure 15: **Sliding Window (SW) Optimization**

Within each window boundary, the total width of the blocks is fixed and the change of the ordering within the window will not affect the positions of the blocks outside. A local optimal ordering will always exist which can give the best interconnect length and routing track congestion.

Different search algorithms can be used to search for a local optimal ordering within one window. When the window size n is small, an exhaustive search will suffice. If n is getting larger, a branch and bound algorithm can be used to reduce the search space. In order to lead quickly to a promising solution, the branching selection is based on the initial ordering of the circuit blocks inside the window. The bounding cost function *Current_Cost* is determined by the number of violations on both wire length budget and routing track budget. A better solution is found if the number of violations on either budgets is reduced. The best solution found so far is stored in the variable *Current_Best*.

In the next section, we will show in most applications, beyond a certain range, increasing the window size does not necessarily improve the result.

6 Experiments

The above mentioned methodology and algorithms have been implemented in C++ and benchmark data path circuits are tested. Each circuit is first abstracted into its APM and

then optimized under different wire length budgets with different window sizes. The results are compared in the tables. The CPU time is the running time on the Ultra 60 Sparc Station. The Max Path represents the maximum wire length in the critical path and different values are tested for different wire length budgets. The length is measured in λ which is the scalable unit. $\#V_{wire}$ and $\#V_{cong}$ are the numbers of violations of wire length budget and routing track budget respectively.

Table 1 is a 32-bit 4-Tap FIR filter in transposed form. The multiplication on each tap is implemented with a Booth multiplier. To reduce the carry propogation time delay, the carry and sum flows on each tap are not merged; instead, they propogate to the next tap in parallel. At the end of the flow, a CPA(Carry Propogation Adder) is used to merge the final carry and sum flows and generate the output. The data path is 3800λ in width and 3200λ in height and composed of about 20K gates. Its APM consists of 42 circuit blocks and 70 interconnects.

Table 2 is a 32-bit 8-Tap FIR filter in direct form. Again the multiplication is implemented with a Booth multiplier. The products of different taps are summed up by the CSA-Tree Adder. The design is composed of about 60K gates and its data path is 7530λ wide and 3200λ high. Its APM consists of 111 circuit blocks with 149 nets.

From these tables, we can see that the optimization will improve as the window size increases (n= 2, 3, 4). However, the window size above 5 will not improve the results. The experiments prove the initial ordering solved by the quadratic objective function generates a pretty good global placement. A small window size (in these cases, n = 5) will generate satisfying results.

Also shown in the tables, under tighter wire length budgets, violations still exist after optimization. This means the constraints are too aggressive. Buffers are needed to give bigger wire length budget or extra overflow tracks should be reserved for more routing spaces.

7 Conclusion

In this paper, a regular global placement methodology for data path is introduced. The interconnect length and routing congestion are planned in parallel with the data path synthesis. There are two contributions in this work: 1) An Abstract Physical Model to extract bit-sliced regularity information from DFG, 2) A two step heuristic algorithm to optimize the linear ordering of circuit blocks in APM. This methodology is tested on the benchmark data path circuits. The results show a feasible placement of the circuit blocks under different wire length budget and routing track budget constraints.

8 Acknowledgement

We would like to thank Felix Huang , Samit Chaudhuri, and Hamid Savoj at Magma Design Automation for their helpful suggestions. We would also like to acknowledge the reviewers for their valuable comments.

Table 1: Placement Results of 32-bit 4-Tap FIR

SW Size	MAX Path = 350λ			MAX Path = 300λ			MAX Path = 250λ		
	T_{CPU} (s)	$\#V_{wire}$ (viol)	$\#V_{cong}$ (viol)	T_{CPU} (s)	$\#V_{wire}$ (viol)	$\#V_{cong}$ (viol)	T_{CPU} (s)	$\#V_{wire}$ (viol)	$\#V_{cong}$ (viol)
0	-	4	1	-	6	1	-	14	1
2	4	2	1	3	4	1	3	9	1
3	5	0	0	4	3	1	6	9	1
4	6	0	0	5	3	1	7	6	0
5	11	0	0	12	0	0	12	6	0
6	52	0	0	54	0	0	57	6	0
7	377	0	0	367	0	0	385	6	0

Table 2: Placement Results of 32-bit 8-Tap FIR

SW Size	MAX Path = 700λ			MAX Path = 500λ			MAX Path = 400λ		
	T_{CPU} (s)	$\#V_{wire}$ (viol)	$\#V_{cong}$ (viol)	T_{CPU} (s)	$\#V_{wire}$ (viol)	$\#V_{cong}$ (viol)	T_{CPU} (s)	$\#V_{wire}$ (viol)	$\#V_{cong}$ (viol)
0	-	5	0	-	11	0	-	11	0
2	4	3	0	3	11	0	3	11	0
3	7	0	0	7	6	0	6	10	0
4	10	0	0	12	2	0	15	5	0
5	29	0	0	30	1	0	36	4	0
6	205	0	0	212	1	0	220	4	0
7	1460	0	0	1438	1	0	1485	4	0

References

[1] Askar, S.; Ciesielski, M. ; *Analytical approach to custom data-path design* Computer-Aided Design, 1999. Digest of Technical Papers. 1999 IEEE/ACM International Conference on , 1999 , Page(s): 98 -101

[2] Serdar, T.; Sechen, C. ; *AKORD: transistor level and mixed transistor/gate level placement tool for digital data paths* Computer-Aided Design, 1999. Digest of Technical Papers. 1999 IEEE/ACM International Conference on , 1999 , Page(s): 91 - 97

[3] Kim, J.; Kang, S.M. ; *A timing-driven data path layout synthesis with integer programming* Computer-Aided Design, 1995. ICCAD-95. Digest of Technical Papers., 1995 IEEE/ACM International Conference on , 1995 , Page(s): 716 -719

[4] Synopsys Module Compiler User Manual

[5] Nijssen, R. X. T. ; van Eijk, C. A. J. ; *Regular layout generation of logically optimized datapaths* Proceedings of the 1997 international symposium on Physical design , 1997, Page(s): 42 - 47

[6] Arikati, S.R.; Varadarajan, R. ; *A signature based approach to regularity extraction* Computer-Aided Design, 1997. Digest of Technical Papers., 1997 IEEE/ACM International Conference on , 1997 , Page(s): 542 -545

[7] Hassoun, S.; McCreary, C. ; *Regularity extraction via clan-based structural circuit decomposition* Computer-Aided Design, 1999. Digest of Technical Papers. 1999 IEEE/ACM International Conference on , 1999 , Page(s): 414 -418

[8] Luk, W.K.; Dean, A.A. ; *Multistack optimization for data-path chip layout* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on Volume: 10 1 , Jan. 1991 , Page(s): 116 -129

[9] Cai, H.; Note, S.; Six, P.; de Man, H. ; *A data path layout assembler for high performance DSP circuits* Design Automation Conference, 1990. Proceedings., 27th ACM/IEEE , 1990 , Page(s): 306 -311

[10] Buddi, N.; Chrzanowska-Jeske, M.; Saxe, C.L. ; *Layout synthesis for data-path designs* Design Automation Conference, 1995, with EURO-VHDL, Proceedings EURO-DAC '95., European , 1995 , Page(s): 86 -90

[11] Yim, J.-S.; Kyung, C.-M. ; *Data path layout optimisation using genetic algorithm and simulated annealing* Computers and Digital Techniques, IEE Proceedings- Volume: 145 2 , March 1998 , Page(s): 135 -141

[12] Nakao, H.; Kitada, O.; Hayashikoshi, M.; Okazaki, K.;Tsujihashi, Y. ; *A high density data path layout generation method under path delay constraints.* Custom Integrated Circuits Conference, 1993., Proceedings of the IEEE 1993 , 199 3 , Page(s): 9.5.1 -9.5.5

[13] Leveugle, R.; Safinia, C.; Magarshack, P.; Sponga, L. ; *Data path implementation: bit-slice structure versus standard cells* Euro ASIC '92, Proceedings. , 1992 , Page(s): 83 -88

[14] Ienne, P.; Griessing, A. ; *Practical experiences with standard-cell based data path design tools. Do we really need regular layouts?* Design Automation Conference, 1998. Proceedings , 1998 , Page(s): 396 -401

[15] Chowdhury, S. ; *Analytical approaches to the combinatorial optimization in linear placement problems* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on Volume: 86 , June 1989 , Page(s): 630 -639

[16] Kang, S. ; *Linear Ordering and Application to Placement* Proc. 20th Design Automation Conf., 1983, pp. 457-464

[17] Sung-Woo Hur; Lillis, J. ; *Relaxation and clustering in a local search framework: application to linear placement* Design Automation Conference, 1999. Proceedings. 36th , 1999 , Page(s): 360 -366

[18] Alpert, C.J.; Chan, T.; Huang, D.J.-H.; Markov, I.; Yan, K. *Quadratic Placement Revisited* Design Automation Conference, 1997. Proceedings of the 34th , Page(s): 752 -757