# A Timing-driven Data Path Layout Synthesis with Integer Programming

Jaewon Kim and S. M. Kang
Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 1308 W. Main St., Urbana, IL 61801

## Abstract

*We propose an efficient data path synthesis algorithm which generates bit-sliced layouts. Since data path circuits have special characteristics which are different from those of random logic circuits, the dedicated synthesis system is required for efficient layouts. Our main goal in the data path synthesis is to satisfy the timing constraints of circuits as well as to reduce layout areas. Timing-driven placement and over-the-cell routing techniques are developed to generate data path modules. Also, signal interfaces between bit-slices are carefully considered to further reduce layout areas. Our synthesis techniques take advantage of the common characteristics of data path structures under timing constraints and applies mixed integer linear programming approach to solve the problem. The superior results from our data path synthesis system are demonstrated through comparison with the layout results with the simulated annealing technique.*

## Introduction

In [1, 2, 3], data path layout synthesis systems were introduced to take advantage of common characteristics of data paths. The layout area minimization with the linear placement of function blocks was the major concern in the previous works. A data path module was assumed to be stacked bit-slices that have leaf cells in one-dimensional array. Even if time delay is an extremely important factor in data path design, they did not address the timing and the control signal interface issues, which may lead to the performance bottleneck.

In this paper, we present an efficient synthesis technique for data path modules. It uses unique performance-driven placement and routing that are specially developed to take full advantage of the common characteristics of regular data path structures. Our placement algorithm uses the path-based approach[4] which tries to minimize the most critical delay path under 0 timing requirements. To enforce timing requirements rigorously, path delays are calculated using the internal MOS channel resistance values of cells and the estimated capacitance values of routed nets. Our routing algorithm, in conjunction with the placement algorithm, tries to minimize the delays of individual nets to ensure the total path delays are within the limits. The physical and delay requirements are represented with linear equations in compact format through several steps to manage large data path circuits. We also pay special attention to signal interfaces between bit-slices, which are usually critical paths in data path modules.

## Data Path Synthesis

We assume that data signals flow horizontally, while control signals flow vertically. Figure 1 shows the usage of the horizontal and vertical signal flows in a four-bit data path left to right and top to bottom, respectively. We note that this characteristic, if harnessed properly, can contribute to the reduction of circuit delays. For the abutment of the duplicated rows, the interface between data path rows should allow direct signal flow without signal reordering.

Control signals that run vertically are divided into two categories in our data path synthesis. One is for *feed-through* signal and the other is for *carry-chain* signal. The feed-through signal

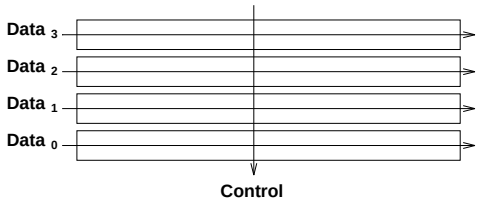
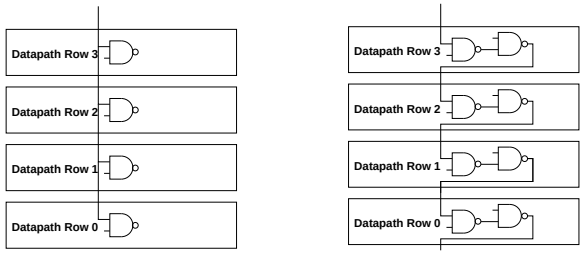



**Figure 1: Four-bit data path.**




(a) feed-through signal (b) carry-chain signal

**Figure 2: Vertical signals.**

is from the control logic unit that is usually located above a data path unit, but the carry-chain is for control flow or special data flow between data path units such as carry-in and carry-out in full adder implementation. Figure 2 shows examples of the feed-through signal and the carry-chain signal in a four bit data path. We note that the incoming and outgoing signal locations in both cases should be consistent for the direct interface between data path rows. In the carry-chain example of Fig. 2, the delay from the carry-input to the carry-output in a data path row may be relatively small. However, if we do not enforce carefully the timing constraints on the local delay, the delay will be amplified four times in the output of the least significant row, which can be unacceptable in global view.

## Placement

Since the net delay in the long path from input signal to output signal can be unacceptable for the given timing requirements, the net delay should be carefully considered in the placement step. In order to reduce the difficulty of the placement problem, we divide the entire problem into three phases; initial placement, slot assignment, cell row assignment.

In the first and second phases, we consider one-dimensional array as the platform for cell placement and try to find the best permutation of cells. In the third phase, each cell is allowed to move to the specific cell row, if the data path row has more than one cell row. We apply the mixed integer programming technique to the first and the second steps separately to utilize available integer programming packages. Suppose a circuit has $n$ basic cells. The objective of the placement is to optimally assign all the basic cells in the circuit into $n$ slots. If we apply the 0-1 integer programming technique to this permutation problem directly, $O(n^2)$ variables and constraints will be necessary to formulate the problem. We divide the integer programming process into smaller problems. The cell positions in one-dimensional array and the delay specifications are primar-

ily considered in the mixed integer programming. An instance of cell placement in a linear array is chosen among $n!$ solution space. However, some cells may share the same position due to the coarse resolution of the constraints. In the second phase, slot assignment, the placement conflict is resolved, where the constraints and the objective function can have finer resolutions. After the conflict resolution, the complete cell locations in a linear array are provided. In the third phase, each cell is assigned to a particular cell row by the row select algorithm, which minimizes unnecessary vertical connections.

## initial placement

Let $c_1, c_2, \cdots, c_n$ denote the cells in a circuit with inputs $f_1, f_2, \cdots, f_m$ and outputs $g_1, g_2, \cdots, g_l$. While the heights of the cells are kept to be the same, the widths of cells can be different from each other. In order to estimate the lumped capacitance value of a net, we define the average width as

$$\bar{w} = \frac{\sum_{i=1}^{n} width(c_i)}{n}.$$

Then, the corresponding one-dimensional array consists of $n$ slots each with width $\bar{w}$. To represent the position of the cells, $n$ integer variables $x_1, x_2, \cdots, x_n$ are defined to be the slot numbers for cells $c_1, c_2, \cdots, c_n$, respectively.

$$1 \le x_i \le n,\ 1 \le i \le n.$$

Since we assume that the primary inputs enter the left side of the data path and the primary outputs exit from the right side of the data path, the main signal flow is mostly in the left-to-right direction. To enforce the signal direction, the cell positions are ordered according to the signal flow.

While we restrict the signal flow in the data path, in some special cases the reversal of direction may be inevitable. Since the associated input and output positions of the carry-chain should be consistent, it is inevitable in this case that the signal flow should be allowed in the reverse direction. In order to represent the timing specifications of a circuit by linear algebraic equations, the delay models has to be established. For delay modeling, each CMOS gate is converted to its equivalent inverter with the corresponding parameters[5]. The parasitic capacitance of each net is assumed to be proportional to the net length. Propagation delay of the net is found based on the cell's load capacitance and inherent resistance using Elmore delay formula[6].

For the critical path delay modeling, we consider the propagation delay from the latest arrival input of the cell to its fanout cells through output[7]. Let $arr(c_i)$ denote the latest signal arrival time in $c_i$. The signal arrival times of the primary inputs are given in the timing specification of the circuit. In Fig. 3, the delay expressions for $c_r$ are denoted as

$$\begin{aligned} arr(c_r) &\ge arr(c_p) + del(c_P) + R_{eq,c_p}\left(\frac{\tilde{C}\bar{w}(x_{c_r} - x_{c_p})}{ROW}\right. \\ &+ \left. C_{int,c_p} + C_{gate,c_r}\right), \\ arr(c_r) &\ge arr(f_r) + R_{eq,f_r}\left(\frac{\tilde{C}\bar{w}x_{c_r}}{ROW} + C_{int,f_r} + C_{gate,c_r}\right), \end{aligned}$$

where $del(c_p)$ is the given internal delay of $c_p$, $ROW$ is the number of cell rows in a data path row, $C_{int,c_p}$ is the parasitic capacitance of cell $c_p$, $C_{gate,c_r}$ is the gate capacitance of cell $c_r$ and $C_{int,f_r}$ is the given capacitance of primary input $f_r$. Note net length term $\tilde{C}\bar{w}(x_{c_r} - x_{c_p})$ is divided by $ROW$ in the first equation.

In order to enforce the maximum path delay in circuits, the required signal arrival time for each primary output and carry-chain output is given. To ensure the required signal arrival time, each output should have timing constraint. In Fig. 3, the delay expressions are represented as

$$arr(g_q) \ge arr(c_q) + del(c_q) + R_{eq,c_q}\left(\frac{\tilde{C}\bar{w}(x_{g_q} - x_{c_q})}{ROW} + C_{int,c_q}\right),$$

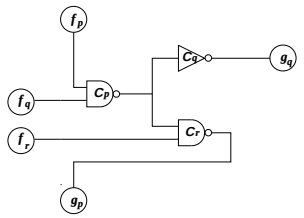



**Figure 3: Data path circuit example.**

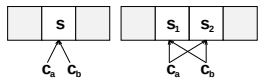
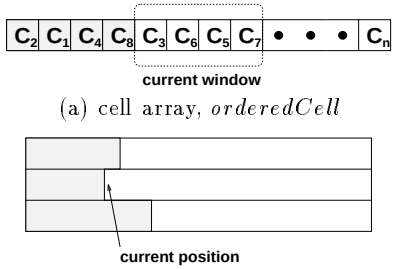



**Figure 4: Slot sharing.**




(a) cell array, $orderedCell$

(b) assigned cell rows

**Figure 5: Row assignment.**

$$arr(g_p) \ge arr(c_r) + del(c_r) + R_{eq,c_r}\left(\frac{\tilde{C}\bar{w}(x_{c_r} - x_{g_p})}{ROW} + C_{int,c_r}\right),$$
$$req(g_q) \ge arr(g_q) + MINSLACK,$$
$$req(g_p) \ge arr(g_p) + MINSLACK,$$

where $MINSLACK$ represents the minimum slack time in all path from the primary inputs to the primary outputs in the circuit. $MINSLACK < 0$ means the violation of the specified timing constraints. While we maintain $MINSLACK \ge 0$, we need to maximize $MINSLACK$ to reduce the critical path delay. With the above equations, the constraints are well established.

In the first phase of placement, both critical timing delay and the total net length should be minimized. $MINSLACK$ can represent the timing term, while the total net length can be induced to the total net delay. The objective function for the integer programming is

$$\text{maximize } MINSLACK - TOTAL\_NET\_DELAY,$$

$$\text{where } TOTAL\_NET\_DELAY = \frac{\tilde{C}\sum_i R_{eq,c_i}\,\text{net\_length}}{ROW}.$$

## slot assignment

While the placement result provides assignment of cell into one dimensional array slots, some cells may share the same slots due to the lack of conflict resolution in the initial placement phase. In order to solve this problem, we generate more detailed placement constraints and objective function and apply them to the 0-1 integer programming technique. Figure 4 shows the example of such slot sharing case in which slot $s$ is shared by two cells, $c_a$ and $c_b$.

Each shared slot is expanded to as many slots as the number of sharing cells. Then, we find the best permutation of the cells to fit them into the expanded slots.

## cell row assignment

The use of multiple cell rows in a data path row can cause in the reduction of the total net length. Even though the vertical delay in a data path row may be negligible, the vertical connection may cost another resource, pass-through between channels, which is limited according to the cell design. While the previous phases are mainly concerned with the net delay in the horizontal direction, in the cell row assignment the main concern is on the pass-throughs. Hence, cells should be placed in the multiple rows reducing the required number of pass-throughs.

The row assignment of cells is performed one cell at a time in the constructive manner from either side of the data path

row as shown in Fig. 5. Among the cells in the current window, the best cell for the current position is chosen and placed. The placed cell is removed from the window and the next candidate cell is inserted in the window.

```
RowAssignment(orderedCell):
    for every row
      initialize length[row] of each cell row to 0
    window = φ
    for i = 1 to WINDOWSIZE
      window = window ∪ PickNextCell(orderedCell)
    while window ≠ φ
      choose row with the minimum length[row]
      candidate = φ
      for every cell in window
        calculate weight(cell)
      choose cell with maximum weight(cell)
      assign cell to row
      length[row] = length[row] + width(cell)
      window = window − cell
      window = window ∪ PickNextCell(orderedCell)
    end while
```

**Figure 6: row assignment**

The row assignment algorithm is shown in Fig. 6. The one-dimensional cell array, *orderedCell* is obtained from the previous placement. *WINDOWSIZE* is the user-specified constant and not less than the number of cell rows in a data path row. In order to balance the length of the rows, the row with the least length is picked for the cell assignment. *window* is the set of the cells which are chosen from *orderedCell* according to the placement order. Given *WINDOWSIZE*, the cardinality of *window* is restricted to the number. To find the best cell for the chosen row, the calculation of weight on each cell should be achieved. The cell with the maximum weight value takes the slot in the row and it makes the row longer as much as the width of the cell. After a cell is assigned to a row, the cell is removed and the next available cell in *orderedCell* is appended to *window*. The time complexity of the row assignment algorithm is linearly proportional to the number of cells in the circuit, $O(n)$.

# Routing

Once each cell is placed in the row-based platform, the next step is to connect the nets between the terminals of the cells. Since we pursue the channel-less routing, the net routing is performed over the cell area with the upper two metal layers, which are metal2 and the metal3 for vertical connection and horizontal connection, respectively. As mentioned earlier, a data path row is a long series of cells. Thus, the capacity of the horizontal connection is limited in the over-the-cell area. Hence, the usage of horizontal tracks should be optimized in order to reduce the layout area. Our routing strategy is to assign one horizontal span for each net as much as possible, which guarantees that the signal delay on each net is minimized.

The routing step is composed of three steps; horizontal segment assignment, pass-through assignment, and detailed routing. The horizontal segment assignment is achieved by forming a horizontal metal segment for each net, which will expand the entire horizontal span of the net in a data path row. The vertical segment assignment covers the connection between the main horizontal segment and the terminals that are not adjacent to the horizontal segment. The first two steps are performed with linear programming technique, which attempts to maximize the efficiency of resource usage on the basis of placement result. If there exist some nets that are not completed in the steps, a maze router handles such nets one at a time. Sometimes, it may not complete the net connection due to the insufficient routing resource. In such a case, the horizontal track capacity or pass-through capacity needs to be expanded depending on the situation and rerun the routing step. After the global routing is done, the conventional channel routing technique is applied to achieve the detailed routing over the cell area.

## horizontal segment assignment

The main objective of the first step in routing is to assign at most one horizontal span to each net. The horizontal span of a net should cover horizontally all the terminals that are associated with the net. Since the height of a data path row is usually much smaller than the width, one horizontal span with some pass-throughs is enough for the connection of a net. This approach reduces the performance degradation which can be caused by the use of doglegs and vias.

The terminals are assumed to be available in the middle of the individual cells to maximize the OTC routing capacity. We define *channel* to be the routing space in the OTC area between two terminal rows. Figure 7 shows an example of four channels on the OTC area. Under our assumption on the positions of the terminals, the terminals should be aligned on the boundaries of the channels. The channel area is also divided vertically into *column blocks*. The size of each column block can be set to an arbitrary positive number. In Fig. 7, each column block provides three vertical tracks. Since the vertical tracks are dedicated to metal2 that is used for the terminals on the channel boundaries, the existence of a terminal reserves the vertical tracks for the corresponding net and terminal connection. If no terminal exists on a boundary of a vertical track, the position of the vertical track can be used as a pass-through between the channels. Initially, all pass-throughs are left unoccupied. Since our individual cell platform allows at least one pass-through in a cell, the size of a column block is typically set to

$$\max(width(c_i)),\ 1 \le i \le n,$$

to evenly distribute the pass-throughs over the column blocks. For the size of the column block described above, each column block can have at least one unoccupied pass-through.

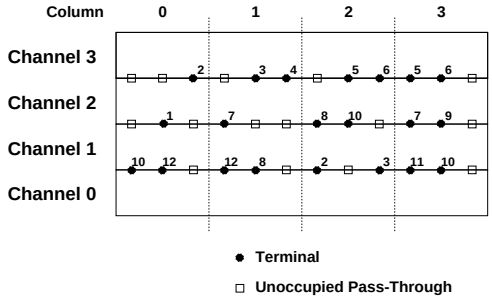



**Figure 7: Channels on data path.**

## pass-through assignment

Once the horizontal segments are assigned for the nets, the terminals should be connected to the associated horizontal segments to complete the net connection. If the associated terminal is on the boundary of the channel where the horizontal segment belongs, the vertical connection between the horizontal segment and the terminal will be made without any need of pass-throughs. Otherwise, the connection should be made through pass-throughs to penetrate the channel boundary. The pass-throughs are the valuable resource and their usage should be optimized.

Since no more than one path should be chosen among the possible routing paths, the following constraint is introduced.

$$\sum_k F_{i,j,k} \le 1,$$

for net $i$ that needs the terminal connection at column block $j$, with $k$ possible routing paths. Note that the terminal connection path may cost horizontal tracks in column blocks as well as pass-throughs. Since horizontal tracks and pass-through are limited, the constraints for the resource usage are specified as

$$\sum_i \sum_j \sum_k q_{i,j,k} F_{i,j,k} \le p_{u,u+1,v},$$

| circuit name | gate number | net number | Data Path Synthesis time (sec) | area ($\lambda^2$) | max path output($\lambda$) | max path carrychain($\lambda$) | SA time (sec) | area($\lambda$) ($\lambda^2$) | max path output($\lambda$) | max path carrychain($\lambda$) |
|---|---|---|---|---|---|---|---|---|---|---|
| circuit1 | 11 | 14 | 3.16 | 34496 | 352 | 288 | 9.9 | 34496 | 352 | 288 |
| circuit2 | 26 | 32 | 14.73 | 75264 | 768 | 1118 | 21.7 | 75264 | 1082 | 2193 |
| circuit3 | 33 | 42 | 8.9 | 53360 | 920 | 1204 | 25.6 | 60720 | 920 | 1204 |
| circuit4 | 61 | 66 | 9.15 | 120516 | 1824 | 2950 | 54.5 | 124032 | 1826 | 4322 |
| ti_alu | 36 | 50 | 29.93 | 105000 | 1250 | 1236 | 34.7 | 112437 | 2282 | 1594 |
| revised_ti_alu | 36 | 50 | 31.56 | 102384 | 1410 | 1268 | 34.3 | 111136 | 1544 | 1616 |

Table 1: Test results.

where $q_{i,j,k} = 1$ if and only if the path represented by $F_{i,j,k}$ needs any pass-through track between channel $u$ and $u+1$ on column block $v$, otherwise $q_{i,j,k} = 0$. Similarly,

$$\sum_i \sum_j \sum_k s_{i,j,k} F_{i,j,k} \leq t_{u,v},$$

where $s_{i,j,k} = 1$ if and only if the path represented by $F_{i,j,k}$ is routed on channel $u$ and column block $v$, otherwise $s_{i,j,k} = 0$. The track capacity $t_{u,v}$ should be reassessed after the horizontal segment assignment step to reflect the pre-routed horizontal segments.

The objective function for the 0-1 integer linear programming in the pass-through assignment step is described as

$$\text{maximize} \sum_i \sum_j \sum_k F_{i,j,k}(1 - \frac{f_{i,j,k} + g_{i,j,k}}{h_i}),$$

where $f_{i,j,k}$ denotes the number of pass-throughs that are necessary for $F_{i,j,k}$, $g_{i,j,k}$ denotes the number of column blocks where $F_{i,j,k}$ needs the horizontal tracks and

$$h_i = \sum_j \sum_k f_{i,j,k} + \sum_j \sum_k g_{i,j,k}.$$

## detailed routing

If all the appropriate connections are made with the previous steps, the conventional channel router can perform the detailed routing in each channel effectively. Since we use only metal2 and metal3 for the inter-cell routing purpose, the channel area on the OTC area can be considered as a conventional channel area, which is empty area between two rows of cells. If all connections are partially made due to resource conflict, the incomplete nets are routed one at a time by a maze router. Each block is regarded as a grid in the maze routing. When the maze router can not complete the connections due to the lack of resources, we have to provide additional horizontal tracks or pass-throughs depending on the situation and start the routing steps again. Based on the result, the complete routing is shown in Fig. 8.

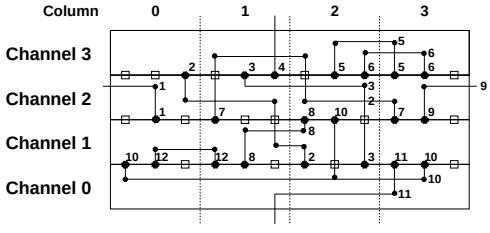



**Figure 8: Data path with complete routing.**

# Experimental Results

We have implemented the performance driven data path placement and routing algorithm with the linear programming technique in C on SPARCstation 10/30. It has been tested with several data path circuits that include MCNC benchmark circuits. The input circuit is described in net lists, gate type specifications and maximum timing delays between circuit inputs and outputs. The physical layout of each circuit has been generated. To compare the quality of the results, we have generated the physical layouts with the simulated annealing technique(SA). For fair comparison, the same layout technology using the fully customized cells and the triple layer over-the-cell routing technique was applied to both methods. The results of the circuits are summarized in Table 1.

The results in Table 1 are produced in one bit layout of each circuit. **ti_alu** and **revised_ti_alu** are from TI74181 ALU. The maximum path of output and the maximum path of carry-chain represent the most critical paths in the circuits, respectively. The distance and the area are measured in terms of $\lambda$ and $\lambda^2$ which are the scalable units. As shown in Table 1, the area differences are rather small, but the critical path delays behave quite differently, especially in the carry-chains. Furthermore, if the multiple bits of the circuits are stacked up, the differences in the total carry-chain delays will be much greater.

As the number of gates in a circuit increases, the cpu time of SA increases. However, the cpu time of our data path synthesis algorithm is more dependent on the circuit behavior rather than on the number of gates.

# Conclusion

In this paper, we have presented a new data path synthesis system that is based on the performance driven placement and routing algorithm with linear programming technique. Our algorithms take account of general characteristics of data paths to generate area-efficient layouts with satisfying given timing constraints. With the restraints on signal flows and track assignment policy, it guarantees minimal signal delays on each net connection. Our synthesis system is shown to generate superior results for regular data path circuits in view of both timing delay and area than the conventional layout approach. Since the data path synthesis has the special characteristics of long structure, carry-chain and feed-through handling which distinguish it from random logic synthesis, dedicated placement and routing schemes need to be used for the improvement of physical layout, circuit performance and the layout area. To generate more area-efficient data path layouts, data path modules with irregular structures should be considered in the future.

# References


[1] H. Cai et al.,"A data path assembler for high performance DSP circuits," *Proc. 27th DAC*, pp. 306-311, 1990.

[2] M. Shiochi et al., "New design approach for configurable data-path," *Proc. 1990 CICC*, pp. 14.5.1-4, 1990.

[3] Y. Tsujihashi et al.,"A high-density data-path generator with switchable cells," *IEEE J. Solid-State Circuits*, pp. 2-8, Jan. 1994.

[4] T. Gao et al., "A performance driven macro-cell placement algorithm," *Proc 29th DAC*, pp. 147-152, 1992.

[5] S. M. Kang and Y. Leblebichi, "CMOS digital integreated circuits: analysis and design," *McGraw Hill*, 1995.

[6] W. C. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifier," *J. Applied Physics*, vol. 19, Jan. 1948.

[7] M. A. B. Jackson and E. S. Kuh,"Performance-driven placement of cell-based IC's," *Proc. 26th DAC*, pp. 370-375, 1989.