AUTOMATIC PLACEMENT OF MICROPIPELINE STANDARD CELLS

Alaaeldin Amin
Computer Engineering Department,
King Fahd University of Petroleum & Minerals (KFUPM),
Dhahran, 31261,
SAUDI ARABIA
amindin@kfupm.edu.sa http://faculty.kfupm.edu.sa/COE/amindin

Abstract: - A new algorithm for automated standard cell placement of asynchronous Micropipeline designs has been developed. The resulting placement solutions are targeted to meet all bundled-data timing constraints while providing efficient chip areas. The placement algorithm utilizes the simulated evolution iterative heuristic. The cost function is a weighted sum of an area factor and a timing factor. Results of five experimental circuits show that full routability with reasonably efficient areas are possible.

Key-Words: - Asynchronous systems; Micropipelnes; Automatic placement and routing; Stochastic Evolution.

1. Introduction

The past decade has witnessed a renaissance of interest in asynchronous design techniques and systems. Several asynchronous processors have been reported by researchers in both the academia and the industry [1] - [6]. Several asynchronous design methodology have been proposed [7]. One such methodology that has been well-received by the design community due to its modularity and simplicity is the micropipeline asynchronous design approach first proposed by Sutherland [8].

Operation of micropipeline systems follows a request-acknowledge handshaking protocol (Fig. 1) where a computation is initiated by a request signal and an output completion signal (acknowledge) indicates completion of this computation. For proper operation, the received request control signal and the data signals must follow the bundled-data constraint where the data bits should arrive at the receiver end prior to the arrival of the request event by some minimum setup time [8].

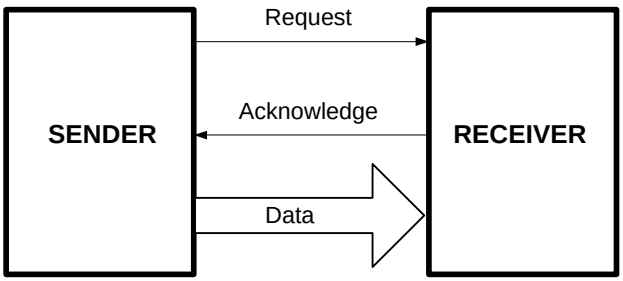

Fig. 1. Bundled-Data Handshaking Protocol

Several chips, including full microprocessors [2] and microcontrollers [3], have been designed using the micropipeline methodology. Asynchronous design methodologies, however, suffer from a considerable shortage of supporting design automation tools including, technology mapping, automatic placement and automatic routing tools.
In this paper, the problem of automatic placement of asynchronous micropipeline standard-cells is addressed. To our knowledge, this problem has never been addressed in the literature.

For proper standard-cell design of micropipeline circuits, it is not enough to provide a solution that minimizes area or delay. More importantly, the cells have to be appropriately placed such that a routing solution where meeting the bundled-data constraints of all communicating cells is possible with no/minimum addition of delay elements.

Given a netlist of micropipeline cells with predefined input and output pins, we have implemented a placement algorithm which is capable of meeting the bundled-data timing constraints [9]. We have adopted the standard-cell design style due to its relative simplicity and the wealth of knowledge available in this area. The design, layout and electrical characteristics of a variety of micropipeline cells were generated using 0.5μ CMOS technology.

Cells corresponding to a particular design are arranged in rows; with horizontal routing channels between rows reserved for cell interconnect. The standard cells are designed such that power and ground buses run horizontally through the top and bottom of the cells. The cells are butted against one

another such that these buses form continuous tracks in each row. The input and output pins of a cell are made available at the top and bottom edges of the cell. The cells are connected by running interconnect wires (nets) through routing channels. Connections from one row of cells to another are done either through vertical wiring channels at row ends or by using feed-through cells, which are standard height cells having few vertical interconnect wires running through them [15].

Given the *netlist* of a micropipeline design, the corresponding set of standard cells should be properly placed on the chip in a way that allows automatic routing of the interconnections while satisfying one or more possibly conflicting goals such as minimizing layout area, maximizing circuit performance, minimizing timing delays on critical nets, etc. [15]. An additional more basic requirement for *micropipelines*, is to meet the bundled-data constraints of various nets [9]. Unless the cells are properly placed to ensure meeting this constraint in an efficient manner, delay elements may have to be added which would unnecessarily increase chip area as well as the total wire length resulting in lower performance. Therefore, the cost function of the placement algorithm should primarily target this timing requirement. For this purpose, an adequate delay model is essential to check for timing constraint violations. Other optimization criteria, e.g. the chip area, the critical path delay, etc. may also be factored into the cost function as desired.

This work reports an evolution-based [16], [17] placement algorithm that is developed to address the case of micropipeline standard-cell designs. For this purpose, a 0.5μ standard cell library of micropipeline modules has been developed and its geometrical, electrical and timing parameters have been fully characterized. In section 2 the adopted timing model is described. Section 3 presents the used wirelength model. Section 4 defines the placement cost function while section 5 gives the details of the simulated evolution placement algorithm. This is followed by discussion of the obtained results in section 6.

2. Timing Model

Micropipeline data path modules (Fig. 2) are designed such that the output *acknowledge* event is asserted only after valid output data are made available at the cell output pin(s) (Data_Out). Thus, the cell output data are valid before the *acknowledge* output event is asserted by a minimum period that will be referred to as the acknowledge margin time (T_{ack}). This timing parameter (T_{ack}) is guaranteed by the cell design and is unaffected by the cell placement or routing, e.g. the self-timed adder reported in [14]. Likewise, for proper functionality, a micropipeline cell has a setup timing constraint specifying the minimum period for which the data inputs must be valid prior to the arrival of the input request event. This data to request setup time will be referred to as the request setup time (T_{req}).

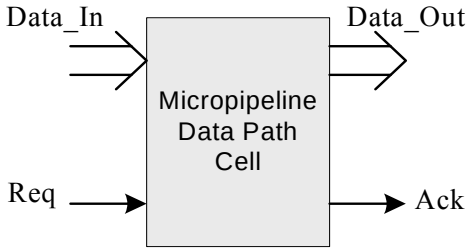

Fig. 2. Micropipeline Data Path Cell

Thus, in addition to the geometrical, electrical and propagation delay parameters, the developed micropipeline standard-cell library defines such important timing constraints between cell terminals (pins).

Typically, as shown in Fig. 3, the output data bus (Data_out(S)) and the associated acknowledge control signal (Ack(S)) of one cell (sender module) are used as input data (Data_in(R)) and its associated request signal (Req(R)) of another cell (receiver module). In other words, the Data_in(R) and Req(R) signals at the receiver module are only delayed versions of the Data_out(S) and acknowledge (Ack(S)) signals of the sender module due to routing. Even though, the sender cell design guarantees to have the data_out valid prior to the associated control signal by a minimum time of T_{ack}, improper placement and/or routing may lead to violation of the input setup time (T_{req}) at the receiver end due to the difference in the interconnect delays of the data and control nets.

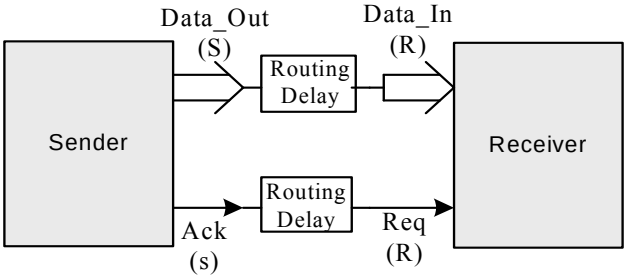

Fig. 3. Data Transfer between Sender & Receiver

The output of a micropipeline standard cell placement program is a list of the physical x-y

coordinates of each cell instance. The generated solution is reported together with slack time estimates for various data nets. The slack time (T_{slk}) of some input data pin represents the amount of additional delay that may still be added to this data net without violating the bundled-data constraint. In other words, it is a measure for the delay margin available for meeting the bundled-data timing constraint. Thus, the slack time of the data input at the receiver cell is given by:

$$T_{slk} = T_{ack}(S) - T_{req}(R) + T_{dly}(req) - T_{dly}(data), \quad (1)$$

where,
T_{ack} (S): is the acknowledge margin time for the sender cell,
T_{req} (R): is the request setup time for the receiver cell,
T_{dly}(req): is the request signal net delay from the sender to the receiver cell.
T_{dly}(data): is the data net delay from the sender to the receiver cell,

A negative value of the slack time indicates a critical condition where the associated request signal needs to be delayed. The generated slack information, and cell coordinates are fed to a detailed router for a complete chip layout.

For proper cell characterization, a proper delay model has to be adopted. The adopted model accounts for the following general delay properties:

1. The cell switching delay has two components:
   a. the intrinsic or base delay (BD) of the cell, and
   b. the load factor (LF) delay which depends on the driving capability of the cell and the load capacitances (input capacitances of load cells as well as the interconnect capacitance).
2. For cells with multiple input and output pins, the switching delay of a cell depends on which pins are active.

The model is less accurate than circuit simulation, but the error introduced is about the same for all cells and nets. This is important, since relative accuracy in timing analysis is sufficient. Even though differentiating delays between rising and falling signals would make the model less pessimistic, in the current work we have assumed equal delay parameters for both the rise and fall times leaving such differentiation for future fine tuning.

Thus, the parameters BD and LF will be assumed to have the same values for different transitions. According to this model, the switching delay of a cell is computed as:

$$Delay = BD + LF * AcLoad \quad (2)$$

where,
BD : the Base Delay of the cell, i.e., its internal intrinsic delay in the absence of any load capacitance,
LF : the cell load factor indicating the amount of delay per unit of load capacitance, and
AcLoad : the overall external load capacitance.

Thus, the delay from a valid input signal x to an output terminal y of some cell consists of two main components; a base delay and a loading delay. The base delay of any cell is placement-independent and should be directly available in the well-characterized cell library. Since for any net, only one source pin may be active at a time, the capacitances of all other pins on the net (usually input pins) represent one component of the loading delay which is also placement-independent. The second loading delay component, however, is due to the interconnect (wiring) capacitance of the net which is placement-dependent. Therefore, the delay from any input pin x to an output pin y has two components, one which is placement-independent T*pi* while the other is placement dependent T*pd*. Thus, the total delay TD can be written as:

$$TD = Tpi + Tpd \quad (3)$$

where,
Tpi : the placement-independent Delay

$$(Tpi = BD + LF * (Cin + Cout) \quad (4)$$

$$Cin : \sum \text{input capacitances of all input pins of the net} \quad (5)$$

$$Cout : \sum \text{output capacitances of all output pins of the net including the active (driving) pin} \quad (6)$$

Tpd : Placement-dependent delay (interconnect/ wiring delay) = $LF * Cw$ (7)

Cw : the wiring capacitance = *Cwa + Cwff*

Cwa : area wiring capacitance (depends on the wire area, i.e. its length and width)
Cwff : fringing-field wiring capacitance (which is considered as independent of the wire width, i.e. it only is a function of the wirelength)

Routing is done assuming two metal layers; Metal1 layer for horizontal tracks and Metal2 for vertical runs. Accordingly, our estimate for the total wire-length distinguishes the total horizontal wire

length component L_h from the total vertical wire length component Lv.

In the above timing model, the metal wire resistance is assumed to be negligible, which imply that an event signal emanating from some source node on a net will reach all sink nodes on the net at the same time. Furthermore, the contact resistance of vias that connect metal layers running horizontally and vertically, has been considered as negligible. It should also be noted that, since the adopted delay model is linear, the total delay of a given path is the sum of component and interconnect delay times along this path.

3. Wirelength Model

The performance of the placement algorithm depends on the way it estimates the wirelength of its nets. In this work, we adopt the semi-perimeter method [19] for estimating the net wirelength assuming Manhattan routing, i.e. routing tracks that are either horizontal or vertical. The wirelength of the net is estimated to be one half the perimeter of the bounding rectangle.

This method works fairly well for nets having two or three pins; however, for nets with more than three pins, the wirelength evaluated by this method is usually an underestimation of the actual value. To improve the accuracy of this estimation, the wirelength estimates of nets with more than three pins are inflated by a factor of 20% which gives closer wirelength estimate on the average. This helps in connecting nets with pins on different rows. Fig. 4 shows an example for a 5-pin net.

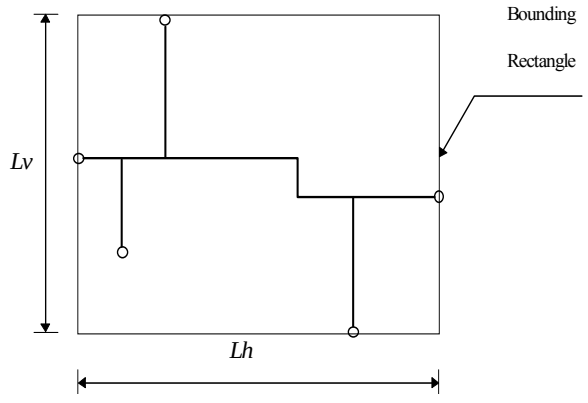

Fig. 4 Bounding Rectangle for a 5-Pin Net
$\{wirelength = 1.2(L_v + L_h)\}$

4. Placement Cost Function

The cost function of the placement algorithm primarily targets routability for proper functionality of micropipeline systems.

This necessarily implies that the cost function should reflect the required bundled-data timing constraints. This necessitates that all T_{req} timing constraints for all cell instances are met. This requires calculations of arrival times of various data and control signals at cell boundaries and comparing these to the specified cell request margin times. We compute a time margin function for all bundled-data constraints T_{ij} of all cell instances. This function yields a normalized estimate of the time margin beyond the specified setup time for the $j^{th.}$ bundled-data constraint of the i^{th} cell instance (margin by which data signals arrive prior to their associated control (Request) signal at cell instance *i* beyond the required T_{req} setup time for this instance. The pseudo code of the procedure which computes T_{ij} is given in Fig. 5. The procedure also computes the total number of constraints *n*, the total number of constraint violations of the i^{th} cell instance ($\boldsymbol{n}_{viol}^{(i)}$), and the total number of constraint violations for all cell instances $N_{violations}$ ($\leq n$). The individually computed time margins are used to compute an average timing margin quantity T_{margin} which is averaged over the total number of constraints *n*.

Placement solutions with negative T_{margin} are infeasible solutions since they correspond to solutions with gross timing violations.

In addition to timing constraints, the cost function also includes a component that accounts for the cost of the resulting chip area. The area cost factor only covers the placement-dependent chip area normalized to a placement-independent analytical-estimate of this area computed according to the procedure described by Kurdahi [20].

Thus, the area cost factor includes the routing channel area[a] as well as the area occupied by the feed-through cells.

The cost function is expressed as a *weighted sum* of the above two factors. The weight factors C_1 and C_2 are selected such that that $C_1 + C_2 = 1$. The used cost function is given by:

$$\boldsymbol{Cost} = \left(1 + \frac{\boldsymbol{N_{violations}}}{\boldsymbol{n}}\right)\left(\boldsymbol{C_1}\frac{1}{1 + \frac{\boldsymbol{T_{margin}}}{\boldsymbol{T_{MAX}}}} + \boldsymbol{C_2}\left(\frac{\boldsymbol{Total\ Area}}{\boldsymbol{Estimated\ Area}}\right)\right) \quad (8)$$

[a] *Width of a routing channel is estimated using the vertical constraint graph of that channel [19]. The length of the longest path in that graph represents a lower bound on the number of tracks required for that channel.*

Procedure Time_Margin
n = 0; / Total number of circuit timing constraints to be satisfied */*
$N_{violations}$ *= 0; /* Total number of violated timing constraints */*
TM = 0;
T_{MAX} *= 0 ;*
FOR *Each Cell_Instance* ***i*** **DO**
$\Delta T(i) = 0$; /* total time margin of i^{th} cell instance*/
$n_{viol}^{(i)} = 0$; /* Number of timing violations at the boundary of the i^{th} cell */
FOR *each timing constraint* ***j*** of *cell_instance* ***i*** **DO**
$T_{req}(i, j) = Setup_i(D_j(i), C_j(i))$ /* j^{th} setup time of cell_instance *i* */
$\Delta_{ij} = T_{Cj}(i) - T_{Dj}(i)$ /* $T_{Dj}(i)$, $T_{Cj}(i)$ = Actual arrival times of the j^{th} data (D_j) and control (C_j) signals at the boundaries of cell_instance *i* */
IF ($\Delta_{ij} < T_{req}(i, j)$) **THEN** $n_{viol}^{(i)} = n_{viol}^{(i)} + 1$;
$T_{ij} = \Delta_{ij} - T_{req}(i, j)$
IF ($T_{ij} > T_{MAX}$) Then $T_{MAX} = T_{ij}$
$\Delta T(i) = \Delta T(i) + T_{ij}$;
$n = n + 1$;
END FOR ***j***
$TM = TM + \Delta T(i)$;
$N_{violations} = N_{violations} + n_{viol}^{(i)}$
END FOR ***i***
$T_{margin} = TM / n$;
IF ($T_{MAX} = 0$) Then $T_{MAX} = 1.01 * T_{margin}$;
END Procedure

Fig. 5. Pseudo code for calculating the timing margins for individual cell instances ($\Delta T(i)$) and the average timing margin (T_{margin})

where,

$N_{violations}$: Total number of timing constraint violations for all cell instances.

n : The total number of timing constraints of all cell instances.

Total Area : Placement-dependent chip area (area of feed-through cells + area of routing channels*).

Estimated Area : Precomputed placement-independent analytical estimate for the area of feed-through cells, and routing channels computed according to the procedure outlined proposed by Kurdahi [20].

C_1, C_2 = Timing and area weight factors chosen such that that $C_1 + C_2 = 1$. Thus if $C_2 = 0$ optimization is performed only for timing while if $C_1 = 0$ only area optimization is targeted.

The cost function is scaled up by the factor $\left(1 + \frac{N_{violations}}{n}\right)$ since timing violations require additional delay cells which translate into higher cost both in timing and in total chip area.

5. Simulated Evolution Placement

This work uses the simulated evolution (SE) algorithm which is a general iterative heuristic method for combinatorial optimization problems

[16]-[18]. The SE method is stochastic in nature providing hill climbing means to allow for escaping local minima of the objective function on the path to a global optimum.

The algorithm is based on an analogy to the natural selection process in biological environments. The cells of a given design are treated as *population* [15]. The biological solution to the adaptation process in the evolution from one generation to the next is done by eliminating ill-suited solutions and keeping near-optimal ones. Every constituent of each generation must constantly prove its fitness under the current conditions in order to remain unaltered. The purpose of this process is to gradually create stable structures which are finally adapted to the given constraints.

The algorithm basically consists of a main loop which comprises three sequentially executed main steps namely; *Evaluation*, *Selection* and *Allocation*. The *evaluation* phase determines a normalized *goodness* of cell instances, i.e. the individual contribution of each cell instance to the overall cost of a given placement solution. For the evaluation phase, the time margin $\Delta T(i)$ of the i^{th} cell instance (see Fig. 5) has been used to determine the goodness of this cell instance as follows:

$$Goodness\ of\ the\ i^{th}\ cell\ instance = \frac{\Delta T(i)}{\left[1+\frac{n_{viol}^{(i)}}{n^{(i)}}\right]^{sign(\Delta T(i))}} \quad (9)$$

where

$n^{(i)}$: The number of timing constraints of the i^{th} cell instance

$n_{viol}^{(i)}$: The number of timing constraint violations for the i^{th} cell instance

$$\text{sign}(\Delta T(i)) = +1 \quad \text{if } \Delta T(i) \text{ is positive} \quad (10)$$

$$= -1 \quad \text{if } \Delta T(i) \text{ is negative} \quad (11)$$

Some micropipeline cells have no bundled-data timing constraints, e.g. cells with only control input and output signals like the Merge-element and the C-element[8]. We refer to such cells as control cells. The goodness of control cells is determined by summing the timing margins of the output control signal at the boundaries of all cell instances using this signal as an input request. Thus, if the i^{th} cell instance is such a control cell, its goodness is computed using the above formula (Eq (9)) but with $\Delta T(i)$, $n^{(i)}$, and $n_{viol}^{(i)}$ computed/defined differently as follows:

$$\Delta T(i) = \sum_{\substack{\forall cells(j)\ with\ the\ output\\ of\ the\ control\ cell\ i\ used\\ as\ input\ request\ signal}} (\Delta_{ij} - T_{req}(i,j)) \quad (12)$$

$n^{(i)}$: The number of cases where the output signal of the control cell instance (i) is used as an input request for other cell instances.

$n_{viol}^{(i)}$: The number of timing constraint violations associated with the output of the control cell instance (i) where this output is used as an input request for other cell instances ($n_{viol}^{(i)} \leq n(i)$).

In the selection phase, some cell instances are probabilistically selected for re-placement based on their goodness values. Thus, the cell's goodness determines its probability of survival in its old location. Cell instances with high goodness values are less likely to be selected compared to others with low goodness values. Finally, the allocation procedure removes selected lower fitness cells from their locations in the current placement solution and attempts to find better locations for them. For each cell individually, a search is performed to find an improved location in the vicinity of its old position. A number of trial placements are evaluated and then a choice is made based on the total cost reduction. When all cells have been replaced, the current iteration is complete and a new solution is formed.

The initial placement is obtained using the Constructive Linear Ordering (CLO) heuristic [21].

6. Results and Discussion

Table 1 lists the sample five micropipeline circuits that were used to test the above described placement algorithm. The resulting placements were passed to a detailed router and timing analysis was performed to verify the validity of bundled-data timing constraints.

Fig. 6 and Fig. 7 show the trend of the cost function versus the number of iterations for different values of C_2 for two of the test circuits (Ckt2 and Ckt5). At higher values of C_2, the area cost component gets more stressed in the cost function at the expense of the timing component. Too high values of C_2, however, generally yield solutions that do not meet the timing constraints.

Table 1. Test Circuits

Circuit	No. of Cells	No. of Nets
Ckt1. 16-bit Self-Timed Adder [14]	673	998
Ckt2. 4x4 Array Multiplier	1108	1714
Ckt3. 8-bit scaling accumulator [22]	1237	2078
Ckt4. 8-bit CORDIC core	2568	4219
Ckt5. MIPS R3000 write-back unit [22]	3842	6147

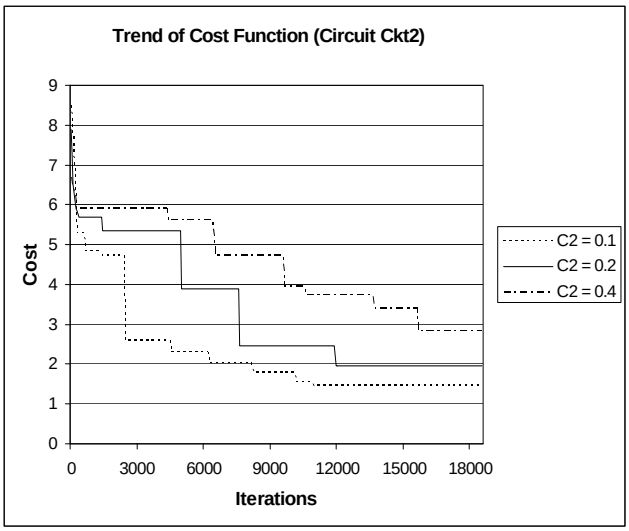

Figure 6. Cost Trend of Circuit Ckt2 for three values of (C_2)

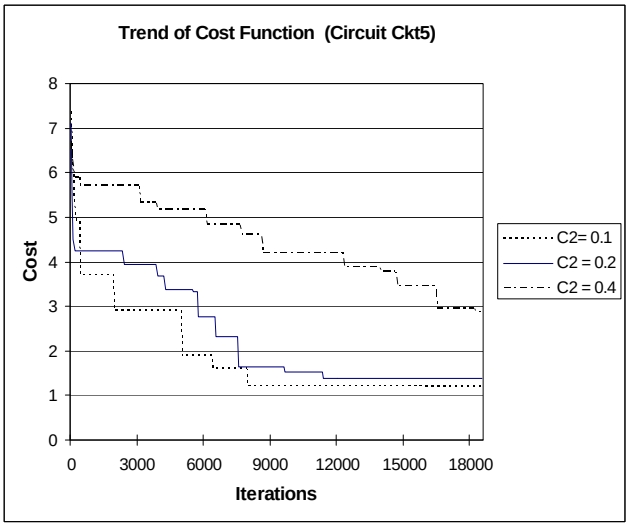

Figure 7. Cost Trend of Circuit Ckt2 for three values of (C_5)

Feasible solutions are those which satisfy all timing constraints, i.e. solutions for which $N_{violations} = 0$. Since delay estimation is based on simplified delay and wirelength models, solutions with small values of T_{margin}, e.g. $T_{margin} = 0$ may end up with timing violations in practice. Thus, higher quality solutions are those with smaller areas and higher T_{margin} values.

Table 2. Quality of Solution; Area (in μicron^2)

	Area (μicron^2)			
Circuit	$C_2 = 0.1$	$C_2 = 0.2$	$C_2 = 0.3$	$C_2 = 0.4$
Ckt1	154464	148910	150692	145036
Ckt2	419784	414548	409535	394163
Ckt3	529012	523822	516097	496724
Ckt4	2248998	2235701	2194092	2111735
Ckt5	5233926	5520975	5106149	4914484
Average	**1717237**	**1768791**	**1675313**	**1612428**

Table 3. Quality of Solution; T_{margin} (in pico-seconds)

	Time (pico-seconds)			
Circuit	$C_2 = 0.1$	$C_2 = 0.2$	$C_2 = 0.3$	$C_2 = 0.4$
Ckt1	30	22	13	7
Ckt2	29	19	14	$2^{\lozenge}$
Ckt3	18	14	9	$-2^{\lozenge}$
Ckt4	15	8	5	$3^{\lozenge}$
Ckt5	11	7	$3^{\lozenge}$	$5^{\lozenge}$

$^{\lozenge}$ Infeasible solution with $N_{violations} \neq 0$

Table 2 and Table 3 compare the quality of solution (circuit area and the T_{margin} parameter) for values of the area weight factor C_2 = 0.1, 0.2, 0.3 and 0.4. For almost all circuits, the best solution has

been obtained at $C_2 = 0.3$. The exception to that were circuits Ckt5, which has an infeasible solution at $C_2 = 0.3$, and Ckt1 which has its best solution at $C_2 = 0.4$. At $C_2 = 0.4$ only the smallest size circuit (Ckt1) has a feasible solution while solutions for other larger circuits are infeasible with $N_{violations} \neq 0$.

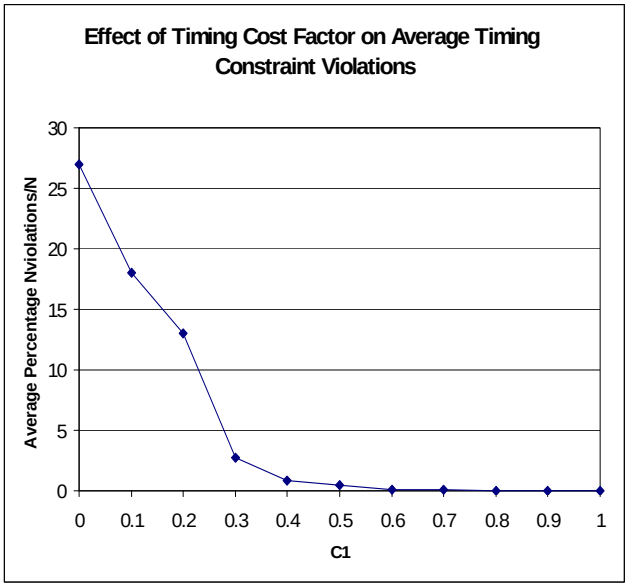

Fig. 8. Effect of the Timing Constraint Weight Factor (C1) on the resulting number of timing violations.

Fig. 8 plots the value of the number of timing violations as a percentage of the total number of timing constraints $\left(\frac{N_{violations}}{n}\right)$ averaged over solutions of the five test circuits versus the timing weight factor of the cost function (C_1). The figure clearly indicates that feasible solutions are not possible for values of $C_1 < 0.6$ ($C_2 > 0.4$).

Fig. 9 plots the resulting average area for all five circuits normalized to the maximum average area obtained at $C_2 = 0$ versus the area weight factor of the cost function (C_2). When the cost function is optimized only for timing constraint requirements (i.e., $C_2 = 0$), the timing constraints are fully met but at the expense of inefficient layouts with largely unequal rows of cells. With the area factor taken into consideration (i.e., for $C_2 = 0$); more area-efficient layouts were produced (Fig. 9).

It is readily seen that feasible solutions are possible for values of C_2 in the range 0.1 - 0.4. Higher values of C_2 have mostly yielded infeasible solutions with increasing number of timing constraint violations (Fig. 8).

For small size circuits, however, higher values of C_2 may yield solutions of more efficient areas.

It should be pointed out, however, that some timing violations may be fixed by inserting extra delay elements into paths of the relevant control signals. For some infeasible solutions with low number of timing constraint violations, this may be utilized to fix reported timing violations thus turning these solutions into feasible ones. Inserting such extra delay elements may, however, increase the layout area as well as disturb the placement solution. One possible solution is to have the cell library include different types of feed-through cells with and without associated delay elements. Thus, different delay values may be inserted conveniently in signal paths without significantly disturbing obtained placement solutions. This issue deserves to be the subject of further future investigations.

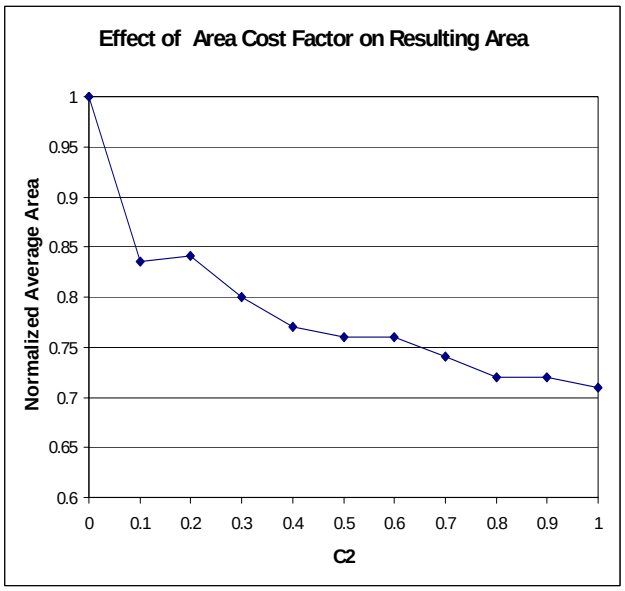

Fig. 9. Effect of the Area Weight Factor (C2) on the resulting area.

7. Conclusions

This work addresses the problem of automated standard cell placement of asynchronous micropipeline designs. With proper choice of area and timing weight factors in the cost function, the resulting solutions can be made to meet all of the required bundled-data timing constraints with reasonably efficient chip areas. The approach deserves more investigation to determine how the current algorithm should be modified / augmented to turn infeasible solutions with small number of timing violations into feasible ones through proper addition of control signal delays with no/minimal disturbance to the obtained placement solution.

Acknowledgement

The author would like to acknowledge the Computer Engineering Department of King Fahd University of Petroleum & Minerals for support of this work. I would also like to acknowledge Mr. F. Ashraf for his valuable help, dedication and insightful comments.

References:

[1] P.A. Beerel, "Asynchronous circuits: an increasingly practical design solution," Proceedings of the International Symposium on Quality Electronic Design, 18-21 March 2002, Pages:367 – 372.

[2] S.B. Furber, D.A. Edwards, and J.D. Garside. AMULET3: a 100 MIPS asynchronous embedded processor. In Proc. International Conf. Computer Design (ICCD), Sept. 2000, pp. 329-334.

[3] H. van Gageldonk, K. van Berkel, A. Peeters, D. Baumann; D. Gloor, G. Stegmann, "An asynchronous low-power 80C51 microcontroller," in Proceedings of the Fourth International Symposium on Advanced Research in Asynchronous Circuits and Systems, 30 March-2 April 1998, Pages:96 – 107.

[4] S. Rotem, K. Stevens, R. Ginosar, P. Beerel, C. Myers, K. Yun, R. Kol, C. Dike, M. Roncken and B. Agapiev, " RAPPID: An Asynchronous Instruction Length Decoder" in Proceedings of the 5th International Symposium on Advanced Research in Asynchronous Circuits and Systems, Apr. 1999, pp. 60-70.

[5] J. Kessels,; and P. Marston, "Designing asynchronous standby circuits for a low-power pager," In Proceedings of the IEEE , Volume: 87 , Issue: 2 , Feb. 1999, Pages:257 – 267.

[6] Tony Werner, and Venkatesh Akella, "Asynchronous Processor Survey," IEEE Computer, November 1997 pp. 67-73.

[7] Scott Hauck, "Asynchronous Design Methodologies: An Overview," Proceedings of the IEEE, 83(1): 69-93, January 1995.

[8] Sutherland, I. E. "Micropipelines," CACM, Vol. 32, No. 6, pp. 720 - 738, June 1989.

[9] Brunvand, E. Translating Concurrent Communicating Programs into Asynchronous Circuits, Ph.D. Thesis., Carnegie Mellon Univ., 1991.

[10] Per Arne Karlsen and Per Torstein Røine, "A timing verifier and timing profiler for asynchronous circuits," In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 13-23, April 1999

[11] Recep O. Ozdag, and Peter A. Beerel, "A Channel Based Asynchronous Low Power High Performance Standard-Cell Based Sequential Decoder Implemented with QDI Templates," 10th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'04) pp. 187-197.

[12] M. Renaudin, P. Vivet, and F. Robin, "ASPRO-216: A standard-cell QDI 16-bit RISC asynchronous microprocessor," In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 22-31, 1998.

[13] Kees van Berkel, Ronan Burgess, Joep Kessels, Ad Peeters, Marly Roncken, Frits Schalij, and Rik van de Wiel, "A single-rail re-implementation of a DCC error detector using a generic standard-cell library," In Asynchronous Design Methodologies, pages 72-79. IEEE Computer Society Press, May 1995.

[14] Amin, A. A. "A High-Speed Self-Timed Carry-Skip Adder," IEE Proceedings - Circuits, Devices and Systems, Volume 153, Issue 6 , December 2006, pp. 574-582.

[15] Shahookar, K. and Mazumder, P. VLSI Cell Placement Techniques, ACM Computing Surveys, Vol. 23, No. 2, June, 1991, pp. 143-220.

[16] King, R. M. and Banerjee, P. ESP: Placement by Simulated Evolution, IEEE Transactions on Computer-Aided Design, Vol. 8, No. 3, March 1989, pp. 245-256.

[17] King, R. and Banerjee, P. Optimization by Simulated Evolution with Applications to Standard Cell Placement, 27th ACM/IEEE Design Automation Conference, 1990, Paper 2.1, pp. 20-25.

[18] Saab, Y. G. and Rao, V. B. Combinatorial Optimization by Stochastic Evolution, IEEE Transactions on Computer-Aided Design, Vol. 10, No. 4, April 1991, pp. 525-535.

[19] Sait, S. M. and Youssef, H., VLSI Physical Design Automation, McGraw Hill, Europe, 1995, pp. 141-205.

[20] Kurdahi, F. J. Techniques for Area Estimation of VLSI Layouts. IEEE Transactions on Computer-Aided Design, Vol. 8, No.1,pp. 81-92, January 1989.

[21] Saab, Y. G. and Rao, V. B. Linear Ordering by Stochastic Evolution, Proceedings of the 4th CSI/IEEE International Symposium on VLSI Design, January 1991, pp. 130-135.
[22] G. Goslin, "A Guide to Using {FPGAs for Application-Specific Digital Signal Processing Performance," Xilinx Application Notes, 1995.