Sharing and Binding for General Circuits

Sheikh Muhammad Adib Bin Sh Abu Bakar
University of Applied Sciences Hamm-Lippstadt
B.Eng. Electronic Engineering
Lippstadt, Germany
sheikh-muhammad-adib.bin-sh-abu-bakar@stud.hshl.de

***Abstract*—Producing an optimum circuit is the dream of every circuit designer. A circuit can be evaluated through its used area and performance. Those aspects are designer concern to be optimized that can be done only during architectural synthesis or known as high level synthesis. Resource sharing and binding are very important process in high level synthesis, especially for circuit optimization. In this paper, sharing and binding will be discussed mainly for general circuit. This includes a sample model circuit to ease the explanation process and the method used for sharing and binding. After that, the strategies for optimization that use all those methods will be explored.**

***Index Terms*—sharing, binding, general circuit, optimization**

I. Introduction

Today, digital circuit are everywhere, from daily fashion accessories to heavy industry, and their demand keep increasing every single day. These phenomena cause evolution in digital circuit production industry. With the help of Computer-Aided Design (CAD), a designer could produce much better circuit at larger scale. The production of circuit involving specification, synthesizing and implementation. In specification stage, the aim of designer is to fulfil all the functionality of the circuit where architectural view of the circuit is produced. The process from architectural model to register transfer level implementation comprise architectural synthesis or known as high level synthesis. A produced circuit is evaluated based on their area and performance. These aspects can only be optimized during high level synthesis where many aspect and algorithm are considered and imply. This is where resource sharing and binding entailed in high level synthesis to help further circuit optimization.

In this paper, resource sharing and binding that are focused for general circuit will be discussed. At the beginning, a sample model of the circuit will be introduced to ease the upcoming explanations in *Sample Model* section. Then, the important aspect of the circuit specification will be discussed as a guide in *Specification* section. Following that, sharing and binding will be introduced, including the methods for sharing and binding in *Resource Sharing and Binding Methods* section. Eventually, we will go into circuit optimization, where estimation of the circuit's aspects and how to achieve the optimization through sharing and binding will be discussed in *Optimization* section.

II. Sample Model

To make every explanation in upcoming sections easy to understand, a sample model of a circuit at the architectural level will be used. The model that has been chosen is circuit that solve the following equation :

$$y\prime\prime + 3xy\prime + 3y \qquad [1] \quad (1)$$

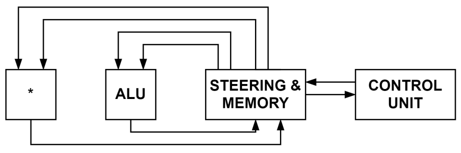

Fig. 1. Example of structural view at the architectural level. [1]

The structural view of the model at architectural level is shown in "Fig. 1" and the behavioural view of the model at architectural level can be written in HDL as follows [1]:

```
diffeq {
    read(x, y, u, dx, a);
    repeat {
        xl = x + dx ;
        ul = u − ( 3∗x∗u∗dx ) − ( 3∗y∗dx );
        yl = y + u∗dx;
        c = xl < a;
        x = xl ; u = ul ; y = yl;
        }
    intil ( c );
    write ( y );
}
```

From the behaviour model, we could see that the solution involving iteration of set of operations. The set of operation in each iteration (within the repeat function) can be broke into 11 simple operation and can be modelled using data-flow graph as shown in "Fig. 2" to represent the abstract model of the solution at architectural level in terms of tasks and their dependencies. The task could be *No-Operation* (NOPs) if it does not entail any operation or can be done instantly with no adverse effects. [1].

Regrading the branching and iteration, the control-flow information can be modelled using a control-flow graph. According to [1]—"Many models have been proposed to represent

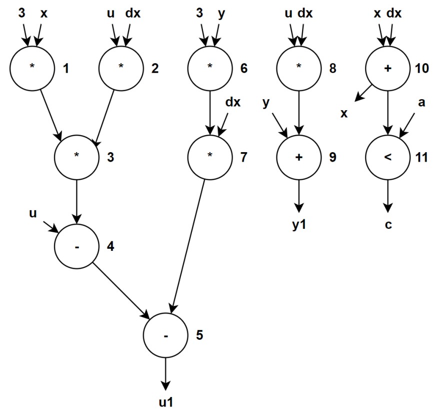

Fig. 2. Example of a data-flow graph. [1]

control data-flow graphs (CDFGs)". One of the main CDFG that will be intensively used in this paper is sequencing graph, $G_s(V, E)$, as illustrate in"Fig. 3" that represent the circuit model where V is the vertex set, $V = \{v_i; i = 0, 1...., n\}$ and E is the set of edges, $E = \{(v_i, v_j); i, j = 0, 1, ..., n\}$. Based on the "Fig. 3", the graph has two properties :

- Polar : The source and sink vertices are labelled with v_0 and v_n representing the start and the last task.Both are NOPs.
- Acyclic : No iteration or loop because the iteration and branching are modelled outside the graph.

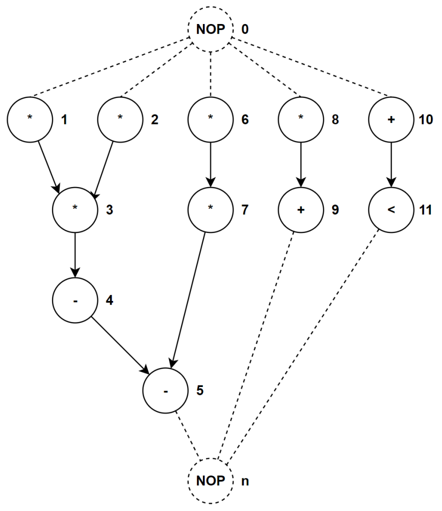

Fig. 3. Example of a sequencing graph. [1]

As stated, the iteration and branching are modelled outside this graph means that path in the graph not representing alternative but concurrent streams of operation [1]. But it does not mean that the sequencing graph cannot model loop and branching. They do, but in a hierarchical way.

After a solution or the circuit have been model, the specification should be further described. The model specification will be explained more in the next sub topic.

III. Specification

In pervious section, one of the important specification already be represented in sequencing graph which is the behavioural-level of the circuit model. In this section, the specification will be furthered detailed, including the type and number of resource that will be used in the circuit and the constraints.

At architectural level, resource have their own definition, where it implements different types of function in hardware and can be classified into 3 classes as listed below [1]:

- Functional resources : Implementing the arithmetic or logic functions.
- Memory resources : Memory array
- Interface resources : Used to support data transfer including busses and interfacing circuit including external interface for input output

Within the scope of this paper, only functional resource will be intensively used. The last specification aspect that will be considered in this paper is constraint. It can be classified into two categories, which are [1]:

- Interface constraint : This specification used to ensure that the circuit can be implemented in provided environment like format and timing used in I/O interface.
- Implementation constraint : This constraint specified the area constraint, performance constraint and binding constraint.

All the constraint are must not specify to give some freedom in synthesizing the architectural model. To narrow the scope of this paper, the performance constraint will be used including cycle time and latency bound, λ that are used to extend the pervious sequencing graph to scheduled sequencing graph as illustrate in "Fig. 4". Scheduled graph is a function $\varphi : V \rightarrow Z^+$, where $\varphi(v_i)$, denotes the operation start time such that $t_i \leq t|_j + d_j, \forall i, j : (v_j, v_i) \in E$ [1]. This graph will be used for further synthesis process. The area constraint entail the number of resource usage, since adding more resource means adding more area to the circuit. All constraint will be further explained in upcoming sections.

As shown in 'Fig. 4", each vertex represent a type of computation or known as operation type and the resource that execute the operation called resource type. It is also possible that a resource type could execute more than one operation type. ALU, for instance, can do addition, subtraction and comparison. In our circuit model, the resource type that we used is ALU and multiplier as shown in "Fig. 1".

Before going further to the synthesizing process, the stage of the synthesizing process need to be understood. The stages will be discussed in the next section.

IV. Architectural Synthesis Stages

In architectural level synthesis, two stages are involved [1], which are :

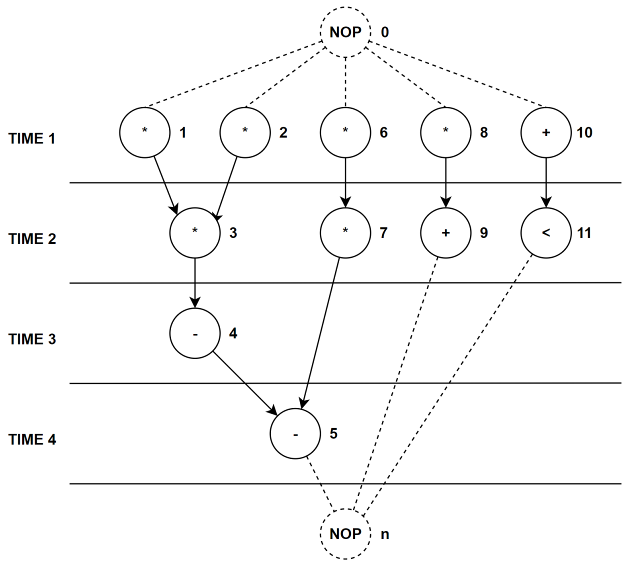

Fig. 4. scheduled sequencing graph. [1]

- Placing the operation in time and space.
- Determining the detailed interconnection of the data path and the logic level specification of the control unit.

To narrow down the scope of this paper, only the first stages will be discussed. In the next section, the important process in the first stage will be explained.

V. Resource Sharing And Binding Methods

In this section one of the most important processes in synthesizing an architecture model will be discussed which is resource sharing and binding. We assume here scheduled sequencing graph and non-hierarchical sequencing graph only. Before going through the methods used to solve resource sharing and binding problem, the definition of them should be comprehended as follows [1]:

- Resource sharing : A resource is assigned to more than one operation
- Binding : Mapping of an operation to a physical resource, $\beta : V \rightarrow R * Z^+$, where $\beta(v_i) = (t, r)$ denotes that the operation corresponding to $v_i \in V$, with type $\tau(v_i) = t$, is implemented by the r^{th} instance of resource type $t \in R$ for each $i = 1, 2, ..., n_{ops}$

If more than one operation with the same type are used, like in our sample model, resource sharing can be implemented as long as the operations are not concurrent, means that $E = \{(v_i, v_j) | \tau(v_i = v_j \text{ and } ((t_i = d_i \leq t_j) \text{ or } t_j = dj \leq t_i)), i, j = 1, ..., n_{ops}\}$. The operation is *compatible* if those conditions are met [1]. The need of sharing is to reduce the area of a circuit. Some circuit specification already have area constraint that need to be followed, means that resource sharing need to be applied. It is also possible that the binding constraint already specified in a circuit specification. To reduce the complexity, our model circuit does not have binding constraint, but area constraint maybe applied and will be explained later. Many methods have been proposed to help designer to decide which operations are compatible and how many resources should be used. Resource sharing implicitly describe the binding of an operation to a specific resource. That means every method used to solve resource sharing problem involve binding problem.

Next, we will discuss the methods used to determine the compatibility of operation for sharing. First, for the sake of simplicity, we assume that our model circuit is a resource dominated circuit, and then we will assume our model circuit is a general circuit to help the reader understand the method.

A. Compatibility Graph

One of the method to analyse the compatibility of operation for sharing is using Compatibility graph. Compatibility graph, $G_+(V, E)$ is a graph whose vertex set $V = \{v_i, i = 1, 2, ..., n_{ops}\}$ is in one to one correspondence with the operation and whose edge set $E = \{v_i, v_j i, j = 1, 2, ..., n_{ops}\}$ denotes the compatible operation pairs [1]. This means that the number of distinct components in the graph is at least equal to the number of resource type. An optimal resource sharing strategy is one that reduces the amount of resources instances required [1].

According to [1]—“A group of mutually compatible operations corresponds to a subset of vertices that are all mutually connected by edges, i.e., to a clique. Therefore, a minimal set of mutually compatible operations is represented by a maximal clique in the compatibility graph. Since we can associate a resource instance to each clique, the problem is equivalent to partitioning the graph into a minimum number of cliques. Such a number is the clique cover number of $G_+(V, E)$, denoted by $\kappa(G_+(V, E))$”

Base on our sample model, the compatibility graph can be drawn as in “Fig. 5. Each vertex representing the operation type. We can see that $\{v_2, v_6\}$ and $\{v_{10}, v_{11}\}$ are examples of compatible operation. Examples of cliques are the sub-graphs induced by $\{v_1, v_3, v_7\}$,$\{v_2, v_6, v_8\}$,$\{v_4, v_5, v_{10}, v_{11}\}$ $\{v_9\}$. So the clique cover number κ of the graph is 4, corresponding to two multipliers and two ALUs. The edges of a clique are thicker than the other, as shown in “Fig. 5.

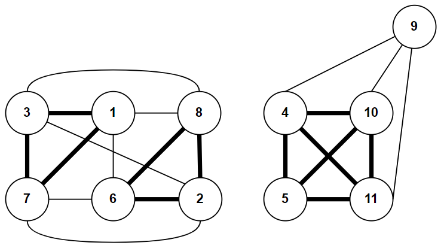

Fig. 5. Compatibility graph. [1]

Since the area of a general circuit is not only depends on resource but also involving the area of steering logic and wiring, we need to extend the compatibility graph to help designer estimate area more accurately. To make explanation much simpler, we only focus on the multiplexer, whose area and propagation delays depend on the number of inputs and wires [1].

We extend the compatibility graph to a weighted compatibility graph. Different types of weighted compatibility graphs can be defined [1]. As mentioned before, we can represent the sub-graph as a clique. Cliques may be connected with weight. The cost of allocating the clique to a single resource comprehend the cost of steering logic and wiring components like multiplexers.

A set inclusion cost function can be used to create a weighted compatibility graph model [1]. Each vertex of the compatibility graph in this model is assigned a set of weights known as a cost set. The total cost is the sum of all weights associated with a partition's cliques. The cost-set approach allows us to compute weights on cliques based on vertices' properties. This model may be used to describe the steering logic and wiring area, which are dependent on operation grouping and binding to a shared resource.

Now we consider the operation in "Fig. 5" within the first clique that use multiplier. Assuming that the cost of multiplexing a signal is $area_{mux} = area^{\triangle}_{mux}(a-1)$ and is expressed as $area^0_M = \sum_{i=1}^{a} area^i_M$, where $area^{\triangle}_{mux} = area^i_M = -area^0_M$, $1 \leq i \leq a$ [1].

If we assume each operation with dedicated resource, the overall cost is $(area_* + area^0_M) + \sum_{i=1}^{3} area^i_M = 3area_*$. If we assume operation v_3 and v_1 share the same resource, and v_7 with dedicated resource, the overall cost is $(area_* + area^0_M + \sum_{i=1}^{3} area^i_M) + (area_*, area^0_M, area^3_M) = 2area_* + area^{\triangle}_{mux}(2-1)$.

So, we can conclude that this method could help designer to decide the compatibility of operations under the constrained area. Another technique that based on compatible graph as mention in [1] that purposed by Tseng and Siewiorek is clique partitioning algorithm that constructs cliques from compatible pairs of operations. Means that they utilized edge weights to reflect the intensity of desire for sharing. The edge-weighted compatibility graph is denoted here by $G_+(V,E,W)$.

Again, we consider the operation in "Fig. 5". The subgraph's two vertices with the most common neighbours are merged. If the edge is not connected to all the member of the new vertex (clique seed), the edge will be deleted. This process is repeated until no connected edge left. Then the formed clique will be saved on a list. This process can be illustrated in "Fig. 6"

Next, we will go through another method used to analyse the compatibility of operation for sharing other than compatibility graph, which is conflict graph.

B. Conflict Graph

Compatibility of operation for sharing can be analysed using Conflict graph. Conflict graph, $G_-(V,E)$ is a graph whose vertex set $V = \{v_i, i = 1,2,...,n_{ops}\}$ is in one-to-one correspondence with the operations and whose edge set $E = \{\{v_i, v_j\} i,j = 1,2,...,n_{ops}\}$ denotes the conflicting

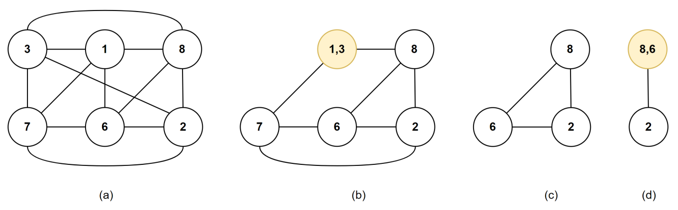

Fig. 6. (a) Compatibility graph for the multiplier type. (b) Reduced compatibility graph with clique seed. (c) Fragment of compatibility graph after one clique has been removed. (d) Reduced fragment with clique seed. [1]

operation pairs.Operations have conflict when they are not compatible [1]. Means that the graph is the compliment of the compatibility graph because a subset of vertices that are not linked by edges corresponds to a set of mutually compatible operations.

The vertexes of the graph are coloured accordingly based on their compatibility, so that it will help the designer to solve the sharing problem. Each colour represent different resource, and it will be helpful if operation with different resource type are distinguished first because they are obviously always conflicting. The optimum resource sharing is when minium number of colour used, which are denoted by $\chi(G_-(V,E))$. $\chi(G_-(V,E))$ is equal to $\kappa(G_+(V,E))$.

Again, using our sample model, the coloured conflict graph can be drawn as illustrate in "Fig. 7.

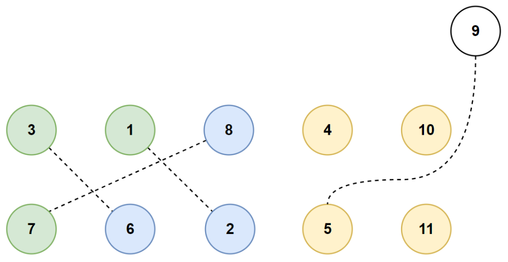

Fig. 7. Conflict graphs for the multiplier and ALU types. [1]

The execution delay for each operation is $\{[t_i, t_i + d_i - 1]; i + 1,2,...,n_{ops}$ and the edges of the conflict graph denote intersections among intervals, hence they are in an interval graph [1]. Means that, in polynomial time, by applying interval sequence graph, it is much easier for designer to find the minimum number of colour used. Back to our circuit model in "Fig. 4", interval conflict graph can be obtained as shown in "Fig. 8".

In the next subtopic, we will discuss the method used to solve the binding problem for a given schedule and resource bound.

C. ILP model

For the ILP model, we will be using a set of binary decision variables with two factors, $B = \{b_{ir}; i = 1,2,...,n_{pos}; r = 1,2,...,a\}$, and a set of continuous and categorical constants

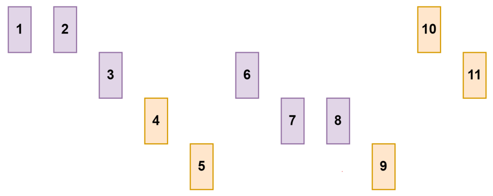

Fig. 8. Intervals corresponding to the conflict graph. [1]

with two factors, $X = \{x_{il}; i = 1, 2, ..., n_{pos}; l = 1, 2, ..., \lambda + 1\}$, where $a \leq n_{ops}$ is an upper bound on the number of resources to be used, if the operation v_i is bound to resource r, i.e., $\beta(v_i) = (1, r)$, b_{ir} will be one 1, otherwise 0 and When operation v_i begins in step 1 of the schedule, i.e., $l = ti$, the binary constant x_{il} equals 1, otherwise 0. Finding a binding that is consistent with a given schedule (expressed by X) and a resource bound a is similar to finding a set of B values that fulfil the following constraints [1]:

$$\sum_{r=1}^{a} b_{ir} = 1, i = 1, 2, ..., n_{ops} \tag{2}$$

$$\sum_{i=1}^{n_{ops}} b_{ir} \sum_{m=l-d_i+1}^{l} x_{im} \leq 1, l = 1, 2, ..., \lambda + 1, r = 1, 2, ..., a \tag{3}$$

$$b_{ir} \in \{0, 1\}, 1 = 1, 2, ..., n_{ops}, r = 1, 2, ..., a \tag{4}$$

Based on our scheduled sequencing graph of “Fig. 4”, the operations have two types and to make it simple we label multiplier with 1 and ALU with 2. So, a feasible binding satisfies the constraints [1]:

$$\sum_{r=1}^{a_1} b_{ir} = 1, \forall_i : \tau(v_i) = 1 \tag{5}$$

$$\sum_{i=:\tau(v_i)=1} b_{ir} x_{il} \leq 1, l = 1, 2, ..., \lambda + 1, r = 1, 2, ..., a_1 \tag{6}$$

$$\sum_{r=1}^{a_2} b_{ir} = 1, \forall_i : \tau(v_i) = 2 \tag{7}$$

$$\sum_{i=:\tau(v_i)=2} b_{ir} x_{il} \leq 1, l = 1, 2, ..., \lambda + 1, r = 1, 2, ..., a_2 \tag{8}$$

The constants in the set X are all zero, except for, $x_{1,1}, x_{1,1}, x_{2,1}, x_{3,2}, x_{4,3}, x_{5,4}, x_{6,2}, x_{7,3}, x_{8,3}, x_{9,4}, x_{10,1}, x_{11,2}$ which are 1. If $a_1 = 1$, multiplier would correspond to finding a solution to [1] :

$$b_{i1} = 1, \forall_i \in \{1, 2, 3, 6, 7, 8\} \tag{9}$$

$$\sum_{i \in \{1,2,3,6,7,8\}} b_{i1} x_{il} \leq 1, l = 1, 2, ..., 5 \tag{10}$$

Because the second constraint as in (3) will entail $b_{1,1} + b_{2,1} \leq 1$, which violates the first constraint as in (2), thus a solution doesn’t really possible. An implementation with $a_1 = 2$ multipliers would correspond to finding a solution to [1]:

$$b_{i1} + b_{i2} = 1, \forall_i \in \{1, 2, 3, 6, 7, 8\} \tag{11}$$

$$\sum_{i \in \{1,2,3,6,7,8\}} b_{i1} x_{il} \leq 1, l = 1, 2, ..., 5 \tag{12}$$

$$\sum_{i \in \{1,2,3,6,7,8\}} b_{i2} x_{il} \leq 1, l = 1, 2, ..., 5 \tag{13}$$

which admits the solution $b_{1,1} = 1, b_{2,2} = 1, b_{3,1} = 1, b_{6,2} = 1, b_{7,1} = 1, b_{8,2} = 1$, all other elements of B with first subscript $i \in \{1, 2, 3.6, 7, 8\}$ being zero. The binding of the ALUs can be computed similarly. The list of binding is shown in “Table I” and can be illustrated using bound sequencing graph as shown in “Fig. 9”. The shaded area represent a resource and the operations within the shaded area are binded to the resource accordingly

TABLE I
BINDING RESULT [1]

$\beta(v_1)$	$(1,1)$
$\beta(v_2)$	$(1,2)$
$\beta(v_3)$	$(1,1)$
$\beta(v_4)$	$(2,1)$
$\beta(v_5)$	$(2,1)$
$\beta(v_6)$	$(1,2)$
$\beta(v_7)$	$(1,1)$
$\beta(v_8)$	$(1,2)$
$\beta(v_9)$	$(2,2)$
$\beta(v_{10})$	$(2,1)$
$\beta(v_{11})$	$(2,1)$

VI. Optimization

The evaluation of a circuit is through its used area and performance. This aspect can be only optimized during the architectural synthesis. So it is important to have the ability to estimate the area used and resulted performance. This then used to make further optimization until the optimum circuit can be produce. The area and performance estimation and optimization during high level synthesis can be done during sharing and binding process. This is due to the fact that a binding provide the information about the area of a circuit and schedule provide the latency with given cycle-time. In this section, we will go through the step in estimating area and performance of a circuit. Then we will go further with the optimization strategies.

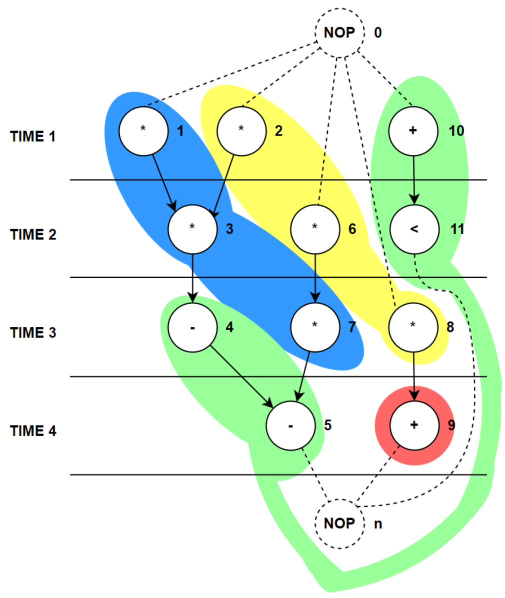

Fig. 9. Scheduled and bound sequencing. [1]

A. Area and performance estimation

In general circuit, the total area of a circuit is the sum of area of resource, steering logic, register, control unit and wiring area. The number of those component used in a circuit is depends on binding as well as schedule. Reducing the number of used resource linearly does not mean that area is reduced linearly. As we discussed in pervious section, resource sharing is applied to reduce used area and reducing the number of resource also cause the increasing number of steering logic, register and control unit. That means the number of every single component affect the used area.

Circuit performance can be evaluated through circuit schedule. As we discussed before in pervious section, schedule sequencing graph provide us the value of latency, λ , propagation delay of an operation and cycle-time. Since the value of latency with defined cycle-time affected by binding, that give us information about maximum number of operation with the same type can run concurrently, we can also estimate the performance through binding. Binding explicitly describe the type and number of component being used. That means every single component affect the performance because the number of resource affect the total of propagation delay and latency.

So it is important to know how the number of every single component such as resource, steering logic, wiring and control unit affect the area and performance. The list below explain how The components affect the area and performance [1].

- Resource : A resource affect the area and performance through its size and its propagation delays of combinational logic.
- Steering logic : Steering logic affect the propagation delay of a resource. The effected area can be estimated depend on the number of input and out put of a multiplexer used as we discuss in pervious section.
- wiring : Wiring contribute to overall area and delay. Estimating the wiring area and length requires the knowledge of the structure. The delay cause by wiring can be estimated through its length.
- Control unit : Control unit contribute to overall area and delay. The area of control unit can be estimate based on latency because the word length used to accommodate all control step can be estimated.

Next, which is the last step in our synthesizing process will be discussed, which is optimization strategies.

B. Optimization strategies

In this subsection, we will discuss circuit optimization. Optimization process involve scheduling and binding in order to optimize the area, latency and cycle-time because of the aim of architectural optimization is to develop a scheduled sequencing graph that has a complete resource binding and meets all of stated constraint [1]. An optimum circuit can be built if the circuit is not influenced only by one aspect, e.g. area, but fairly influenced by all aspect. As explained in pervious section, particularly in compatibility graph subsection, we can see that reducing number of resource cause more operation need to share some resource, thus the operation cannot run concurrently, and the latency need to be increased. On the other hand, increasing the number of resource could reduce the latency but increase the total area of a circuit. Reducing the cycle time could cause an operation to take two cycles to complete. All the possibilities can be structure in design evaluation space. To have an optimum circuit, we need to find a point where all aspects are fairly dominate the circuit. This point is called Pareto point [1].

To have this point, a designer should traverse the design evaluation space. Exploring the (area/latency) trade-off for various cycle-time values is a common method of architectural exploration. Another method is to look for the (cycle-time/latency) tradeoff for a certain binding or the (area/cycle-time) tradeoff for specific schedules.

Next we will go through for each design evaluation space.

C. (area/latency) trade-off

In (area/latency) trade-off design evaluation space, the value of cycle time is fixed. The extreme point of each aspect, area and latency, can be found by finding the minimum value of latency and resource accordingly. The design evaluation space can be explored by reducing one parameter as a function of the other.

Based on our circuit model as in “Fig. 4”, we make assumption as below :

- Area constraint is 20 units.
- Latency is less than 8 cycles.
- ALU propagation delay is 25 ns.
- Multiplier propagation delay is 35 ns.

Since the cycle time should be fixed, if we set the cycle-time to 40ns, every operation will take one cycle time, but if we set it to 30ns, multiplier will take two cycles to complete. Now we fix our cycle time to 40ns.

If we took only one multiplier, the resulting latency is 13, and it will break the constraint. if we choose two multiplier and one adder, the resulting latency and area is 8 cycles and 12 unit. The exploration can be continued by trying all the possibilities as shown in "Fig. 10".

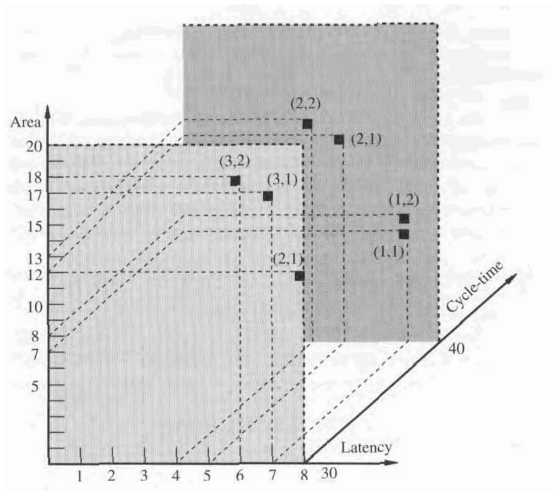

Fig. 10. Design evaluation space: area/latency trade off points for two values of the cycle time. [1]

D. *(cycle-time/latency) trade-off*

By fixing the value of total area, we could try to draw a possible point by changing the value of number of component and see how the value of cycle-time and latency affect each other as shown in "Fig.11". From our discussion in the pervious section, the method that used weighted compatibility graph could be helpful.

E. *(area/cycle-time) trade-off*

By fixing the value of latency, we could try to draw a possible point by changing the value of number of component and see how the value of cycle-time and area affect each other as shown in "Fig.12" by applying possible methods from pervious section.

VII. Conclusion

As a conclusion, since the optimization of a circuit can only be done during high level synthesis, it is important to have concrete understanding how sharing and binding affect the area and performance of a circuit and the methods to implement them. The design evaluation space also needs to be comprehended to allow a designer to find the Pareto point in order to produce an optimum circuit. The exploration of design evaluation space can be improved not only through learned theory, but also experience.

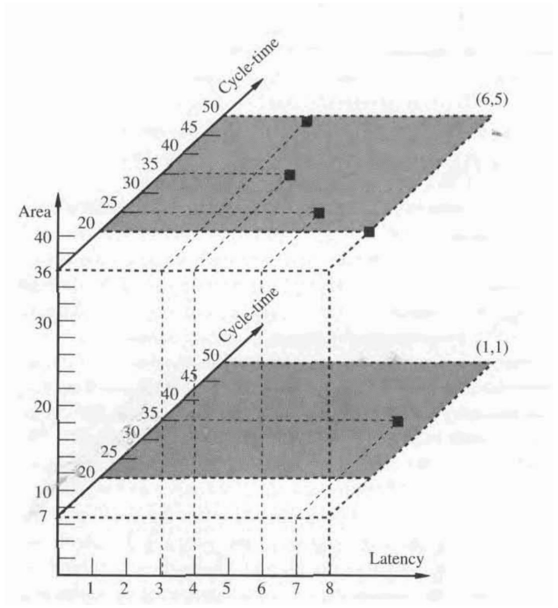

Fig. 11. Design evaluation space: cycle-time/latency trade off points for two values of the area. [1]

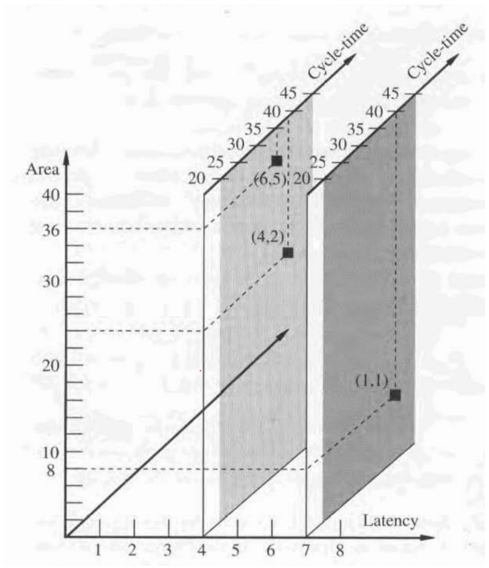

Fig. 12. Design evaluation space: area/cycle-time trade off points for two values of the latency. [1]

Acknowledgment

I would like to express my very great appreciation to Dr Achim Rettberg for his valuable and constructive suggestions during the planning and development of this research work. His willingness to give his time so generously has been very much appreciated.

References

[1] G. D. Micheli, Synthesis and optimization of digital circuits. New York: McGraw-Hill, 1994.
[2] Shilpa, K. and Lakshminarayana, C and Singh, Manoj. (2019). Optimal Resource Allocation and Binding in High-Level Synthesis Using Nature-Inspired Computation. 10.1007/978-981-13-5802-9-95.