Profit Maximization through Process Variation
Aware High Level Synthesis with Speed Binning

Zhao Mengying
City University of Hong Kong

Orailoglu Alex
University of California, San Diego

Xue Chun Jason
City University of Hong Kong

Abstract—As integrated circuits continuously scale up, process variation plays an increasingly significant role in system design and semiconductor economic return. In this paper, we explore the potential of profit improvement under the inherent semiconductor variability based on the speed binning technique. We first accordingly propose a set of high level synthesis techniques, including allocation, scheduling and resource binding, thus essentially constructing designs that maximize the number of chips that can be sold at the most advantageous price, leading to the maximization of the overall profit. We explore subsequently the optimal bin placement strategy for further profit improvement. Experimental results confirm the superiority of the high level synthesis results and the associated improvement in profit margins.

I. INTRODUCTION

With the continuous scaling of integrated circuits, fabrication size is shrinking in nanometer regimes. As a result, production to ensure predictable performance can no longer be guaranteed. Transistor parameters, such as channel length, gate-oxide thickness and threshold voltage, deviate from nominal values, thus introducing ambiguities on the optimal course to be taken in processor design. Intel lab results show a twenty-fold variation in leakage power for a 30% variation in performance based on a design in 180nm technology [1]. When looked at from the vantage point of total power, a 40%-70% variation is associated with a 20%-50% variation in frequency, as [2] reports. Due to the transistor parameter fluctuations resulting from process variation, fabricated chips vary from each other in performance. Manufactured products of the same design may end up being used as high performance 1 GHz chips, or end up being used as lower-performance 600 MHz chips. In such a manufacturing environment, the issue of speed binning is brought to the forefront.

Speed binning refers to the test procedures that help qualitatively categorize working chips into different bins according to the highest speed test that they could pass, so that chips could be offered to customers with the appropriate frequency grades [3]. For instance, the MPC7455 microprocessor has been offered in 6 grades, i.e. 6 bins: 600, 733, 800, 867, 933 MHz and 1GHz [4]. Chips in different bins are correspondingly offered with various price grades, thus delivering distinct economic returns. The profit is defined as follows:

\[ \text{Profit} = \text{income} - \text{cost} = \sum_{i=1}^{n} p_i \cdot n_i - \text{cost} \]

where \( n \) is the number of bins, \( p_i \) and \( n_i \) represent the price of the \( i_{th} \) bin and the number of chips falling into this bin, and \( \text{cost} \) denotes the cost of the design. Figure 1 shows one example with 3 bins. The price is a stair-case function of \( T_{\text{clk}} \) delay. Profit is maximized by intelligently distributing chips into bins for aggressive income and retaining a low design cost at the same time.

As one indispensable step in system design, high level synthesis (HLS) translates the behavioral description into a corresponding register level structure description, including resource allocation (functional unit type selection), scheduling (assigning operations into clock steps) and binding (resource instance mapping) [5]. Due to the process variation, the concept of performance yield is developed and widely used as HLS optimization criterion. It describes the probability of a certain design meeting the predefined performance constraints space [6], e.g. 90% performance yield means 90% of the chips statistically satisfy design constraints. A number of researchers conduct variation-aware HLS based on this yield theory to deal with process variation, like minimizing latency [7] [8] or area [9] while guaranteeing satisfactory performance yield. Our objective is to build a satisfactory circuit performance distribution (\( T_{\text{clk}} \)) by HLS solutions, which determines the economic profit in the context of speed binning.

The binning result is affected by all the system design levels. Prior work on profit-aware design considers economic issues for profit maximization at the circuit design level [10] [11], whereas we focus at HLS. In this paper, we explore the potential HLS approaches to strive for profit improvement under process variation, and build up a set of HLS techniques to maximize the number of chips that can be sold at the most advantageous price, thus maximizing overall profit. To the best of our knowledge, this is the first work discussing speed binning in the context of embedded system processors. In particular, our contributions are summarized as follows:

- Introduce speed binning into the HLS domain to derive maximal economic return.
- Develop a set of HLS approaches for profit maximization, including allocation, scheduling and binding.
- Propose a strategy for optimal bin placement.

The remainder of this paper is organized as follows. Section II shows an example to illustrate how the HLS decisions affect economic profit. Section III accordingly introduces profit-aware HLS strategies. Section IV presents the optimal bin
selection approach. Section V presents the experimental setup and results. Section VI offers a brief set of conclusions.

II. MOTIVATION

In this section, we present an example to illustrate how the HLS decisions affect profit, i.e., the difference between income and the expense associated in generating it.

Figure 2(a) shows the given DFG and resource library. Each operation can be mapped to either of two types, mnemonically denoted as Fast and Slow. Figure 2(b) presents four distinct HLS solutions. \( S_1 \) achieves the best performance with the fastest modules. \( S_2 \) ignores process variation and is implemented with the lowest cost. However, applied to speed binning, neither deliver a satisfactory profit. Two bins are set: bin1 is set at a clock cycle delay of 9 and bin2 is set at 12, with the price of 200 and 70, respectively. All chips slower than 12 are discarded and deliver no economic return. Consequently, a tradeoff between income and cost so as to achieve profit maximization. An optimal bin placement strategy is also proposed for further profit improvement.

III. PROFIT-AWARE HLS UNDER PROCESS VARIATION

In this section, we present the profit-aware HLS considering process variation.

![Algorithm 1](https://www.example.com)

### Algorithm 1: Profit-aware HLS under process variation.

**Input:** DFG, Resource Library, price profile \( P \), bin setting  
**Output:** HLS solution for DFG  
1: map all the operations to the fastest components  
2: do scheduling; //objective: equalizing slack of all clock cycles  
3: while 1 do  
4: assign each node one corresponding nodePriority \( NP \);  
5: for each operation calculate the operationPriority \( OP \);  
6: \( operationToSlowDown = \max \{ \text{OP}(i) \mid i \in \text{operation of DFG} \} \);  
7: if \( operationToSlowDown < 0 \) then  
8: break; //no operation can be slowed down  
9: else  
10: slowDown(operationToSlowDown);  
11: end if  
12: end while  
13: do binding; //objective: resource sharing among critical paths among all clock cycles

Intuitively, we start with the design with the highest income, by mapping all the operations to the fastest components, and then iteratively slow down some of them based on defined priorities, until we pinpoint the best tradeoff between performance and cost, thus maximizing the profit (Algorithm 1). The slowdown procedure is conducted for all clock cycles simultaneously. For example, if each clock cycle has one adder candidate to slow down, they should be degraded at the same time. Differentially slowing down a component across clock cycles consistently leads to guaranteed suboptimal solutions, as allocation is determined by the maximum cardinality of each type. Consequently, for one operation, only the global slowdown of a component in all clock cycles may conceivably deliver profit\(^1\). The operation to slow down is selected by operation priority (\( OP \)), which is computed based on node priority (\( NP \)). We will apply the proposed HLS solutions on the example shown in Figure 2 for a detailed explanation.

#### A. Equal-slacks guided scheduling

As the initial allocation before scheduling, all the nodes are mapped to the fastest components, with traditional ASAP (as soon as possible) and ALAP (as late as possible) applied to determine mobilities for each node.

Then scheduling assigns each node into an appropriate clock cycle. We define the nodeList of one clock cycle as the set of nodes it possibly holds. It is initialized with all nodes whose mobility spans the clock cycle in question. At each step, one unscheduled node will be removed from one of the nodeList thus narrowing down the scheduling space. At termination, the nodeList contains the exact scheduling result for all the cycles.

For a particular clock cycle, the most problematic case (with the smallest slack) would happen when the nodes in its nodeList are all scheduled in this cycle. A variable, denoted

\(^1\)The strategy of uniformly degrading component performance can be relaxed when clock cycles do not fully utilize resources of the particular component type.
danger Slack, is proposed to describe this characteristic. It is defined as the slack of one clock cycle when all nodes in its nodeList are scheduled into it. Danger Slack outperforms the real slack by considering the potential danger to clock cycles, because it considers not only the nodes that have been scheduled into this cycle, but also those having possibilities to be scheduled in. In the proposed strategy, scheduling starts from the most dangerous clock cycle, namely the one with the smallest danger Slack, and excludes components out of its nodeList to relieve the impact of overcrowding. Thus the excluded node loses one degree of freedom in its mobility and is eventually fixed to a particular clock cycle when only one possibility is left.

Algorithm 2 Equal-slacks Guided Scheduling

Input: an original DFG
Output: the DFG scheduled with all nodes scheduled into appropriate clock cycles

1: apply ASAP and ALAP on the DFG
2: for each clock cycle C, do
3: initialize available resource of Ci with resource library;
4: initialize nodeList of Ci, the set of nodes having mobility across Ci, and corresponding startNodeList, endNodeList;
5: end for
6: while not all nodes are fixed do
7: //danger Slack exclusion
8: for each clock cycle C, do
9: derive danger Slack of Ci;
10: end for
11: get the most dangerous clock cycle Cmax with the smallest danger Slack;
12: find the functional unit FUex to exclude;
13: exclude FUex from Cmax;
14: update nodeList, startNodeList and endNodeList of Cmax;
15: update the ASAP or ALAP of related nodes accordingly;
16: end while

Algorithm 2 describes the scheduling process. Danger Slack exclusion (Line 7-15) protects dangerous clock cycles that potentially have the smallest slack by excluding one component at each iteration. The functional unit (FU) to be excluded is chosen as follows (Line 12). Firstly, its exclusion should impose least impact on others, so it is chosen from either startNodeList or endNodeList of the danger cycle. Secondly, it is suboptimal to exclude one FU whose ASAP (or ALAP) is the danger step, since it imposes a significant restriction for both itself and its successors (or predecessors). Last but not least, the exclusion should bring the current clock cycle the maximum delay reduction to get this cycle a bigger danger Slack. Even if no delay reduction can be obtained by excluding any component because of the parallelism, the exclusion may still be reasonable because delay reduction can possibly be attained in cognition with subsequent exclusions.

Table I shows the scheduling procedure of the example in Figure 2. The scheduling result based on the DFG equipped with fastest components is shown in Figure 3(a).

After scheduling, every node of the DFG will have been fixed into the appropriate clock cycle and the slacks of each clock step are at that point typically evenly distributed. In the following section, we will introduce approaches to replace some of the modules with slower and cheaper module types. Prior to that, we expose the profit benefit measure of slowing one node, i.e. the node priority.

Algorithm 2 Equal-slacks Guided Scheduling

Input: an original DFG
Output: the DFG scheduled with all nodes scheduled into appropriate clock cycles

1: apply ASAP and ALAP on the DFG
2: for each clock cycle C, do
3: initialize available resource of Ci with resource library;
4: initialize nodeList of Ci, the set of nodes having mobility across Ci, and corresponding startNodeList, endNodeList;
5: end for
6: while not all nodes are fixed do
7: //danger Slack exclusion
8: for each clock cycle C, do
9: derive danger Slack of Ci;
10: end for
11: get the most dangerous clock cycle Cmax with the smallest danger Slack;
12: find the functional unit FUex to exclude;
13: exclude FUex from Cmax;
14: update nodeList, startNodeList and endNodeList of Cmax;
15: update the ASAP or ALAP of related nodes accordingly;
16: end while

Algorithm 2 describes the scheduling process. Danger Slack exclusion (Line 7-15) protects dangerous clock cycles that potentially have the smallest slack by excluding one component at each iteration. The functional unit (FU) to be excluded is chosen as follows (Line 12). Firstly, its exclusion should impose least impact on others, so it is chosen from either startNodeList or endNodeList of the danger cycle. Secondly, it is suboptimal to exclude one FU whose ASAP (or ALAP) is the danger step, since it imposes a significant restriction for both itself and its successors (or predecessors). Last but not least, the exclusion should bring the current clock cycle the maximum delay reduction to get this cycle a bigger danger Slack. Even if no delay reduction can be obtained by excluding any component because of the parallelism, the exclusion may still be reasonable because delay reduction can possibly be attained in cognition with subsequent exclusions.

Table I shows the scheduling procedure of the example in Figure 2. The scheduling result based on the DFG equipped with fastest components is shown in Figure 3(a).

After scheduling, every node of the DFG will have been fixed into the appropriate clock cycle and the slacks of each clock step are at that point typically evenly distributed. In the following section, we will introduce approaches to replace some of the modules with slower and cheaper module types. Prior to that, we expose the profit benefit measure of slowing one node, i.e. the node priority.

Algorithm 2 Equal-slacks Guided Scheduling

Input: an original DFG
Output: the DFG scheduled with all nodes scheduled into appropriate clock cycles

1: apply ASAP and ALAP on the DFG
2: for each clock cycle C, do
3: initialize available resource of Ci with resource library;
4: initialize nodeList of Ci, the set of nodes having mobility across Ci, and corresponding startNodeList, endNodeList;
5: end for
6: while not all nodes are fixed do
7: //danger Slack exclusion
8: for each clock cycle C, do
9: derive danger Slack of Ci;
10: end for
11: get the most dangerous clock cycle Cmax with the smallest danger Slack;
12: find the functional unit FUex to exclude;
13: exclude FUex from Cmax;
14: update nodeList, startNodeList and endNodeList of Cmax;
15: update the ASAP or ALAP of related nodes accordingly;
16: end while

Algorithm 2 describes the scheduling process. Danger Slack exclusion (Line 7-15) protects dangerous clock cycles that potentially have the smallest slack by excluding one component at each iteration. The functional unit (FU) to be excluded is chosen as follows (Line 12). Firstly, its exclusion should impose least impact on others, so it is chosen from either startNodeList or endNodeList of the danger cycle. Secondly, it is suboptimal to exclude one FU whose ASAP (or ALAP) is the danger step, since it imposes a significant restriction for both itself and its successors (or predecessors). Last but not least, the exclusion should bring the current clock cycle the maximum delay reduction to get this cycle a bigger danger Slack. Even if no delay reduction can be obtained by excluding any component because of the parallelism, the exclusion may still be reasonable because delay reduction can possibly be attained in cognition with subsequent exclusions.

Table I shows the scheduling procedure of the example in Figure 2. The scheduling result based on the DFG equipped with fastest components is shown in Figure 3(a).

After scheduling, every node of the DFG will have been fixed into the appropriate clock cycle and the slacks of each clock step are at that point typically evenly distributed. In the following section, we will introduce approaches to replace some of the modules with slower and cheaper module types. Prior to that, we expose the profit benefit measure of slowing one node, i.e. the node priority.

Algorithm 2 Equal-slacks Guided Scheduling

Input: an original DFG
Output: the DFG scheduled with all nodes scheduled into appropriate clock cycles

1: apply ASAP and ALAP on the DFG
2: for each clock cycle C, do
3: initialize available resource of Ci with resource library;
4: initialize nodeList of Ci, the set of nodes having mobility across Ci, and corresponding startNodeList, endNodeList;
5: end for
6: while not all nodes are fixed do
7: //danger Slack exclusion
8: for each clock cycle C, do
9: derive danger Slack of Ci;
10: end for
11: get the most dangerous clock cycle Cmax with the smallest danger Slack;
12: find the functional unit FUex to exclude;
13: exclude FUex from Cmax;
14: update nodeList, startNodeList and endNodeList of Cmax;
15: update the ASAP or ALAP of related nodes accordingly;
16: end while

Algorithm 2 describes the scheduling process. Danger Slack exclusion (Line 7-15) protects dangerous clock cycles that potentially have the smallest slack by excluding one component at each iteration. The functional unit (FU) to be excluded is chosen as follows (Line 12). Firstly, its exclusion should impose least impact on others, so it is chosen from either startNodeList or endNodeList of the danger cycle. Secondly, it is suboptimal to exclude one FU whose ASAP (or ALAP) is the danger step, since it imposes a significant restriction for both itself and its successors (or predecessors). Last but not least, the exclusion should bring the current clock cycle the maximum delay reduction to get this cycle a bigger danger Slack. Even if no delay reduction can be obtained by excluding any component because of the parallelism, the exclusion may still be reasonable because delay reduction can possibly be attained in cognition with subsequent exclusions.

Table I shows the scheduling procedure of the example in Figure 2. The scheduling result based on the DFG equipped with fastest components is shown in Figure 3(a).

After scheduling, every node of the DFG will have been fixed into the appropriate clock cycle and the slacks of each clock step are at that point typically evenly distributed. In the following section, we will introduce approaches to replace some of the modules with slower and cheaper module types. Prior to that, we expose the profit benefit measure of slowing one node, i.e. the node priority.

Algorithm 2 Equal-slacks Guided Scheduling

Input: an original DFG
Output: the DFG scheduled with all nodes scheduled into appropriate clock cycles

1: apply ASAP and ALAP on the DFG
2: for each clock cycle C, do
3: initialize available resource of Ci with resource library;
4: initialize nodeList of Ci, the set of nodes having mobility across Ci, and corresponding startNodeList, endNodeList;
5: end for
6: while not all nodes are fixed do
7: //danger Slack exclusion
8: for each clock cycle C, do
9: derive danger Slack of Ci;
10: end for
11: get the most dangerous clock cycle Cmax with the smallest danger Slack;
12: find the functional unit FUex to exclude;
13: exclude FUex from Cmax;
14: update nodeList, startNodeList and endNodeList of Cmax;
15: update the ASAP or ALAP of related nodes accordingly;
16: end while

Algorithm 2 describes the scheduling process. Danger Slack exclusion (Line 7-15) protects dangerous clock cycles that potentially have the smallest slack by excluding one component at each iteration. The functional unit (FU) to be excluded is chosen as follows (Line 12). Firstly, its exclusion should impose least impact on others, so it is chosen from either startNodeList or endNodeList of the danger cycle. Secondly, it is suboptimal to exclude one FU whose ASAP (or ALAP) is the danger step, since it imposes a significant restriction for both itself and its successors (or predecessors). Last but not least, the exclusion should bring the current clock cycle the maximum delay reduction to get this cycle a bigger danger Slack. Even if no delay reduction can be obtained by excluding any component because of the parallelism, the exclusion may still be reasonable because delay reduction can possibly be attained in cognition with subsequent exclusions.

Table I shows the scheduling procedure of the example in Figure 2. The scheduling result based on the DFG equipped with fastest components is shown in Figure 3(a).

After scheduling, every node of the DFG will have been fixed into the appropriate clock cycle and the slacks of each clock step are at that point typically evenly distributed. In the following section, we will introduce approaches to replace some of the modules with slower and cheaper module types. Prior to that, we expose the profit benefit measure of slowing one node, i.e. the node priority.
9, 12 respectively, when the expectation delay of the critical path is 8, the income is estimated by $T_{clk}$ following $(8,0.8^2)$, which computes to 186. Note the critical path here refers to the longest path in one clock cycle instead of the whole DFG.

Thus the incomeReduction is defined as the income difference between critical path delays of the current clock cycle before and after one node slowing down. In this example, if $A$ is slowed down, the duration of the critical path ‘A-B-C’ would increase from 4 to 8, resulting in incomeReduction = income(4) - income(8) = 14.

3) Critical Factor (criF): CriF, ranging in [0,1], represents the impact on the critical path delay of this slowdown. CriF $= 1$ means the node is in the critical path and its slowdown directly results in the income reduction.

$$criF = \frac{\max\{\text{delay of paths that contain this node}\}}{\text{delay of the critical path of } DFG}$$

Due to process variation and multi-types for each operation, the critical path is not readily recognizable. Our solution is to derive the expectation delay of one component under process variation. It is derivable because the delays follow certain delay distributions, either Gaussian or non-Gaussian. As we map all the nodes to the fastest components at first and then slow them down, at each step, the module type is determined. In this way, the delays of each path can be calculated and then used for criF calculation. For node $A$ in this example, $criF = 1$ because the longest path containing $A$ is ‘A-B-C-E-F’, which is exactly the critical path of this DFG.

4) Commonality Factor (comF): For a vertex, to measure how much its replacement with a slower component is going to prevent other vertices in the DFG from also being replaced by slower components, we apply the identical concept of commonality factor proposed in [13]. In Figure 3(b), $comF(C) > comF(E)$ since the slowdown of $C$ prevents all others’ slowdown, whereas the slowdown of $E$ still gives a chance to $D$ because of their parallelism.

$NP(A) = \frac{14 - income(4) - income(8) + \frac{1}{2}}{2} = 5.5$;
$NP(B) = \frac{14 - income(4) - income(7) + \frac{1}{2}}{2} = 6.5$.
Similarly, $NP(D) = 25$; $NP(E) = NP(F) = 11$.

C. Component slowdown based on Operation Priority (OP)

OP is the criterion of choosing the operation to slow down. It is determined as the minimal positive NP of nodes belonging to this operation, implying the lower bound of the profit benefit of degrading this operation globally. In the previous example: $OP(<) = \min\{NP(B), NP(C)\} = 6.5$;
$OP(+) = \min\{NP(A), NP(D), NP(E), NP(F)\} = 5.5$.

The operation to slow down would be the one with the biggest OP, i.e. ‘<’ in this example. According to the procedure described in Algorithm 3, at each clock cycle, among all the components of ‘<’, the one having the biggest NP is picked to slow down. In cycle 1, B and C have the same NP and we replace B with one slow unit. Clock cycle 2 has no < operation and is thus not subject to component degradation.

Past the first iteration, the algorithm would encounter $NP(A) = 56$; $NP(C) = -48$; $NP(D) = 25$; $NP(E) = NP(F) = 11$. So ‘+’ is chosen as the slowdown operation and component $D$ is degraded to the slow type. Notice that the negative NP indicates that the reduction in income can not compensate the reduction in cost, degrading profit. After another slowdown iteration, in which $E$ is degraded to the slow type, all the NPs become negative, indicating that the allocation has been completed, as shown in Figure 3(c).

Algorithm 3 SlowDown

Input: DFG, operation to slow down oper picked based on OP

1: for each clock cycle $j$ of DFG do
2: if there are components of operation oper then
3: node $\leftarrow$ the one with biggest NP in all oper components;
4: $t$ = module type of node;
5: $m$ = currently occupied resource # of operation oper, type $t$ in cycle $j$;
6: end if
7: end for

D. Binding

The binding procedure maps the operations to particular hardware resources. Our objective is the resource sharing among different clock cycles, so as to tighten the correlations among all cycles, which benefits the performance according to timing analysis. The longest path dominates $T_{clk}$, so nodes in the longest paths from all clock cycles ideally should be merged into the same hardware as much as possible.

To implement this approach, each clock step maintains one longPath, which is the longest path in this cycle with at least one component unbound. Among the longPaths of all clock cycles, the longest one is referred to as the criticalPath of the flow graph. The objective of the binding algorithm is to match the same FU-instance mapping between all other longPaths and the criticalPath. The match starts from the biggest unbound component in the criticalPath, because it has the biggest criticality to determine the chip frequency. So it is assigned one current available resource. Then for every longPath, if there is one unbound component that shares the same type with the first assigned one, it will be matched to the same instance if the resource is available. After each mapping iteration, the criticalPath and longPaths are updated, to replace paths that have completed binding with the second longest one. The binding result of the previous example is shown in Figure 3(d), which can be seen to be $S_1$ in Figure 2(b).

IV. OPTIMAL BIN PLACEMENT

In this section we present the optimal bin placement strategy under the assumption that the bin placement can be adjusted by designers. The optimal bin placement problem is defined as follows. Given the circuit $T_{clk}$ distribution $D$, the price profile $P$ and the desirable bin number $n$, to find the corresponding $n$ places for the bin setting, so as to maximize profit.

In this problem formulation, system design is assumed to have been completed, implying that the exact $T_{clk}$ distribution has been derived and the cost is fixed. Profit maximization is consequently solely converted to income maximization, which is implemented by adjusting bin positions and thereby chip numbers in each bin based on the cognizant $T_{clk}$ distribution (Equation 1).

If the number of bins is not restricted, the ideally maximal income would be derived with infinite bins, as every chip is then placed in an individual bin and sold at the most advantageous price. The ideal income is calculated by the

\[ \text{Income} = \sum_{i=1}^{n} \text{price}(i) \times \text{quantity}(i) \]

The detailed derivation of $comF$ can be found in [13].
convolution of circuit distribution and price: \( \text{Income} = \text{convolution}(D, P) \).

Figure 4 shows that the \( \text{Income} \) rises at various rates for different delay regions. For example, in [7, 9] the profit rises sharply, which means that setting a bin here benefits more than setting one in the range [9, 11]. The bin density in a certain range is directly determined by the profit improvement.

**Algorithm 4 Optimal bin boundary selection (OBBS).**

**Input:** circuit \( T_{\text{clk}} \), distribution \( D \), price profile \( P \), bin number \( n \)

**Output:** bin boundaries \( BB \)

1. \( \text{Income} = \text{convolution}(D, P) \); //derive the ideal income
2. \( a = \text{min}(\text{Income}) \);
3. \( b = \text{max}(\text{Income}) \);
4. equally divide region \([a, b]\) by \( n-1 \) boundaries, stored in \( R \);
5. \( \text{Income} = \text{inverse}(\text{Income}) \); //derive the inverse function of \( \text{Income} \)
6. for \( i = 1 : n - 1 \) do
7. \( BB(i) = \text{Income}'(R(i)) \);
8. end for
9. in \([BB(n-1), b]\), find a boundary \( p \) that makes \( \text{set} \ BB(1:n-1) \) give highest income; //determine the last bin
10. \( BB() = p \);

Algorithm 4 presents the optimal bin selection strategy (OBBS). After deriving the ideal \( \text{Income} \) function (Line 1), the \( \text{Income} \) region is equally divided into \( n \) partitions, where \( n \) denotes the bin number. These corresponding partition boundaries, calculated by the inverse function of \( \text{Income} \), are selected as bins. Note that the last bin is somewhat special. If it is selected at the rightmost boundary, which is close to the clock cycle time, almost all the remaining chips would fall into this bin. It delivers a good economic return when the price curve is flat. However, when the price curve is sharp in this region, placing the last bin at the biggest delay results in a much lower pin price, which significantly degrades the chip values in the last bin. To handle this, we use the exhaustive search for the last bin in range \([2\text{-last-bin, clock-cycle-time}]\) (Line 4 - 10). The last bin then will typically be selected as somewhat in the middle of this region when the price curve is sharp and the exact clock cycle time when the price is extremely flat. Figure 4 illustrates the OBBS for \( S_4 \) in Figure 2(b) when 3 bins are needed.

Based on the fact that the bins are essentially selected as the tradeoff between chip numbers in each bin and the bin price, the algorithm places bins according to the derivative of the ideal \( \text{Income} \) function. The algorithm can be understood as follows. First, we derive the derivative curve of \( \text{Income} \) and tend to put more bins in the range with bigger derivative values. Then we distribute the bins proportionally according to the area under the derivative curve, as the integral of the derivative function would signify the original \( \text{Income} \). Consequently, the bins are selected as the boundaries of the equal division of the \( \text{Income} \) range.

V. Experiments

We have tested the proposed design approaches on the same HLS benchmarks used in [14]. They provide a representative spectrum because of the differences in size and parallelism (nodes/critical path). We adopt the price profile of Intel Prescott processors, which is simulated from the data given in [10].

\[
 p = 0.0000139 \cdot e^{3.384 \cdot \text{fre}^2} + 1.473;
\]

where \( \text{fre} \) denotes the labeled speed of one chip when sold.

A. Evaluation of the profit-aware HLS under process variation

The proposed strategy is both process variation aware and profit aware, which is denoted as double-aware (DA). To evaluate the necessity of process variation awareness, we compare the profit derived by the proposed DA strategy with process variation unaware (PV-unA) approach. To confirm the importance of tradeoff between income and cost, we also compare the profit performance between DA and performance-yield-optimization (PYO) based method.

The PV-unA method is implemented by aggressive scheduling and blind binding. The aggressive scheduling step schedules components to the current clock cycle as long as the overall delay is smaller than the clock cycle time. The blind binding step matches available components to operations in descending speed order and subject to the constraint of ensuring no timing constraint violations in the clock cycle. The PYO strategy maximizes the chip numbers in the most expensive bin. It selects the components with best performance, performs equal-slab guided scheduling and maximizes the resource sharing among critical paths of all the clock cycles in binding.

Monte Carlo methods are applied to simulate the inherent correlations between clock cycle distributions that share the same resource. To make the experiment consistent and efficient, for all test benches, the clock cycle time is chosen to be 18, with the two bins set at 14 and 18, respectively.

**Table III**

**Profit improvement with components following Gaussian delay distributions.**

<table>
<thead>
<tr>
<th>Bench</th>
<th>Clock cycle time = 18. Bin setting: (14, 18).</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \text{Profit} )</td>
</tr>
<tr>
<td></td>
<td>( \text{PV-unA} )</td>
</tr>
<tr>
<td>ARF</td>
<td>0.97</td>
</tr>
<tr>
<td>EWF</td>
<td>0.86</td>
</tr>
<tr>
<td>Cos1</td>
<td>0.58</td>
</tr>
<tr>
<td>Cos2</td>
<td>0.55</td>
</tr>
<tr>
<td>jWBH</td>
<td>0.48</td>
</tr>
<tr>
<td>mMM</td>
<td>0.39</td>
</tr>
<tr>
<td>jFDCT</td>
<td>0.79</td>
</tr>
<tr>
<td>Average</td>
<td>0.92</td>
</tr>
</tbody>
</table>

Table III shows the profit improvement with components following Gaussian distributions. On average, the proposed DA has a 63.5% profit improvement in comparison to PV-unA, and a 19.7% improvement in comparison to the PYO method. The improvement over PV-unA benefits from the handling with process variation, whereas the advantage over PYO results from design cost reduction. Take jFDCT, for example. Due to the process-variation-unaware strategy, PV-unA aggressively schedules too many components in one clock cycle, resulting in only 59% of the chips eligible for Bin2 and no chip in
Evidently the low income is the reason for unsatisfactory profit. PYO achieves the highest income but is also equipped with the most expensive cost. DA slows down 27% of the components used by PYO, thus significantly lowering the cost. At the same time, it keeps 98% chips sold at the higher price and the other 2% falling into the second bin, thus retaining a satisfactory income.

We also conduct the identical experiments with components that follow a non-Gaussian delay distribution, shown in Table IV. Comparing with Uniform and PYO, DA achieves 66.0% and 22.0% for Uniform component distributions; 70.2% and 26.0% for Triangle distributions. This evinces that the proposed HLS approach is suitable for both Gaussian and non-Gaussian distributions and superior at dealing with big variations, furthermore.

### B. Evaluation of the optimal bin placement

One previous work also presents a heuristic of the optimal bin placement for profit maximization, in which the bins are initialized by equally dividing the performance yield and the bins are shifted gradually until no profit improvement can be observed [10]. We refer to it as EY (equal-yield based) approach in this paper.

Though we have the same problem definition, the EY strategy is applicable only when \( T_{clk} \) follows a Gaussian distribution. In contrast, the proposed OBBS (optimal bin boundary selection) method is suitable for any kind of distribution. In order to compare with the EY strategy, we first evaluate the OBBS with Gaussian distributions, and then test it on the benchmarks with complicated non-Gaussian \( T_{clk} \) distributions, to evaluate the profit compared with the exhaustive bin boundary search.

### VI. Conclusion

In this paper, we introduce speed binning into the HLS domain to help derive maximal economic return. Taking process variation into account, we develop a set of HLS approaches for profit maximization, including allocation, scheduling and binding. Experimental results confirm the superiority of HLS results and the associated improvement in profit margins, when said components follow Gaussian, Uniform and Triangle delay distributions. We also propose an optimal bin boundary selection (OBBS) algorithm, delivering a near-optimal performance. To sum it, this paper constructs both process variation aware and profit aware HLS designs in the context of speed binning, delivering maximal economic profit.

### Acknowledgment

This work is partially supported by grants from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 123811 and 123210].

### REFERENCES


#### Table IV

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Uniform Bin Improvement (%)</th>
<th>Triangle Bin Improvement (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DA vs. PYO</td>
<td>DA vs. EY</td>
</tr>
<tr>
<td>ARP</td>
<td>84.8</td>
<td>5.4</td>
</tr>
<tr>
<td>EWP</td>
<td>66.5</td>
<td>14.0</td>
</tr>
<tr>
<td>Cos1</td>
<td>50.9</td>
<td>11.5</td>
</tr>
<tr>
<td>Cos2</td>
<td>53.8</td>
<td>14.0</td>
</tr>
<tr>
<td>JWBH</td>
<td>80.9</td>
<td>45.9</td>
</tr>
<tr>
<td>mMM</td>
<td>68.6</td>
<td>19.2</td>
</tr>
<tr>
<td>jWBH</td>
<td>84.6</td>
<td>24.8</td>
</tr>
<tr>
<td>Average</td>
<td>66.0</td>
<td>22.0</td>
</tr>
</tbody>
</table>

#### Table V

<table>
<thead>
<tr>
<th>Bin#</th>
<th>Uniform σ = 0.8</th>
<th>Uniform σ = 1.0</th>
<th>Uniform σ = 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n=2, n=4</td>
<td>n=2, n=4</td>
<td>n=2, n=4</td>
</tr>
<tr>
<td></td>
<td>n=2, n=4</td>
<td>n=2, n=4</td>
<td>n=2, n=4</td>
</tr>
<tr>
<td>EY</td>
<td>4.17, 5.11, 6.11</td>
<td>4.25, 5.35, 6.39</td>
<td>4.32, 5.39, 6.90</td>
</tr>
<tr>
<td>OBBS</td>
<td>4.46, 5.47, 6.33</td>
<td>4.43, 5.66, 6.81</td>
<td>4.21, 5.08, 7.49</td>
</tr>
<tr>
<td>optimal</td>
<td>4.48, 5.48, 6.36</td>
<td>4.44, 5.67, 6.83</td>
<td>4.22, 5.04, 7.30</td>
</tr>
</tbody>
</table>

Table V presents the income results for Gaussian distributions. Without loss of generality, we set the mean as 10 and test various deviations. The results show that the proposed OBBS strategy has a 5.7% income improvement on average in comparison to the EY method and always performs better than 99.5% of the optimal results. OBBS outperforms EY more significantly with severe process variation (bigger \( \sigma/\mu \)) and more bins.

We also apply OBBS on test benches HLS solutions. To evaluate the profit performance, exhaustive search is conducted to obtain the optimal bin placement. Figure 5 presents the profit relative to the optimal solution, implying 98.9% of the optimal solution on average.

![Profit relative to the optimal bin placement, when applying OBBS on test benches](image-url)