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:

$$Profit = income - cost = \sum_{i=1}^{n} p_i \cdot n_i - cost \qquad (1)$$

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 *cost* denotes the cost of the design. Figure 1 shows one example with 3 bins. The price is a stair-case function of T_{clk} delay. Profit is maximized by intelligently distributing chips into bins for aggressive income and retaining a low design cost at the same time.

978-3-9815370-0-0/DATE13/©2013 EDAA

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_{clk}) by HLS solutions, which determines the economic profit in the context of speed binning.

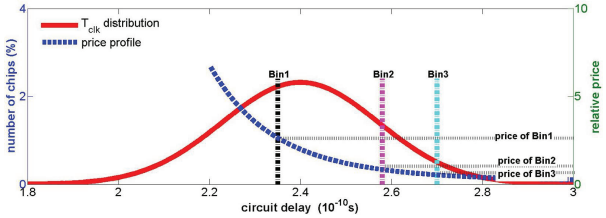


Fig. 1. Income calculation.

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.

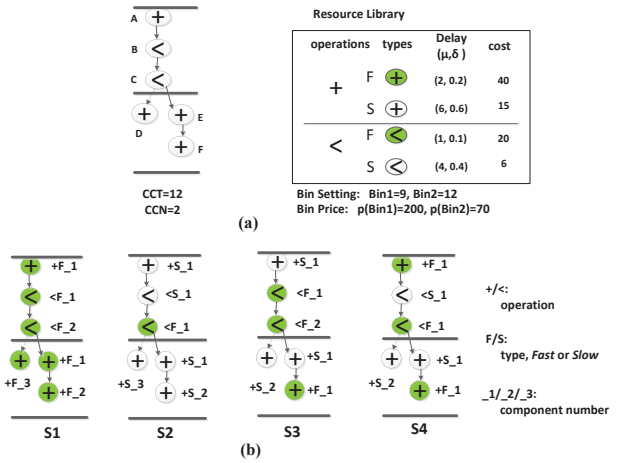
The remainder of this paper is organized as follows. Section II shows an example to illustrate how the HLS decisions affect economic profit. Section III accordingly introduces profit-aware HLS strategies. Section IV presents the optimal bin

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, when 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. Figure 2(c) lists the economic features. Though S_2 has the lowest implementation cost, due to the uncertainties resulting from process variation, 51% of the chips are discarded, and the other 49% are only eligible for the slower bin, resulting in a comparatively lower income, which falls short of compensating for the cost and thus delivers no profit. In contrast, S_1 derives the best income (with 100% eligible chips sold at the higher price), yet also with the most expensive design cost, resulting in an unsatisfactory profit. Consequently, a tradeoff between high income and low cost should be taken into account for profit optimization. In this case, by intelligently selecting the appropriate cost-effective components and an associated binding strategy, S_3 and S_4 attain various tradeoffs. Both of them deliver a quite high income, which not only suffices to compensate for the comparatively high cost, but also delivers additional profit.



	Chip #		income	cost	profit
	Bin1	Bin2			
S_1	100%	0%	1.00*200+0.00*70=**200**	40*3+20*2=**160**	200-160=**40**
S_2	0%	49%	0.00*200+0.49*70=**34.3**	15*3+20*1+6*1=**71**	34.5-71= **-36.7**
S_3	93%	7%	0.93*200+0.07*70=**190.9**	40*1+15*2+20*2=**110**	190.8-110=**80.9**
S_4	95%	5%	0.95*200+0.05*70=**193.5**	40*1+15*2+20*1+6*1=**96**	193.5-96=**97.5**

(c)

Fig. 2. HLS example. Clock cycle time=12. Clock cycle number=2. (a) DFG and settings. Each operation has two types: Fast and Slow. (b) Four possible HLS solutions. (c) Economic features of these four solutions.

To sum it, as HLS steps, allocation decides the cost; scheduling and binding determine the income by intelligently distributing chips into various speed bins. In this paper, we explore the intertwined HLS solutions for the best 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 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 slacks 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{OP(i)| i∈ 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*s thus narrowing down the scheduling space. At termination, the *nodeList*s denotes 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.

dangerSlack, 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. *DangerSlack* 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 dangerSlack, 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* with all nodes scheduled into appropriate clock cycles

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

Algorithm 2 describes the scheduling process. *DangerSlack* 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 *dangerSlack*. 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.

TABLE I
SCHEDULING PROCEDURE OF THE EXAMPLE IN FIGURE 2. GIVEN 2 CLOCK CYCLES.

Step		NodeList	DangerSlack	Action / Result
1	CC1	ABCDEF	*4*	exclude F / F fixed in CC2
	CC2	ABCDEF	4	-
2	CC1	ABCDE	6	-
	CC2	ABCDEF	*4*	exclude A / A fixed in CC1
3	CC1	ABCDE	*6*	exclude D / D fixed in CC2
	CC2	BCDEF	6	-
4	CC1	ABCE	*6*	exclude E / E fixed in CC2
	CC2	BCDEF	6	-
5	CC1	ABC	8	-
	CC2	BCDEF	*6*	exclude B / B fixed in CC1
6	CC1	ABC	8	-
	CC2	CDEF	*7*	exclude C / C fixed in CC1
7	CC1	ABC		
	CC2	DEF	scheduling completed.	

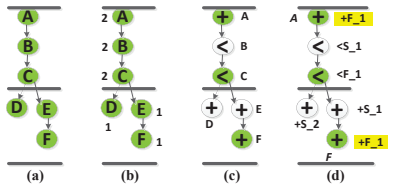


Fig. 3. HLS decisions. (a) Scheduling. (b) Commonality Factors. (c) Allocation. (d) Binding. *A* and *F* share the same *adder*.

B. Node Priority (NP)

Node priority of one vertex is developed to denote the potential profit benefit when slowed down. When a node is replaced with a slower component, both the cost and income would potentially drop, with the profit benefit being determined by the severity of the individual decreases.

$$NP = \frac{costReduction - incomeReduction * criticalFactor}{commonalityFactor} \tag{2}$$

1) CostReduction: The *costReduction* is the price difference between the previously assigned fast type and its replacement. In the previous example, it would be $40 - 15 = 25$ if the fast *adder* is replaced with the slow type.

2) IncomeReduction: Before defining *incomeReduction*, we explore approaches to estimate *income*. Given the price profile and bin settings, *income* is determined by the T_{clk} distribution. However, it is hard to derive the exact T_{clk} distribution without allocation, scheduling and binding information from HLS. Based on the fact that T_{clk} is mainly decided by the critical path, we approximate T_{clk} by the expectation delay of the critical paths of each clock cycle. In the *income* estimation, the T_{clk} distribution is assumed to follow a Gaussian distribution (μ, σ) with $\sigma/\mu = 0.1$. We choose Gaussian because component delays are widely estimated as Gaussian distributions and the circuit *SSTA* result can accurately be approximated by Gaussian distributions [12].

TABLE II
T_{clk}-INCOME ESTIMATION TABLE FOR *incomeReduction* ESTIMATION

$T_{clk}(10^{-10}s)$	≤ 6	7	8	9	10	11	12
income (price unit)	200	199	186	134	89	62	36

Table II shows the estimation with settings in Figure 2. For example, with clock cycle time being 12 and two bins set at

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.

$$criF = \frac{max\{delay\_of\_paths\_that\_contain\_this\_node\}}{delay\_of\_the\_critical\_path\_of\_DFG} \quad (3)$$

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[2].

Based on all the previously defined variables:
NP(A)=$\frac{25-(income(4)-income(8))*\frac{12}{12}}{2}$ = 5.5;
NP(B)=NP(C)=$\frac{14-(income(4)-income(7))*\frac{11}{11}}{2}$ = 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).

[2]The detailed derivation of *comF* can be found in [13].

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* ← 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 *longPath*s 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 *longPath*s 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 *longPath*s 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_4 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 thus 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

convolution of circuit distribution and price: $Income = convolution(D, P)$.

Figure 4 shows that the *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_{clk} distribution *D*, price profile *P*, bin number *n*
Output: bin boundaries *BB*

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

Algorithm 4 presents the optimal bin selection strategy (OBBS). After deriving the ideal *Income* function (Line 1), the *Income* region is equally divided into *n* partitions, where *n* denotes the bin number. These corresponding partition boundaries, calculated by the inverse function of *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 bin 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 [2nd-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.

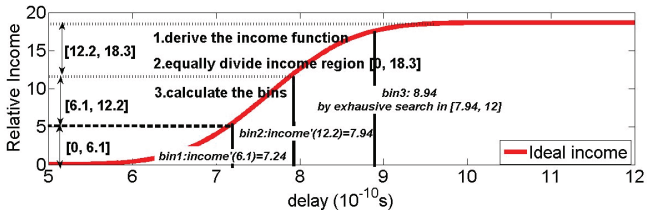


Fig. 4. OBBS for S_4 in Figure 2(b). Given bin number n=3.

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 *Income* function. The algorithm can be understood as follows. First, we derive the derivative curve of *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 *Income*. Consequently, the bins are selected as the boundaries of the equal division of the *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].

$$p = 0.0000139 * e^{3.384*fre} + 1.473; \quad (4)$$

where *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-slack 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.

test Bench	Clock cycle time = 18. Bin setting: (14, 18).				
	profit			profit improvement (%)	
	PV-unA	*PYO*	*DA*	*DA v.s. PV-unA*	*DA v.s. PYO*
ARF	0.67	0.90	0.98	46.6	8.9
EWF	0.66	0.96	1.10	67.2	14.6
Cos1	0.58	0.70	0.93	61.5	32.4
Cos2	0.55	0.77	0.86	56.4	12.0
jWBH	0.48	0.60	0.76	60.4	26.6
mMM	0.39	0.56	0.66	68.0	18.9
jFDCT	0.79	1.16	1.44	83.9	24.4
Average				63.4	19.7

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

Bin1. 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*.

TABLE IV
PROFIT IMPROVEMENT WITH COMPONENTS FOLLOWING NONGAUSSIAN (UNIFORM AND TRIANGLE) DELAY DISTRIBUTIONS.

test Bench	profit improvement (%)			
	nonGaussian-Uniform		nonGaussian-Triangle	
	DAv.s.PV-unA	*DAv.s.PYO*	*DAv.s.PV-unA*	*DAv.s.PYO*
ARF	44.8	7.5	54.1	8.6
EWF	66.5	14.0	67.1	22.9
Cos1	60.9	31.1	61.4	47.1
Cos2	55.8	11.5	67.4	13.4
jWBH	80.9	45.9	71.8	41
mMM	68.6	19.2	80.0	20.3
jFDCT	84.6	24.8	89.6	28.9
Average	66.0	22.0	70.2	26.0

We also conduct the identical experiments with components that follow a non-Gaussian delay distribution, shown in Table IV. Comparing with *PV-unA* 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.

TABLE V
INCOME RESULT FOR GAUSSIAN DISTRIBUTIONS (μ, σ). $\mu = 10$.

	$\sigma = 0.8$			$\sigma = 1.0$			$\sigma = 1.2$		
Bin#	n=2	n=4	n=10	n=2	n=4	n=10	n=2	n=4	n=10
EY [10]	4.17	5.17	6.13	4.23	5.35	6.49	4.32	5.59	6.96
OBBS	4.46	5.47	6.33	4.43	5.65	6.81	4.51	6.00	7.49
optimal	4.48	5.48	6.36	4.44	5.67	6.83	4.52	6.01	7.50

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 σ/μ) 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.

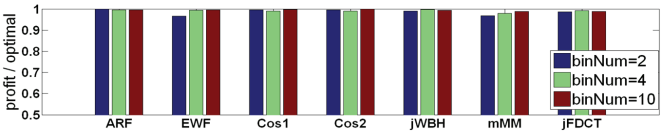


Fig. 5. Profit relative to the optimal bin placement, when applying OBBS on test benches.

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, delievering 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

[1] T. Karnik, S. Borkar, and V. De, "Sub-90 nm technologies-challenges and opportunities for cad," in *ICCAD*, 2002, pp. 203–206.
[2] R. Teodorescu and J. Torrellas, "Variation-aware application scheduling and power management for chip multiprocessors," in *ISCA*, 2008, pp. 363–374.
[3] B. Cory, R. Kapur, and B. Underwood, "Speed binning with path delay test in 150-nm technology," *IEEE Design and Test of Computers*, vol. 20, pp. 41–45, 2003.
[4] D. Belete, A. Razdan, W. Schwarz, R. Raina, C. Hawkins, and J. Morehead, "Use of DFT techniques in speed grading a 1 GHz+ microprocessor," in *ITC*, 2002, pp. 1111–1119.
[5] D. D. Gajski, N. D. Dutt, A. C.-H. Wu, and S. Y.-L. Lin, *High-level synthesis: introduction to chip and system design*. Kluwer, 1992.
[6] F. Wang, C. Nicopoulos, X. Wu, Y. Xie, and N. Vijaykrishnan, "Variation-aware task allocation and scheduling for MPSoC," in *ICCAD*, 2007, pp. 598–603.
[7] Y. Chen, J. Ouyang, and Y. Xie, "ILP-based scheme for timing variation-aware scheduling and resource binding," in *International SOC Conference*, 2008, pp. 27–30.
[8] J. Jung and T. Kim, "Scheduling and resource binding algorithm considering timing variation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, pp. 205–216, 2011.
[9] F. Wang, Y. Xie, and A. Takach, "Variation-aware resource sharing and binding in behavioral synthesis," in *ASPDAC*, 2009, pp. 79–84.
[10] A. Datta, S. Bhunia, J. H. Choi, S. Mukhopadhyay, and K. Roy, "Speed binning aware design methodology to improve profit under parameter variations," in *ASPDAC*, 2006, pp. 712–717.
[11] A. Davoodi and A. Srivastava, "Variability driven gate sizing for binning yield optimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, pp. 683–692, 2008.
[12] A. Datta, S. Bhunia, S. Mukhopadhyay, N. Banerjee, and K. Roy, "Statistical modeling of pipeline delay and design of pipeline under process variation to enhance yield in sub-100nm technologies," in *DATE*, 2005, pp. 926–931.
[13] S. Bakshi and D. Gajski, "Component selection for high-performance pipelines," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 4, pp. 181–194, 1996.
[14] J. Jung and T. Kim, "Timing variation-aware high-level synthesis considering accurate yield computation," in *ICCD*, 2009, pp. 207–212.