Register Allocation for High-Level Synthesis Using Dual Supply Voltages

Insup Shin
Dept. of Electrical Engineering, KAIST
Daejeon 305-701, Korea

Seungwhun Paik
Dept. of Electrical Engineering, KAIST
Daejeon 305-701, Korea

Youngsoo Shin
Dept. of Electrical Engineering, KAIST
Daejeon 305-701, Korea

ABSTRACT

Reducing the power consumption of memory elements is known to be the most influential in minimizing total power consumption, since designs tend to use more memories these days. In this paper, we address a problem of high-level synthesis with the objective of minimizing power consumption of storage using dual-V_{dd}. Specifically, we propose a complete design framework that starts from dual-V_{dd} scheduling, dual-V_{dd} allocation, and controller synthesis down to the final layout. Its main feature is dual-V_{dd} register allocation, which exploits timing slacks left in the data-path after operation scheduling. In experiments on benchmark designs implemented in 1.08 V (with V_{ddl} of 0.8 V), 65-nm CMOS technology, both switching and leakage power were reduced by 20% on average, respectively, compared to data-path with dual-V_{dd} applied to functional units alone. Detailed analysis of slack histogram, area, wirelength, and congestion were performed to assess feasibility of the design framework.

Categories and Subject Descriptors: B.5.2 [**Register-transfer-level implementation**]: Design Styles—*Automatic synthesis*

General Terms: Algorithms, Design, Performance

Keywords: High-level synthesis, register allocation, dual supply voltage, low power

1. INTRODUCTION

Reducing supply voltage (V_{dd}) is the most effective way to reduce switching power due to its quadratic dependence on V_{dd}; it is also effective in reducing subthreshold leakage, which is roughly proportional to ${V_{dd}}^3$, and in reducing gate leakage, which is proportional to ${V_{dd}}^4$ [1]. Since reducing single overall V_{dd} increases circuit delay, multiple-V_{dd} technique, which applies higher V_{dd} to timing-critical elements and lower V_{dd} to elements that are not critical to timing, is employed instead to maintain clock frequency. Due to complexity of building power network [2], two V_{dd}s (dual-V_{dd}) are typically used at gate-level of granularity, i.e. V_{dd} is assigned gate by gate; when more than two V_{dd}s are used, they are used at block-level, where each block is put in its own voltage island [3].

Several algorithms have been proposed to determine V_{dd}-type (high- or low-V_{dd}) at gate-level netlist [4–6]; these algorithms are applied after netlist is determined, while netlist itself is derived without dual-V_{dd} in mind (e.g. only high-V_{dd} is assumed during logic synthesis), which limits the scope low-V_{dd} can be applied.

DAC'09, July 26-31, 2009, San Francisco, California, USA

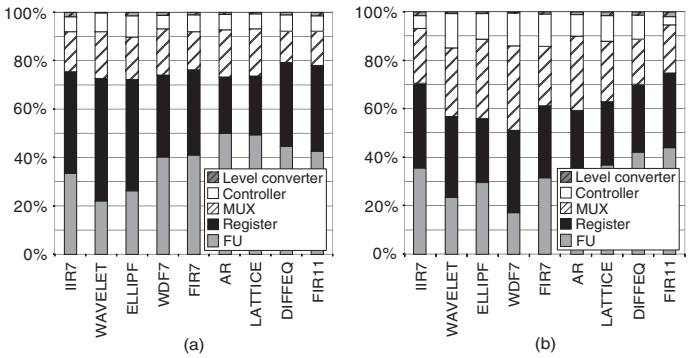

Figure 1: Distribution of (a) switching- and (b) leakage-power of several benchmark circuits when dual-V_{dd} is applied to functional units alone.

Therefore, it is natural to consider dual-V_{dd} at earlier design stage such as during architectural design. Many algorithms have been proposed to determine V_{dd}-type of functional units (such as adders and multipliers) during high-level synthesis (HLS) [7–12]; the problem can be cast to conventional operation scheduling except that the same functional unit with different V_{dd}-type is considered as different units. Surprisingly, however, no research effort has been made toward dual-V_{dd} register allocation, even though the proportion of memory elements (in recent complex designs such as SoCs and microprocessors) is substantial, about 80% of total transistors [13].

To assess the importance of storage (together with multiplexers that are located between storage and functional units) in power consumption, we took a set of of behavioral benchmark designs. Each design was transformed into a DFG (data flow graph); dual-V_{dd} scheduling [11] was applied to determine V_{dd}-type of each operation, which was followed by allocating functional units; remaining steps of HLS were performed when we assume that only high-V_{dd} can be used for registers, multiplexers, and controller; RTL design was then synthesized to obtain a gate-level netlist, which was taken for fast SPICE simulation [14]; simulation was repeated 100 times using different input vectors in commercial 65-nm technology to obtain both switching- and leakage-power consumption. Figure 1(a) shows that the registers contribute, on average, 36% to the total switching power of these circuits (multiplexers contribute 17%), which is substantial; they contribute 30% on average to the total leakage power (26% for multiplexers) as shown in Figure 1(b).

Selecting V_{dd}-type of register and functional unit affects each other when they are on the same timing path from register to register. Since register allocation is usually performed after scheduling, in this paper, we take a scheduled DFG, where V_{dd}-type of each operation is already determined, and try to determine V_{dd}-type of variables, which are then mapped to registers. Central to this approach is how we exploit timing slacks that are left in the data-path

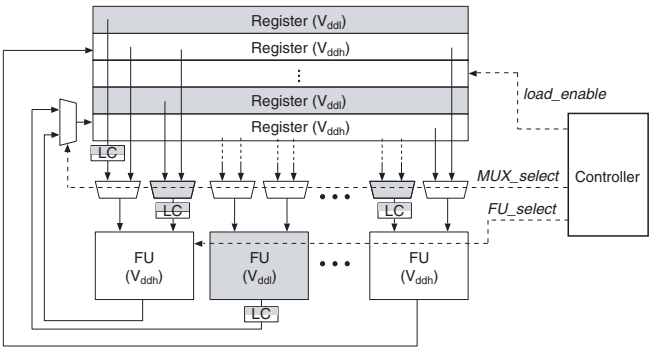

Figure 2: Target architecture for dual-V_{dd} HLS.

after dual-V_{dd} scheduling is performed, so that we can respect the latency and the number of functional units, which are either given by designers as constraints or determined as a result of scheduling. Our main contributions are:

- A complete framework for dual-V_{dd} HLS that covers design processes from scheduling, allocation, and controller synthesis to the final circuit layout (Section 2.2).
- A heuristic solution to dual-V_{dd} register allocation, which is a main feature of the framework, minimizing the number of high-V_{dd} registers as well as total number of registers (Section 3); a heuristic for assigning V_{dd}-type to multiplexers (Section 4).
- Extensive experimental results from commercial 65-nm technology applied to behavioral benchmark designs to assess effectiveness of dual-V_{dd} register allocation on power, and to assess the framework on timing and physical design aspects (Section 5).

2. PRELIMINARIES

2.1 Target Architecture for HLS

We consider a register file-based architecture as a target of dual-V_{dd} HLS. As shown in Figure 2, it consists of a data-path (functional units, registers, and their connections) and a controller. Each functional unit is assigned either high-V_{dd} (V_{ddh}) or low-V_{dd} (V_{ddl}) as a result of scheduling and allocating functional units [11]. For simplicity of presentation, the controller is assumed to be assigned V_{ddh}, which eliminates the need for level converters between data-path and controller.

The delay of timing path across data-path has several components:

$$D_{dp} = T_{cq} + \alpha \cdot T_{mux} + \beta \cdot T_{lc} + T_{FU} + T_{su}, \quad (1)$$

where T_{cq} and T_{su} are clock-to-Q delay and setup time of a register, respectively, and T_{FU} is a (maximum) delay through a functional unit; T_{mux} and T_{lc} denote the delay of a multiplexer and a level converter, respectively. The number of multiplexers, α, and the number of level converters, β, on the timing path are determined after resource allocation and dual-V_{dd} assignment on all data-path components. The controller delay can also be factored in (1) through estimating its delay [15]; or, it can be assumed to be negligible if all control signals (see Figure 2) are ready before data actually arrive (e.g. if MUX_select arrives before multiplexer inputs do), which can be done via adjusting clock arrival times to controller.

Note that (1) is needed during operation scheduling, i.e. the delay (as a number of control steps) required for executing i-th operation (when it is executed on a resource having T_{FU} delay) is given by

$$d_i = \left\lceil \frac{D_{dp}}{P_c} \right\rceil, \quad (2)$$

where P_c is a designer-specified clock period. The V_{dd}-type of a functional unit can be determined during scheduling by assuming two T_{FU}s (one for V_{ddh} and the other for V_{ddl}). However, the values of parameters in (1) except for T_{FU} are unknown when they are needed in (2), since resource allocation is typically performed after scheduling. Therefore, during scheduling, we assume that V_{ddh} is initially assigned to all registers, which gives us the values for T_{cq} and T_{su}, and α=2 (one before each FU input and the other after FU output); we also assume V_{ddh} for multiplexers†; β=1 is assumed when functional unit is assigned V_{ddl}, i.e. we assume a level converter after the output of functional unit.

Due to the way operation delay is defined as in (2), some timing paths across data-path may have timing slacks that can be utilized to assign V_{ddl} to some registers (thus increasing T_{cq} and T_{su}) during dual-V_{dd} register allocation in Section 3.

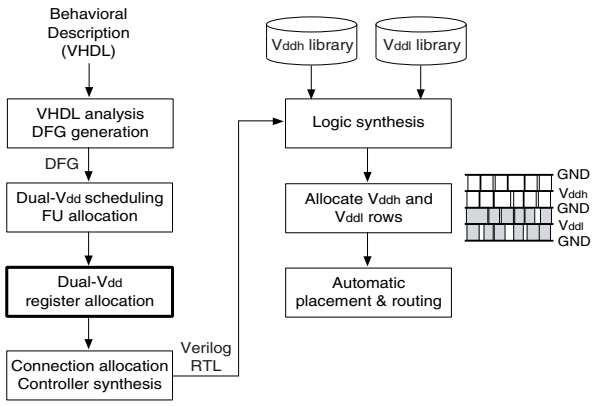

Figure 3: Overall design flow based on dual-V_{dd} HLS.

2.2 Dual-V_{dd} Design Framework

The overall design flow based on dual-V_{dd} HLS is shown in Figure 3. A behavioral description written in VHDL is first analyzed [16] and is then transformed into a DFG [17]. The DFG is an input to dual-V_{dd} operation scheduling [11], which, together with subsequent step of allocating functional units, determines V_{dd}-type of each functional unit, while minimizing latency under resource constraints (or its dual). The scheduled DFG will then be an input to register allocation, which will be described in Section 3. The connection allocation, which uses multiplexers to connect registers to functional units and functional units to registers, is formulated as vertex coloring of a connection conflict graph [18], which is performed by left edge algorithm [19]. The FSM controller is synthesized as a hard-wired sequential circuits; thus, it is described as a state transition graph [18]. Allocation and control synthesis generates the data-path and controller as a Verilog HDL, which goes through a standard logic synthesis [20] to create a gate-level netlist. Two libraries of gates were created and used: one for V_{ddh} (1.08 V in our experiment) and the other for V_{ddl} (0.80 V).

For the physical design of dual-V_{dd} netlist, we first determine the number of V_{ddh} and V_{ddl} placement rows that will be required,

†We assume 10-to-1 multiplexer during scheduling. The exact number of inputs of each multiplexer can only be determined after register allocation and connection allocation. The difference of delay arising from this assumption is utilized when we determine V_{dd}-type of multiplexers in Section 4.

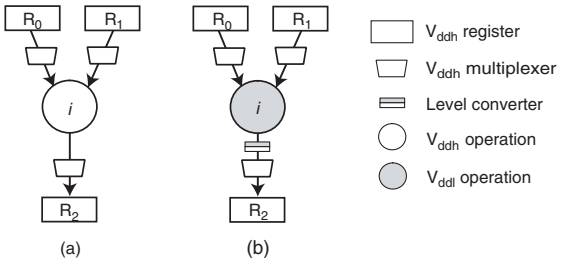

Figure 4: Data-path after dual-V_{dd} scheduling: operation is assigned (a) V_{ddh} and (b) V_{ddl}.

based on the area of the cells to be placed in each type of row. This is followed by decision of how to interleave them, i.e. how many V_{ddh}-rows are followed by how many V_{ddl}-rows. In order to apply double-back layout pattern, which helps reduce area, we try to group even number of consecutive V_{ddh}- or V_{ddl}-rows, e.g. every four V_{ddh}-rows followed by every two V_{ddl}-rows, which is then followed by automatic placement and routing [21] of the netlist.

3. DUAL-V_{dd} REGISTER ALLOCATION

We accept a scheduled DFG, where each operation is assigned either V_{ddh} or V_{ddl}, as an input to dual-V_{dd} register allocation, which consists of the following three steps. We use Figure 5 as an example to illustrate each step.

3.1 Find Feasible V_{dd} Combinations of Operands

After dual-V_{dd} scheduling, data-path can be either in the form of Figure 4(a) when operation i is assigned V_{ddh} or Figure 4(b) when it is assigned V_{ddl}. In this data-path, we want to determine whether input or output operands can be assigned V_{ddl}[†]. We compute slack along each timing path from input operand to output operand. In Figure 4, we first compute the timing slack from R_0 to R_2,

$$S_l = d_i \cdot P_c - D_{dp,l}, \tag{3}$$

where $D_{dp,l}$ is the delay of timing path across data-path from R_0 to R_2 ($D_{dp,r}$ is the delay from R_1 to R_2; thus D_{dp} in (1) represents the maximum of $D_{dp,l}$ and $D_{dp,r}$). We also compute the amount of time required to assign V_{ddl} to R_0,

$$\Delta T_{cq} = T_{cq}(V_{ddl}) - T_{cq}(V_{ddh}) + T_{lc}\text{[‡]}, \tag{4}$$

and the amount of time required to assign V_{ddl} to R_2,

$$\Delta T_{su} = T_{su}(V_{ddl}) - T_{su}(V_{ddh}). \tag{5}$$

We then compare S_l to ΔT_{cq}, ΔT_{su}, and $\Delta T_{cq} + \Delta T_{su}$ to see if we can assign V_{ddl} to R_0, R_2, or both of them. We repeat the process for the path from R_1 to R_2.

In Figure 5(b), combinations of operand V_{dd}-type are illustrated within each operation, where the first two letters correspond to input operands (for binary operations) and the last letter to output operand. Since V_{dd}-type of each operand is determined independently in each operation and operand can be shared in different operations, there can be inconsistencies of V_{dd}-type. In Figure 5(b), z_1 always takes V_{ddh} in op_1, which is why the third combination in op_4 (where z_1 is V_{ddl}) is crossed out; we cross out the second combination in op_6 because z_2 has to be assigned V_{ddh} in op_2.

[†] V_{dd}-type is assigned to functional unit (not to operation) and register (not to operand). Assigning V_{dd} to operation and operand implies assignment before allocation.

[‡] Note that we need a level converter to use V_{ddl} for R_0, since multiplexer is initially assigned V_{ddh}. We may instead try to locate level converter at the output of multiplexer while we use V_{ddl} for multiplexer; we postpone the decision of V_{dd}-type of multiplexer (see Section 4), since exact number of multiplexer inputs can only be determined after connection allocation.

3.2 Categorize Variables

3.2.1 Initial Categorization

Once we derive candidate V_{dd}-types of operands, we categorize operands (or variables) into three groups: V_{ddh}-variables, independent variables, and dependent variables. V_{ddh}-variables (HV) are those whose V_{dd}-types are fixed to V_{ddh} (note that there are no V_{ddl}-variables, because if any variable can be assigned V_{ddl}, it can be assigned V_{ddh} as well): z_1, z_2, and y_1 belong to this category as shown in Figure 5(c). Independent variables (IV) can take either V_{ddh} or V_{ddl}, and the decision does not affect V_{dd}-type of other variables; if the decision affects other variables (even though they can take both types of V_{dd}), they are called dependent variables (DV). In Figure 5(c), x_1 and x_2 belong to IV, because op_1 has all four combinations of V_{dd}-type (HH, HL, LH, and LL) for them, while x_2 is fixed to V_{ddh}; z_3 is DV because if it is assigned V_{ddl}, x_3 has to be assigned V_{ddh} (see op_4).

The decision of IV or DV can be made as follows: for each variable (which is not HV), we check each operation that takes the variable as one of its operands one by one; in each operation, we remove the letters corresponding to HVs; if there are all possible combinations of V_{dd}-type for the remaining variables, the variable is IV in that particular operation; if the variable is IV in all the operations, it is IV, otherwise it is DV. In op_4 of Figure 5(c), we remove the letters (H) for z_1 that leaves us three combinations (HH, LH, and HL), thus both x_3 and z_3 are DV; z_6 is IV in op_7 but it is DV in op_5, thus it is DV.

3.2.2 Refinement by Extending Operation Delay

Our objective of dual-V_{dd} register allocation is to minimize the number of registers assigned to V_{ddh} as well as the total number of registers. We have better chance to achieve this goal (during the next step of register allocation) when we have less number of HVs and less number of DVs (note that allocating V_{ddl} to some DVs may cause other DVs to be allocated to V_{ddh}). This can be accomplished by extending the delay of operations having some HV or DV operands, which do not have any successor operations scheduled directly in the next control step (thus have a room for extending its delay): in Figure 5(d), delays of op_2 and op_6 are extended; as a result, z_2 becomes IV (while it was HV in Figure 5(c)) and z_5 also becomes IV (it was DV).

Note that allocation of functional units remains intact: in Figure 5(a), we need 1 V_{ddh} adder (for op_1, op_2, and op_7), 1 V_{ddh} multiplier (for op_4 and op_6), and 1 V_{ddl} multiplier (for op_3 and op_5); the same is true in Figure 5(d) because we simply delay loading the results of op_2 and op_6 to output registers by one cycle, which can be done by generating `load_enable` signals (during controller synthesis) one cycle later. This can be checked by counting the number of operations of the same type in the control step where operation will be extended and comparing that to resource constraint; in Figure 5(a), there are no additions in the third control step, which allows op_2 to be extended.

Extending operation delay increases the lifetime interval of input operands (and decreases that of output operands), which may increase the total number of registers. This is simply checked by counting the maximum number of lifetimes that are overlapped in any control steps, which serves as a lower bound of total number of registers, before and after extending operation delay; if the number increases, extending operation delay is not performed.

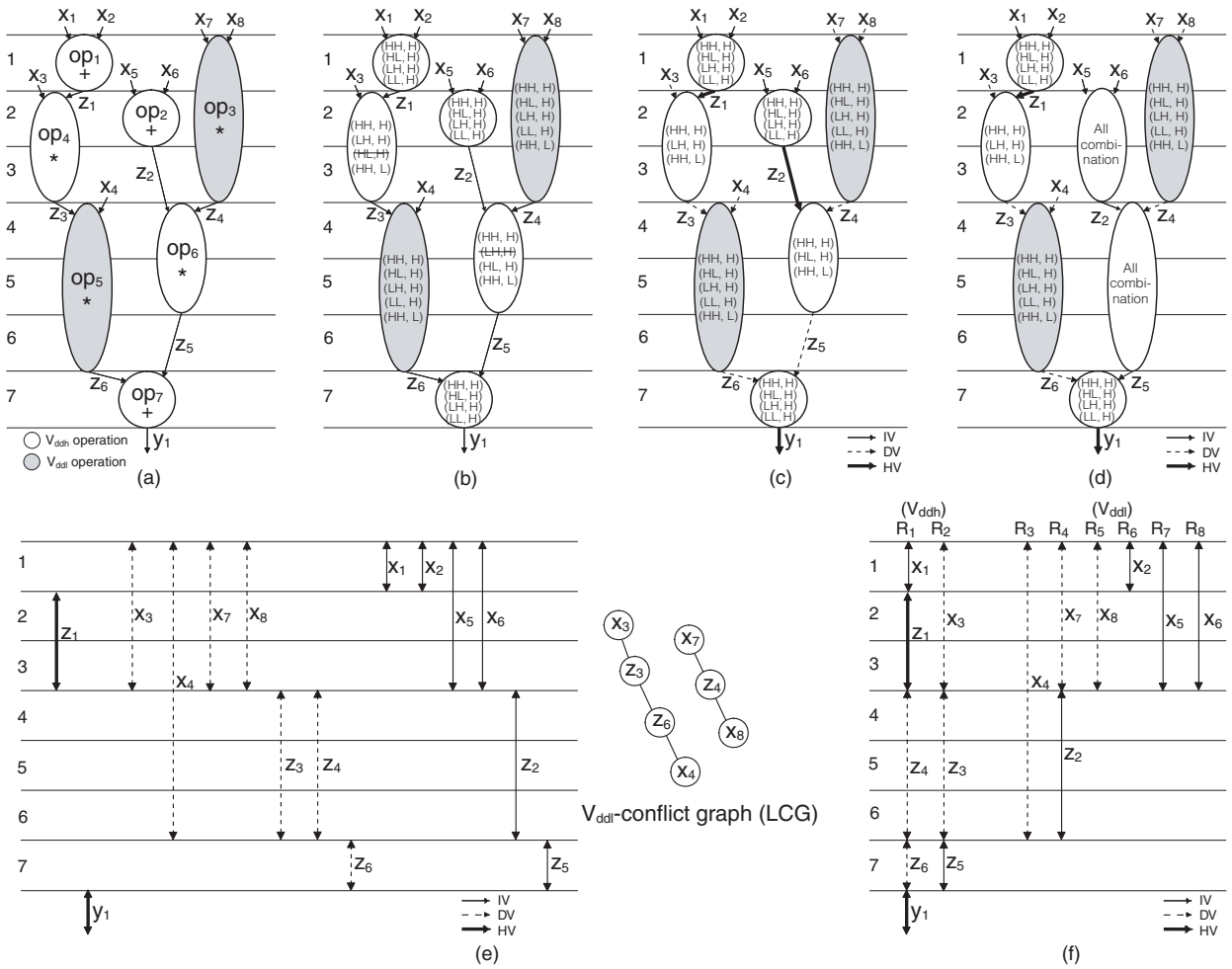

Figure 5: (a) An initial DFG, which is scheduled with dual-V_{dd}, (b) finding feasible V_{dd} combinations of operands, (c) categorizing variables as HV, DV, and IV, (d) re-categorizing variables after extending delay of some operations, (e) variable lifetimes and V_{ddl}-conflict graph of DVs, and (f) result of register allocation.

3.3 Register Allocation

Once we categorize variables as HV, DV, and IV, we derive their lifetimes as shown in Figure 5(e). We also derive a V_{ddl}-conflict graph (LCG), where each vertex corresponds to DV and an edge signifies conflict of allocating V_{ddl}, e.g. if x_3 is allocated to V_{ddl}-register, z_3 has to be allocated to V_{ddh}-register (see Figure 5(e) and (d)).

Since we want to minimize the number of V_{ddh}-registers, we first perform register allocation using HVs alone (by using left edge algorithm [19, 22] if lifetime conflict graph is interval graph [18] or heuristic vertex coloring algorithm [23] otherwise). In Figure 5(f), z_1 and y_1 are assigned to R_1. Since R_1 is already V_{ddh}-register, we want to allocate as many (DV and IV) variables as possible to it (in general, to all V_{ddh}-registers). We first consider DVs and their LCG; in Figure 5(e), z_3, z_4, and z_6 are candidates because their lifetimes do not overlap with those of z_1 and y_1, which are already in R_1. Since only one of z_3 and z_4 can be allocated to R_1, we select the one that removes the most number of edges in LCG when it is taken away, which is why z_4 (together with z_6) is allocated to R_1. Note that once z_4 and z_6, with their edges, are removed from LCG, x_4, x_7, and x_8 become IVs, thus are removed from LCG as well. We continue to pick candidates from IVs that can be allocated to R_1; we select x_1 in Figure 5(f).

If we still have some DVs left in LCG, we need more V_{ddh}-registers. We remove the node having the most number of edges in LCG, and allocate it to a new V_{ddh}-register; we then take DVs and IVs in turn until the register is filled in; we repeat the process until LCG becomes empty; in Figure 5(f), x_3, z_3, and z_5 are allocated to V_{ddh}-register R_2. Remaining variables are all IVs, which are submitted to standard register allocation minimizing the number of V_{ddl}-registers.

4. DUAL-V_{DD} ALLOCATION FOR MULTIPLEXERS

Even after dual-V_{dd} register allocation, there may be some slacks left in the data-path, which can be used to assign V_{ddl} to multiplexers. This is formulated as a problem of finding maximum-weight independent set (MWIS) from a conflict graph $G_{mux} = (V, E)$; each vertex in V correspond to a multiplexer that can be assigned V_{ddl} (after checking timing slack similar to what we did in Section 3.1) and there is an edge in E between two multiplexers if assigning V_{ddl} to one multiplexer has a conflict with assigning V_{ddl} to the other. Each vertex has a weight, which reflects the amount of power saving from using V_{ddl}-multiplexer; weight varies from multiplexer to multiplexer depending on the number of inputs and possibility of

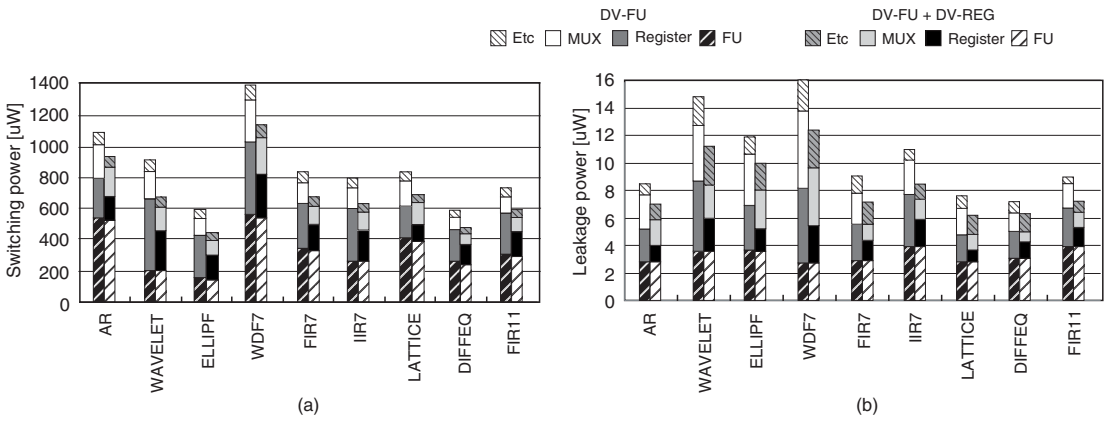

Figure 6: Comparison of (a) switching- and (b) leakage-power between DV-FU (left-hand bars) and DV-FU+DV-REG (right-hand bars).

Table 1: Area and delay of (32-bit) circuit elements used in the experiment. Register delay is denoted by T_{cq}/T_{su}.

Circuits		Area (um^2)	Delay (ps)
Adder/subtractor	V_{ddh}	387	6880
	V_{ddl}	387	27320
Multiplier	V_{ddh}	3593	6950
	V_{ddl}	3593	27500
Register	V_{ddh}	378	230/150
	V_{ddl}	378	700/630
10-to-1 MUX	V_{ddh}	933	500
	V_{ddl}	933	2130
Level converter		82	350

reducing the number of level converters when level converters are moved from inputs to output (details are omitted). MWIS problem is solved by employing a heuristic algorithm [24].

5. EXPERIMENTAL RESULTS

We carried out experiments on a set of behavioral benchmark designs to assess the effectiveness of dual-V_{dd} register allocation (DV-REG) in reducing switching as well as leakage power consumption. For dual-V_{dd} scheduling and allocation of functional units, the algorithm presented in [11] (referred to as DV-FU) was used; it was also used as a reference of comparison, i.e. we compared DV-FU and DV-FU+DV-REG. DV-REG was implemented in C under Centos 5.0 Linux machine. Each design was synthesized in commercial 1.08V, 65-nm bulk CMOS technology; V_{ddl} was set to 0.8V (74% of V_{ddh} of 1.08) based on the observation that V_{ddl} should be set to about 70% of V_{ddh} for maximum saving of power [25]. For physical design presented in Section 5.3, we forced at least 70% of the placement region to be occupied by cells during automatic placement; metal layers up to M5 were allowed for routing.

Table 1 summarizes circuit elements used in the experiment, each one with its area and delay; single-supply level converter [5] was designed and used. For all benchmark designs, we used 10 ns of clock period (see P_c in (2)); this is slightly larger than maximum delay along the data-path (multiplier, register, and two multiplexers; see Figure 2), which is $6950 + 210 + 700 + 2 \cdot 500 = 8860$ ps when all circuit elements are implemented in V_{ddh}.

5.1 Power Consumption

We compared results from DV-FU+DV-REG against those from DV-FU alone (thus, registers and multiplexers remain in V_{ddh}) under the same latency and the same number of functional units used. The comparisons are illustrated in Figure 6; Figure 6(a) for switching power and Figure 6(b) for leakage power. Each bar consists of four components: power consumption from FUs, registers, multiplexers, and the remaining circuit elements such as level converters, and controller. Both switching and leakage power were obtained by simulating each circuit with fast transistor-level circuit simulator [14], which was repeated 100 times using different random input vectors and the average was taken as a result.

To summarize Figure 6, switching power is reduced by 20.4% on average; leakage power is reduced by 19.9%. If we consider registers and multiplexers alone, switching power is reduced by 43.6%, and leakage power is reduced by 37.3%.

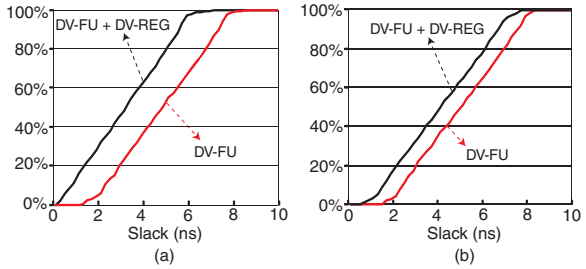

Figure 7: Cumulative slack histogram of (a) LATTICE and (b) IIR7 in DV-FU+DV-REG and DV-FU.

5.2 Slack Histogram

In two implementations we compared in Figure 6, clock period (10 ns), as well as latency, is the same. This implies that DV-REG utilizes timing slacks that are left in the implementation from DV-FU. In Figure 7, we compare DV-FU+DV-REG and DV-FU of the benchmark designs LATTICE and IIR7 in terms of cumulative slack (e.g. about 50% of register-to-register timing paths in the data-path have slacks less than 5 ns in DV-FU implementation of LATTICE). From DV-FU to DV-FU+DV-REG, we see the clear shift of histogram to the left, which shows that we indeed use the slacks left from DV-FU implementation.

Note that even after DV-REG consume some slacks left by DV-FU, there are still plenty of slacks left in the data-path. This can be understood as follows. During scheduling, operation delay (see (2)) is computed from maximum delay through combinational path, D_{dp}, which is based on the maximum delay values in Table 1; maximum delay of adder/subtractor (in V_{ddh}) is 6880 ps while minimum

Table 2: Comparison of area, total wirelength, and average congestion of DV-FU and DV-FU+DV-REG

Circuits	% of V_{ddl}-rows		Area			Wirelength			Average congestion		
	DV-FU	DV-FU+DV-REG	DV-FU (um^2)	DV-FU+DV-REG (um^2)	Inc. (%)	DV-FU (mm)	DV-FU+DV-REG (mm)	Inc. (%)	DV-FU (%)	DV-FU+DV-REG (%)	Inc.
AR	18	57	20834	21325	2.4	224	270	20.6	44	51	7
WAVELET	11	34	33526	34837	3.9	340	454	33.5	42	53	11
ELLIPF	15	51	25191	26338	4.6	284	318	12.2	46	50	4
WDF7	10	51	35400	37448	5.8	477	555	16.4	55	61	6
FIR7	18	59	20588	21244	3.2	235	280	19.2	46	53	7
IIR7	27	68	27560	28543	3.6	246	348	41.1	38	53	15
LATTICE	20	59	17977	18930	5.3	195	213	12.0	44	46	2
DIFFEQ	22	49	16754	17410	3.9	168	223	33.1	41	51	10
Average	18	54			4.1			23.5			8

is 50 ps (adder sum bit of LSB). This also applies to register allocation (see (3)) when we use $D_{dp,l}$. In other words, bit-level of granularity is exposed in the slack histograms shown in Figure 7, while scheduling and register allocation do not perform in that granularity (e.g. scheduling considers a multiplier as a single entity). Therefore, the netlist we obtain after performing DV-FU and DV-REG may be submitted to gate-level dual-V_{dd} allocation for further reducing power, which is left for future work.

5.3 Physical Design

Physical design aspects of DV-FU+DV-REG are compared to DV-FU in Table 2. The area, which is the sum of the areas of all the cells in the design, increases by 4.1% on average; more use of level converters due to DV-REG is a main reason for the increase. The total wirelength, reported after running detailed routing [21], increases by 23.5% on average. This relatively large increase of wires can be understood from the percentage of V_{ddl}-rows; it occupies 18% on average in DV-FU (second column in Table 2) and 54% on average in DV-FU+DV-REG (third column). As V_{ddh}- and V_{ddl}-rows are more evenly used, we can expect more connections between cells placed in each type of rows; placement is more constrained, which makes shorter connections more difficult to be achieved. Due to increased wirelength, average congestion also increases by 8% on average, as reported in the last three columns of Table 2.

6. CONCLUSION

We have presented dual-V_{dd} register allocation, focusing on the problem of minimizing power consumption of storage. Dual-V_{dd} HLS framework includes the complete design flow from dual-V_{dd} scheduling to circuit layout, which was tested using commercial 65-nm technology. Main feature of the framework is a heuristic solution to the problem of finding a register allocation, which minimizes the number of high-V_{dd} registers as well as total number of registers. Dual-V_{dd} allocation for multiplexers was formulated as MWIS problem, which was then solved by using a heuristic algorithm. Experiments on benchmark designs showed that DV-REG can reduce switching- and leakage-power by 20% on average, respectively.

References

[1] R. K. Krishnamurthy et al., "High-performance and low-power challenges for sub-70nm microprocesssor circuits," in *Proc. CICC*, May 2002, pp. 125–128.

[2] C. Yeh et al., "Layout techniques supporting the use of dual supply voltages for cell-based designs," in *Proc. DAC*, June 1999, pp. 62–67.

[3] J. Hu et al., "Architecting voltage islands in core-based system-on-a-chip designs," in *Proc. ILSPED*, Aug. 2004, pp. 180–185.

[4] K. Usami et al., "Automated low-power technique exploiting multiple supply voltages applied to a media processor," *JSSC*, vol. 33, no. 3, pp. 463–472, Mar. 1998.

[5] R. Puri et al., "Pushing ASIC performance in a power envelope," in *Proc. DAC*, June 2003, pp. 788–793.

[6] S. Kulkarni, A. Srivastava, and D. Sylvester, "A new algorithm for improved VDD assignment in low power dual VDD systems," in *Proc. ILSPED*, Aug. 2004, pp. 200–205.

[7] S. Raje and M. Sarrafzadeh, "Variable voltage scheduling," in *Proc. Int. Symp. on Low Power Design*, Apr. 1995, pp. 9–14.

[8] Y. R. Lin et al., "Scheduling techniques for variable voltage low power designs," *TODAES*, vol. 2, no. 2, pp. 81–97, Apr. 1997.

[9] M. C. Johnson and K. Roy, "Datapath scheduling with multiple supply voltages and level converters," *TODAES*, vol. 2, no. 3, pp. 227–248, July 1997.

[10] J. Chang and M. Pedram, "Energy minimization using multiple supply voltages," *TVLSI*, vol. 5, no. 4, pp. 436–443, Dec. 1997.

[11] W. Shiue and C. Chakrabarti, "Low-power scheduling with resources operating at multiple voltages," *TCAS-II*, vol. 47, no. 6, pp. 536–543, June 2000.

[12] A. Manzak and C. Chakrabarti, "A low power scheduling scheme with resources operating at multiple voltages," *TVLSI*, vol. 10, no. 1, pp. 6–14, Feb. 2002.

[13] K. Nose and T. Sakurai, "Optimization of VDD and VTH for low-power and high-speed applications," in *Proc. ASP-DAC*, Jan. 2000, pp. 469–474.

[14] Synopsys, "NanoSim User Guide," Dec. 2007.

[15] G. Gupta et al., "Rapid estimation of control delay from high-level specifications," in *Proc. DAC*, July 2006, pp. 455–458.

[16] T. Ahn et al., "Incremental analysis and elaboration of VHDL description," in *Proc. APCHDL*, Jan. 1996, pp. 128–131.

[17] J. Jeon et al., "High-level synthesis under multi-cycle interconnect delay," in *Proc. ASP-DAC*, Jan. 2001, pp. 662–667.

[18] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, Inc., 1994.

[19] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," in *Proc. Design Automation Workshop*, June 1971, pp. 155–169.

[20] Synopsys, "Design Compiler User Guide," Mar. 2007.

[21] Synopsys, "Astro User Guide," June 2006.

[22] F. J. Kurdahi and A. C. Parker, "REAL: a program for register allocation," in *Proc. DAC*, June 1987, pp. 210–215.

[23] D. Brelaz, "New methods to color the vertices of a graph," *Comm. of the ACM*, vol. 22, no. 4, pp. 251–256, Apr. 1979.

[24] D. Kagaris and S. Tragoudas, "Maximum independent sets on transitive graphs and their applications in testing and CAD," in *Proc. ICCAD*, Nov. 1997, pp. 736–740.

[25] T. Kuroda and M. Hamada, "Low-power CMOS digital design with dual embedded adaptive power supplies," *JSSC*, vol. 35, no. 4, pp. 652–655, Apr. 2000.