Chapter 4

Variable-Precision Arithmetic
Circuit Implementation

4.1 Motivation

As I showed in Chapter 3, the vertex and pixel shaders consume substantial amounts of energy in the graphics pipeline. Since these two shaders actually run on a single pool of general-purpose processors in modern hardware, developing an energy-saving strategy for one stage will likely lead to energy savings in the other, too. I update Hao and Varshney’s variable-precision rendering techniques to today’s GPUs, so a natural step is to find or build hardware capable of trading off precision for energy savings. Since 32-bit floating-point numbers with full precision are not necessary to perform many rendering tasks, the goal will be to reduce the amount of computation that is done in order to save energy.

This variable-precision arithmetic hardware must be able to limit both its dynamic and leakage power (see Chapter 2.1), so clock gating by itself is not sufficient; some form of power gating will be necessary. Furthermore, initial experiments revealed that the precision of the arithmetic must be variable at a very fine level, possibly down to a per-bit granularity, but also must be able to operate at full-precision for some graphical and scientific applications. (Not every program can tolerate having its precision reduced.) Clearly, a high-performance environment like a GPU cannot afford to have its throughput decreased by the inclusion of variable-precision hardware; this hardware must not negatively impact performance. The necessary circuits are the building blocks of a full FPU: integer adders and multipliers. These basic arithmetic circuits constitute more complex units used in graphics hardware and are responsible for the precision of
a given floating-point operation.

To put a fine point on the requirements of the variable-precision arithmetic hardware needed for saving energy in a GPU, let me enumerate them here. The circuits need to be:

1. integer adders and multipliers (that will be used in a full FPU),
2. power gated, so that leakage power will be reduced, as well as dynamic power,
3. variable-precision at a fine granularity,
4. not significantly slower than the original hardware, and
5. dynamically reconfigurable.

I will go over many past techniques and approaches for tackling this problem in the next section and will show that no existing work addresses each of the requirements listed above. So, the rest of this chapter will describe new circuits to enable precision-energy tradeoffs by not computing successive least significant bits (LSBs).

### 4.2 Related Research

Many methods of power gating have been presented, from simple header and footer transistors to more complex techniques. For instance, if there is a need to save the current state and data stored within a circuit while it is power-gated, Liao et al. and Kim et al. have both proposed structures allowing for this capability (Liao et al., 2002; Kim et al., 2004). However, this is far beyond what is needed for my approach to variable-precision arithmetic; there is no need to store intermediate results in the lower, power-gated bits. So, I chose simpler techniques with lesser overheads that can be applied to each bit of an arithmetic circuit, rather than the circuit as a whole.

There has also been research directed towards low power arithmetic circuit design. Sheikh and Manohar thoroughly examined a floating-point adder and designed a new one piece by piece with aggressive optimizations for energy savings (Sheikh and Manohar, 2010). Liu and Furber presented a low power multiplier (Liu and Furber, 2004), while Callaway and Swartzlander detailed the relationship between power and operand size in CMOS multipliers (Callaway and Swartzlander, 1997). Tong et al. suggested a digit-serial multiplier design with three discrete bit-widths, resulting in a linear power savings (Tong et al., 2000). Lee et al. proposed a variable-precision constant
multiplier that uses shifting in the place of multiplication by powers of 2, realizing an energy savings of between 16% and 56% (Lee et al., 2007). Most similar to my work is that of Huang and Ercegovac, who developed two-dimensional signal gating for variable bit-width multipliers, realizing up to 66% power savings over a baseline implementation (Huang and Ercegovac, 2002; Huang and Ercegovac, 2003). However, their work does not address leakage power, which is a large component of nanometer-scale CMOS hardware. They also look at the layout of the parts of an array multiplier from an energy standpoint, but do not perform any power gating (Huang and Ercegovac, 2005).

Phatak et al. presented a low power adder and included a treatment of the adder’s power usage dependent on the number of significant bits (Phatak et al., 1998). Kwong filed a patent for a variable-precision digital differential adder for use in graphics rendering, but has not reported any power results (Kwong, 2005). Park et al. have proposed a scheme in which energy can be traded for quality (similar to this dissertation) in a discrete cosine transform (DCT) algorithm using only three “tradeoff levels” (Park et al., 2010). Other research by Usami et al. and Sjalande et al. has led to variable-precision power-gated multipliers, which will save leakage current in smaller processes (Usami et al., 2009a; Sjalander et al., 2005). However, both of these papers only allow for two different operating precisions, while the ability to operate at a full range of precisions is necessary for rendering. (In experiments for Chapter 5, there were many shader programs that could be reduced to, say, 17 or 18 bits of precision, which would not see any savings with hardware that accommodates only 2 or 3 precision.)

Kulkarni et al. use building blocks that are slightly numerically inaccurate to create a multiplier with bounded error characteristics that saves power over a traditionally precise multiplier (Kulkarni et al., 2011). What’s more, they offer a method for trading off error for power, allowing the designer to choose a point along the error-power curve that their application can tolerate, and they allow for exact computations with the use of a residual adder. This is very promising! However, these design choices must be made as the hardware is being built, which precludes the use of this approach for general-purpose hardware. The precision necessary for a GPU’s applications can vary wildly from frame to frame, even from one stage of the pipeline to the next, and cannot be fixed in the hardware.

None of these approaches have all design characteristics mandated at the beginning of this chapter. My targeted applications need very fine-grained control over the operating precision; thus, coarse-grained designs which allow for, for example, 8, 16, and 24 bits of precision simply do not offer the necessary degree of control. The use
of power-gating will offer significant returns when also considering the savings in decreased leakage current (Kim and Shin, 2006). Finally, the ability to reconfigure the hardware for different precisions at runtime is imperative for use in a GPU.

The VFloat library is meant to address some of these problems - application-specific precisions, reduced leakage current - but has only been implemented for field-programmable gate array (FPGA) devices (Wang and Leeser, 2010). So, these problems are only solved by actually reprogramming the hardware, which is not possible at runtime.

Specialized hardware for other domains has also been developed to reduce leakage current by power gating the arithmetic hardware in certain ways, such as Ngo and Asari’s video processing convolution hardware (Ngo and Asari, 2009). There are key differences between our approaches, though; the convolution of image data lets Ngo and Asari use a priori knowledge, such as the magnitude of common filter coefficients, that I cannot count on in my design. They use this knowledge to optimize circuit paths such as one and zero detection. Also, they can count on the dynamic range of neighboring pixels to be relatively small, leading to optimizations taking advantage of transforming this spatial coherence to temporal coherence from the point of view of the arithmetic logic unit (ALU). However, in a massively parallel GPU, it is not guaranteed that neighboring pixels will be processed on the same ALU, rendering this approach infeasible for my designs.

Other low-power techniques, such as DVFS (Mao et al., 2004) and unit-level power gating (Chowdhury et al., 2008), can be used for energy-efficient graphics hardware. These techniques are orthogonal to this work on fine-grained power gating for variable-precision arithmetic.

4.3 Hardware Implementation

To create new hardware that meets the criteria detailed in Section 4.1, I modified existing arithmetic circuits. I chose three common integer adder designs and looked into different ways of adapting a standard array multiplier. The adders used are a ripple carry, uniform carry-select, and Brent-Kung adders (Brent and Kung, 1982), each with their own strengths and weaknesses. The ripple carry adder is a simple design that uses very little hardware, but has the longest critical path and therefore the longest propagation delay. The carry-select adder is faster but, depending on the implementation, can use nearly twice as much area. The Brent-Kung adder, although
Figure 4.1: A standard full adder modified for use in a power-gated variable-precision arithmetic circuit. Depending on the value supplied on the “Enable” line, the transistors in the gates either receive an actual power source ($V_{dd}$) or just a floating input, which does not provide a path for current to follow. The transistors connected to the outputs only pull the values low if the block is disabled, providing components downstream from the adder with a constant value.
it has the highest area requirements, is the fastest of the three and is easily pipelined,
making it a popular and commonly-used design. It is one of many parallel-prefix adders
(Harris, 2003).

Three key modifications were applied to any single component subject to power
gating. First, the arithmetic logic transistors were supplied with either a virtual power
(header switch) or ground (footer switch) signal controlled by sleep transistors driven
by an enable signal, rather than actual power or ground rails. This modification allows
the power to the element to be cut off, thereby practically eliminating the dynamic
power consumption and potentially reducing leakage power loss through the circuit.
When deciding whether to use either a header or footer switch, I consider the power
and timing implications of each (Shi and Howard, 2006), as well as the desired output
in the disabled state. In the second modification, the outputs were either pulled down
(for a header) or pulled up (for a footer switch), depending on the larger context
of the element, so that any downstream hardware will see a consistent signal. This
both reduces downstream switching and allows for transparent integration with existing
hardware; no special treatment of floating signals needs to be considered because the
outputs of disabled gates are not floating. Since the state of the output does not need
to be preserved when disabled, no extra registers are necessary. Lastly, the logic and
gating transistors in the circuit were manually resized in order to minimize the power
or timing overheads of the modified designs (Mao et al., 2004; Shi and Howard, 2006;
Sathanur et al., 2008). Figure 4.1 shows these changes applied to a standard full adder.

Fine-grained power gating, such as I propose, is subject to problems with ground
bounce if large portions of the circuit are switched at the same time. Rush-current
suppression can be implemented by skewing the switching of portions of the circuit
(Usami et al., 2009b). For my design, I can skew the switching by disallowing very
large changes in precision at one time. A possible approach is to have the software
driver monitor precision changes and sequence overly large ones as a series of smaller
changes.

The operating precision is chosen by setting enable lines to each gated unit. Several
approaches are available for correctly setting these enable signals. The most straight-
forward is to drive each gated element based on a symbolic constant in a register.
Alternatively, any manner of decoding circuitry can be used to translate a shorter en-
able code bundled with operands into individual enable/disable signals. The specific
technique used will depend heavily on the application and the usage patterns of the
unit. It is highly likely, however, that whatever area overheads are incurred by the
control circuitry will be shared over several functional units, over an entire ALU, or even over multiple ALUs.

4.3.1 Modified Adder Designs

Differences in each of the three adders targeted led to distinct approaches to power gating for each. I explore designs of 24-bit integer adders, which are used in single-precision floating-point addition, a common operation in many applications. Past research has shown that, for some target applications, the most readily available savings appear in the first twelve least significant bits of a 24-bit adder, where reduced precision will not have an overly negative impact on application utility (Yoshizawa and Miyanaga, 2006; Chittamuru et al., 2003). I therefore limit the precision control of my proposed designs to the least significant sixteen bits. I note here that though two of the adder designs I explore are rudimentary and not often used in high-performance systems, I show later that they can be more energy-efficient than faster designs. Furthermore, their relatively high latency does not render them useless in a GPU; performance in a GPU is a function of throughput, which can be achieved by many pipelined ALUs with any given latency (within reason).

Ripple Carry Adder

First, let’s examine a ripple carry adder. This is a very basic adder whose functionality is immediately discernible, and it will serve as a baseline implementation. A ripple carry adder simply uses one full adder per bit of precision needed by the operands. I modify each full adder as previously described and shown in Figure 4.1. Disabling
Figure 4.3: A portion of the double full adder chain of a carry-select adder block. Each gated unit is two modified full adders which share the same gating transistor, saving area and timing penalties. The final sum is chosen with a multiplexer driven by the carry-in of the previous block.

Each successive full adder has the effect of reducing the precision by a single bit. The modified design is shown in Figure 4.2.

The interested reader may continue in this section for details of the other adder and multiplier designs; otherwise, results are presented in Section 4.5.

**Carry-Select Adder**

Carry-select adders are slightly more complicated than simple ripple carry adders. They employ several banks of smaller ripple carry adders to make up one full-width adder; each bank computes two carry paths in parallel. When the carry out signal from one block enters the next, multiplexers select the correct carry path to output to the next stage, and so on. The first ripple carry block does not have the second carry path, since its carry-in signal is always ‘0.’ It is treated like the modified ripple carry adder above. The other type of block is made up of two ripple carry chains in parallel. Applying my technique to these blocks involves power gating each parallel pair of full adders as one unit, leading to less power and area overhead than simply using the single full adder approach. Specifically, the tested design was a uniform carry-select adder which uses four blocks of six full adders, with all but the least significant block performing additions in parallel chains. Figure 4.3 shows the details of a carry-select block with
Figure 4.4: Power gating applied to the first stage of a Brent-Kung adder, the carry generation and propagation signal generation stage. Note the use of the NMOS to supply a virtual ground to the logic gates, and the PMOS to tie the output signals to a logical ‘1,’ characteristics of a footer switch. The outputs are sent further down the computation chain of the current bit, as well as to the next stage of the next significant bit, as complementary (inverted) signals.

two layers of full adders gated as a single unit.

Brent-Kung Adder

Last, I modify a 24-bit Brent-Kung adder, one of several parallel adder designs. In contrast to the first two adder designs I explored, which generate a single bit’s sum in one functional unit (a full adder), Brent-Kung adders perform addition on a single bit in several stages (Brent and Kung, 1982). Intermediate stages’ outputs are used as inputs to later stages of the same bit, as well as later stages of more significant bits. So, in order to freeze the switching activity in the computation of a single bit, it is only necessary to gate the power of the first stage of that specific bit. I used a footer switch to gate this computation in order to tie the outputs high, as they are treated as complementary (inverted) signals by other signal paths. So, the eventual sums generated will be ‘0’ in the disabled bits, which results in the same behavior as my other adder designs. While it is possible to explicitly power gate the subsequent
stages along a bit’s computation path, I found that the extra power savings obtained are minimal and do not justify the additional area and speed overheads incurred. The details of these modifications to the first stage can be seen in Figure 4.4 and are the only modifications necessary for applying my technique to this adder.

4.3.2 Modified Multiplier Designs

Integer multipliers are used in many different application domains with similarly varied usage patterns. So, I explored several approaches to modifying a 24x24-bit array multiplier for variable-precision operation. A carry-save array multiplier, abstracted in Figure 4.5, is constructed with a matrix of cells (blue squares) composed of an AND gate, to generate the partial products, and a full adder. The final summation step (dark blue rectangle) of the design is performed with a ripple carry adder for simplicity. This adder is not variable-precision, in order to fully separate the two designs (adder and multiplier), though it would certainly make sense to combine my designs in practice. An \( n \times n \) multiplier produces \( 2^n \) product bits, but, in the larger context of a floating-point multiplier, only the high \( n \) bits (green squares) are used, while the low \( n \) bits (red squares) are ignored.

The full adder of each of these cells is gated in a fashion similar to that shown in Figure 4.1, but I also designed versions that have separate gating controls for the signals that propagate downwards and those that propagate to higher bits. First, I tested simply suppressing the low order bits in the operands. Next, I gated the power to just one operand’s lower bits, then the lower bits of both operands. Finally, I adapted a truncation multiplier with correction constant and extended the column truncation to provide variable-precision operation with power gating. Each of the accompanying illustrations represents the gating applied to an 8x8 adder operating at 5 bits of precision.

Operand Bit Suppression

Suppressing the data entering the arithmetic units can be done in different ways. In my tests, I assumed bit suppression at the source registers or before; I do not include specialized circuitry for this purpose. My results, then, will simply show the dynamic power saved. Since there is no power gating performed, the leakage power will not be reduced.
Figure 4.5: An abstracted representation of an 8x8 carry-select array multiplier, showing partial product generation (blue squares), final adder (dark blue rectangle), used product bits (light green squares), and ignored product bits (dark red squares).

Figure 4.6: When gating only one operand, the multiplicand, diagonal slices of the partial product matrix are disabled. This allows for more precise rounding if required.
Figure 4.7: When gating both operands, entire rows of the multiplier’s partial product matrix are disabled in addition to the diagonal slices of the multiplicand.

Figure 4.8: Column truncation extends the premise of a truncation multiplier by applying power gating to entire columns at a time. In addition, not every column needs to be implemented in hardware, saving significant circuit area, though this will make full-precision operation impossible.
Single Operand Power Gating

Only varying the precision of one operand (the multiplicand) shows that my design allows for handling operands of different precisions. This yields more precise rounding, if necessary, while still achieving significant power savings. For each bit of reduced precision, another diagonal slice of the multiplication matrix can be gated, as shown in Figure 4.6. Each diagonal slice consists of half of a full adder from the lower bit and half a full adder from the higher bit of the slice, so that the signals that would propagate further left are not affected. This mode will also have the lower bound for energy savings in handling operands of different precisions (one operand at full precision).

Double Operand Power Gating

By gating the low-order bits of both operands, even more circuitry is shut down with each bit of reduced precision. As in single operand power gating, a diagonal slice of the partial product matrix is gated for each bit of the multiplicand. Additionally, an entire row is gated for each reduced bit of the multiplier. This gating scheme is shown in Figure 4.7.

Column Truncation

A truncation multiplier saves area and power by simply not implementing low-order columns of the partial product generation stage. A correction constant which reasonably handles the average case is added to the carry-in stage of the existing circuitry to correct for the incurred error, but errors can still be large when the generated partial product in a column would all be ‘0’ or ‘1.’ I extended the idea of a truncation multiplier (Ercegovac et al., 2000; Walters and Schulte, 2005) by applying power gating to entire columns in order to reduce the operating precision (Figure 4.8). As more columns are gated, the correction constant (supplied in a similar manner to the precision selection) is changed by software to minimize the average error. Since this scheme has an immediate loss of precision, it is not likely a reasonable choice for hardware that may need to operate at full-precision, but I have included it as another example of a design to which fine-grained power gating can be applied.
4.4 Simulation Setup

I used LT Spice IV (Linear Technology, 2010), built on the well-known Spice III simulator (The University of California at Berkeley, 2010), to simulate the netlists generated by Electric (used for rapid prototyping of smaller circuits) for power and timing figures for a 0.13µm TSMC library with a $V_{dd}$ of 1.3V, frequency of 100MHz, and load capacitances of 0.01pF. The Spice models were at the TT corner and simulated at a standard 25C. (A higher temperature and voltage would exacerbate leakage effects.) First, I tested a smaller 8-bit version of each adder exhaustively for correctness, and then I compared the results of adding 200 random operands to a baseline 24-bit ripple carry adder and visually compared the results to waveforms produced by the operations in software. I repeated these steps for the multipliers. In this way, I verified the functionality of my designs. The same set of random 24-bit operands was used for the power usage simulations of each modified unit at each operating precision. The current drain through the supply voltage source was tracked to determine the power consumed and energy used over these operations. Next, a set of worst-case operands was used to find the longest propagation delay of each adder, measured from the 50% level of the input’s voltage swing to the 50% level of the slowest output’s voltage swing. Leakage power was found by first performing an operation on random 24-bit operands to approximate the average case current draw. Then, power was measured 500ms after the operation to allow for the dynamic current draw to fade away, leaving only quiescent current. I also devised an experiment to time the worst case delay in enabling/disabling all 16 controllable bits at a time. This will be, in effect, the timing penalty incurred for dynamically changing precisions. It may be necessary to slow this down in order to avoid ground bounce, as described above, but it will serve as a worst-case penalty.

4.5 Results

I now present the power savings and area/timing overheads of my designed circuits from simulation. These results are from simulations of pre-layout circuit designs with realistic load capacitances and transistor sizes. While a more detailed, post-layout simulation would also include the effects of wire capacitances, the results presented are strong indicators of the trends of energy savings realizable as arithmetic precision is reduced. Area and timing overheads are difficult to classify as either acceptable or unacceptable (Sathanur et al., 2008), so I compare my overheads with those in other
techniques. Finally, I compare my power savings with other approaches.

4.5.1 Energy and Power Savings

The overall energy consumption for my adder designs as a function of precision is shown in Figure 4.9(a). To demonstrate that these designs help suppress leakage power, which is likely to become increasingly significant as transistor technologies continue to shrink (Roy et al., 2003), Figure 4.9(b) shows the leakage power for each adder circuit as a function of the operating precision. Similar graphs are shown for the results of the modified multiplier power savings in Figures 4.10(a) and 4.10(b). For reference, single full-precision ripple carry, carry-select, and Brent-Kung additions require 3.5, 6.7, and 8.2 pJ, respectively, and a single full-precision multiplication requires 196.1 pJ.

Adders

The desired linear power savings are very apparent and significant in my proposed adder designs. When using a Brent-Kung adder, for example, reducing the precision by just four bits will cause each operation to use roughly 80% of the energy used by full precision operations. In many applications, the precision can often be reduced by more than just four bits without sacrificing fidelity. I will show in Chapter 5 that up to 12 bits can be lost without causing several graphics applications to become unusable. This would give energy savings of close to 50% for additions. Also, though there were energy overheads caused by the circuits becoming slightly slower (see Section 4.5.3), these were overcome after reducing the precision by just 3 bits in the worst case, and only 1 bit in the case of the Brent-Kung adder.

There are some expected characteristics of the energy per operation versus precision trends worth noting. Firstly, the ripple carry adder has an almost perfectly linear slope. This is exactly what one would expect, since precisely one full adder per bit is gated. Second, the carry-select adder has two different regions of savings, due to the structure of its design. The first is seen in precisions 24 through 18, which corresponds to the single layer of full adders being gated in succession. After bit 18, at a precision of 17 and below, the savings are more aggressive as two full adders per bit are gated and consume minimal power.

Leakage power consumption (Figure 4.9(b)) shows analogous trends. Firstly, all the adders show linear savings, as expected. Also, the carry-select adder displays the same dual-slope that was seen in the total power results. Furthermore, while there are
some overheads, due to the added transistors, they are overcome with a reduction in precision by only 4-6 bits.

**Multipliers**

The power savings for the multiplier designs (Figure 4.10) are even more promising than those of the adders, due to the quadratic complexity of the multiplier’s hardware. Just as the adders displayed interesting behavior, the multipliers show trends that warrant remark. The design with the lowest energy savings is that with only one gated operand (“X Gating”), which naturally results in linear energy savings. Simple operand suppression is more useful, but, as previously noted, does not stop leakage current (see Figure 4.10(b)), which will be more of a problem when using a smaller technology. Gating both operands (“XY Gating”) performs better than suppression with a similar inverse quadratic decay, expected from the gating pattern. Using this approach, one must only reduce the precision by 5 bits in order to see a 50% decrease in power consumption. Column gating exhibited even more dramatic power savings, which is to be expected, as roughly half of the multiplier was disabled (or not implemented) from the start. However, it must be noted that the precision is not guaranteed to be exactly the number specified, since the correction constant does not change with operands, only with precision. Errors of one to a few low-order bits must be acceptable when using this scheme, which limits its utility somewhat but gives it the greatest power savings.

The leakage power versus precision curves, in Figure 4.10(b), resemble those of the full energy per operation versus precision curves. While operand suppression does not reduce leakage power, as was expected, the other designs save significant power and overcome very small power overheads after only one bit of precision reduction. So, the power savings will be immediately realized.

### 4.5.2 Area Overheads

The extra area incurred by the gating and control circuitry must not overshadow the power savings they enable. Table 4.1 shows the overheads, as extra transistor area, for each adder type, and Table 4.2 shows the same figures for the multiplier designs. I have not included the area penalty for precision control circuitry, as it is dependent on the implementation chosen. Also, any overhead of the control hardware would likely be shared among several units; the amortized impact on a single unit, such as an adder, would likely be acceptably small.
Figure 4.9: Energy per operation and leakage power versus precision of the different adder designs. The ripple carry adder uses very little energy per operation, while the carry-select and Brent-Kung adders use nearly double this amount. These two, however, are significantly faster. Like the energy per operation, leakage power declines roughly linearly with precision.
Figure 4.10: Energy per operation and leakage power versus precision of the different multiplier designs. Simply gating one operand ("X Gating") leads to a linear savings, while gating both operands ("XY Gating") and taking advantage of the multiplier’s quadratic complexity yields more aggressive savings with minimally reduced precision. Suppressing operand data does not reduce leakage power at all, but the other curves show trends similar to those seen in the energy per operation savings.
Table 4.1: Extra area needed for modified adders.

<table>
<thead>
<tr>
<th>Adder type</th>
<th>Transistor area ($\mu m^2$)</th>
<th>Unmodified</th>
<th>Modified</th>
<th>Increase (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ripple Carry</td>
<td>4.606</td>
<td>5.383</td>
<td>16.9</td>
<td></td>
</tr>
<tr>
<td>Carry-Select</td>
<td>9.165</td>
<td>10.319</td>
<td>12.6</td>
<td></td>
</tr>
<tr>
<td>Brent-Kung</td>
<td>13.487</td>
<td>14.735</td>
<td>9.3</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: Extra area needed for modified multipliers.

<table>
<thead>
<tr>
<th>Gating type</th>
<th>Transistor area ($\mu m^2$)</th>
<th>Unmodified</th>
<th>Modified</th>
<th>Increase (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>128.65</td>
<td>172.78</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>XY</td>
<td>128.65</td>
<td>172.78</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>Column</td>
<td>69.10</td>
<td>80.10</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

Overheads in the on-chip area are not of a degree to prohibit my designs from being used. To control 16 bits of a 24-bit unit, the areas of ripple carry, carry-select, and Brent-Kung adders increase by 16.9, 12.6, and 9.3%, respectively, and the multiplier’s area increases by 16 or 34%, depending on configuration. 16 bits is likely at the upper threshold of bits of precision that can be safely lost without adversely affecting the function of an application that normally operates at 24 bits of precision. Choosing a design that controls fewer than 16 bits will use even less extra hardware, both by reducing the number of gating network transistors needed and also by simplifying the control logic. For comparison, simpler signal-gating approaches have incurred overheads of 5-16% (Huang and Ercegovac, 2002) (measured by counting the number of inverters with the simple assumption that each sequential unit has five inverters, while offering only a fixed reduced precision). Only the circuitry to gate at a certain bit (22 in the $X$ dimension and 16 in the $Y$ dimension) was included in the cited work. Overheads would be much higher were their circuits to allow a full range of operating precisions, as mine do.

### 4.5.3 Timing Overheads

The proposed variable-precision units incur two types of delay penalties. The first is the extra time needed for the input signals to propagate through the resized gates to the output. The second is the time taken to change operating precisions, or the turn-on
Table 4.3: Time overheads of the modified adders.

<table>
<thead>
<tr>
<th>Adder type</th>
<th>Critical path delay (ns)</th>
<th>Unmodified</th>
<th>Modified</th>
<th>Increase (%)</th>
<th>Turn-on time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ripple Carry</td>
<td>Critical path delay (ns)</td>
<td>5.6</td>
<td>5.9</td>
<td>6.9</td>
<td>2.1</td>
</tr>
<tr>
<td>Carry-Select</td>
<td>Critical path delay (ns)</td>
<td>2.4</td>
<td>2.5</td>
<td>6.9</td>
<td>1.4</td>
</tr>
<tr>
<td>Brent-Kung</td>
<td>Critical path delay (ns)</td>
<td>1.066</td>
<td>1.069</td>
<td>0.4</td>
<td>1.069</td>
</tr>
</tbody>
</table>

Table 4.4: Time overheads of the modified multipliers.

<table>
<thead>
<tr>
<th>Gating type</th>
<th>Critical path delay (ns)</th>
<th>Unmodified</th>
<th>Modified</th>
<th>Increase (%)</th>
<th>Turn-on time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Critical path delay (ns)</td>
<td>6.99</td>
<td>7.26</td>
<td>3.8</td>
<td>7.15</td>
</tr>
<tr>
<td>XY</td>
<td>Critical path delay (ns)</td>
<td>6.99</td>
<td>7.26</td>
<td>3.8</td>
<td>7.15</td>
</tr>
<tr>
<td>Column</td>
<td>Critical path delay (ns)</td>
<td>6.99</td>
<td>7.26</td>
<td>3.8</td>
<td>7.15</td>
</tr>
</tbody>
</table>

time. Table 4.3 lists these figures and compares the propagation delays of the modified and original designs for the new adders, and Table 4.4 reports my findings for the new multiplier designs.

These timing overheads are also acceptable. Firstly, the worst-case turn-on time due to precision changing is a cycle or less for each of the modified designs; allowing that my simulations are pre-layout, this is reasonable. The propagation delay penalty is also quite acceptable, less than 7% at maximum for the adders and less than 4% at maximum for the multipliers. While this overhead is already quite low, in low-power devices, a high clock speed is usually not the primary concern. In fact, the clock may be dynamically slowed to take advantage of lighter workloads. My techniques are orthogonal to DVFS; both can be used on the same circuitry to gain energy savings. As before, my designs are competitive compared with a signal-gated approach that shows delay overheads of 7-11% (Huang and Ercegovac, 2002).

4.5.4 Comparison with Other Techniques

Here, I compare the energy savings of my proposed circuits with the savings of other variable-precision techniques. This is a difficult task, as other reported findings differ in technology sizes and other factors. I offer comparisons of my approach versus both coarse-grain power gating and signal gating.

I first look at one representative coarse-grain power gating technique, a twin-precision
multiplier, which is nearly directly comparable with my results, thanks to the same size process (130nm) and similar driving voltages (my 1.3V versus their 1.2V) (Sjalander et al., 2005). There are several differences between our two approaches: Sjalander et al.’s circuit is based on a tree multiplier, while mine is a simpler array multiplier. Also, their approach allows for only two different precisions to be used, whereas my design offers a continuum of operating precisions. While they do not report all the necessary results, such as power consumption of the multiplier in 16-bit mode, one metric that I can compare is the power consumption of a standard 16-bit multiplier operating on 8 bit operands compared to their twin-precision cutoff multiplier operating on 8-bit operands. The ratio between these two is 3.2, whereas the ratio between my multiplier operating at full and half precisions is 6.8, indicating that I see more savings for the same reduction in precision. However, this comparison is unfair, as I do not implement power gating below 8 bits. So, if I treat 8 bits as ‘0’ and find the ratio between the new full and half precisions (24 and 16, respectively), I arrive at a ratio of 3.4. This is slightly better than the twin-precision multiplier. Lastly, even though my unpipelined multiplier has a delay of 4 to 5 times that of Sjalander et al.’s, depending on configuration and despite my 50% larger bit width, my design is more flexible and has an energy efficiency 1.7 times higher than their design.

I now compare my results against a signal-gated approach by Huang and Ercegovac (Huang and Ercegovac, 2002). In this compared work, a 32-bit multiplier is signal-gated in both the X and Y dimensions, and is the technique on which I have based my “XY” power gating approach. However, they hardwire gating lines at the 22\textsuperscript{nd} bit of one dimension and then 16\textsuperscript{th} bit of the other. I have only reported results for symmetric power gating, though my circuit could be driven with two different precisions. So, to choose a comparison, I first observe that they report results when gating, on average, 40% of each operand. This equates, in my design, to an operating precision of 14.4 bits. So, I will compare their reported results with my results linearly interpolated between 14 and 15 bits. They report energy savings of 67% when using their most low-power design, and I show savings of 76% for my analogous “XY” gating technique. (Column gating would yield better savings, but incurs computational errors not seen in their approach.) As expected, my own “Suppression” technique, which mimics their coarse-grain signal-gating approach, has an energy savings of 69%, which agrees closely with their results.
4.6 Conclusion

I have applied power-gating techniques to several standard integer adders and an array multiplier, converting them to be dynamic, fine-grained variable-precision circuits. My designs show significant savings when reducing the precision of integer adders and multipliers in order to save dynamic and static power consumption. I have shown that the overheads caused by this power gating are modest, and that the precision only needs to be reduced by 2 or 3 bits in order to start seeing energy savings. I will use the energy versus precision characteristics of these circuits in Chapter 5 to build an energy model of the vertex and pixel shader stages of a GPU that can trade precision for energy savings.

There is significant remaining work in the area of variable-precision arithmetic circuits. First, none of my designs are pipelined, which is a common optimization in throughput-oriented devices like GPUs. Second, my designs are only the foundation for an FPU; they will need to be assembled into a variable-precision ALU with floating-point specific hardware to handle exponents, rounding, etc. Lastly, while I have presented several adder designs, I am confident my approach will apply to other adders, as well, including carry-save adders or Kogge-Stone and other parallel adders (Harris, 2003). Likewise, the application of my techniques to different multiplier designs, such as Wallace or Dadda trees, may reveal an even more useful design.