Abstract

Increasing prominence of commercial, financial and internet-based applications, which process decimal data, there is an increasing interest in providing hardware support for such data. In this paper, new architecture for efficient binary and Binary Coded Decimal (BCD) adder/subtractor is presented. This employs a new method of subtraction unlike the existing designs which use 10’s complements, to obtain a much lower latency. Though there is a necessity of correction in some cases, the delay overhead is minimal. A complete discussion about such cases and the required logic to process is presented. The architecture is run-time reconfigurable to facilitate both BCD and binary operations, including signed and unsigned numbers. The proposed circuits are compared (both qualitatively as well as quantitatively) with the existing circuits in literature and are shown to perform better. Simulation results show that the proposed architecture is at least 11% faster than the existing designs.

1. Introduction

Due to growing importance of decimal arithmetic in commercial, financial and internet-based applications, which cannot tolerate errors from converting between binary and decimal formats, hardware support for decimal arithmetic is receiving an increased attention. Recently, specifications for decimal floating point arithmetic have been added to the draft revision of the IEEE-754r Standard for Floating Point Arithmetic [1]. Despite the widespread use of binary arithmetic, decimal computation remains essential for many applications. Not only is it required whenever numbers are presented for human inspection, but it is also often a necessity when fractions are involved. Decimal fractions are pervasive in human endeavors, yet most cannot be represented by binary fractions. The value 0.1, for example, requires an infinitely recurring binary number. If a binary approximation is used instead of an exact decimal fraction, results can be incorrect even if subsequent arithmetic is correct. [2]

It is anticipated that once the IEEE-754r Standard is finally approved, hardware support for decimal floating point arithmetic will be incorporated on processors for various applications. Still, the major consideration while implementing Binary Coded Decimal (BCD) arithmetic will be to enhance its speed as much as possible which is being addressed in this paper.

But to facilitate even binary applications on the same hardware a reconfigurable approach needs to be adopted. This paper deals with the design of an architecture that can perform both binary and BCD addition/subtraction. It also supports both signed and unsigned operations.

All the existing architectures, as per the knowledge of the authors, use 10’s or 9’s complement to implement subtraction in BCD. But this has been found to have a very high latency. Hence a new approach has been proposed to overcome this problem.

The architecture has been designed to have maximum hardware utilization. The rest of the paper is organized as follow: Section 2 provides a brief mathematical background of BCD while section 3 gives a brief explanation about the existing architectures. The proposed algorithm for the unified BCD and binary adder/subtractor is given in section 4. In section 5, the proposed architecture is presented. Simulation results for the proposed and existing circuits are given in section 6 and comparisons are carried out. Finally a conclusion is presented in section 7.

2. Mathematical Background for BCD

BCD is a decimal representation of a number directly coded in binary, digit by digit. For example the number \( (9527)_{10} = (1001\ 0101\ 0010\ 0111)_{\text{BCD}} \). It can be seen that each digit of the decimal number is coded in binary and then concatenated, to form the BCD representation of the decimal number.

To use this representation all the arithmetic and logical operations need to be defined. As the decimal
number system contains 10 digits, at least 4 bits are needed to represent a BCD digit. Consider a BCD digit \(A\). The BCD representation of \(A\) is \(A_4A_3A_2A_1\) where all \(A_k \in \{0,1\} \). The only point of note is that the maximum value that can be represented by a BCD digit is 9. The representation of \((10)_{10}\) in BCD is \((0001\ 0000)\).

Addition in BCD can be explained by considering two decimal digits \(A\) and \(B\) with BCD representations as \(A_4A_3A_2A_1\) and \(B_4B_3B_2B_1\) respectively. In the conventional algorithm, these two numbers are added using a 4-bit binary adder. It is possible that the resultant sum can exceed 9 which results in overflow. If the sum is greater than 9, the binary equivalent of 6 is added to the resultant sum to obtain the exact BCD representation. This can be illustrated with the following example:

\[
\begin{align*}
A &= 0110 \quad (6) \\
B &= 0101 \quad (5) \\
Sum &= 1011 \quad (11) \\
Add &= 0110 \quad (6) \\
BCD &= 10001 \quad (11 \text{ in } BCD) \\
Answer &= (0001 \ 0001)
\end{align*}
\]

3. Related Work

There is a wide range of literature available in field of BCD arithmetic. Some of the first contributions were made by Schmookler et. al.[10] and Adiletta et. al. [14]. An approach towards to architecture dealing with both BCD and binary was shown by Levine et. al. and Anderson, while one of the first BCD sign-magnitude adder/subtractor architecture was presented by Grupe [17]. An area efficient sign-magnitude adder was later developed by Hwang [13]. In his approach two additional conversions were introduced before and after the binary addition. Area occupied by this design was least amongst all the previous designs.

Flora [11] presents an adder similar to the carry-select adder. This employs the use of duplicate hardware to compute the output in the presence of a carry and in its absence. It then selects the appropriate one as the carry is computed. This was improvised by Fischer et. al. where only a single adder was employed to reduce the area over-head. But there was a higher latency due to the additional correction block employed.

The approach to construct BCD architectures in many IBM processors is based on the work presented by Haller et. al. in [12]. This architecture shown in Fig. 3 operated in a single cycle, though requiring corrections in some cases. In the case of subtraction there is a need for the computation of the complement after the subtraction to obtain the correct difference, hence increasing the latency. Another improvement in the same architecture was the optimization of the carry chain resulting in a slight delay improvement with an increased area of the unit.
4. Proposed Algorithm

The proposed algorithm aims at performing both BCD and binary addition/subtraction. The major concern is to avoid 10’s complement to perform subtraction which is the reason for the high latency in the existing architectures. The proposed design can be divided into three major parts, the pre-computation stage, the prefix network [20] and the post-computation stage.

The pre-computation stage generates control signals named propagate (P) and generate (G). These control the operation of the prefix network. These control signals are generated for every significant stage in the N-digit number i.e. there 2*N( P*N + G*N) control signals. These denote whether the kth stage propagates the carry/borrow signal or generates it respectively.

The concept of propagate and generate in both binary and BCD is illustrated below with equations and examples. In the case of addition of binary numbers the following equations denote propagate and generate for A and B (two bits at the kth stage):

\[ P = A \oplus B \]
\[ G = A \cdot B \]

Propagate Case
\[
\begin{array}{ccc}
A+B=9 & \rightarrow & 0 \ 0 \\
A+B>9 & \rightarrow & 1 \ 1
\end{array}
\]

Generate Case
\[
\begin{array}{ccc}
A+B=9 & \rightarrow & 1 \ 0 \\
A+B>9 & \rightarrow & 1 \ 1
\end{array}
\]

In the case of addition of BCD digits A and B (two digits at the kth stage) the following equations denote propagate and generate:

\[ P \Rightarrow A+B=9 \]
\[ G \Rightarrow A+B>9 \]

This operation can be better illustrated from the following example:
In the case of subtraction of BCD digits $A$ and $B$ (two digits at the $k^{th}$ stage), the following equations denote propagate and generate:

- $P \Rightarrow A = B$
- $G \Rightarrow A < B$

This operation can be better illustrated from the following example:

![Fig 7. Examples of BCD subtraction illustrating the concept of propagate and generate](image)

These propagate and generate bits are sent to the prefix network which has a network of blocks which calculates the group propagate and generate bits. The group $P_{k:0}$ and $G_{k:0}$ bits denote whether the first $k$ stages propagate or generate the carry/borrow.

After the calculation of the group propagate and generate, the carry/borrow can be calculated based on the carry/borrow input based on the following equation.

$$ C_{out} = G + P \cdot C_{in} $$

Thus, the carry/borrow at every significant stage is obtained. These bits are sent to the post-computation blocks which compute the final sum/difference based on these bits. In case of binary, after obtaining the carry/borrow the sum/difference is the XOR of the carry/borrow and the propagate bit.

In case of BCD, it is a little complicated. As mentioned in Section 2, for the addition of two BCD digits if there is an overflow then a correction value of $0110(6)$ has to be added. For the subtraction of BCD digits, all the existing architectures are employing 10’s complement subtraction. But computing 10’s complement induces a very high latency in the operation.

Hence the proposed architecture uses propagate and generate bits defined specifically for subtraction to compute the borrow bits for every stage using the prefix network. After the borrow bit is computed, the individual digits at every significant stage are subtracted by using 2’s complement using 4-bit binary adders. But then if a borrow output is generated then the output has to be corrected by adding 1010 (10) to the difference. This is checked by inspecting the borrow input of the subsequent stage which has already been generated by the prefix network.

This operation is illustrated in the following example:

![Fig 8. Examples of Binary and BCD operations illustrating the dataflow of the proposed architecture](image)

5. Architecture of the proposed BCD and binary Adder/Subtractor

The proposed architecture performs both BCD and binary addition/subtraction including signed and unsigned numbers. This architecture can be divided into three major parts, the pre-computation stage, the prefix network and the post-computation stage. This architecture is illustrated by a block diagram in Fig. 9.
For the case of BCD computation it can be observed from the diagram that the pre-computation block for every significant stage consists of logic to generate (P, G) and (P*, G*). Depending on whether addition or subtraction is selected the corresponding propagate and generate bits are sent to the prefix network using an array of multiplexers.

The selection of the prefix network can be made according to the requirements of area, power and delay from the wide range available in literature. For simulation purposes Sklansky network is used in the design. [19] Though Sklansky network is being used, any available prefix network [20] can be chosen depending upon the area, power and delay criterion.

This generates the group propagate and generate which when combined with the carry/borrow input generates carry/borrow for every stage.

These bits are taken by the final post-computation BCD Full Adder/Subtractor blocks shown in Fig 10.

The BCD Full Adder/Subtractor computes the sum if selected by the control signal. The addition operation is performed by adding the two BCD digits using the 4-bit binary carry look-ahead adder and the correction (mentioned in Sec. 2) block. This diagram is shown in Fig. 11.

The subtraction operation is done by 2’s complement addition. But the difference generated needs to be corrected. The borrow input for the subsequent stage that has already been generated by the prefix network is checked and the correction value (1010) is added. The correction block is shown in Fig. 12. The final set of multiplexers select the output based on the operation selected. The only thing that needs to be made sure in this logic is that during subtraction the subtrahend is always the smaller number (in magnitude). This is the case even in all the existing architectures except the design of Humberto et. al. In the proposed architecture this is managed by using the outputs of the (P*, G*) blocks which have comparators in their logic. Thus essentially there is no additional latency overhead due to this comparison.
pre-computation of the binary numbers consists of an array of XOR gates which compute the 2’s complement when subtraction needs to be performed. Then the propagate and generate bits are generated based on the equations mentioned in the previous section. These bits are sent to the prefix network which generates the carry at every stage. These individual carry bits are XORed to the corresponding propagate bits for every stage to compute the sum/difference.

Thus the prefix network that is the major block both in the BCD and binary computation is shared between the two operations to facilitate re-configurability. The signed numbers are taken care by the control logic at the beginning which take the two sign bits and OpSelect(Operation Select) as inputs to compute the control signal that selects the appropriate multiplexers depending on the operation.

6. Simulation and Results

6.1 Simulation Environment

The proposed architectures have been described using Verilog HDL and 8, 16 and 32 digits wide operands. The design was synthesized using Mentor Graphics’ Leonardo spectrum targeting Xilinx Virtex-E v50ecs144 (speed grade-8) FPGA. The approximate values of area have also been mentioned. The inputs are given at a clock frequency of 500 MHz.

6.2 Results and Discussion

The proposed architectures have been simulated in the simulation environment mentioned above and the results for latency and area for 32-bits architectures are given in Table 1.

Table 1. Comparison of the existing and the proposed architectures in terms of latency and area

<table>
<thead>
<tr>
<th>Design</th>
<th>Latency(ns)</th>
<th>Area (LUTs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hwang[13]</td>
<td>10.5</td>
<td>158</td>
</tr>
<tr>
<td>Haller[11]</td>
<td>10</td>
<td>584</td>
</tr>
<tr>
<td>Humberto[17]</td>
<td>12.1</td>
<td>495</td>
</tr>
<tr>
<td>Proposed</td>
<td>8.9</td>
<td>523</td>
</tr>
</tbody>
</table>

In terms of latency Haller turned out to be better of the all the existing designs. Compared to it, this design is 11% faster and uses 10% to 11% less hardware. This is due to the absence of 10’s complement approach. In terms of area Hwang’s proposal is by far the best one presented, as it uses only a single adder in its design.

7. Conclusion

Existing and proposed architectures for the BCD and binary reconfigurable adders are presented, simulated and compared. A novel way of implementing subtraction in Binary Coded Decimal without the use of 10’s complement is explained. Though there is the necessity for the check of magnitude of the two numbers before subtraction it is implemented in a way as not to affect the latency. All the cases where correction is necessary and the logic correction blocks The proposed BCD and binary adder/subtractor is at least 11% faster than the fastest one till now while occupying a considerable amount of area.

8. References


