DESIGN PROJECT 2 –
BCD CALCULATOR –
Part 1: Datapath Design


This part not to be submitted separately.
Overall Project: Due Class Time, Friday, April 30, 1999; 15% of course grade.

This project is to be submitted and will be graded. It is to be performed by teams of two students to permit a more significant project and foster collaboration on difficult parts and in Mentor Graphics usage. The team is not to copy designs or results from others. However, sharing knowledge and tips on how to use Mentor Graphics with other teams is allowed and encouraged.

As a part of the project report, a table is to be included that reports in detail what parts of the project were performed by each team member; the table to be completed and submitted is provided at the end of this write-up. Ordinarily the same grade will be given to each team member, but if there is a significant imbalance in overall contribution, individual grades will be given.

This project write-up is organized as follows. Initially, we will give specifications for the BCD Calculator. The specifications will be followed by a number of design exercises that will lead you through the design of the datapath part of the calculator. Part 2, to be given out later this week, will guide you through the design of the control of the calculator and the integration and testing of the datapath and the control. You are strongly encourage to study this entire write-up before beginning.

EXTERNAL SPECIFICATIONS

The BCD Calculator user interface is shown in Figure 1; it is modeled after a Texas Instruments™ TI™-1768II calculator. Differences are that it has only four digits, handles only integers, and lacks multiplication, division, memory, and an error indicator (Well, it was to have multiplication, but it was just too much work!) In addition to the Power/Reset button, the calculator has 10 digit buttons, 0 through 9, for entering decimal digits, and four operation buttons, Clear, +, -, and =. There is a 4 and 1/2 digit LCD
The calculator has the following properties:

1. It uses “infix” notation, e.g., entry sequence 5, +, 6, ×, 7 gives \((5 + 6) \times 7 = 77\).
2. When the first digit of an operand is entered, the most significant three digits go to 0.
3. If more than four digits are entered, the last four digits are used as the operand.
4. If a succession of operators is entered without entering any digits, the last of the operator entered is the one executed.
5. The CLEAR operator clears the result register and the entry register and attaches the entry register to the display.
6. The +, − operators perform their operations just once. If they are repeated before entering new digits, no operation is executed.
7. In contrast, when the = is pressed multiple times, the last computation is repeated, e.g., \(5, +, 6, =, =, =\) gives \((5 + 6) + 6 = 23\).
8. There is no error or overflow indicator.

**INTERNAL SPECIFICATIONS**

The top level of the design of the BCD Calculator appears in Figure 2.
You are to design only the CORE LOGIC! The I/O Logic and Circuitry is specified functionally as follows and is not to be designed! This description of the interface to the outside world will permit you to understand the environment for the CORE LOGIC.

The external pushbuttons and the display connect to the I/O Logic and Circuitry which does the following:

1. provides a clock signal, CK.
2. provides a master reset signal, Reset, that is activated when the power is turned on.
3. debounces and otherwise conditions signals from the pushbuttons.
4. for each decimal digit pushbutton provides, when the button is pushed:
   a. \( \text{en}_\text{dig} = 1 \); otherwise, \( \text{en}_\text{dig} = 0 \).
   b. \( \text{in}_\text{dig}(3:0) \), the 4-bit BCD representation for the decimal digit, both for one clock cycle,
5. for each operation pushbutton provides, when the button is pushed:
   a. \( \text{en}_\text{op} = 1 \); otherwise, \( \text{en}_\text{op} = 0 \).
   b. \( \text{in}_\text{op}(2:0) \), both for one clock cycle (The 3-bit representation for the operation defined as:

   Clear: 100, =: 010, -: 001, and +: 000.), and
6. takes the 4-digit output \texttt{dis3(3:0), dis2(3:0), dis1(3:0), and dis0(3:0)} and \texttt{sign} from the \texttt{CORE LOGIC} and provides the display drivers for driving the 4-1/2 digit display.

The \texttt{CORE LOGIC} is divided into two pieces, the \texttt{datapath} and the \texttt{control}. You will design the \texttt{datapath} first, followed by the \texttt{control}.

**DATAPATH SPECIFICATIONS**

A schematic for the \texttt{datapath} appears in Figure 3. The \texttt{datapath} performs serial operations as illustrated in Figure 5-5 of Mano and Kime except that it processes BCD digits instead of bits. Thus, each connection in Figure 5-5 which carries a bit will carry a 4-bit BCD digit in the datapath. So many of the connections in Figure 3 are 4-bit buses which transfer one BCD digit at a time.

Note that the \texttt{datapath} contains a number of components:

1. \texttt{bcd_digit}, which is the component you designed in Project 1.
2. two \texttt{sh_reg}, which are four BCD digit bidirectional shift registers with hold and clear.
3. \texttt{carry}, which initializes and stores the carry for the serial processing, and
4. four \texttt{q_mux_2}, which are quad 2-to-1 multiplexers for selecting the contents of one or the other of the \texttt{sh_reg}'s to appear on the calculator display.

All flip-flops within these components have clock \texttt{CK} and power-up reset \texttt{RESET_B}.

The component, \texttt{bcd_digit}, has a role corresponding to the full adder in Figure 5-5, except that it 1) processes BCD digits, and 2) handles subtraction and correction by complementation as well as addition. It has control signals \texttt{sub} and \texttt{comp_a} that permit subtraction and complementation of operand \texttt{A} as well as addition.

The top \texttt{sh_reg} component which we will call \texttt{ENT_R} for \texttt{Entry Register}:

1. captures entered digits from input \texttt{in_dig(3:0)} by using right digit shift,
2. stores entered operands before and during processing, the latter by “recirculating” digits back into \texttt{sh_reg} during left digit shifts,
3. delivers these operands from its \texttt{D0(3:0)} output to \texttt{bcd_digit} and \texttt{carry} for processing serially, least significant digit first, by using four left digit shifts and
4. provides the partially entered and entered operands \texttt{D3(3:0), D2(3:0), D1(3:0)} and \texttt{D0(3:0)} in parallel to the display outputs to the multiplexers for display output.

Carefully note the connections to and from \texttt{ENT_R} that are used to perform 1 through 4 above. Note that \texttt{ent_f(1:0)} and \texttt{ent_c(1:0)} are control signals used to control the operation of \texttt{ENT_R}.

The bottom \texttt{sh_reg} component which we will call \texttt{ACC_R} for \texttt{Accumulator Register} (An accumulator was the central processing register in early computers.):

1. stores entered operands, transferred from \texttt{ENT_R} by adding them to 0, by using four left shifts,
2. stores intermediate operands and results of processing, by using hold,
3. delivers entered, intermediate, and result operands from its \texttt{D0(3:0)} output to \texttt{bcd_digit} and \texttt{carry} for processing serially, least significant digit first, by using four left shifts, and
Figure 3: datapath
4. provides results to D3(3:0), D2(3:0), D1(3:0) and D0(3:0) in parallel to the display outputs via the multiplexers.

Carefully note the connections to and from ACC_R that are used to perform 1 through 4 above.

The component carry which we have labeled A_Carry:

1. stores carry values between processing of digit positions and operations, and
2. at output CQ, provides a) fixed values 0 and 1 for addition and subtraction of the least significant digits, and b) values appearing on a prior cycle on input CD for other digit positions.

A_Carry had as_cc(1:0) as its control inputs.

**DATAPATH COMPONENT SPECIFICATIONS**

The following specifications for the datapath components will be used in designing them in the next section. The specifications for bcd_digit are given for reference only.

bcd_digit. Component bcd_digit performs BCD digit addition, subtraction, and complementation. Its operation is defined in the following table:

<table>
<thead>
<tr>
<th>COMP_A</th>
<th>SUB</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>BCD Addition: (COUT, S) = A + B + CIN</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>BCD Subtraction: (COUT, S) = A + 9’s comp(B) + CIN</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>BCD Complementation (COUT, S) = 9’s comp(A) + CIN</td>
</tr>
</tbody>
</table>

sh_reg. Component sh_reg consists of 16 flip-flops and associated logic. It performs hold, right digit shift, left digit shift, and clear operations. The input to the rightmost bit for left digit shift is L_IN(3:0) and the input for the right digit shift is R_IN(3:0). All four digits stored D3(3:0), D2(3:0), D1(3:0), D0(3:0) are available as outputs. For right digit shift, D0(3:0) is considered the output. For left digit shift, D3(3:0) is the output, but is never used in the datapath for this purpose. The operation of the sh_reg is described by the following table:

<table>
<thead>
<tr>
<th>Name for Operation</th>
<th>F(1:0) or S(1:0)</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>hold</td>
<td>0 0</td>
<td>Hold the contents unchanged</td>
</tr>
<tr>
<td>sr</td>
<td>0 1</td>
<td>Shift all digits one position to the right</td>
</tr>
<tr>
<td>shl</td>
<td>1 0</td>
<td>Shift all digits one position to the left</td>
</tr>
<tr>
<td>clr</td>
<td>1 1</td>
<td>Clear the contents to all 0 digits</td>
</tr>
</tbody>
</table>

Note that there are two control fields F(1:0) and S(1:0). F(1:0) controls the least significant digit position D0(3:0) and S(1:0) controls the other three digit positions. This is because of one combined operation used when the first digit is entered in ENT_R. For this operation, F(1:0) = 10 and S(1:0) = 11, loading the value on L_IN into D0(3:0) and clearing to 0 the rest of the digit positions. Otherwise, F(1:0) and S(1:0) will be equal.
**carry.** Component carry has a single flip-flop for storing the carry bit for BCD addition. This flip-flop also provides the 0 or 1 needed as input for the least signification bit addition or subtraction, respectively. The operation of carry is described by the following table:

<table>
<thead>
<tr>
<th>Name for Operation</th>
<th>CC(1:0)</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>hold</td>
<td>0 0</td>
<td>Hold the contents of the flip-flop with output COUT unchanged</td>
</tr>
<tr>
<td>reset</td>
<td>0 1</td>
<td>Reset the flip-flop with output COUT to 0</td>
</tr>
<tr>
<td>set</td>
<td>1 0</td>
<td>Set the flip-flop with output COUT to 1</td>
</tr>
<tr>
<td>load</td>
<td>1 1</td>
<td>Load value on CIN into the flip-flop with output COUT</td>
</tr>
</tbody>
</table>

Note that all of these operations are synchronous with the clock CK.

**q_mux_2.** This is a straightforward quad 2-to-1 multiplexer. Its operation is described by the following table:

<table>
<thead>
<tr>
<th>S</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Select input A(3:0)</td>
</tr>
<tr>
<td>1</td>
<td>Select input B(3:0)</td>
</tr>
</tbody>
</table>

**DATAPATH DESIGN**

Use the specifications above to obtain the data path components and the datapath. If a simulation of any component designed indicates an error, debug the design until it functions correctly.

---

**Figure 4: sh_cell**

---

**DESIGN PROJECT 2 – BCD CALCULATOR – Part 1: Datapath Design**
This circuit has been predesigned for you. It is a hierarchical design using a component `sh_cell` and a component `sh_cell_4`. Enter and validate the design by performing the following steps:

1. Open a new sheet in `da` called `sh_cell`. Enter the design from Figure 4 into the sheet and check and save the design.

   Validate `sh_cell` by applying combinations to perform the following sequence of operations on successive clock cycles: Asynchronous reset using `RESET_B`, shift right with `R_IN = 1`, hold, clear, shift left with `L_IN = 1`, shift right with `R_IN = 0`, hold, shift right with `R_IN = 1`, shift left with `L_IN = 0`.

2. Open a new sheet in `da` called `sh_cell_4`. Enter the design from Figure 5 into the sheet and check and save the design.

   Validate `sh_cell_4` by applying combinations to perform the following sequence of operations on successive clock cycles: Asynchronous reset using `RESET_B`, shift right with `R_IN = 7`, hold, clear, shift left with `L_IN = 7`, shift right with `R_IN = 8`, hold, shift right with `R_IN = 7`, shift left with `L_IN = 8`, shift left with `L_IN = 7`, Asynchronous reset using `RESET_B`.

3. Open a new sheet in `da` called `sh_reg`. Enter the design from Figure 6 into the sheet and check and save the design.
Validate sh_reg by applying combinations to perform the following sequence of operations on successive clock cycles: Asynchronous reset using RESET_B, shift right four times with R_IN = 7, hold, clear, shift_left four times with L_IN = 7, shift right four times with R_IN = 8, hold, shift right four times with R_IN = 7, shift_left four times with L_IN = 8, shift left four times with L_IN = 7, Asynchronous reset using RESET_B.

Submit: Logic schematics and Quicksim trace file for sh_cell, sh_cell_4 and sh_reg. Make sure that each sheet is clearly labeled with the circuit name.

Figure 6: sh_reg

Perform, enter and validate the design by performing the following steps:
1. Design the circuit using the mux41 and the dff from gen_lib as used in sh_cell.
2. Open a new sheet in da called carry. Enter the design into the sheet and check and save the design.

Validate carry by applying combinations to perform the following sequence of operations on successive clock cycles: Asynchronous reset using RESET_B, hold, set,
DATAPATH DESIGN

hold, reset, hold, load \( CD = 1 \), hold, load \( CD = 0 \), set, asynchronous reset using \( \text{RESET}_B \).

Submit: Logic schematic and Quicksim trace file outputs for carry, annotated to demonstrate correct operation.

q_mux_2

Perform, enter and validate the design by performing the following steps:

1. Design the circuit by using mux21 from gen_lib and making the quad (4-bit) inputs and outputs 4-bit buses using the names \( A(3:0) \), \( B(3:0) \), \( Y(3:0) \). Use \( S \) as the select input such that it selects \( B \) for \( S = 1 \).

2. Open a new sheet in da called q_mux_2. Enter your design into the sheet and check and save the design.

Validate q_mux_2 by applying combinations to perform the following sequence of operations on successive clock cycles: With \( A = 7 \) and \( B = 8 \), select \( A \). With \( A = 7 \) and \( B = 8 \), select \( B \). With \( A = 8 \) and \( B = 7 \), select \( A \). With \( A = 8 \) and \( B = 7 \), select \( B \).

Submit: Logic schematic and Quicksim trace file outputs for q_mux_2, annotated to demonstrate correct operation.

Data Path Integration

Open a new sheet in da called datapath and interconnect the components developed thus far as in Figure 3 to implement datapath. Generate a symbol for datapath.

Validate datapath by applying force file datapath.frc from the Project 2 section of the course Web page using Quicksim and checking the Quicksim output to make sure the outputs are correct.

Submit: Logic schematic and Quicksim list file outputs for datapath, annotated to demonstrate correct operation.
Submit: This page as the **next to last** page of your project report; the **last** page is the Team Effort Report for Part 2. Each **row** of the table containing the task contributions must sum to 100%.

<table>
<thead>
<tr>
<th>Team Member Names:</th>
<th>%</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>sh_cell: Entry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sh_cell: Validation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sh_cell: Debug</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sh_cell_4: Entry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sh_cell_4: Validation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sh_cell_4: Debug</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sh_reg: Entry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sh_reg: Validation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sh_reg: Debug</td>
<td></td>
<td></td>
</tr>
<tr>
<td>carry: Design/Entry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>carry: Validation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>carry: Debug</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_mux_2: Design/Entry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_mux_2: Validation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q_mux_2: Debug</td>
<td></td>
<td></td>
</tr>
<tr>
<td>datapath Integration: Design/Entry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>datapath Integration: Validation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>datapath Integration: Debug</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Comments: