Abstract

Data Processors are arithmetic circuits supplemented by local memory registers that hold intermediate calculations results. In CE1911, data processors are implemented as Moore finite state machines that calculate simple algebraic functions on multiple variables. The design of the controller consists of identifying register-transfer-level (RTL) instructions that move data, complete calculation using an ALU, and store results. Each instruction becomes the output voltages of one state in the controller finite state machine. The process of determined the output voltages is called machine coding and thus the voltage transitions through time represent the machine code of the system.

Introduction

A simple special purpose data processor can be constructed using an ALU, two storage registers, and a data bus multiplexer used to move numbers from outside the processor into the local memory registers. Figure 1 shows a simple organization of these components into a useful processor.

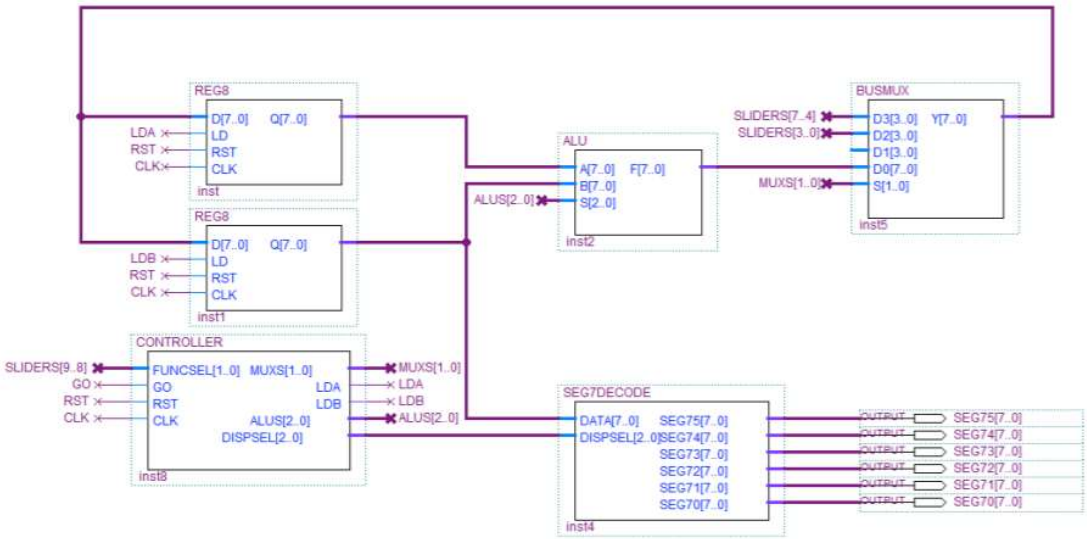

Figure 1: The CE1911 Data Processor

Basic arithmetic uses two data values. For example, A+B requires number A and number B. Similarly, A-B also requires numbers A and B. Thus, the ALU in Figure 1 is supplemented by two local memory registers called REGA and REGB. The data processor controller uses these registers to store intermediate numbers created during the algebraic calculation selected by the function select input bus. The intermediate numbers can originate outside the processor as input data or can be a number calculated by the ALU. A multiplexer is used to select which origination point provides numeric data during each state of the algebraic calculation. In Figure 1, the multiplexer is attached to slider switches that provide two numeric inputs and also to the ALU so that calculation results can be stored again for reuse. Figure 1 also shows a seven-segment decoder used to display the number stored in REGB on a display panel.

Design Example

Consider a modified version of the Figure 1 data processor where the slider switches are known as input numbers T and J, and where the controller has only a single function select bit called FS. The desired algebraic functions are summarized in Table 1.

Table 1: Algebraic Functions

FS	Algebraic Function Selected
0	F(T) = 4T-9
1	G(J,T) = 2J+T+2

Microsoft Excel or a similar spreadsheet program is a great way of organizing the output truth table for the controller finite state machine. Table 2 shows one implementation using the given ALU and MUX selection behavior. A VHDL implementation using enumerated state names immediately follows.

Table 2: Output Logic Truth Table

CURRENT	RTL	REGA	REGB		LDA	LDB	MUXS	ALUS		ALUS	ALUF
STATE	BEHAVIOR									0	A-1
RESET		0	0		0	0	0	0		1	B+1
F0	A <-- 0	0	0		1	0	3	3		2	A+B
F1	A <-- A-1	-1	0		1	0	3	0		3	constant 0
F2	A <-- A-1	-2	0		1	0	3	0			
F3	A <-- A-1	-3	0		1	0	3	0			
F4	A <-- A-1	-4	0		1	0	3	0		MUXS	MUXY
F5	A <-- A-1	-5	0		1	0	3	0		0	
F6	A <-- A-1	-6	0		1	0	3	0		1	J
F7	A <-- A-1	-7	0		1	0	3	0		2	T
F8	A <-- A-1	-8	0		1	0	3	0		3	ALU
F9	A <-- A-1	-9	0		1	0	3	0			
F10	B <-- T	-9	T		0	1	2	0			
F11	A <--A+B	T-9	T		1	0	3	2			
F12	A <--A+B	2T-9	T		1	0	3	2			
F13	A <--A+B	3T-9	T		1	0	3	2			
F14	A <--A+B	4T-9	T		1	0	3	2			
F15	HOLD	4T-9	T		0	0	0	0			
G0	A <-- J, B <-- J	J	J		1	1	1	0			
G1	A <-- A + B	2J	J		1	0	3	2			
G2	B <-- T	2J	T		0	1	2	0			
G3	B <-- A+B	2J	2J+T		0	1	3	2			
G4	B<--B+1	2J	2J+T+1		0	1	3	1			
G5	B<--B+1	2J	2J+T+2		0	1	3	1			
G6	HOLD	2J	2J+T+2		0	0	0	0			

```
--*************************************************************
--* CE1911                                                    *
--* In-Class Example - Special Purpose Computer Controller    *
--* Name: Dr. Meier                                           *
--* Description:                                              *
--*                                                           *
--*   This controller implements two functions using an       *
--*   input called function select (FS).                      *
--*                                                           *
--*   FS = 0: F(T) = 4T-9                                     *
--*   Fs = 1: G(J,T) = 2J+T+2                                 *
--*                                                           *
--*   This controller has states prefixed F0, F1, for F(T).   *
--*   This controller has states prefixed G0, G1, for G(T).   *
--*************************************************************
--*   The data path controlled by this controller uses a      *
--*   restricted ALU with only four functions.                *
--*                                                           *
--*   ALUS = 0: A-1                                           *
--*   ALUS = 1: B+1                                           *
--*   ALUS = 2: A+B                                           *
--*   ALUS = 3: CONSTANT 0                                    *
--*************************************************************
--*   The data path provides numeric values to the register   *
--*   file using a four item multiplexer.                     *
--*                                                           *
--*   MUXS = 0: no connection                                 *
--*   MUXS = 1: input number J                                *
--*   MUXS = 2: input number T                                *
--*   MUXS = 3: ALUF                                          *
--*************************************************************
--*   This controller uses VHDL 2008 standard for process(all) *
--*************************************************************

-- load IEEE libraries
library ieee;
use ieee.std_logic_1164.all;

-- entity declaration
entity control is
port(rst,clk,go,fs: in std_logic;
     lda,ldb: out std_logic;
     muxs: out integer range 0 to 3;
     alus: out integer range 0 to 3);
end entity control;

architecture behavioral of control is
  type states is (reset,f0,f1,f2,f3,f4,f5,f6,f7,f8,
                  f9,f10,f11,f12,f13,f14,f15,
                  g0,g1,g2,g3,g4,g5,g6);
  signal d, q: states;
begin

  transitions: process(all)
  begin
    if rst = '0' then q <= reset;
     elsif rising_edge(clk) then
       case q is
          -- handle getting out of reset: jump to correct function when go=0.
          when reset =>
             if go = '0' and fs = '0' then q <= f0;
             elsif go = '0' and fs = '1' then q <= g0;
             else q <= reset;
```

```
            end if;

           -- states for fs = 0: f(t) = 4t-9
                                  -- RTL Version 1: shorter exist
            when f0 =>  q <= f1;  -- A <- 0
            when f1 =>  q <= f2;  -- A <- A-1
            when f2 =>  q <= f3;  -- A <- A-1
            when f3 =>  q <= f4;  -- A <- A-1
            when f4 =>  q <= f5;  -- A <- A-1
            when f5 =>  q <= f6;  -- A <- A-1
            when f6 =>  q <= f7;  -- A <- A-1
            when f7 =>  q <= f8;  -- A <- A-1
            when f8 =>  q <= f9;  -- A <- A-1
            when f9 =>  q <= f10; -- A <- A-1
            when f10 => q <= f11; -- B <- T
            when f11 => q <= f12; -- A <- A+B
            when f12 => q <= f13; -- A <- A+B
            when f13 => q <= f14; -- A <- A+B
            when f14 => q <= f15; -- A <- A+B
            when f15 => q <= f15; -- hold lock

            -- states for fs = 1: g(j,t) = 2j+t+2
                                  -- RTL Version 1
            when g0 => q <= g1;   -- A <- J, B <-J
            when g1 => q <= g2;   -- A <- A+B
            when g2 => q <= g3;   -- B <- T
            when g3 => q <= g4;   -- B <- A+B
            when g4 => q <= g5;   -- B <- B+1
            when g5 => q <= g6;   -- B <- B+1
            when g6 => q <= g6;   -- hold lock

            -- correct for any error states
            when others => q <= reset;

         end case;
      end if;
    end process;

    -- active-high lda output logic
    with q select
    lda <= '0' when reset|f10|f15|g2|g3|g4|g5|g6,
           '1' when others;

    -- active-high ldb output logic
    with q select
    ldb <= '1' when f10|g0|g2|g3|g4|g5,
           '0' when others;

    -- multiplexer selector for data bus value
    with q select
    muxs <= 2 when f10|g2, -- declared as integer
            0 when reset|f15|g6,
              1 when g0,
              3 when others;

    -- alu selector for arithmetic or logic function
    with q select
    alus <= 3 when f0,
            2 when f11|f12|f13|f14|g1|g3,
              1 when g4|g5,
              0 when others;
end architecture behavioral;
```

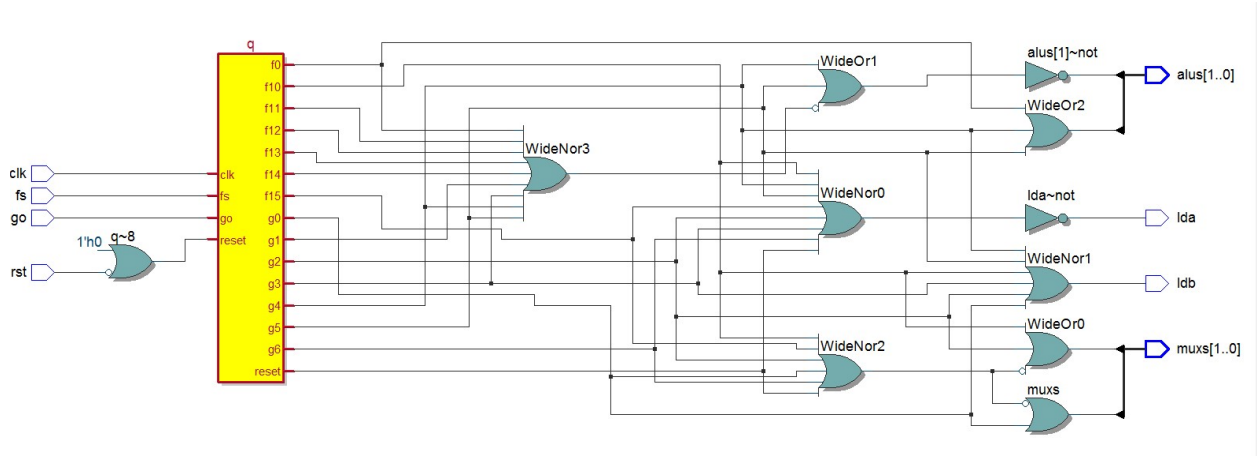

Figure 2: Altera Quartus Synthesizer RTL View of the Data Path Controller

Figure 2 shows the Moore finite state machine created by the Quartus synthesizer. No compiler options were used to set the encoding so the synthesizer has used its default behavior and implemented a one-hot machine. The synthesizer has added an OR gate to implement the active-low reset behavior and created output logic gates for each output signal set that depend only on the current state of the machine.

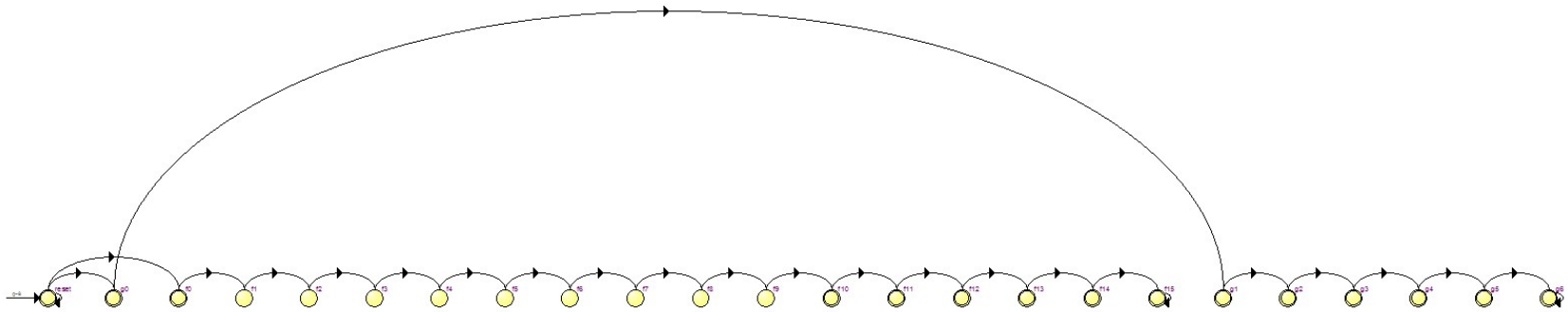

Figure 3: Altera Quartus Synthesizer State Machine View of the Data Processor Controller

Figure 3 shows the Quartus generated state machine diagram. Zoom in to more clearly see the state names and transition arcs. The two function calculation paths are easy to identify as the machine leaves reset for either the function F or the function G when GO is asserted. Two saturated hold states are visible terminating each of the functions. These two hold states, F15 and G6, produce identical outputs according to Table 2 and represent the exact same calculation behavior – "do not make any more changes." Thus, they can be combined to reduce the size of the machine. Figure 4 shows an alternate state diagram when this modification is made to the VHDL source code and Quartus re-synthesizes the circuit. Both calculation paths, F(T) and G(J,T) are still clearly visible but this time they both end in the common hold state.

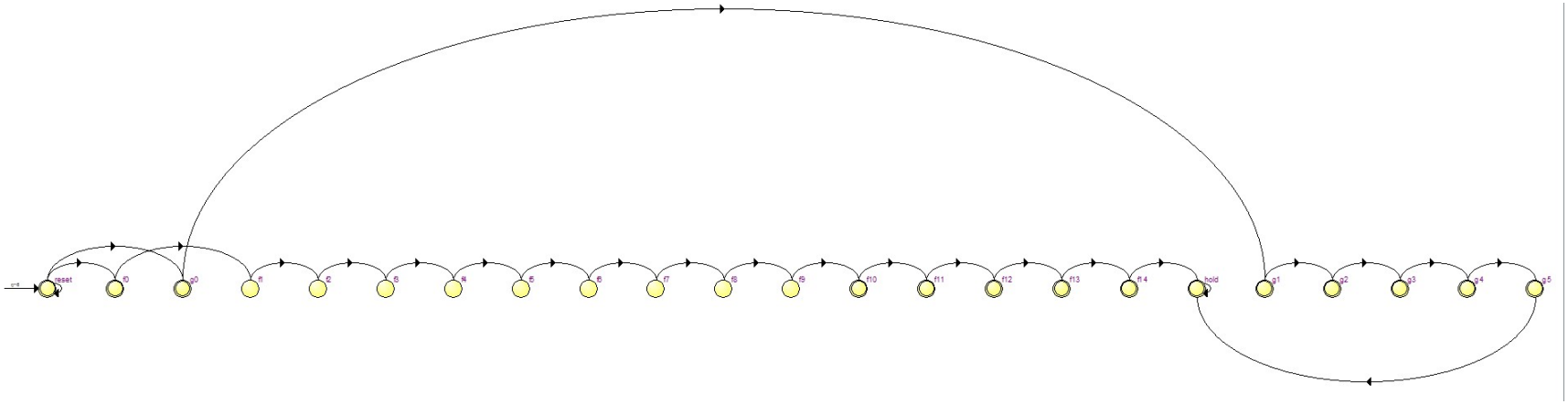

Figure 4: Altera Quartus Synthesizer State Machine View of the Data Processor Controller with Common Hold State

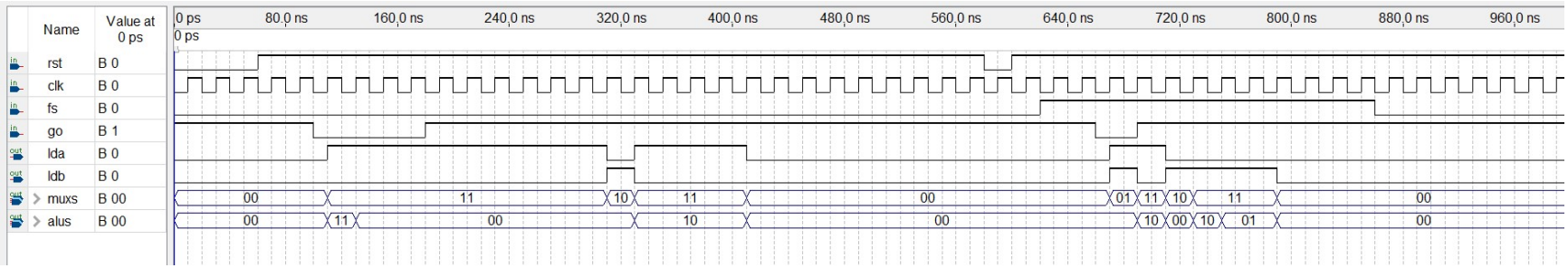

Figure 5: Altera Model-Sim Simulation of Data Processor Controller – Entity Inputs and Outputs Only

Figure 5 shows a basic simulation of the data processor controller. The reset signal is asserted to force the machine into a known starting state. The machine waits for GO to be asserted before transitioning to the function selected by the FS input. In the first half of the simulation, FS = 0 selected F(T) = 4T-9 and the machine creates outputs through time before arriving at the hold condition – identifiable as the long period of time when both LDA and LDB are not asserted from around 400 nanoseconds to 640 ns. The second half of the simulation shows the machine behavior when FS = 1 selects G(J,T) = 2J+T+2.

This simulation does show useful information because it demonstrates output behavior as a response to input behavior. But, the simulation does not show the state transitions to help verify that each state occurs in the proper order and – for most of the states – without repetition during a calculation sequence.

A better simulation would insert the Q outputs of the one-hot state machine flip flops. This can be done in Quartus by using node-finder with the node-finder filter set to “Design Entity – All Names.” Figure 6 documents the results after inserting all of the Q outputs and organizing them appropriately in the waveform editor before running the simulation. This advanced simulation technique now allows the design engineer to position a vertical marker by dragging it from its starting location directly below 0 picoseconds to any state within the machine. In Figure 6, this marker has been advanced to state F10. Comparing the output voltages in state F10 shows matching behavior to Table 2. This technique can be used to verify each state of the machine.

It should be noted that the simulations of Figures 5 and 6 are for the original VHDL design that includes repeated hold states. This is clearly seen in Figure 6 as both F15 and G6 Q-outputs are active for long periods of hold time.

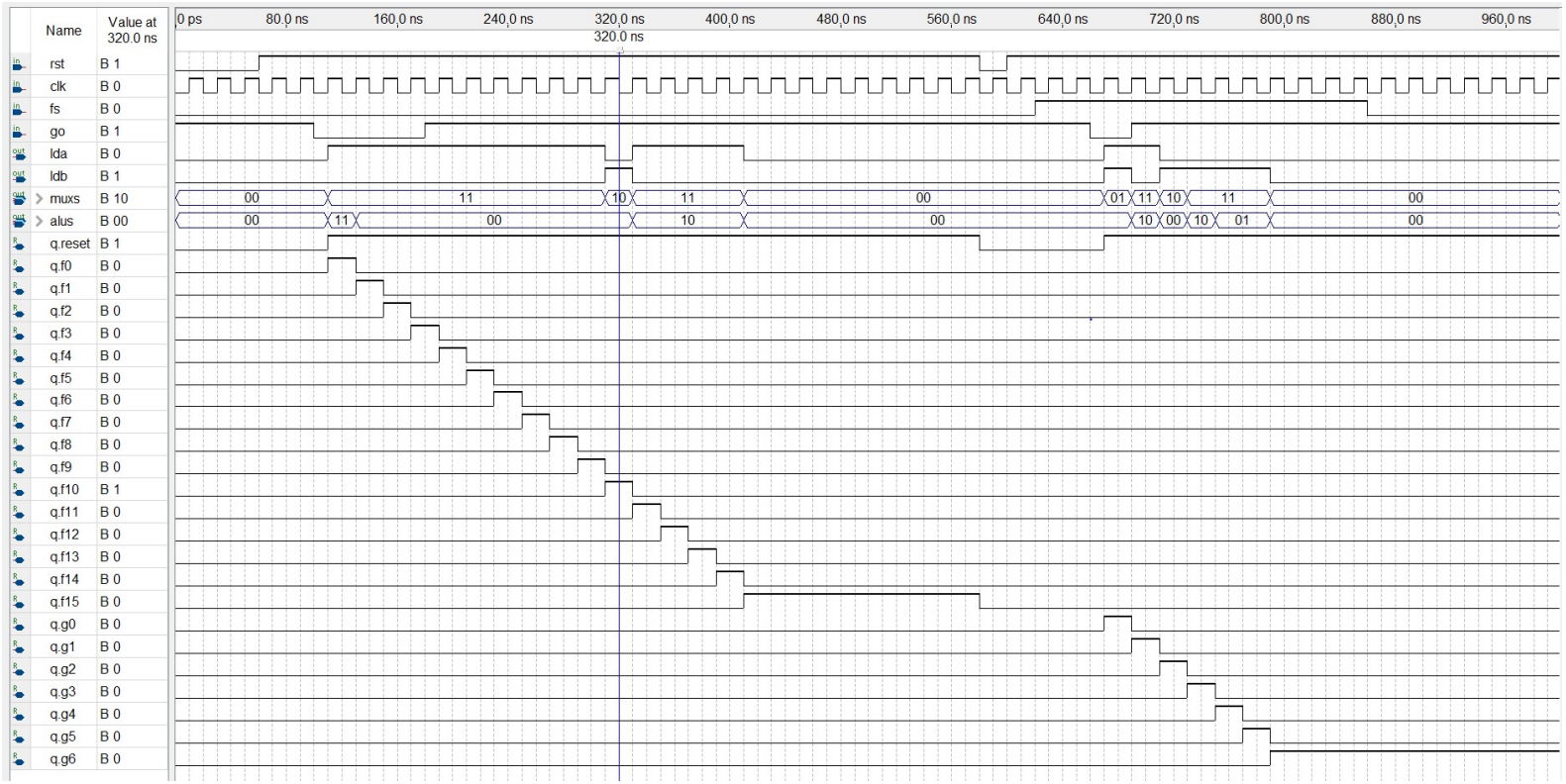

Figure 6: Altera Model-Sim Simulation of Data Path Controller – Entity Inputs, Outputs, and Internal One-Hot State Machine