HW3 Solutions (CS552 Spring 2013)

```
Grade distribution: (Total: 100)
- Verilog submission of problems 1), 2) and 3) carry 15 points each(Total: 45
points)
- Written part of problem 1), 2) and 3) carry 5 points each (Total: 15 points)
- Written part of problems 4) through 11) ( only four are graded) carry 10 points
each (Total: 40 points)

Grading of verilog submission:
Total points for each problem :15
- Points for compiling design : 3
- Points for functional tests: 12 (.25 points each for 48 tests)
- Penalty for incorrect directory structure: -5
- Penalty for missing files: (50% of points scored for the problem)
- Penalty for vcheck failures: (50% of points scored for the problem)

Grading of written part:
Problems 4), 5), 9) and 11) are graded and each carries 10 points.
```

Problems 1, 2 and 3

Solutions not to be provided

Problem 4

The order, from easiest to hardest, is Bit Equal, Split Register, Replace Under Mask.

Bit Equal is equivalent to an XNOR operation. Assuming that the internal ALU has an XOR function and can also invert an input, not changes would be required to the datapath. The control would need a slight modification to perform and XOR with one operand inverted.

Split Register requires changes to the datapath and control path. The biggest change would be a register file with two write ports (this instruction writes to both $rd and $rt). Additionally, there would have to be a change in the datapath to accommodate writing the bottom half of a register.

Replace Under Mask would require a complete redesign of the ISA because it uses three register specifiers and a 16-bit literal, which, when added to the opcode, cannot be expressed in 32-bits. Thus, the control and datapath of every instuction would change!

Problem 5

The key to solving this problem was recognizing that the critical path through the 4-bit adder was NOT equal to the sum of the critical paths of individual 1-bit full adders.

The initial XOR and NAND gates in the 1-bit full addder switches independently of the incoming cin value. Therefore, only the first 1-bit full adder in the ripple chain contributes the entire critical path delay. The remaining adders have a shortened critical path that goes through the last two NAND gates.

The critical path for a 4-bit ripple carry adder: total ripple delay:
(XOR + 2 * NAND) + (3 * (2 * NAND)) = (12 + 2^2) + 8*(8 + 2^2) = 112

The delay through the carry-select is equal to the delay through a 4-bit ripple carry plus the delay through a 2-1 mux. The critical path through a mux is 1 NOT and 2 NANDs (8+1) + 2*(8+2^2) = 33
total carry-select delay: 4-bitrippledelay+2-1muxdelay = 112+33 = 145

Problem 6
<See end of document>

Problem 7 (4.1.1 – 4.1.3)

4.1.1 a) RegWrite (1)
MemRead(0)
ALUMux(0)
MemWrite(0)
ALUop(AND)
RegMux(0)
Branch(0)

b) RegWrite (0)
MemRead(1)
ALUMux(1)
MemWrite(1)
ALUop(ADD)
RegMux(X)
Branch(0)

4.1.2 a) All except data memory and branch add unit
b) All except branch add unit and write port of the register file

4.1.3 a) Outputs not used (branch add), No outputs (data memory)
b) Outputs not used (branch add,reg write port), No outputs (none)

Problem 8 (4.1.4 – 4.1.6)

4.1.4 Out of the many paths in the circuit, the following two are identified as the longer ones. Hence the calculations are shown only for these paths. You can do similar calculations on other paths and verify that they are not critical

Path X: Imem + RegRead + Mux + ALU + Mux
Path Y: Imem + Control + Mux + ALU + Mux

a) Path X: 200ps + 90ps + 20ps + 90ps + 20ps = **420ps (Critical)**
Path Y: 200ps + 40ps + 20ps + 90ps + 20ps = 370ps

b) Path X: 750ps + 300ps + 50ps + 250ps + 50ps = **1400ps (Critical)**
Path Y: 750ps + 300ps + 50ps + 250ps + 50ps = **1400ps (Critical)**

4.1.5 Path X: Imem + RegRead + Mux + ALU + DmemRead + Mux
Path Y: Imem + Control + Mux + ALU + DmemRead + Mux

a) Path X: 200ps + 90ps + 20ps + 90ps + 250ps + 20ps = **670ps (Critical)**
Path Y: 200ps + 40ps + 20ps + 90ps + 250ps + 20ps = 620ps

b) Path X: 750ps + 300ps + 50ps + 250ps + 500ps + 50ps = **1900ps (Critical)**
Path Y: 750ps + 300ps + 50ps + 250ps + 500ps + 50ps = **1900ps (Critical)**

4.1.6 Path X: Imem + RegRead + Mux + ALU + AND(ignored) + MUX
Path Y: Imem + Control + Mux + ALU + AND(ignored) + MUX
Path I: Add + Add + Mux
Path J: Imem + Add + Mux

a) Path X: 200ps + 90ps + 20ps + 90ps + 0ps + 20ps = **420ps (Critical)**
Path Y: 200ps + 40ps + 20ps + 90ps + 0ps + 20ps = 370ps
Path I: 70ps + 70ps + 20ps = 160ps
Path J: 200ps + 70ps + 20ps = 290ps

b) Path X: 750ps + 300ps + 50ps + 250ps + 0ps + 50ps = **1400ps (Critical)**
Path Y: 750ps + 300ps + 50ps + 250ps + 0ps + 50ps = **1400ps (Critical)**

Path I: 200ps + 200ps + 50ps = 450ps
Path J: 750ps + 70ps + 20ps = 840ps

Problem 9 (4.9.1 – 4.9.3)

4.9.1 a) 0xAE04FF9C
b) 0x0043082A

4.9.2 a) Read register 1 (10000), Actually read.
Read register 2 (00100), Actually read.
b) Read register 1 (00010), Actually read.
Read register 2 (00011), Actually read.

4.9.3 a) Can't be sure about the value of the control signal RegDst. Depending on the implementation of the controller, this signal can be 1 or 0. Hence either instruction[20:16] or instruction[15:11] can reach the 'write register' input of the register field.
So, one should expect either 00100 or 11111.
And, no, the register is not actually written
b) 00001, yes this register is written

Problem 10 (4.10.1 – 4.10.3) (Only part A)

To solve the problems in this exercise, it helps to first determine the latencies of different paths inside the processor. Assuming zero latency for the Control unit, the critical path is the path to get the data for a load instruction, so we have:

Original critical path = Imem + Mux + RegRead + Mux + ALU + D-Mem + Mux
= 200ps + 20ps + 90ps + 20ps + 90ps + 250ps + 20ps
= 690ps

4.10.1 The Control unit can begin generating MemWrite only after I-Mem is read. It must finish generating this signal before the end of the clock cycle. Note that MemWrite is actually a write-enable signal for D-Mem flip-flops, and the actual write is triggered by the edge of the clock signal, so MemWrite need not arrive before that time. So the Control unit must generate the MemWrite in one clock cycle, minus the I-Mem access time: 690ps – 200ps = 490ps

4.10.2 All control signals start to be generated after I-Mem read is complete. The most slack a signal can have is until the end of the cycle, and MemWrite and RegWrite are both needed only at the end of the cycle, so they have the most slack. Othe control signals are needed before the end of the cycle and hence have lower amounts of slack.

Hence the time to generate both signals without increasing the critical path is the one computed is exactly **490ps** (as computed in 4.10.1)

4.10.3
- MemWrite and RegWrite are only needed by the end of the cycle.
- RegDst, Jump, and MemtoReg are needed one Mux latency before the end of the cycle, so they are more critical than MemWrite and RegWrite.
- Branch is needed two Mux latencies before the end of the cycle, so it is more critical than these.
- MemRead is needed one D-Mem plus one Mux latency before the end of the cycle, and D-Mem has more latency than a Mux, so MemRead is more critical than Branch.
- ALUOp must get to ALU control in time to allow one ALU Ctrl, one ALU, one D-Mem, and one Mux latency before the end of the cycle. This is clearly more critical than MemRead.

- Finally, ALUSrc must get to the pre-ALU Mux in time, one Mux, one ALU, one D-Mem, and one Mux latency before the end of the cycle. Again, this is more critical than MemRead.
- Between ALUOp and ALUSrc, ALUOp is more criti-cal than ALUSrc if ALU control has more latency than a Mux.

a)ALUCtrl (30ps) > Mux (20ps). **Hence ALUOp is the most critical control signal.**

Current critical path = Imem + Mux + RegRead + Mux + ALU + D-Mem + Mux
The longest path with ALUOp = Imem + Controller(ALUOp) + ALUCtrl + ALU + D-Mem + Mux

ie, Controller(ALUOp) = Mux + RegRead + Mux -ALUCtrl
= 20ps + 90ps + 20ps – 30ps
= **100ps**

Problem 11 (4.11.1 to 4.11.3)

4.11.1	a)	Output of 'Sign-extend':	(32 bits) 0000 0000 0000 0000 0000 0000 0001 0100
		Output of 'Shift left 2':	(28bits) 0001 1000 1000 0000 0000 0101 0000
	b)	Output of Sign-extend:	(32 bits) 0000 0000 0000 0000 0000 1000 0010 1010
		Output of 'Shift left 2':	(28bits) 0010 0000 1000 0010 0000 1010 1000
4.11.2	a)	Instruction[5:0] = 010100	
		ALUOp[1:0] = 00 [Opcode is 0x2c which is sw and refer to Figure 4.22]	
	b)	Instruction[5:0] = 101010	
		ALUOp[1:0] = 10 [Opcode is 0x00 which is R-format and refer to Figure 4.22]	
4.11.3	a)	PC+4 [PC->ADD->Mux->Mux->PC]	
	b)	PC+4 [PC->ADD->Mux->Mux->PC]	

Problem 6

Refer to the following:

1) the schematic
2) The controller state machine
3) The truth table for the state machine

States:
Idle: The idle state when the divider just waits for new data to operate on.
Error: Encountered a divide by zero error. Go to idle in the next cycle.
Done: The divide operation has been completed successfully. Go to Idle in next cycle.
S-eq: the equilibrium state where to which the divider returns after each iteration. The subtraction is performed in this stage. After 33 iterations proceed to Done state.
S0: When result of subtraction is positive, write 1 to lsb of quotient (and shift left). Divisor is right shifted in this stage. Next state is S-eq.
S1: When result of subtraction is negative, correct the remainder and write 0 to lsb of quotient (and shift left). Divisor is right shifted in this stage. Next state is S-eq.

Problem 6 Divider Schematic

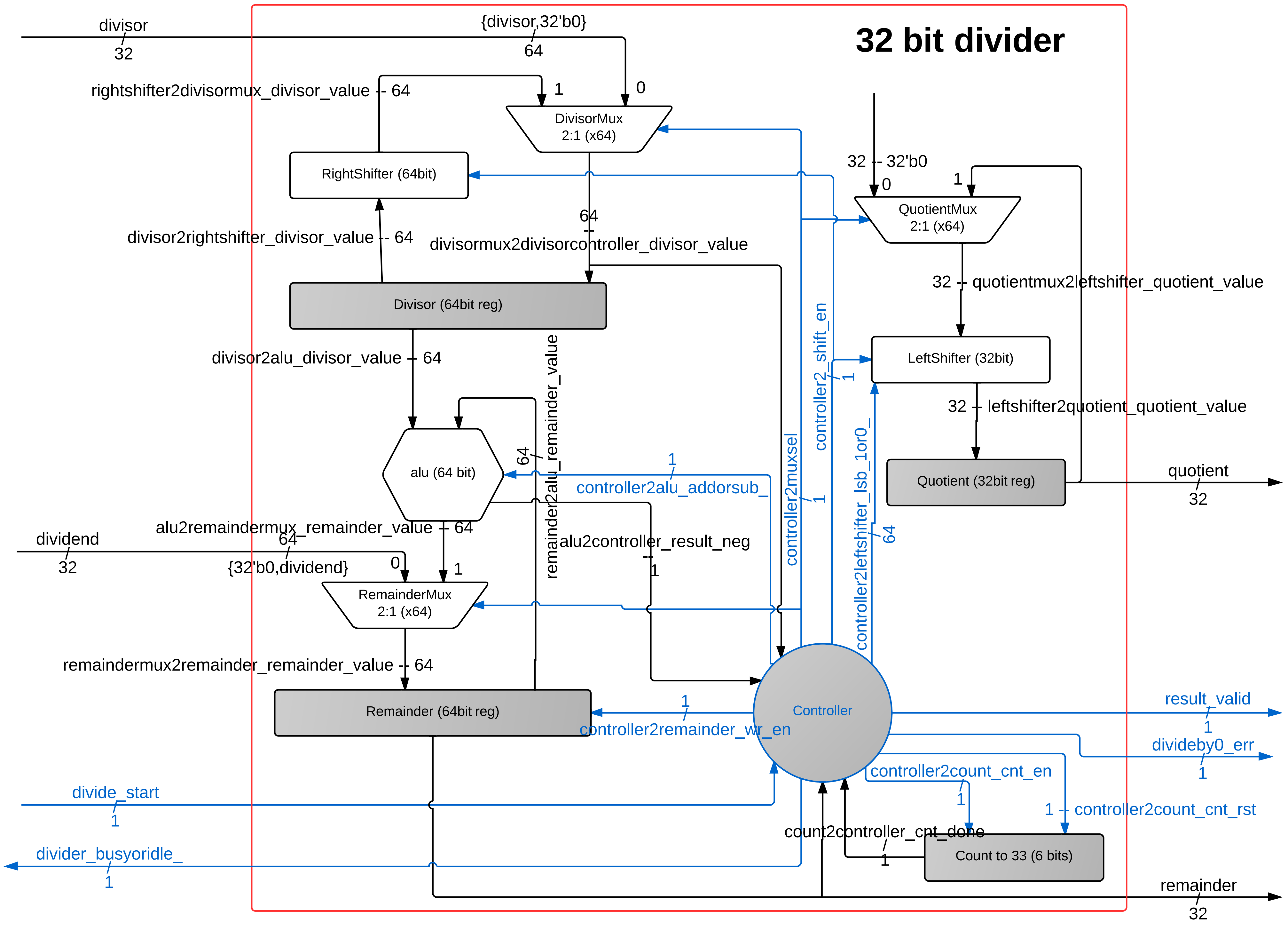
Legends

1) Boxes with white shading indicates combinational logic

2) Boxes with gray shading indicates clocked elements (All clocked elements are fed by clk and rst which is not indicated in the schematic)

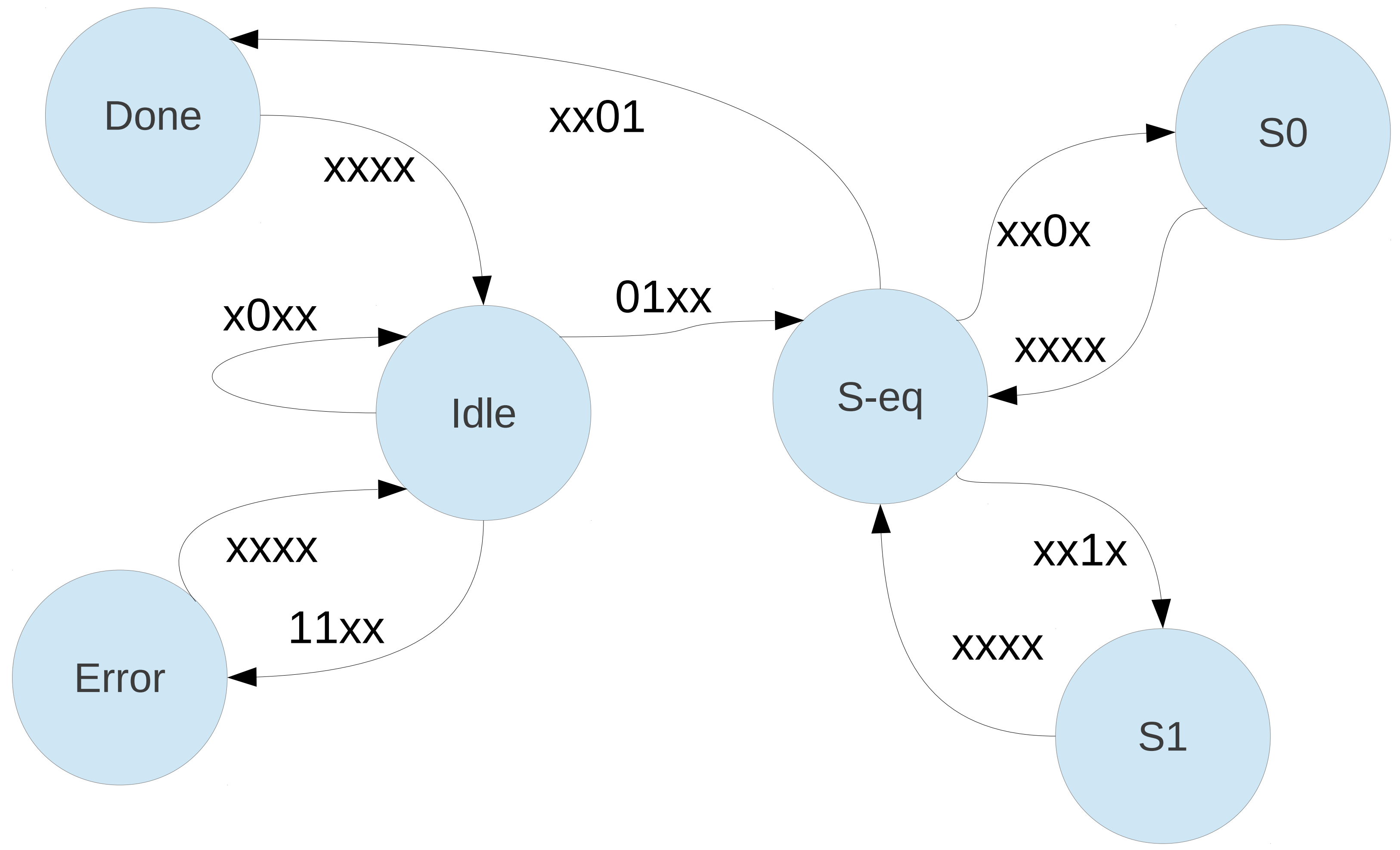
3) Black lines indicate signals in the data path

4) Blue lines indicate signals in the control path



32 bit divider
divisor
32
{divisor,32'b0}
64
rightshifter2divisormux_divisor_value -- 64
1
0
DivisorMux 2:1 (x64)
RightShifter (64bit)
64
divisor2rightshifter_divisor_value -- 64
divisormux2divisorcontroller_divisor_value
Divisor (64bit reg)
divisor2alu_divisor_value -- 64
alu (64 bit)
64 -- remainder2alu_remainder_value
1
controller2alu_addorsub_
alu2remaindermux_remainder_value -- 64
dividend
32
64
{32'b0,dividend}
0
1
alu2controller_result_neg
1
RemainderMux 2:1 (x64)
remaindermux2remainder_remainder_value -- 64
Remainder (64bit reg)
1
controller2remainder_wr_en
controller2muxsel
1
controller2_shift_en
1
controller2leftshifter_lsb_1or0_
64
32 -- 32'b0
0
1
QuotientMux 2:1 (x64)
32 -- quotientmux2leftshifter_quotient_value
LeftShifter (32bit)
32 -- leftshifter2quotient_quotient_value
Quotient (32bit reg)
quotient
32
Controller
result_valid
1
divideby0_err
1
controller2count_cnt_en
1
1 -- controller2count_cnt_rst
count2controller_cnt_done
1
Count to 33 (6 bits)
divide_start
1
divider_busyoridle_
1
remainder
32

Problem 6 Controller state machine



Inputs: (divisor==0, divide_start, remainder<0, count2controller_cnt_done)

Problem 6 Controller Truth Table

Moore state machine's current state	State machine outputs									
	divider_busyoridle_	divideby0_err	result_valid	controller2count_cnt_en	controller2count_cnt_rst	controller2leftshifter_lsb_1or0_	controller2alu_addorsub_	controller2remainder_wr_en	controller2muxsel	controller2_shift_en
Idle	0	0	0	0	1	0	x	1	0	0
Error	1	1	1	x	x	x	x	x	x	x
Done	1	0	1	x	x	x	x	x	x	x
S-eq	1	0	0	0	0	x	0	1	1	0
S0	1	0	0	1	0	1	x	0	1	1
S1	1	0	0	1	0	0	1	1	1	1