

13.1 Synchronous Reset with Enable

Friday, March 9, 2001

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all; -- needed to add an integer to std_logic_vector
entity sctr is
  port (reset : in std_logic;
        enb   : in std_logic;
        clk   : in std_logic;
        count : out std_logic_vector(3 downto 0));
end sctr;
architecture behavioral of sctr is
  signal count_int : std_logic_vector(3 downto 0);
begin -- behavioral
  count <= count_int;
-- rest of architecture in the next slide
```

13.2 Simulation of Synchronous Reset

```
process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' then
      count_int <= x"0";
    elsif enb = '1' then
      count_int <= count_int + 1;
    else count_int <= x"2";
    end if;
  end if;
end process;
end behavioral;
```



13.3 Asynchronous Reset with Enable

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
-- needed to add an integer to std_logic_vector
entity actr is
  port (reset : in std_logic;
        enb   : in std_logic;
        clk   : in std_logic;
        count : out std_logic_vector(3 downto 0));
end actr;
architecture behavioral of actr is
  signal count_int : std_logic_vector(3 downto 0);
begin -- behavioral
  count <= count_int;
-- rest of architecture in the next slide
```

13.4 Simulation of Asynchronous Reset

```
process (clk, reset)
begin
  if reset = '1' then
    count_int <= x"0";
  elsif rising_edge(clk) then
    if enb = '1' then
      count_int <= count_int + 1;
    else count_int <= x"2";
    end if;
  end if;
end process;
end behavioral;
```



13.5 NG Asynchronous Reset with Enable

```
-- same library, entity as before
architecture behavioral of actr is
    signal count_int : std_logic_vector(3 downto 0);
begin -- behavioral
    count <= count_int;
    process (clk, reset)
    begin
        if reset = '1' then
            count_int <= x"0";
        elsif rising_edge(clk) then
            if enb = '1' then
                count_int <= count_int + 1;
            end if;
        else count_int <= x"2";
        end if;
    end process;
end behavioral;
```

13.6 NG Asynchronous Reset Report

□ Compiling: ngaetr.vhd

- ngaetr.vhd (line 20, col 13): Warning: (W479) 'enb' should be referenced in the sensitivity list.
- ngaetr.vhd (line 21, col 36): Warning: (W479) 'count_int' should be referenced in the sensitivity list.

□ /u07/warp4.3/lib/ieee/stdlogic.vhd (line 923, col 20): Abort: (E512) Can't handle expression 's'event' in final equations.

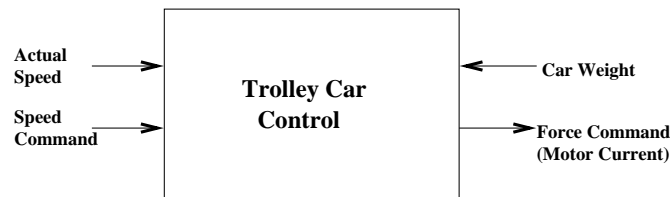
```
diff aetr.vhd ngaetr.vhd

22d21
<     else count_int <= x"2";
23a23
>     else count_int <= x"2";
```

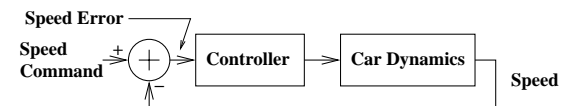
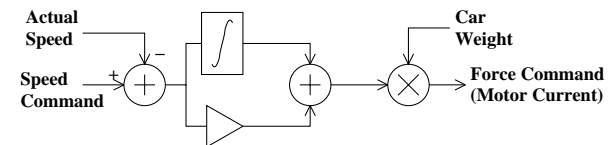
13.7 Example Using Microcode

□ Control for a trolley car (LRV)

- Control FORCE applied to wheels
- Speed command by driver
- Accommodates car weight
- Uses feedback control

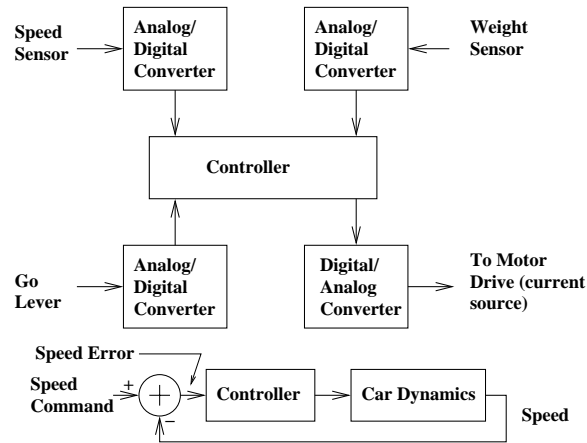


13.8 Approach Uses a PI Controller



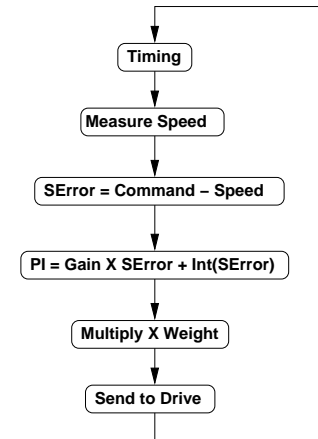
Integral part of control drives speed error to zero
P+I makes the system stable

13.9 High Level Block Diagram



Integral part of control drives speed error to zero
P+I makes the system stable

13.10 High Level Flow Diagram



13.11 A Diversion: Binary Arithmetic

Counting 00000000 = 0,
 00000001 = 1,
 00000010 = 2 etc.

Twos Complement for negative numbers:
Invert all digits and add one

```

16 = 00010000
-16 = 11101111
+   1
= 11110000
16 = 00001111
+   1
= 00010000
    
```

Addition works:

12	00001100	12	00001100	-20	= 11101011
+ 8	00001000	-20	11101100		+ 1
=20	00010100	=-8	11111000		= 11101100
				- 8	= 11110111
					+ 1
					= 11111000

13.12 Multiplication

□ Shift and Add

- You learned this technique in elementary school.
- Takes as many cycles as there are bits.
- Multiplier uses an 8-bit register.
- Accumulator uses a 16-bit shift register.

□ Each cycle, we will rotate the accumulator right.

- This puts the LSB into the high order part of the accumulator.
- We also shift the multiplier register right to get at the next bit of the multiplier.

□ At the end we rotate the accumulator right 8 times to get the answer repositioned.

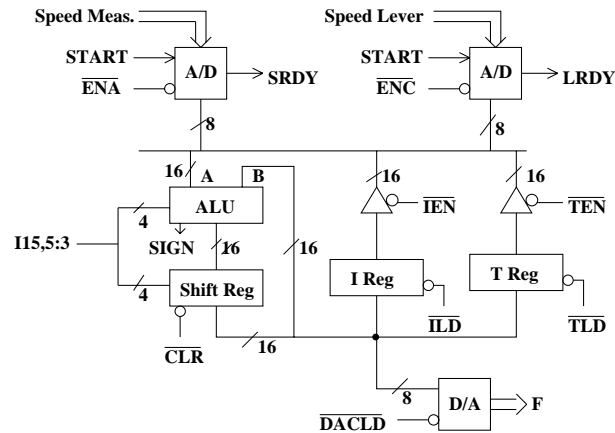
□ With this technique we can only multiply two positive numbers.

- If one number is negative then we invert it before and after the multiply.

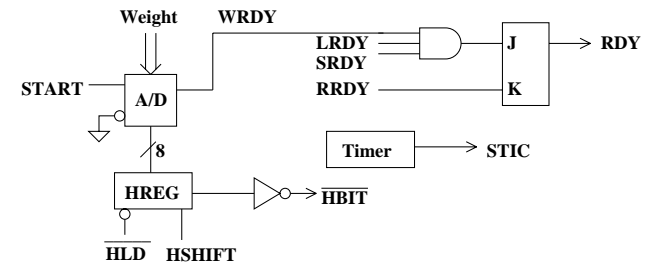
13.13 PI Controller Data Path (A)

□ One implementation

- Integral approximated by sum
- Note signals are summarized later



13.14 PI Controller Data Path (B)



$$F = \text{Weight} * [(set - speed)/4 + \sum_{t=-\infty}^{\text{now}} (set - speed)/16]$$

13.15 ALU and Shift Register Control

□ Directly from the instruction word PROM

- Enabled by I15

□ Shift register is the accumulator

op	I5	I4	I3	ALU Function	SR Function
	0	0	0	don't care	Hold
ADD	0	0	1	A + B	Load
ADD1	0	1	0	A + 1	Load
INV	0	1	1	/A	Load
SRA	1	0	0	don't care	Shift Right Arithmetic
ROT	1	0	1	don't care	Rotate
SUB	1	1	0	A - B	Load
	1	1	1	don't care	don't care

13.16 Signals From/To Data Path

□ Signals required for the PI Controller

○ Condition Signals

- ▷ /HBIT Used in Multiply: this bit of weight is high
- ▷ RDY A/D Converters are ready to be read
- ▷ STIC Timer Tick: used to start the process
- ▷ SIGN Most Significant Bit is 1: Negative Number

○ Control Signals

- ▷ /ENA Enable A/D Converter to drive the bus
- ▷ /HLD Load multiplier register (Use /ENA for this signal)
- ▷ /ENC Enable Control Handle to drive the bus
- ▷ /IEN Enable integrator register to drive the bus
- ▷ /TEN Enable temporary register to drive the bus
- ▷ HSHIFT Shift multiplier register right
- ▷ RRDY Reset Ready Signal

○ Coded into a '138: used one at a time

- ▷ /CLR Clear Accumulator (Shift Register)
- ▷ /ILD Load integrator register
- ▷ /TLD Load temporary register
- ▷ /DACLD Load Output DAC
- ▷ START Start A/D Conversion

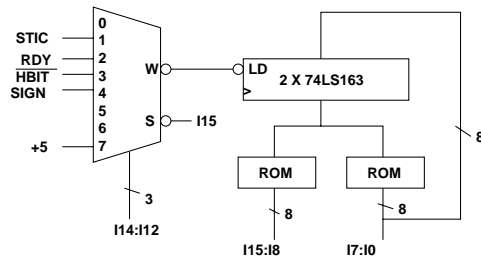
○ Directly from instruction word to control the ALU and shift register

- ▷ I5, I4, I3 Directly from the instruction ROM
- I15 Enables the ALU and Shift Register

13.17 Counter-Based Control Machine

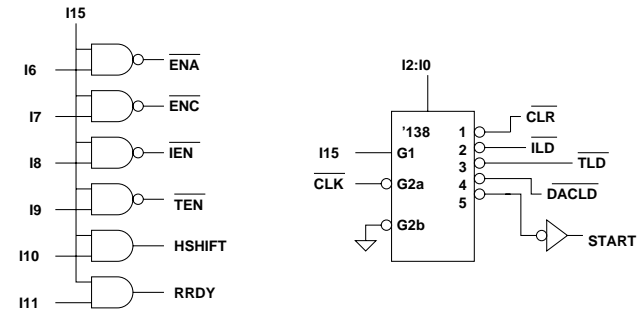
□ Based on 8 bits of counter

- 256 instruction addresses
- Uses 16-bit instruction word
- 8 possible conditions (7 + 'true')



13.18 Combinational Logic for Outputs

- Six assertions done together
- Enabled by Bit 15 (opcode)



13.19 Instruction Specification (Part A)

□ In language of microcode assembler

- For readability, one file is split onto four pages
- Page 1: heading, basic specs, and conditions

```

/* 6.111 L13.sp           Friday, March 9, 2001 */

/* Instruction Set: */
/*      I15 I14 I13 I12 I11 I10 I9  I8  I7  I6  I5  I4  I3  I2  I1  I0 */
/* Assert  1  S  S  S  S  S  S  S  S  S  S  S  S  S  S  S */
/* If C jmp A 0  C  C  C  -  -  -  A  A  A  A  A  A  A  A */

op<15:0>; /* sixteen bit wide word          */
address op<7:0>; /* eight bit address space    */
jmp nop; /* jmp is a convenient mnemonic */
do op<15>=1; /* do means set MSB                */
if op<15>=0; /* if implies setting MSB low        */

/* Condition Signals: */
stic op<14:12>=1; /* timer tic is true                */
rdy op<14:12>=2; /* A/D converters are done                */
/hbit op<14:12>=3; /* multiplicand bit is zero                */
sign op<14:12>=4; /* MSB of accumulator is one            */
true op<14:12>=7; /* wired high for jump instructions      */

```

13.20 Instruction Specification (Part B)

```

/* Assertions */
/* One at a time, coded into 138 */
clr op<2:0> = 1; /* clear accumulator          */
ild op<2:0> = 2; /* load integrator register          */
tld op<2:0> = 3; /* load temporary register          */
daclد op<2:0> = 4; /* load DAC: final output          */
startc op<2:0> = 5; /* start a2d conversion          */
/* ALU and SR control: coded into bits 5:3 */
add op<5:3> = 1; /* add to accumulator          */
add1 op<5:3> = 2; /* add 1 to A                    */
invert op<5:3> = 3; /* invert A and store          */
sra op<5:3> = 4; /* arithmetic shift right      */
rot op<5:3> = 5; /* rotate right one bit        */
sub op<5:3> = 6; /* subtract B from A          */
/* single bit assertions */
ena op<6> = 1; /* A/D converter (speed) drives bus */
enc op<7> = 1; /* speed command drives bus      */
ien op<8> = 1; /* integrator register drives bus */
ten op<9> = 1; /* temporary register drives bus  */
hshift op<10> = 1; /* shift multiplicand register  */
rrdy op<11> = 1; /* reset ready latch            */

```

13.21 Microcode for Controller (page 1)

```
/* 6.111 L13                                Friday March 9, 2001 */

start: if stic jmp conv; /* wait for timer tic */
      if true jmp start;
conv:  do startc;
cwait: if rdy jmp ld1; /* end of conversion? */
      if true jmp cwait;
ld1:   do clr rrdy; /* first, clear accumulator */
      do ena add; /* put speed into accumulator */
      do enc sub; /* command minus speed */
      do sra; /* divide by 4 */
      do sra;
      do tld; /* save this in temp */
      do sra; /* another divide by four */
      do sra; /* for the integration step */
      do ien add; /* add to integrator reg */
      do ten add; /* now add in temp */
      do tld; /* save number in temp */
      if sign jmp nmul;
```

13.22 Microcode for Controller (page 2)

```
      do clr; /* clear register */
      if /hbit jmp mul1; /* jump if hbit is zero */
      do ten add; /* otherwise, add number */
mul1:  do hshift; /* then shift h register */
      do rot; /* rotate accumulator right */
      if /hbit jmp mul2; /* jump if hbit is zero */
      do ten add; /* otherwise, add number */
mul2:  do hshift; /* then shift h register */
      do rot; /* rotate accumulator right */
      if /hbit jmp mul3; /* jump if hbit is zero */
      do ten add; /* otherwise, add number */
mul3:  do hshift; /* then shift h register */
      do rot; /* rotate accumulator right */
      if /hbit jmp mul4; /* jump if hbit is zero */
      do ten add; /* otherwise, add number */
mul4:  do hshift; /* then shift h register */
      do rot; /* rotate accumulator right */
      if /hbit jmp mul5; /* jump if hbit is zero */
      do ten add; /* otherwise, add number */
mul5:  do hshift; /* then shift h register */
      do rot; /* rotate accumulator right */
      if /hbit jmp mul6; /* jump if hbit is zero */
      do ten add; /* otherwise, add number */
```

13.23 Microcode for Controller (page 3)

```
mul6:  do hshift; /* then shift h register */
      do rot; /* rotate accumulator right */
      if /hbit jmp mul7; /* jump if hbit is zero */
      do ten add; /* otherwise, add number */
mul7:  do hshift; /* then shift h register */
      do rot; /* rotate accumulator right */
      if /hbit jmp mul8; /* jump if hbit is zero */
      do ten add; /* otherwise, add number */
mul8:  do rot; /* put number back */
      do rot;
      do rot;
      do rot;
      do rot;
      do rot;
      do rot;
      do rot;
      if true jmp done; /* go done and load the output dac */
nmul:  do clr; /* clear register */
      do ten invert; /* complement temp register */
      do tld; /* store it in temp register */
      do ten addl; /* add 1 to it */
      do tld; /* put it in temp register */
      if /hbit jmp nmul1; /* jump if hbit is zero */
      do ten add; /* otherwise, add number */
```

13.24 Microcode for Controller (page 4)

```
nmul1: do hshift; /* then shift h register */
      do rot; /* rotate accumulator right */
      if /hbit jmp nmul2; /* jump if hbit is zero */
      do ten add; /* otherwise, add number */
nmul2: do hshift; /* then shift h register */
      do rot; /* rotate accumulator right */
      if /hbit jmp nmul3; /* jump if hbit is zero */
      do ten add; /* otherwise, add number */
nmul3: do hshift; /* then shift h register */
      do rot; /* rotate accumulator right */
      if /hbit jmp nmul4; /* jump if hbit is zero */
      do ten add; /* otherwise, add number */
nmul4: do hshift; /* then shift h register */
      do rot; /* rotate accumulator right */
      if /hbit jmp nmul5; /* jump if hbit is zero */
      do ten add; /* otherwise, add number */
nmul5: do hshift; /* then shift h register */
      do rot; /* rotate accumulator right */
      if /hbit jmp nmul6; /* jump if hbit is zero */
      do ten add; /* otherwise, add number */
```

13.25 Microcode for Controller (page 5)

```
nmul6: do hshift; /* then shift h register */
       do rot; /* rotate accumulator right */
       if /hbit jmp nmul7; /* jump if hbit is zero */
       do ten add; /* otherwise, add number */
nmul7: do hshift; /* then shift h register */
       do rot; /* rotate accumulator right */
       if /hbit jmp nmul8; /* jump if hbit is zero */
       do ten add; /* otherwise, add number */
nmul8: do rot; /* rotate right to put number back */
       do rot;
       do rot;
       do rot;
       do rot;
       do rot;
       do rot;
       do rot;
       do ten invert; /* invert the answer */
       do tld;
       do ten addl;
       do tld;
done:  do dacl; /* put into output */
       if true jmp start; /* go back for more */
```

13.26 Error

```
troxel@sunpall 25 % assem16to8 113
```

```
READING SPECIFICATION FILE
```

```
ASSEMBLING
```

```
assembler : line 68: do ten addl;
                warning, command fields overlap when adding 'do',
Check for a missing semicolon.
```

```
assembler : line 110: do ten addl;
                warning, command fields overlap when adding 'do',
Check for a missing semicolon.
```

```
troxel@sunpall 26 %
```