

13.1 Example Using Microcode

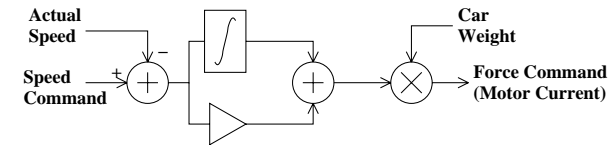
Friday, March 8, 2002

- Control for a trolley car (Light Rail Vehicle)
- Three Inputs
 - Desired Speed
 - Actual Speed
 - Car Weight
- One Output
 - Force Command



13.2 Approach Uses a PI Controller

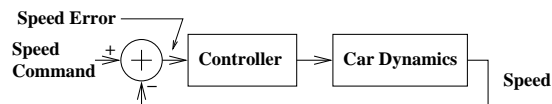
We want to control a trolley car (Light Rail Vehicle). We will do this by controlling the FORCE applied to the wheels. We assume this force is proportional to the motor current. The driver positions a 'go lever' to specify the desired speed. We will use this speed command along with a measurement of the actual speed to calculate the motor current. The actual weight of the car (plus passengers) is a variable in the computation of the motor current. The difference between commanded speed and actual car speed should be the acceleration. $F = MA$. We will use PI control. The integral part drives the error to zero. The proportional part gives stability.



13.3 Feedback Control

By measuring the actual speed and feeding that back to the PI controller, we can drive the speed error (exponentially) to zero.

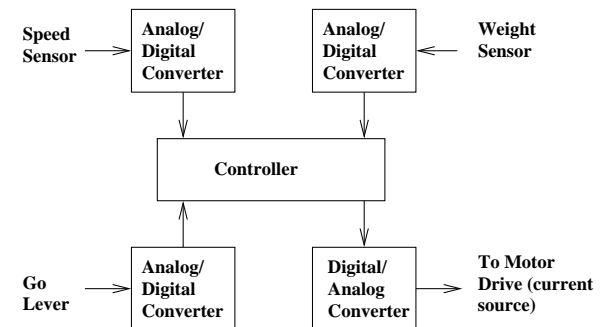
So, in this example, we will examine how to build the controller. We assume a highly ideal drive, in which the drive force is directly proportional to the commanded motor current.



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

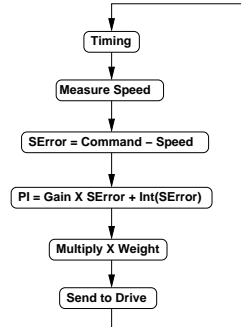
13.4 High Level Block Diagram

The first thing we do (as digital engineers) is to convert all analog signals to their digital representation and to convert the (digital) computed force command (motor drive current) to the required analog voltage.



13.5 High Level Flow Diagram

Timing establishes a fixed interval for our control.
 Speed error is the difference between the measured and commanded speeds.
 PI is just the addition of the proportional and integrated signal.
 Multiply is (perhaps) the most interesting step.



13.6 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
    
```

13.7 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 (dropping the 8 LSB).

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

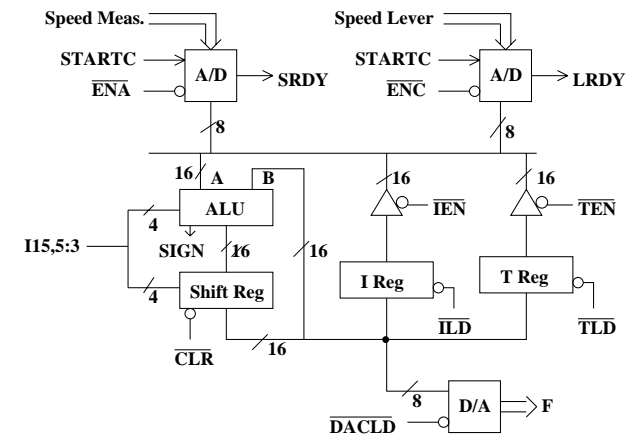
- If one number is negative then we invert it before and after the multiply.
- In our case the weight is always positive.

13.8 PI Controller Data Path (A)

□ One implementation

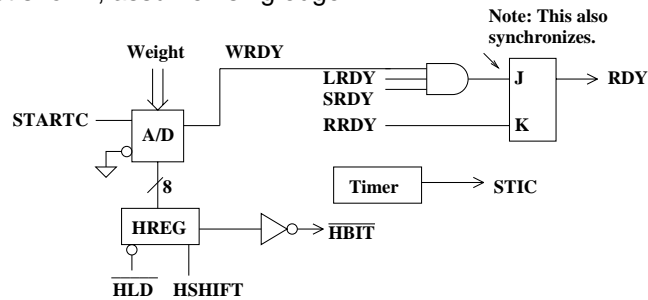
clocks not shown, assume rising edge

- Integral approximated by sum
- Note signals are summarized later



13.9 PI Controller Data Path (B)

- Again, One implementation clocks not shown, assume rising edge



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

13.10 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

13.11 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
- ▷ STARTC 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

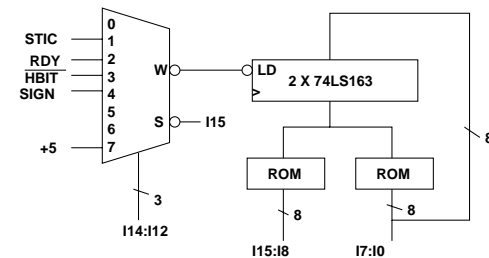
13.12 Counter-Based Control Machine

- Based on 8 bits of counter

○ 256 instruction addresses

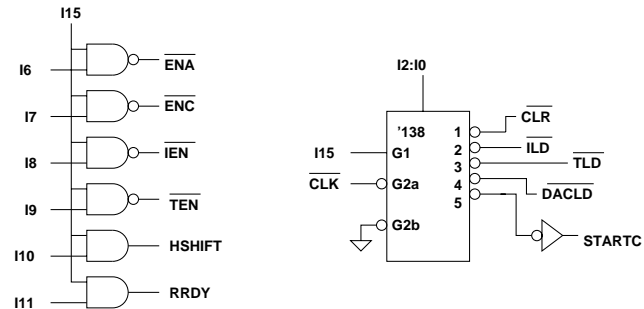
○ Uses 16-bit instruction word

○ 8 possible conditions (7 + 'true')



13.13 Combinational Logic for Outputs

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



13.14 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 8, 2002 */

/* 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 */

```

13.15 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          */
dacl  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          */

```

13.16 Microcode for Controller (page 1)

```

/* 6.111 L13                      Friday March 8, 2002 */

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 */
      /* and JK flip-flop */
      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;

```


13.21 Error

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

```
READING SPECIFICATION FILE
```

```
ASSEMBLING
```

```
assembler : line 68:  do ten add1;  
                warning, command fields overlap when adding 'do',
```

```
Check for a missing semicolon.
```

```
assembler : line 110: do ten add1;  
                warning, command fields overlap when adding 'do',
```

```
Check for a missing semicolon.
```

```
troxel@sunpall 26 %
```