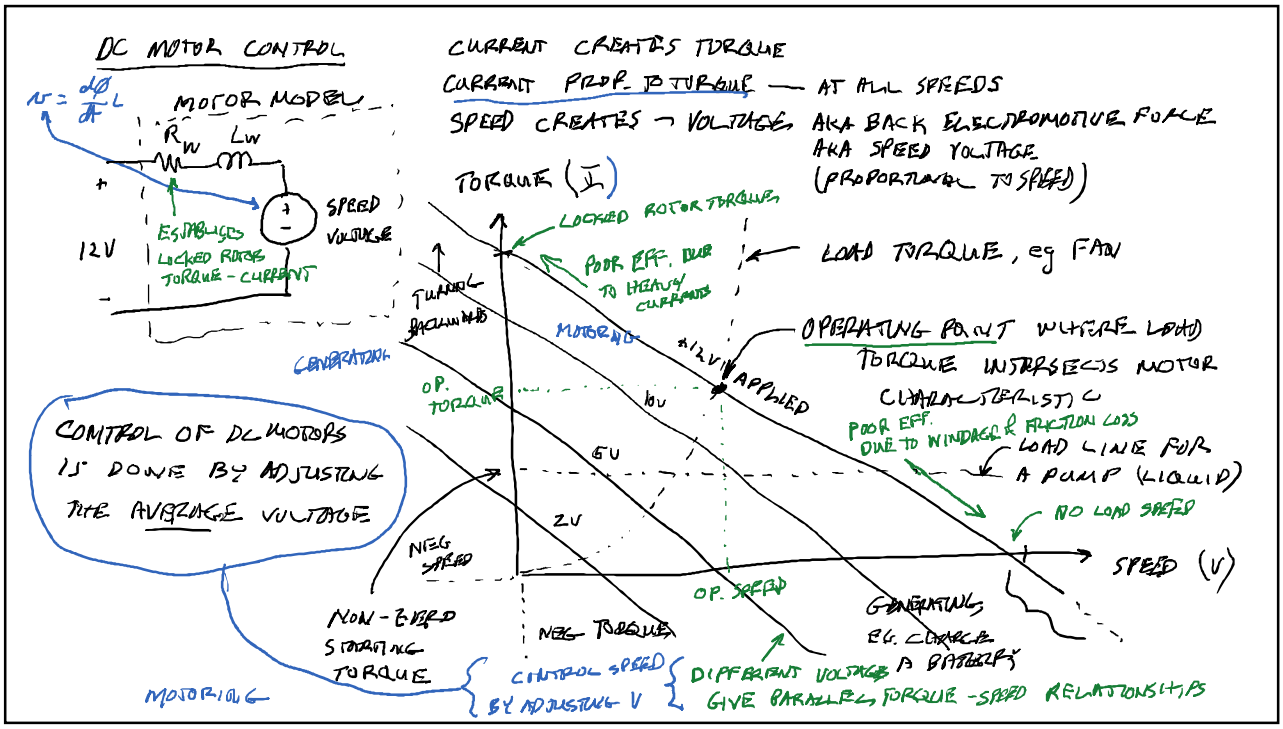
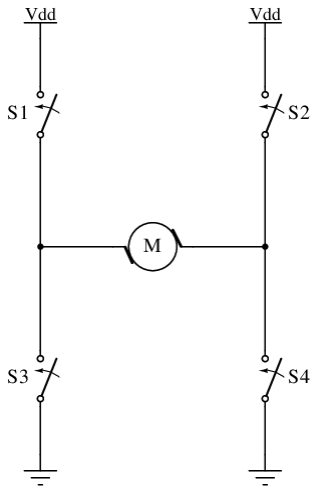


1

H-Bridge (application to DC motor control)

By closing various switches voltage can be applied to the motor in either polarity.

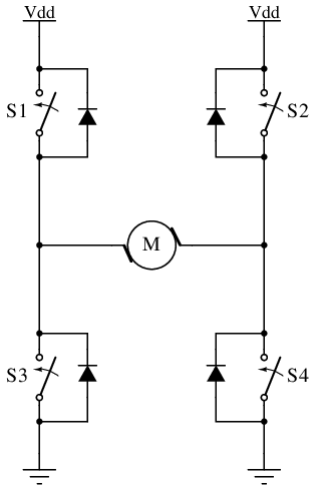


2

H-Bridge (application to DC motor control)

By closing various switches voltage can be applied to the motor in either polarity.

But switches are not enough. The motor is an inductive load (has coils of wire). Thus you **MUST** incorporate flyback diodes.



3

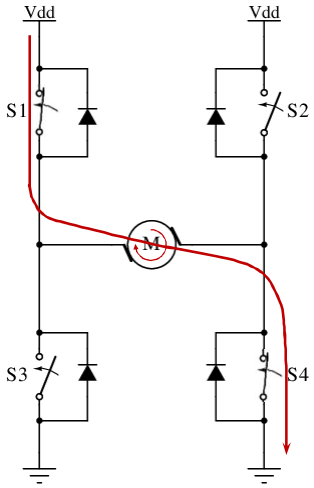
H-Bridge (application to DC motor control)

To run the motor in one direction, say clockwise, close S1 and S4.

To run the motor in the other direction, say counterclockwise, close S2 and S3.

To allow the motor to coast, open all four switches.

To brake the motor, close S3 and S4 (or alternatively, close S1 and S2).



4

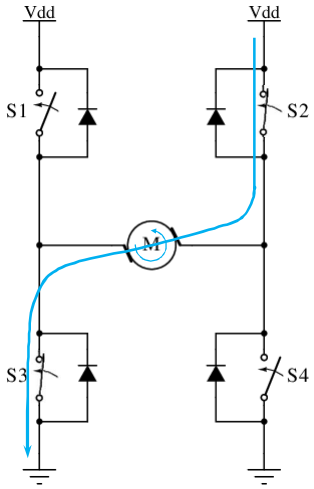
H-Bridge (application to DC motor control)

To run the motor in one direction, say clockwise, close S1 and S4.

To run the motor in the other direction, say counterclockwise, close S2 and S3.

To allow the motor to coast, open all four switches.

To brake the motor, close S3 and S4 (or alternatively, close S1 and S2).



5

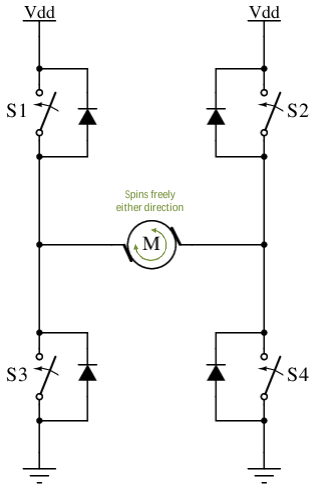
H-Bridge (application to DC motor control)

To run the motor in one direction, say clockwise, close S1 and S4.

To run the motor in the other direction, say counterclockwise, close S2 and S3.

To allow the motor to coast, open all four switches.

To brake the motor, close S3 and S4 (or alternatively, close S1 and S2).



6

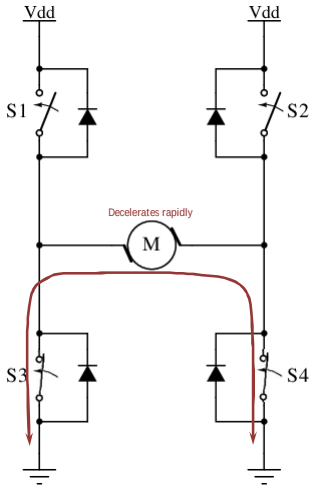
H-Bridge (application to DC motor control)

To run the motor in one direction, say clockwise, close S1 and S4.

To run the motor in the other direction, say counterclockwise, close S2 and S3.

To allow the motor to coast, open all four switches.

To brake the motor, close S3 and S4 (or alternatively, close S1 and S2).



7

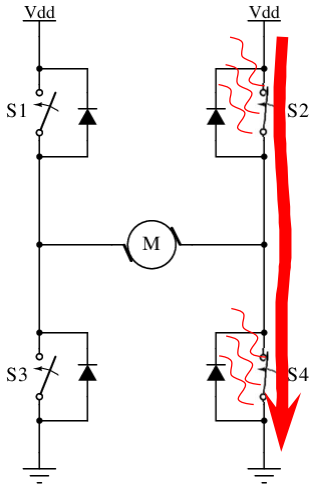
H-Bridge (application to DC motor control)

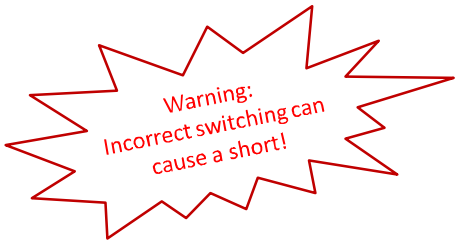
To run the motor in one direction, say clockwise, close S1 and S4.

To run the motor in the other direction, say counterclockwise, close S2 and S3.

To allow the motor to coast, open all four switches.

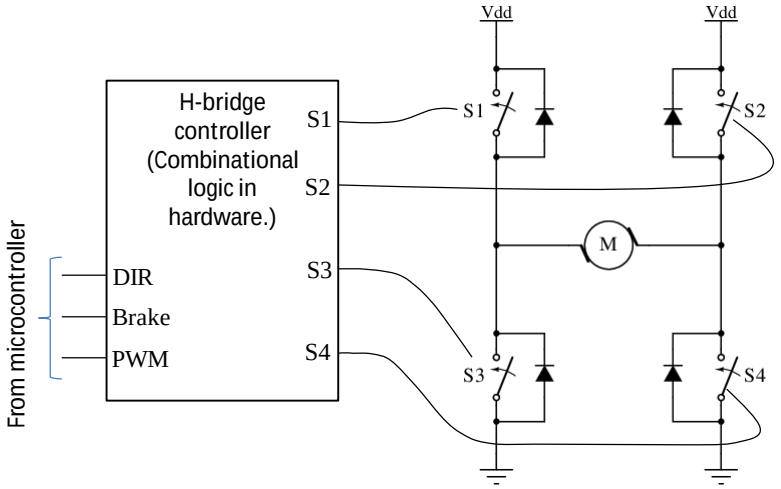
To brake the motor, close S3 and S4 (or alternatively, close S1 and S2).





8

H-Bridge (application to DC motor control)



Warning: Incorrect switching can cause a short!

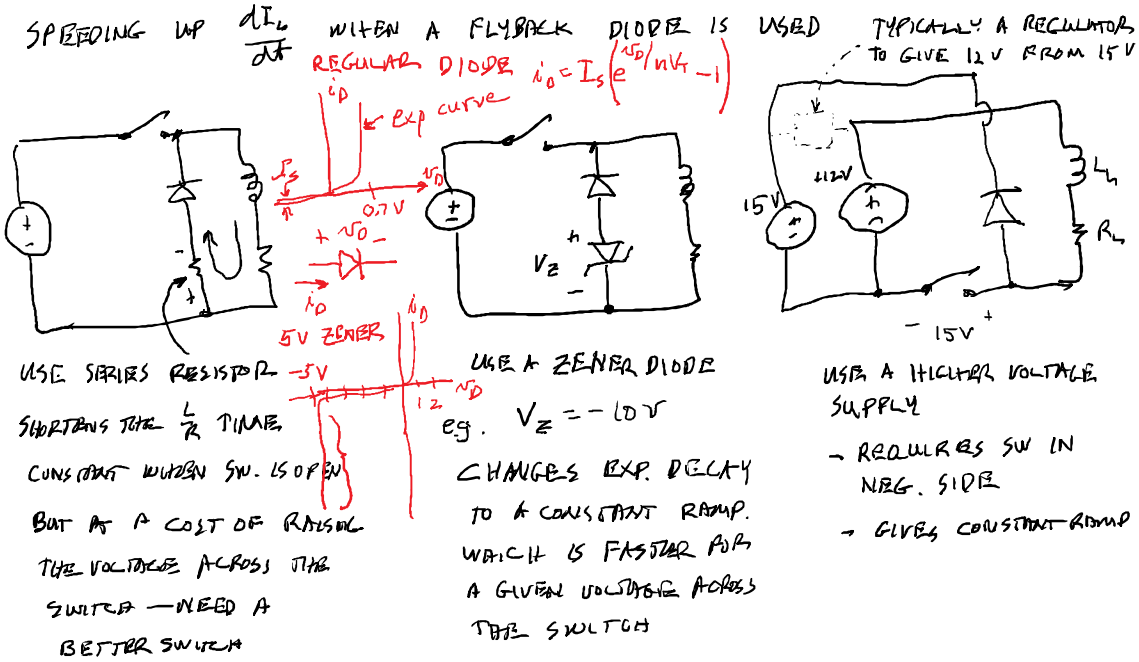
To prevent a short, combinational logic (NOT SOFTWARE) is commonplace.

Inputs			Outputs				Result
Dir	Brake	PWM	S1	S2	S3	S4	
0	0	0	open	open	open	open	Coast
0	0	1	close	open	open	close	Fwd
0	1	0	open	open	open	open	Coast
0	1	1	open	open	close	close	Brake
1	0	0	open	open	open	open	Coast
1	0	1	open	close	close	open	Rev
1	1	0	open	open	open	open	Coast
1	1	1	open	open	close	close	Brake

Never drive H-bridge switches directly from a microcontroller.
Physical destruction, maybe even fire,
is assured due to software bugs and crashes.

Microcontroller produces Dir, Brake, PWM
No matter what it produces the H-bridge
cannot short out the power supply.

9



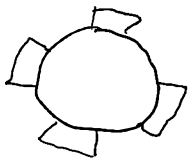
10

POSITION ENCODERS — TO SENSE LINEAR OR ANGULAR POSITION

RELATIVE POSITION — A SIGNAL FOR EVERY x DEGREES OF ROTATION
(SOME OTHER MECHANISM IS USED ONE TIME TO EST. ABS. POSITION)

ABSOLUTE POSITION — A MULTI-BIT VALUE FOR EACH POSITION

RELATIVE POSITION



IN THIS EXAMPLE A SENSOR SIGNAL EVERY $\frac{1}{8}$ OF A TURN

RESOLUTION CAN BE $\frac{360^\circ}{\#\text{EDGES}} = \frac{360}{2 \times \#\text{TEETH}} = \frac{360^\circ}{2 \times 4} = 45^\circ$

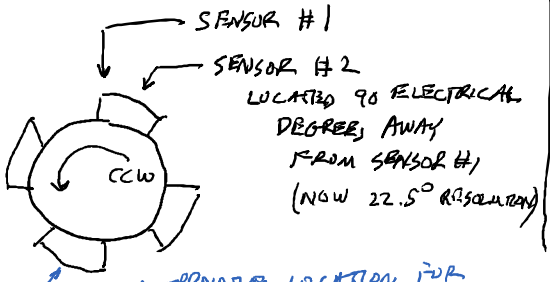
TOO SENSITIVE TO VIBRATION

IF YOU STOP ON A BOUNDARY VIBRATION CAN CAUSE FALSE COUNTS.

(TOO SIMPLE FOR ACTUAL USE UNDER ANY CONDITION)

11

QUADRATURE ENCODER POPULAR

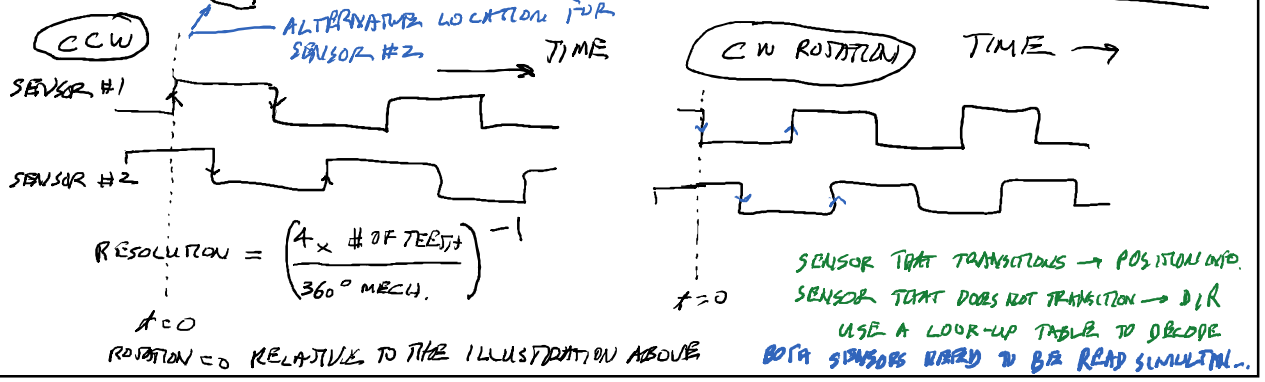


LOCATED 90 ELECTRICAL DEGREES AWAY FROM SENSOR #1
(NOW 22.5° RESOLUTION)

DETECTING DIRECTION OF ROTATION

CCW
- SENSOR #1 L→H AND #2 = H1
- SENSOR #2 H→L AND #1 = H1
- SENSOR #1 H→L AND #2 = LO
- SENSOR #2 L→H AND #1 = LO

CW
- #1 H→L AND #2 = H1
- #2 H→L AND #1 = LO
- #1 L→H AND #2 = LO
- #2 L→H AND #1 = H1



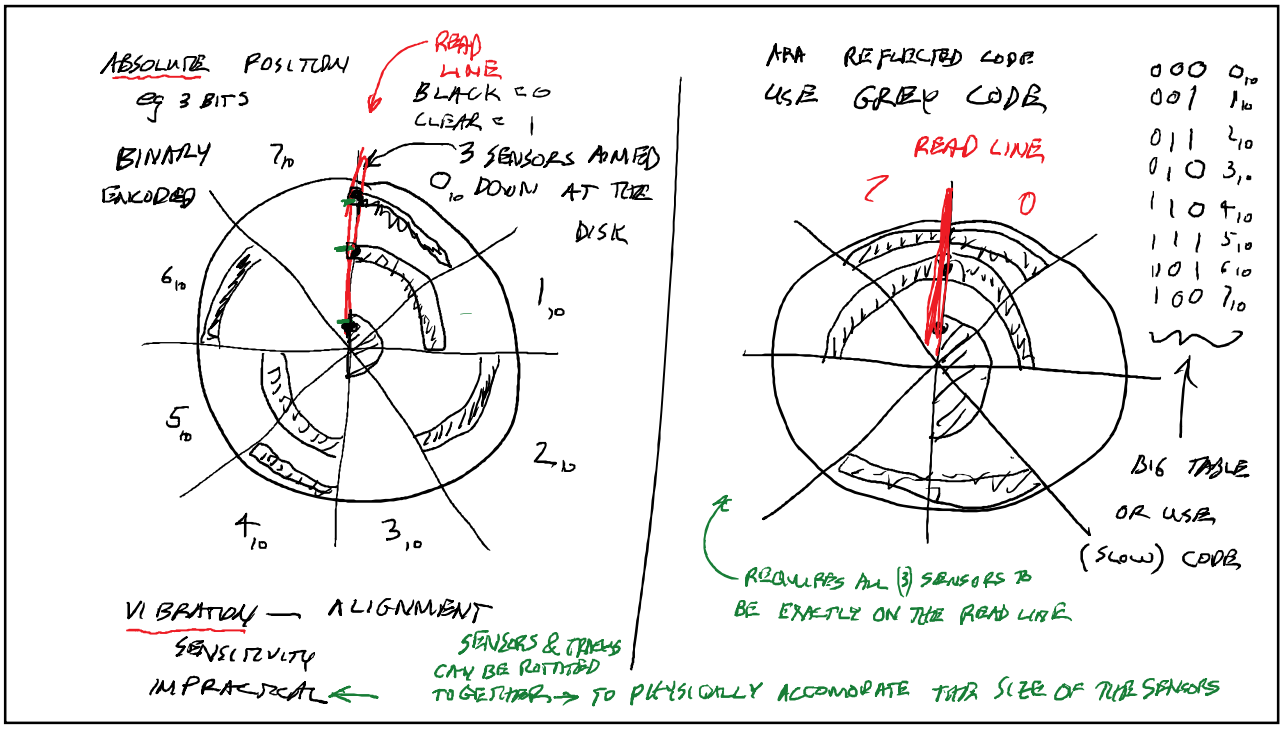
$\text{RESOLUTION} = \left(\frac{4 \times \#\text{ OF TEETH}}{360^\circ \text{ MECH.}}\right)^{-1}$

$t = 0$

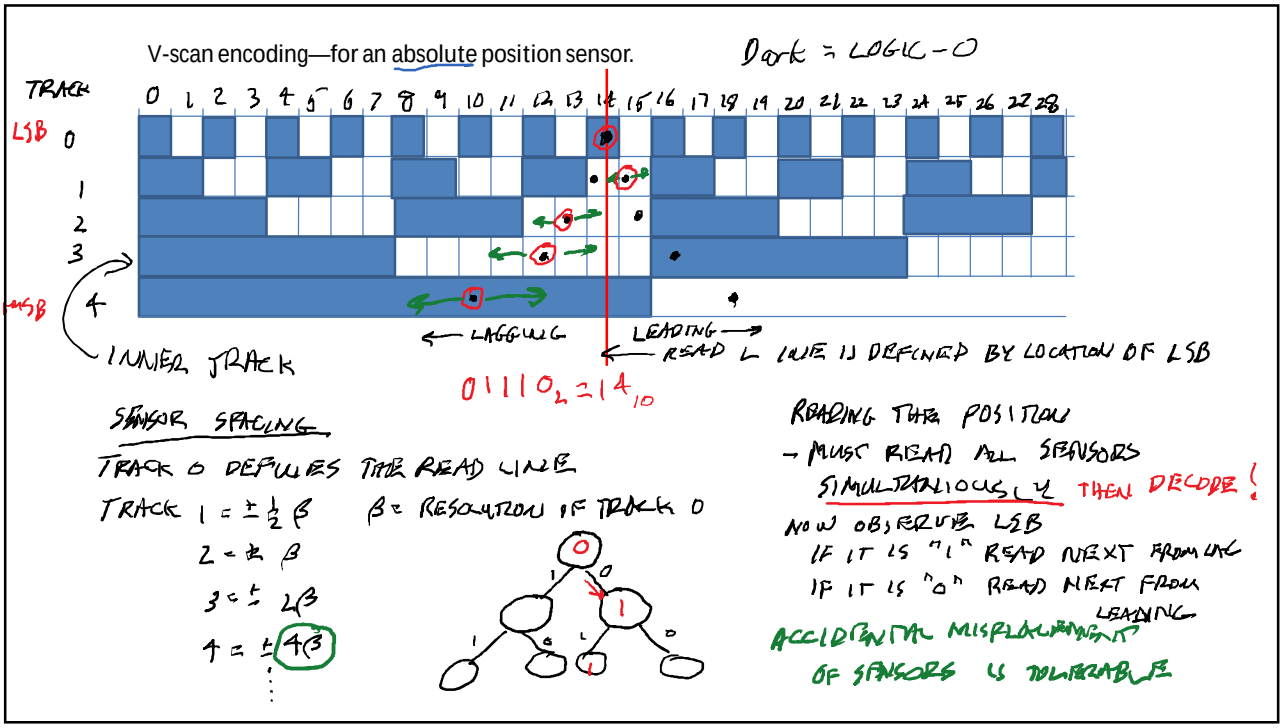
ROTATION = 0 RELATIVE TO THE ILLUSTRATION ABOVE

SENSOR THAT TRANSITIONS → POSITION INFO.
SENSOR THAT DOES NOT TRANSITION → D/R
USE A LOOK-UP TABLE TO DECODE
BOTH SENSORS NEEDED TO BE READ SIMULTANEOUSLY.

12


ABSOLUTE POSITION
eg 3 BITS
BINARY ENCODED
READ LINE
BLACK = 0
CLEAR = 1
3 SENSORS ADMED DOWN AT THE DISK
7₁₀
6₁₀
1₁₀
5₁₀
2₁₀
4₁₀
3₁₀
VIBRATION — ALIGNMENT
SENSITIVITY
IMPRACTICAL
SENSORS & TRACKS CAN BE ROTATED TOGETHER → TO PHYSICALLY ACCOMODATE THE SIZE OF THE SENSORS
USE REFLECTED CODE
USE GREY CODE
READ LINE
7
0
000 0₁₀
001 1₁₀
011 2₁₀
010 3₁₀
110 4₁₀
111 5₁₀
101 6₁₀
100 7₁₀
BIG TABLE OR USE (SLOW) CODE
REQUIRES ALL (3) SENSORS TO BE EXACTLY ON THE READ LINE

13


V-scan encoding—for an absolute position sensor.
Dark = LOGIC-0
TRACK
LSB
MSB
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 19 19 20 21 22 23 24 25 26 27 28
INNER TRACK
LAGGING
LEADING
READ LINE IS DEFINED BY LOCATION OF LSB
0111₂ = 14₁₀
SENSOR SPACING
TRACK 0 DEFINES THE READ LINE
TRACK 1 = ± ½ β
2 = ± β
3 = ± 2β
4 = ± 4β
β = RESOLUTION OF TRACK 0
READING THE POSITION
- MUST READ ALL SENSORS SIMULTANEOUSLY THEN DECODE!
NOW OBSERVE LSB
IF IT IS "1" READ NEXT FROM LAG
IF IT IS "0" READ NEXT FROM LEADING
ACCIDENTAL MISPLACEMENT OF SENSORS IS TOLERABLE

14

U-scan encoding—Same as V-scan except at some point the sensor spacing stops increasing.

eg

TRACK	SPACING
0	0
1	$\pm \frac{1}{2}\beta$
2	$\pm\beta$
3	$\pm\frac{3}{2}\beta$
4	$\pm 2\beta$
5	$\pm 2\beta$
6	$\pm 2\beta$

↑ V SCAN UP TO THIS POINT

↓ U SCAN

CALLED "U-SCAN"

MULTIPLE TURN SCANNING IS POSSIBLE USING GEAR MECHANISMS

15

Doorbell. . .
Better go answer it.

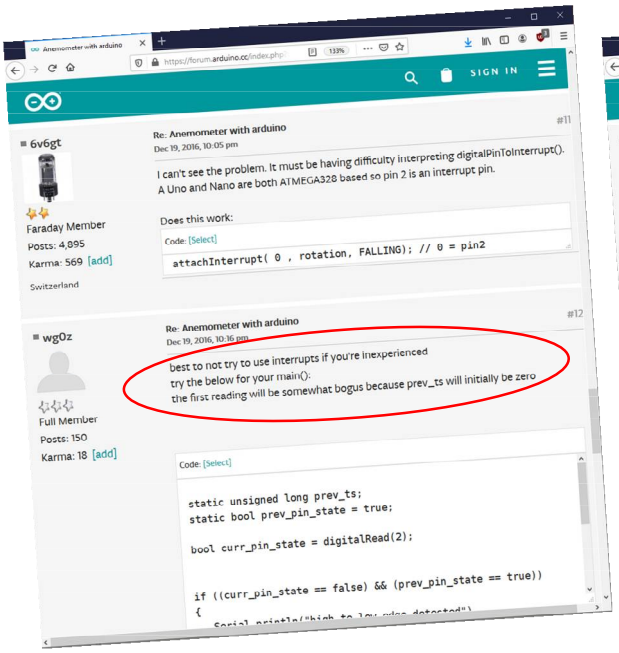
Ahh. . .
Delivery guy is bringing slides about Interrupt Driven I/O!

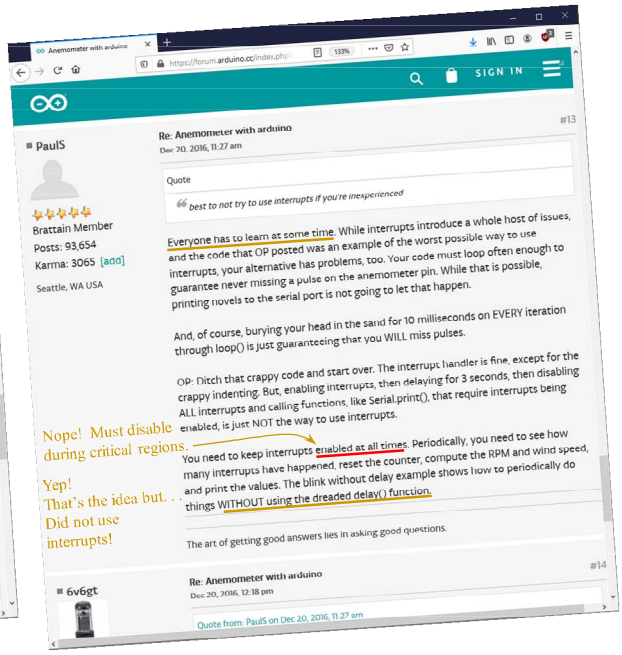
(Here we leave most of the hobby-oriented Web pages behind.)

https://www.explainthatstuff.com/how-electric-doorbells-work.html

16

Words from the Arduino Forum:





17

SUMMARY SLIDE

How can the port be controlled? How is I/O accomplished with this hardware?

This slide repeated from 1/17

It is a two-step process

1.) Setup: place bits in various control registers to establish . . .
--direction of I/O, input or output
--if input, and no connection to it, default high (enable pull up resistor), or default low (if available), or random
--if output, what is the initial output before the first write after power-up?, 1, 0 or X

2.) Do the actual I/0. There are various strategies

a.) *Blind-cycle*: Just do it immediately as the code runs. Not in coordination with the I/O device.
(Ready or not, hear I come!)

Gadfly: an annoying person (among other meanings)

b.) *Busy-waiting* (aka *gadfly* I/O): Use a status bit to check the I/O device before reading or writing to it. Result: while I/O device is busy, CPU needs to wait and monitor, hence the name.
(The CPU is analogous to kids in the back seat, "Are we there yet? Are we there yet? Are we there yet?. . ." The kids don't do homework while waiting, they busy themselves only with the pestering question.)

c.) *Periodic polling*: Similar to Busy-waiting, but CPU may work on other threads of code while waiting on a busy I/O device. (Requires a timer interrupt—i.e requires additional hardware.)
(The kids do homework while waiting. Every five minutes a bell rings and they ask the question.)

d.) *Interrupt driven:* The I/O device has a method in hardware to request I/O service.
(The kids stick to their homework until told that they have arrived at their destination.)

e.) *Direct Memory Access*: The I/O device takes over the CPU bus and writes directly into memory without CPU supervision. (The kids are not in the car!)

18

The agenda—understanding interrupt-driven I/O (and by extension, multitasking)

An example to give some context

Memory capabilities needed for subroutines (functions, procedures, interrupts, are types of subroutines)

Sources of interrupts including counter-timer systems

Advantages of using interrupt-driven I/O—so obvious this section is hardly needed.
- --Alternatives to interrupt driven I/O are gadfly (uncontrolled—annoying) I/O or various polling techniques, all of which waste processor cycles prodigiously.
- --Interrupts are foundational to object-oriented programming
- --Many embedded systems that use interrupts have very little other code to run!

Risks of interrupt-driven I/O
- --density limit
- --latency and resolution limits
- --interval restrictions
- --critical regions in code
- --deadlock

19

The agenda—understanding interrupt-driven I/O (and by extension, multitasking)

An example to give some context

Memory capabilities needed for subroutines (functions, procedures, interrupts, are types of subroutines)

Sources of interrupts including counter-timer systems

Advantages of using interrupt-driven I/O—so obvious this section is hardly needed.
- --Alternatives to interrupt driven I/O are gadfly (uncontrolled—annoying) I/O or various polling techniques, all of which waste processor cycles prodigiously.
- --Interrupts are foundational to object-oriented programming
- --Many embedded systems that use interrupts have very little other code to run!

Risks of interrupt-driven I/O
- --density limit
- --latency and resolution limits
- --interval restrictions
- --critical regions in code
- --deadlock

20

An example of interrupt-driven I/O to give us some context

Goal: Use the 60 Hz powerline frequency as a time-reference for measuring time intervals.
(Could be extended to keeping real-time if ways to initially set the date and time-of-day are provided.)

A transformer will be used to step the mains voltage down to a convenient level. (About 9 V peak)

A wave-shaping circuit will be used to convert the sinusoidal shape (±9 V) to a TTL-level square wave (0 V to 5 V).

The square-wave signal will be connected to a digital I/O pin of the Arduino Uno.

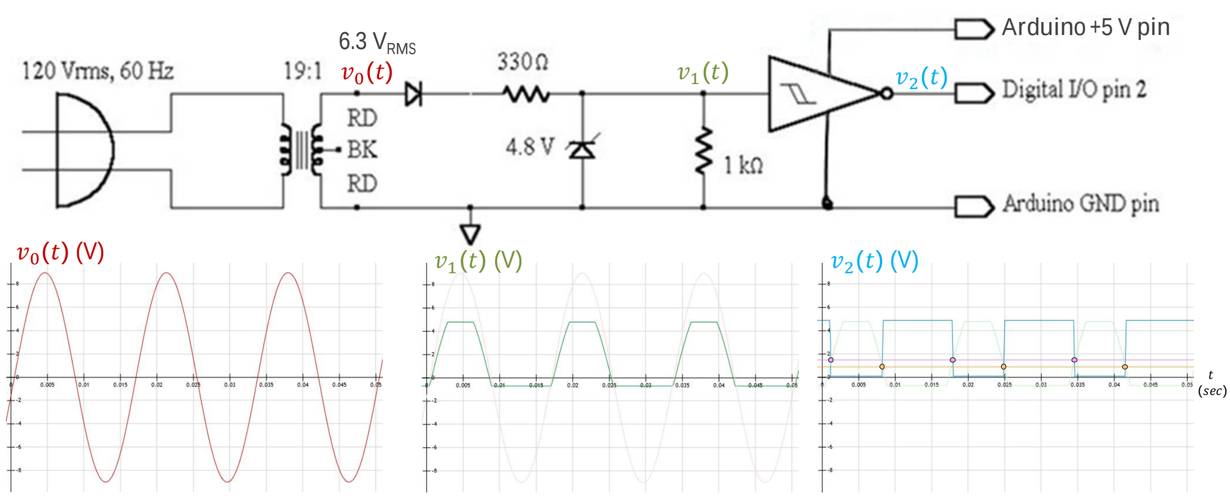
The Arduino Uno will used to count the cycles of the mains power. Every 60 counts represents 1 second.

The above will be accomplished in the background so that a program operating in the `loop()` section of the code can reference the relative time. (Relative time is the time since the Arduino Uno user program was loaded, reset, or since the relative time register rolled over.)

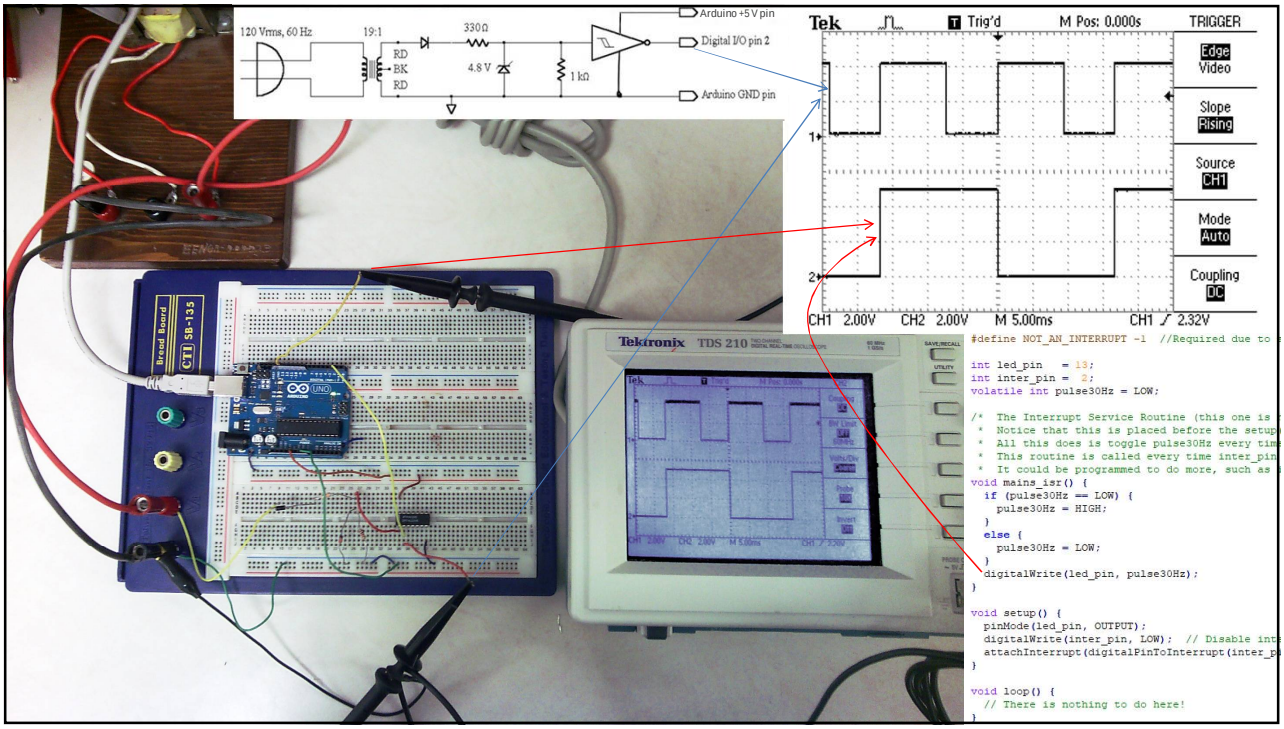
21

An example of interrupt-driven I/O

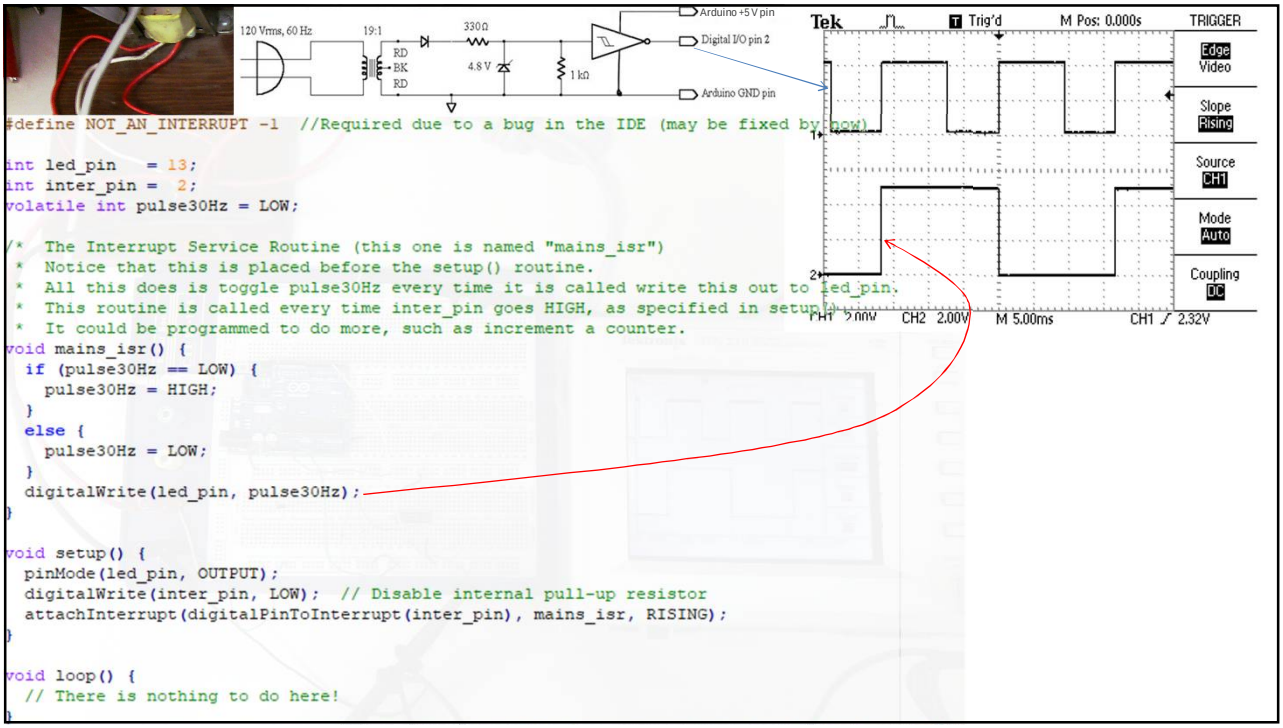
Goal: Use the 60 Hz powerline frequency as a time-reference for measuring time intervals.
(Could be extended to keeping real-time if ways to initially set the date and time-of-day are provided.)



22


120 Vrms, 60 Hz
19:1
330Ω
4.8 V
1 kΩ
RD
BK
RD
Arduino +5 V pin
Digital I/O pin 2
Arduino GND pin
Tek
Trig'd
M Pos: 0.000s
TRIGGER
Edge
Video
Slope
Rising
Source
CH1
Mode
Auto
Coupling
DC
CH1 2.00V
CH2 2.00V
M 5.00ms
CH1 ∕ 2.32V
Tektronix
TDS 210
#define NOT_AN_INTERRUPT -1 //Required due to
int led_pin = 13;
int inter_pin = 2;
volatile int pulse30Hz = LOW;
/* The Interrupt Service Routine (this one is
* Notice that this is placed before the setup
* All this does is toggle pulse30Hz every tim
* This routine is called every time inter_pin
* It could be programmed to do more, such as
void mains_isr() {
if (pulse30Hz == LOW) {
pulse30Hz = HIGH;
}
else {
pulse30Hz = LOW;
}
digitalWrite(led_pin, pulse30Hz);
}
void setup() {
pinMode(led_pin, OUTPUT);
digitalWrite(inter_pin, LOW); // Disable int
attachInterrupt(digitalPinToInterrupt(inter_p
}
void loop() {
// There is nothing to do here!
}

23


120 Vrms, 60 Hz
19:1
330Ω
4.8 V
1 kΩ
RD
BK
RD
Arduino +5 V pin
Digital I/O pin 2
Arduino GND pin
Tek
Trig'd
M Pos: 0.000s
TRIGGER
Edge
Video
Slope
Rising
Source
CH1
Mode
Auto
Coupling
DC
CH1 2.00V
CH2 2.00V
M 5.00ms
CH1 ∕ 2.32V
#define NOT_AN_INTERRUPT -1 //Required due to a bug in the IDE (may be fixed by now)
int led_pin = 13;
int inter_pin = 2;
volatile int pulse30Hz = LOW;
/* The Interrupt Service Routine (this one is named "mains_isr")
* Notice that this is placed before the setup() routine.
* All this does is toggle pulse30Hz every time it is called write this out to led_pin.
* This routine is called every time inter_pin goes HIGH, as specified in setup()
* It could be programmed to do more, such as increment a counter.
void mains_isr() {
if (pulse30Hz == LOW) {
pulse30Hz = HIGH;
}
else {
pulse30Hz = LOW;
}
digitalWrite(led_pin, pulse30Hz);
}
void setup() {
pinMode(led_pin, OUTPUT);
digitalWrite(inter_pin, LOW); // Disable internal pull-up resistor
attachInterrupt(digitalPinToInterrupt(inter_pin), mains_isr, RISING);
}
void loop() {
// There is nothing to do here!
}

24