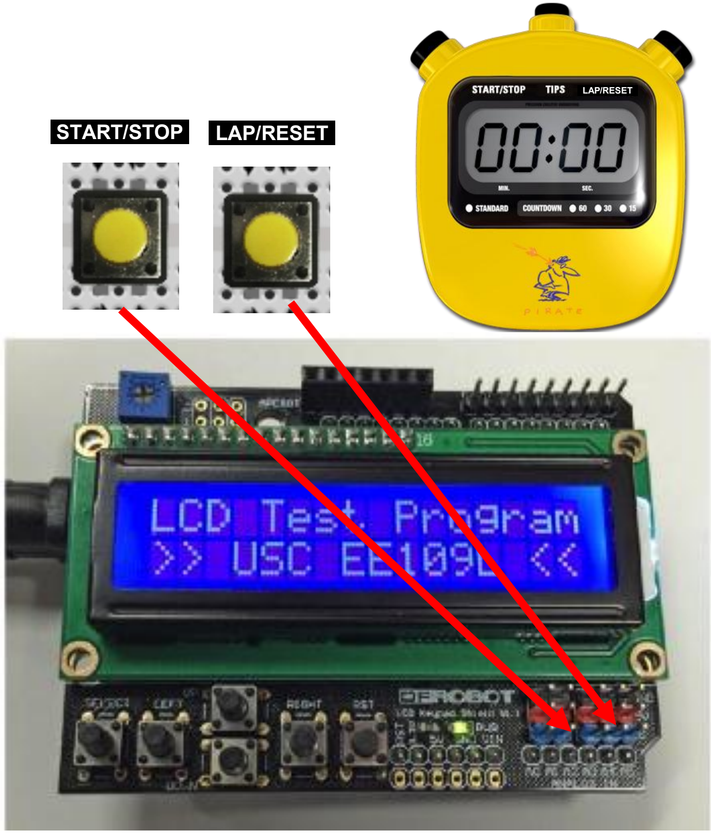
Timers Lab
(Stopwatch Application)

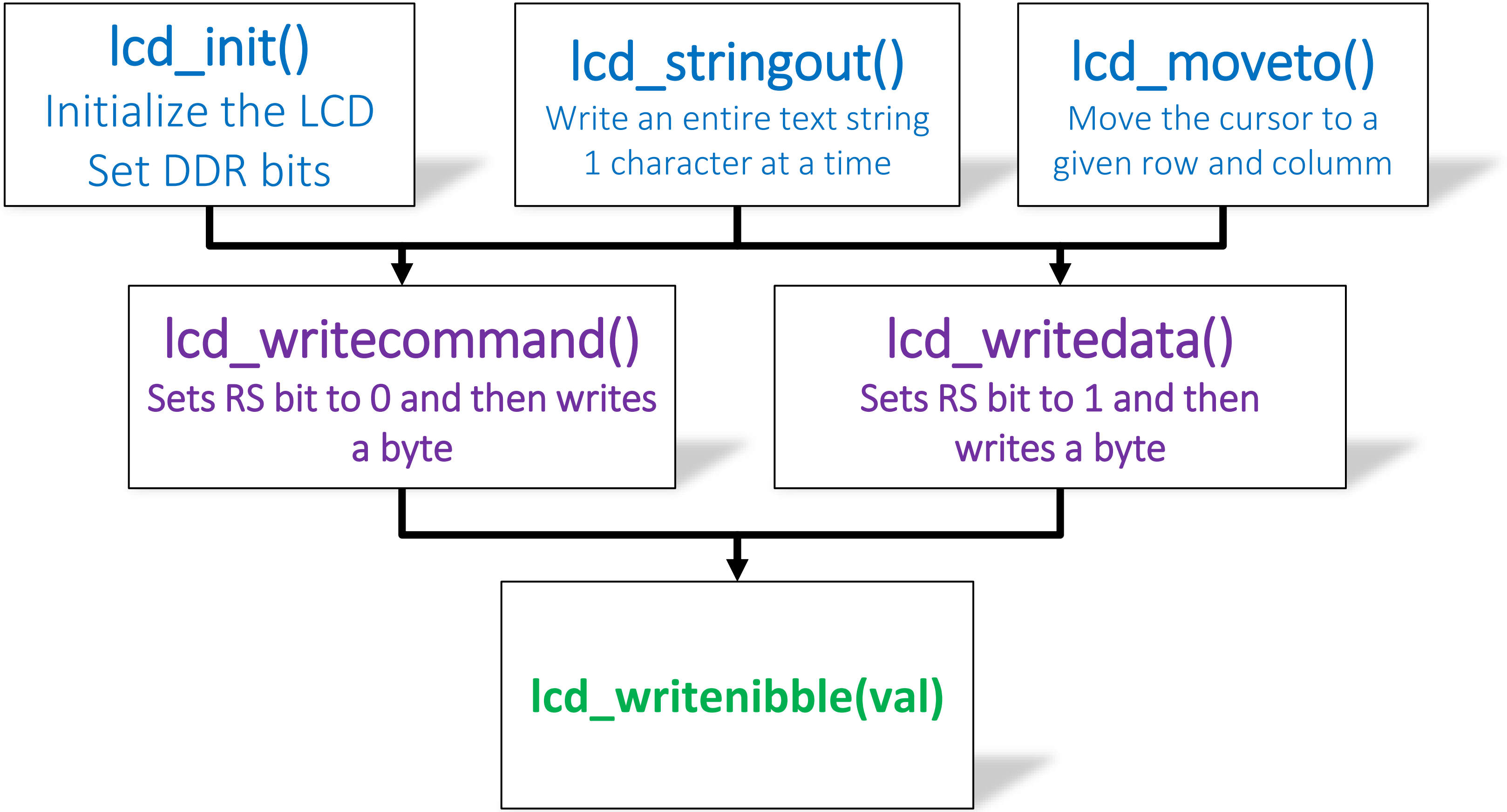
Stopwatch Lab

- Let's design a stopwatch (0.1s units)
- Inputs
  - Buttons for
    - Start/stop
    - Lap/reset
  - Timer interrupt
- Outputs
  - LCD [XX.Y format]
  - XX=seconds,
  - Y=Tenths of a second



This Photo by Unknown Author is licensed under CC BY-SA

Code Organization



lcd_init()
Initialize the LCD
Set DDR bits

lcd_stringout()
Write an entire text string
1 character at a time

lcd_moveto()
Move the cursor to a
given row and columm

lcd_writecommand()
Sets RS bit to 0 and then writes
a byte

lcd_writedata()
Sets RS bit to 1 and then
writes a byte

lcd_writenibble(val)

Character Strings

- Last lab you only output single characters ('0'-'9') but now let's learn how to print out entire strings that may contain numeric or other variable values
- A string in C is just a character array terminated with a null character ('\0' or 0)
  - Writing a string in double quotes ("EE 109") automatically appends the null character
- Printing out a string means printing one character at a time until we hit the null character

Addr: Index:	520 [0]	521 [1]	522 [2]	523 [3]	524 [4]	525 [5]	526 [6]
str1:	'E'	'E'	' '	'1'	'0'	'9'	'\0'

Computer Memory

```
void lcd_stringout(char *str)
{
    int i = 0;
    // Loop until NULL character
    while (str[i] != '\0') {
        // Send the character
        lcd_writedata(str[i]);
        i++;
}   }
```

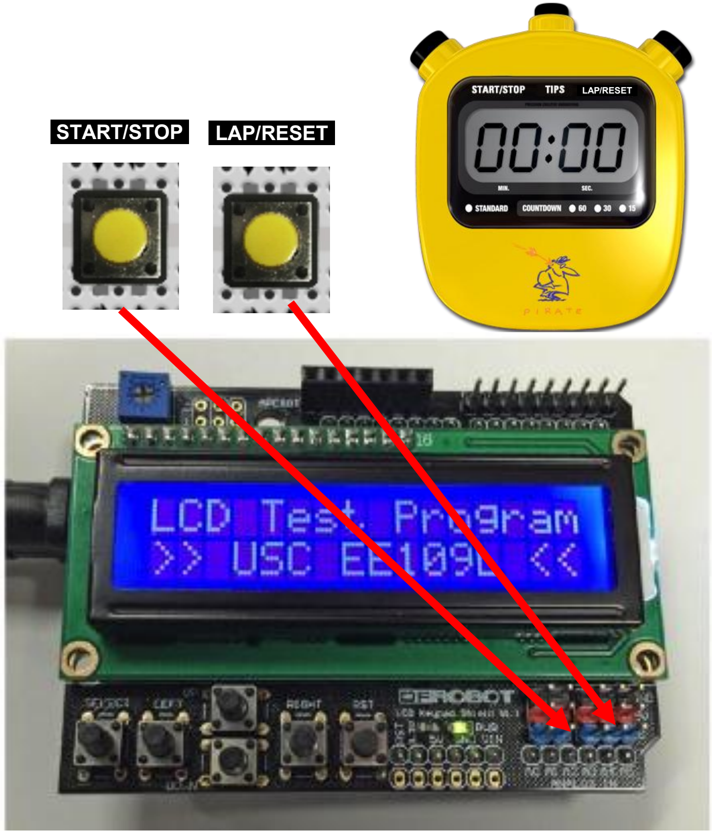
lcd_stringout & lcd_moveto

- lcd_stringout() prints the string starting wherever the cursor is currently located
- It does NOT wrap to the next line
  - You must ensure strings are of the appropriate length to fit on the screen
  - You must use lcd_moveto to position the cursor before printing a string

WATCH APPENDIX C: TIMER VIDEO

Stopwatch Lab

- Question:
  - What do I need state for in this design?
- Answer:
  - Anytime you provide the same input and different outputs/actions occur, there is state inside
  - Different actions for same button press



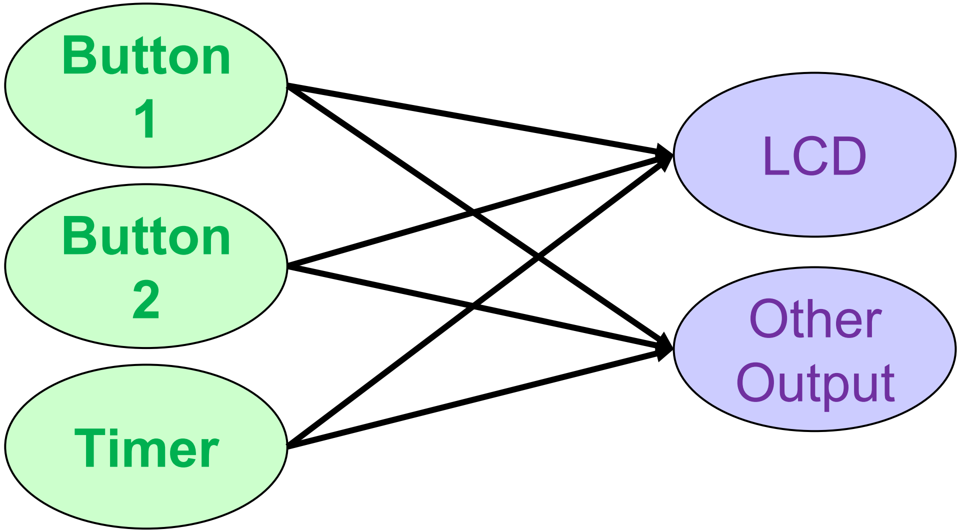
This Photo by Unknown Author is licensed under CC BY-SA

Stopwatch Lab

USING STATE MACHINES TO SIMPLIFY & ORGANIZE DESIGNS

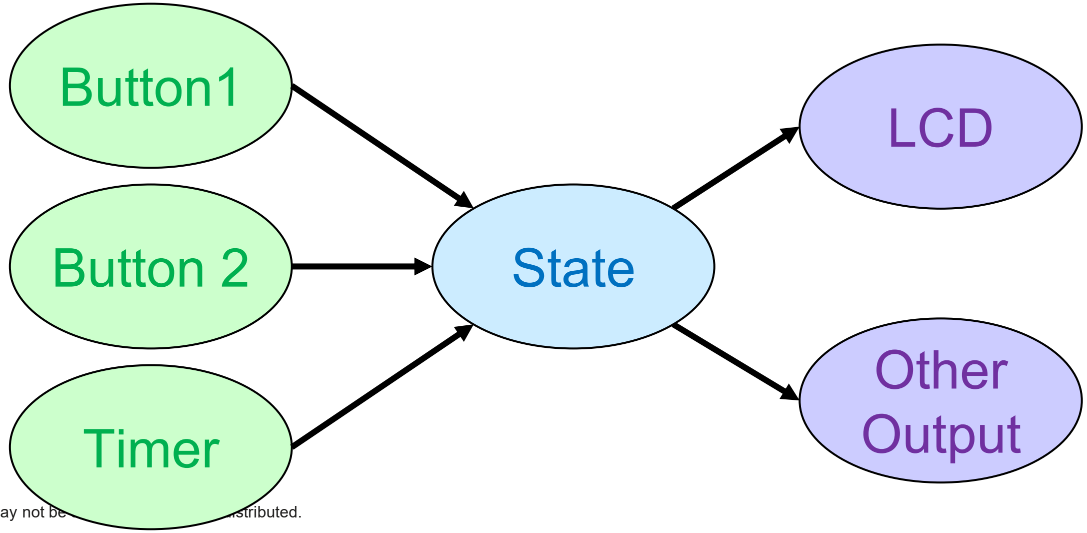
Why Use State Machines

- It can be very hard/difficult to design a system where all the inputs can affect each of the outputs (i.e. an all-to-all relationship)
  - If **m**-inputs & **n**-outputs then all-to-all => **m*n** cases to account for



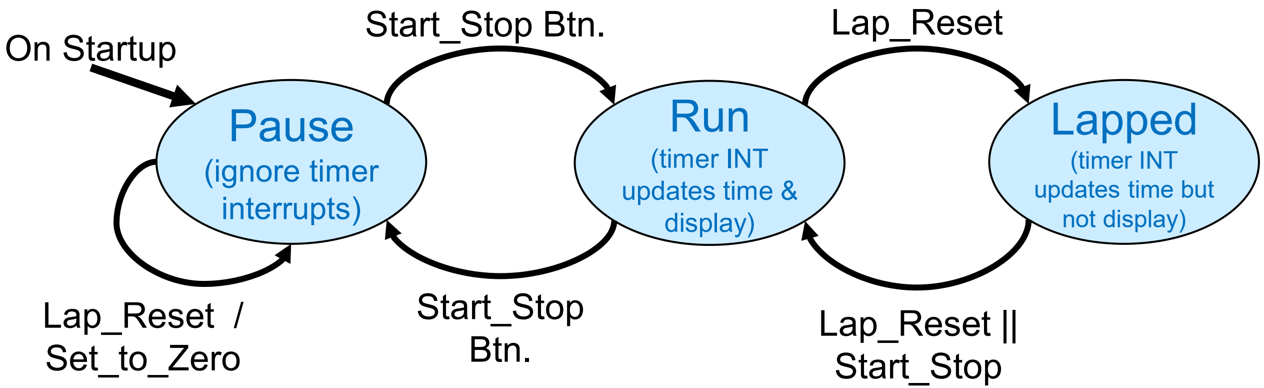
Why Use State Machines

- Easier to decouple relationship between input and output
- Let inputs update state, then examine the state to decide what outputs should be or do => **m+n** cases to account for
- Similar to the popular MVC GUI & Web app design approach
  - Model->View->Controller (MVC design)
  - **Model (State)**, **View (Output)**, **Controller (Input)**



Stopwatch Application

- What states do we need to differentiate button presses



- When timer interrupt occurs examine the state to decide how to update the display (or just leave current displayed time)
- What else in this design is technically "state"?
  - **Time format: `SS.Tenths`**
  - Every time the timer interrupts check to see if time needs to update & increment the time if necessary

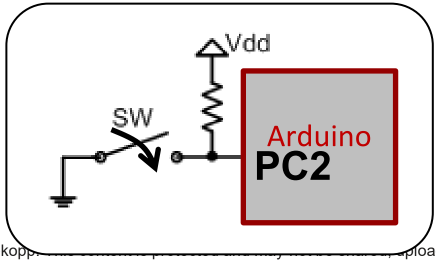
DEBOUNCING AND WHEN AN ACTION TAKES PLACE

Counting Presses

- **Recall**: If we simply check a button each time through our main loop, our code is fast enough to sense a single press as MANY presses.
- **But remember, we can WAIT THROUGH a press by looping until it is released!**

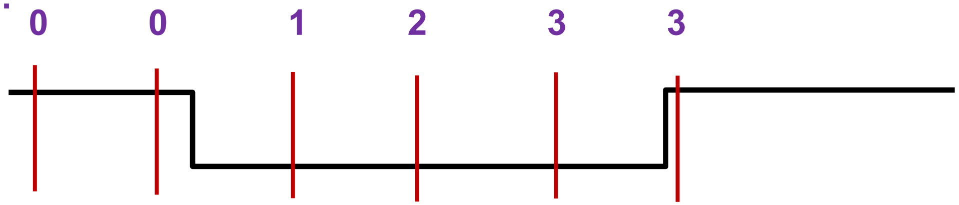
Option 1

```
#include <avr/io.h>
int main()
{
   PORTC |= (1 << PC2);
   int cnt = 0;
   while(1){
       char pressed = (PINC & 0x04);
       if( pressed == 0 ){
           cnt++;
       }
   }
   return 0;
}
```



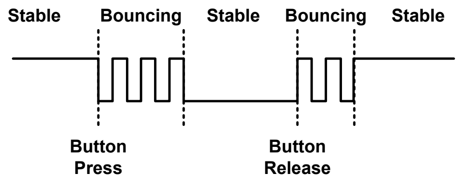
Option 1a:
cnt

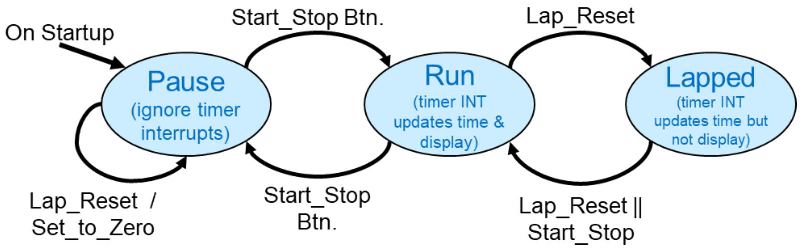
PC2



Why Waiting & Debouncing Are Necessary

- When I press start/stop the system should make a SINGLE transition to the opposite state
  - If we don't handle bouncing and waiting through a press, what could happen?




On Startup
Pause (ignore timer interrupts)
Start_Stop Btn.
Run (timer INT updates time & display)
Lap_Reset
Lapped (timer INT updates time but not display)
Lap_Reset / Set_to_Zero
Start_Stop Btn.
Lap_Reset || Start_Stop

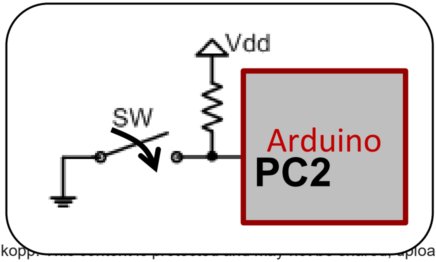
Act on Press or Release

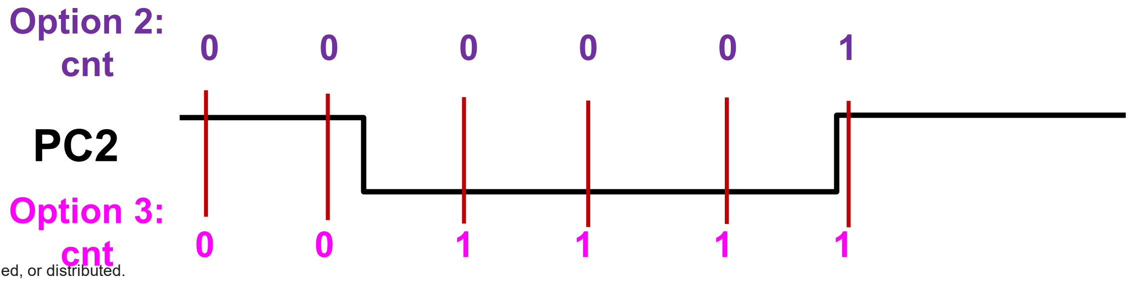
```
#include <avr/io.h>
int main()
{
   PORTC |= (1 << PC2);
   int cnt = 0;
   while(1){
      char pressed = (PINC & 0x04);
      if( pressed == 0 ){
         while( (PINC & 0x04) == 0 ) {}
         cnt++;
      }
   }
   return 0;
}
```

Option 2: cnt incremented on RELEASE

```
#include <avr/io.h>
int main()
{
   PORTC |= (1 << PC2);
   int cnt = 0;
   while(1){
      char pressed = (PINC & 0x04);
      if( pressed == 0 ){
         cnt++;
         while( (PINC & 0x04) == 0 ) {}
      }
   }
   return 0;
}
```

Option 3: cnt incremented on PRESS





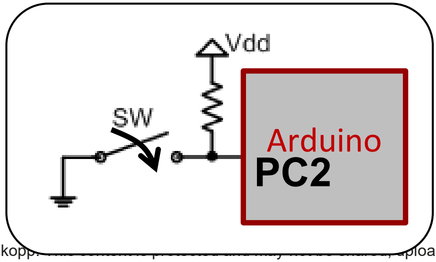
Connection to our Stopwatch

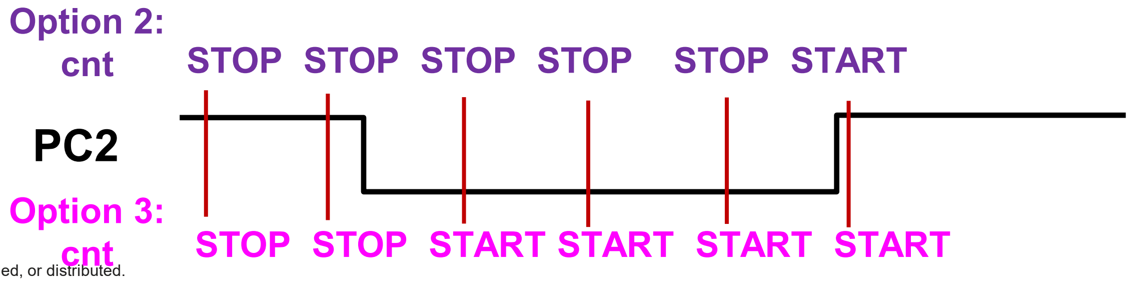
```
#include <avr/io.h>
int main()
{
   PORTC |= (1 << PC2);
   int cnt = 0;
   while(1){
      char pressed = (PINC & 0x04);
      if( pressed == 0 ){
         while( (PINC & 0x04) == 0 ) {}
         // update state AND
         // start timer
      }
   }
   return 0;
}
```

Option 2: Start timer on release

```
#include <avr/io.h>
int main()
{
   PORTC |= (1 << PC2);
   int cnt = 0;
   while(1){
      char pressed = (PINC & 0x04);
      if( pressed == 0 ){
         // update state AND
         // start timer
         while( (PINC & 0x04) == 0 ) {}
      }
   }
   return 0;
}
```

Option 3: start timer on press





Option 2: cnt

STOP STOP STOP STOP STOP START

PC2

Option 3: cnt

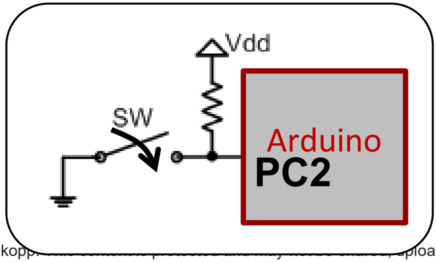
STOP STOP START START START START

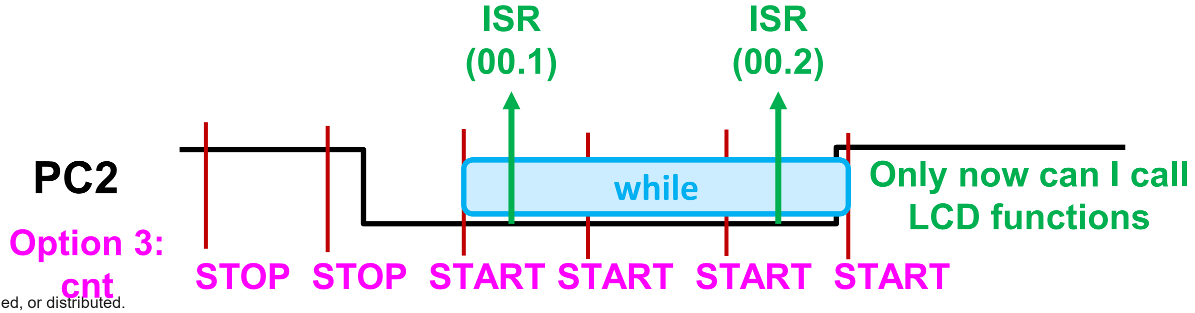
Connection to our Stopwatch

Option 3: start timer on press

```
#include <avr/io.h>
int main()
{
   PORTC |= (1 << PC2);
   int cnt = 0;
   while(1){
      char pressed = (PINC & 0x04);
      if( pressed == 0 ){
         // update state AND
         // start timer
         while( (PINC & 0x04) == 0 ) {}
      }
   }
   return 0;
}
```

Even if I update state and start the timer BEFORE WAITING FOR RELEASE and ISR's start triggering during the press, we will get stuck in the while loop waiting for release and WON'T BE ABLE to update the LCD with the new time.



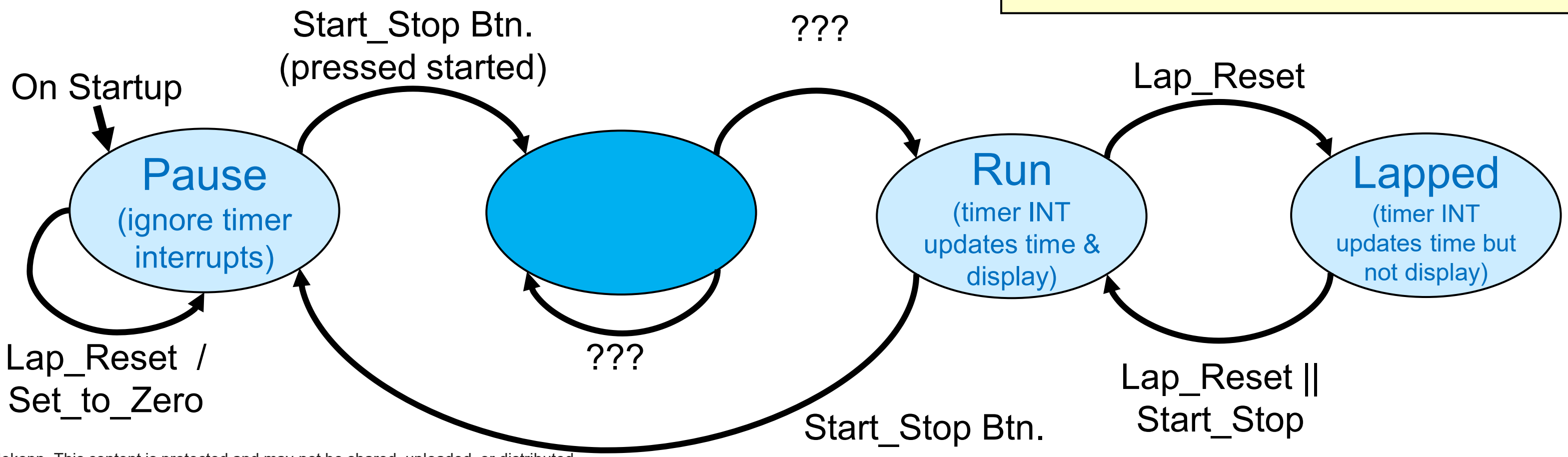


Only now can I call LCD functions

Adding More States

- Remember in our state machines lecture we said let's avoid other loops and just use 1 main while loop and STATE + COUNTERs!

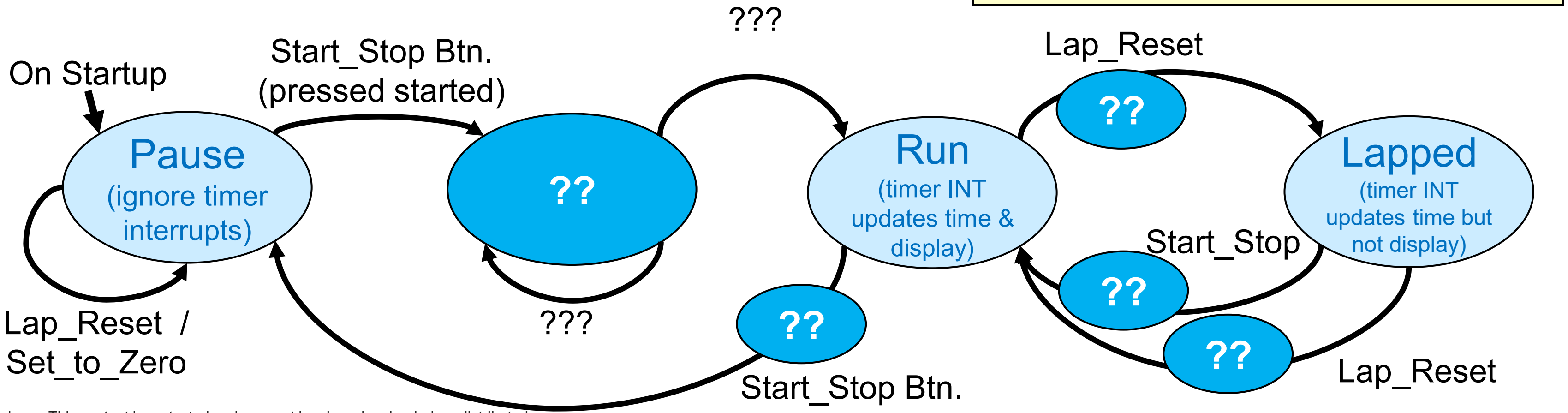
```
#include <avr/io.h>
int main()
{

    while(1){
       ...
       char pressed = (PINC & 0x04);
       if( pressed == 0 ){
          // update state AND
          // start timer
          while( (PINC & 0x04) == 0 ) {}
       }
    }
    return 0;
}
```



Even More States

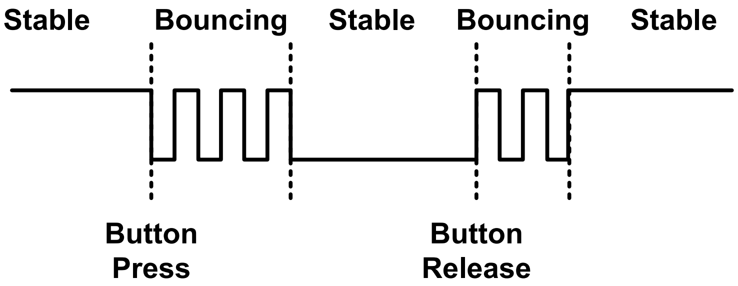
- Can the same problem happen when we press STOP during the RUN state?
  - Should we stop the timer and change states BEFORE or AFTER the release?
  - Is getting stuck in a while loop detrimental?
- What about the Lap Button when going into or out of the **Lapped** state.

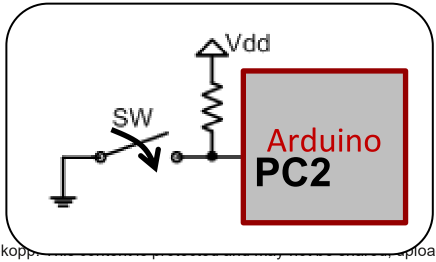
```
#include <avr/io.h>
int main()
{

    while(1){
        ...
        char pressed = (PINC & 0x04);
        if( pressed == 0 ){
            // update state AND
            // start timer
            while( (PINC & 0x04) == 0 ) {}
        }
    }
    return 0;
}
```

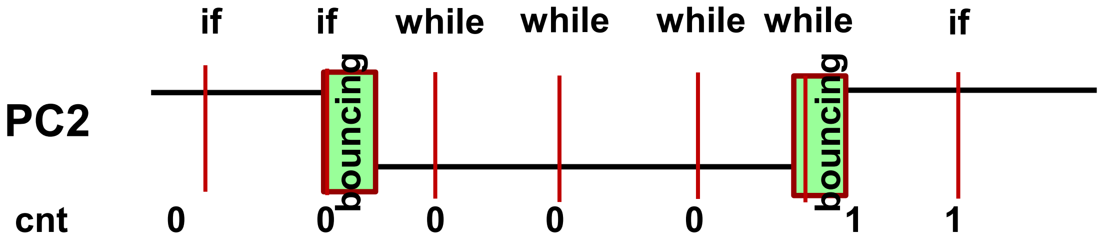


Interfacing Mechanical Switches/Buttons

- Recall that mechanical buttons bounce.
- But we said those bounces can ONLY occur for a short time (1-5 ms) after we press or release the button.
- If we simply set out delay around the main while loop to be more than 5 ms, we should not need to worry about bouncing.


Stable
Bouncing
Stable
Bouncing
Stable
Button Press
Button Release


Vdd
SW
Arduino
PC2


if
if
while
while
while
while
if
PC2
bouncing
bouncing
cnt
0
0
0
0
0
1
1