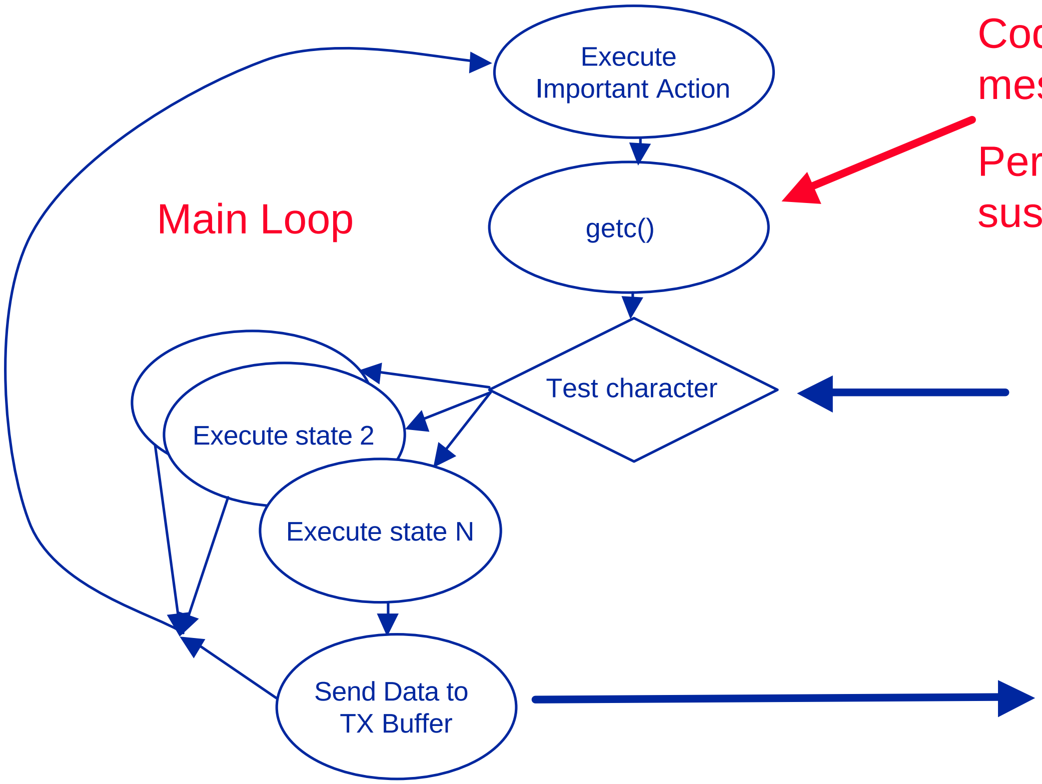
# RS232 - In Line

- **PUTC / GETC / PUTS / GETS**
  - **Functions to allow passing information back and forth to PC via serial**
  - **putc, getc, puts, gets are blocking functions**
  - **OK for simple code flow**



Main Loop

Execute Important Action

getc()

Test character

Execute state 2

Execute state N

Send Data to TX Buffer

Code flow stops here until a message is received from user

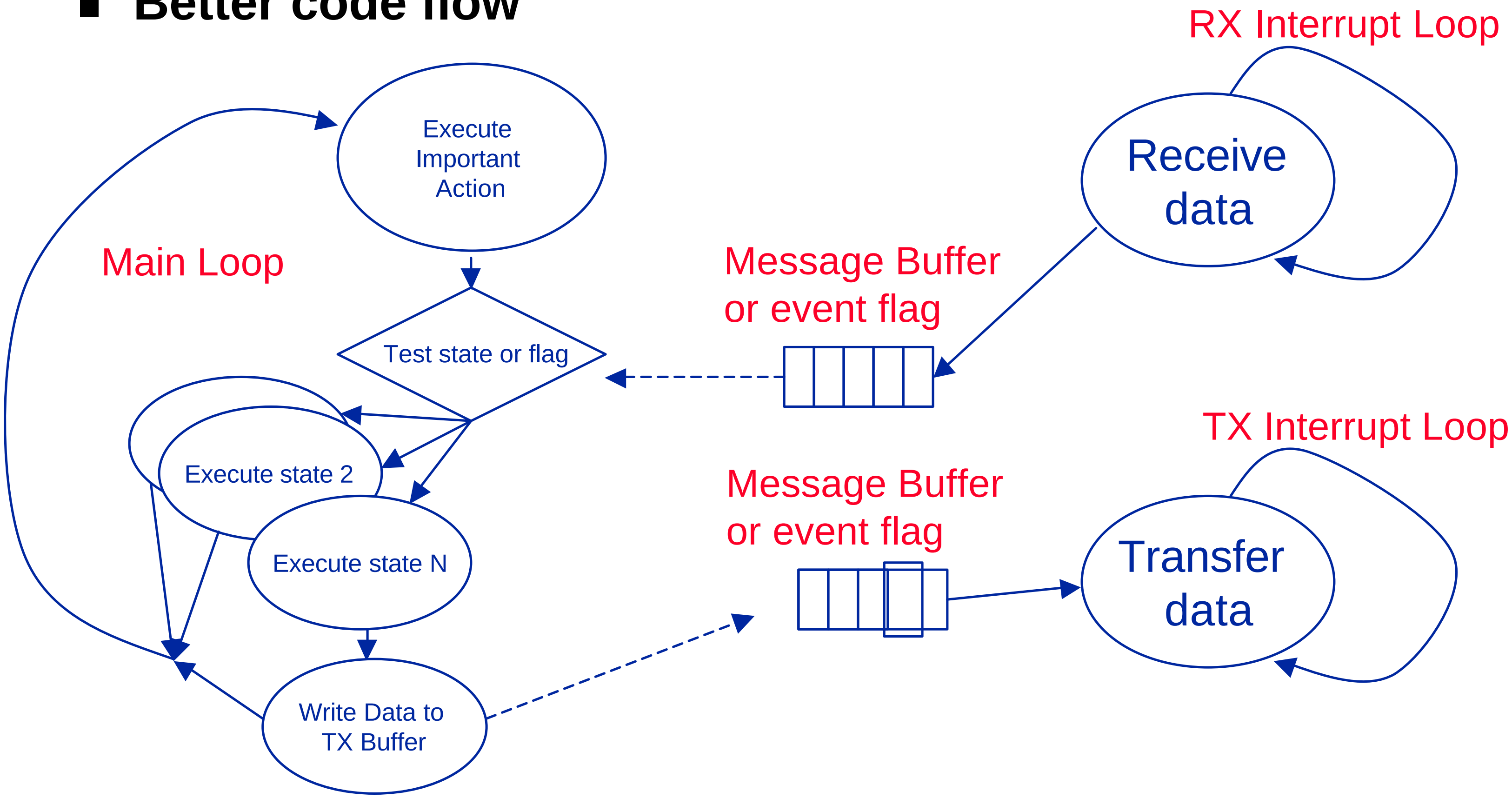
Periodic action (like sensing) is suspended

User input (keyboard)

User display (screen)

# RS232 - Interrupts

- **RX / TX Interrupt**
  - **Better way to handle communications**
  - **Interrupts handle monitor of comms channel**
  - **Better code flow**



# RS232 - Hardware

- ## Comms Code
  - **Utilize both RX and TX interrupts**

Define RX, TX pins to hardware

```
#use rs232(baud=4800, xmit=PIN_C6, rcv=PIN_C7)
```

Define RX, TX Buffers and ptrs to head and tail

```
byte r_buffer[R_BUFFER_SIZE]; // receive buffer
byte r_head;  // head of the queue
byte r_tail;  // tail of the queue
byte t_buffer[T_BUFFER_SIZE];  // transmit buffer
byte t_head;  // head of the transmit queue
byte t_tail;  // tail of the transmit queue
```

`HandleCharacter(rxbyte);`
Function that appends new character to message string and tests whether it is complete

TX interrupt allows next byte to be sent as soon as previous byte clears TX

```
#int_tbe t_handler() {
  if(t_head == t_tail) disable_interrupts(INT_TBE);
  else {
    putc(t_buffer[t_tail]);
    t_tail++;
    if(t_tail == T_BUFFER_SIZE) t_tail = 0;
  }
}
```

RX interrupt - signals when new byte is in receive buffer. Byte passed to state machine to concatenate and test

```
#int_rda receive_handler()  {
  byte rxbyte;
  rxbyte = getch();
  HandleCharacter(rxbyte);
}
```

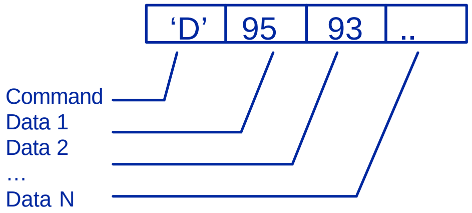
Appending characters to transmit buffer (called faster than info being sent)

```
void send_byte(byte txbyte) {
  t_buffer[t_head] = txbyte;
  t_head++;
  if(t_head == T_BUFFER_SIZE) t_head = 0;
  enable_interrupts(INT_TBE);
}
```

# RS232 - Interrupts

- **Receive Data Interrupt**
  - **Collect bytes from serial line**
  - **Allows complete message to come in before acting**
  - **Byte order sets state machine**

## Example (Set Motors)

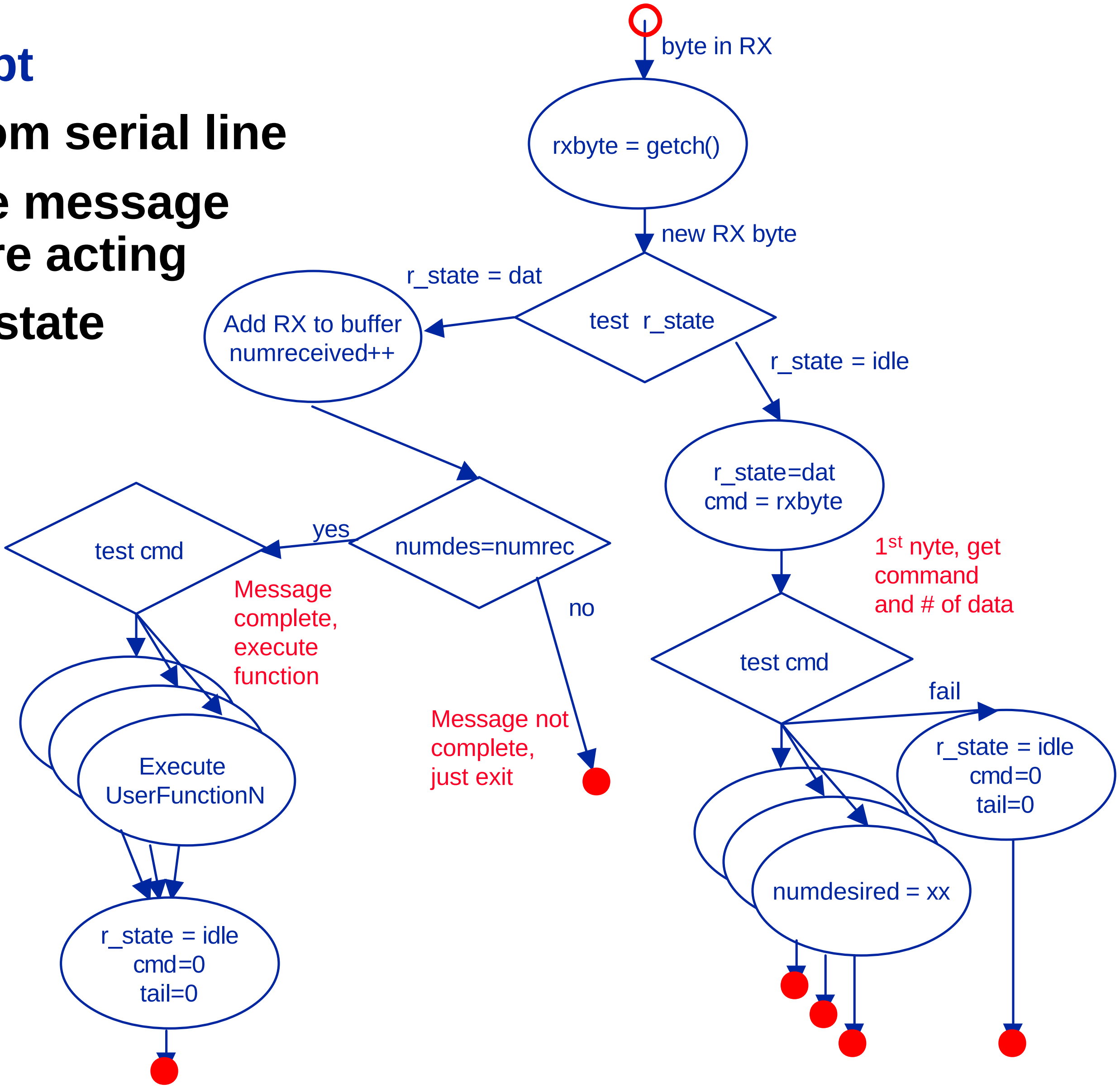


## States

r_state = {idle, dat}

## State Variables

byte r_state, cmd  
byte numdesired, numreceived  
byte r_buffer[size]  
byte r_tail, r_head



# RS232 - Interrupts

- **Receive Data Interrupt**
  - **Simple State Machine**

```
#int_rda receive_handler() {
  byte rxbyte;
  rxbyte = getch();
  if(r_state == idle){
    r_state = dat;
    cmd = rxbyte;
    numreceived = 0;
    switch ( cmd ) {
      case 1:
        numdesired = 2;
        break;
      default:
        r_state = idle;
        break;
  }
  else{
    numreceived++;
    r_buffer[r_head]=input;
    r_head++;
    if(numreceived == numdesired){
      switch ( cmd ) {
        case 1:
          dat1 = r_buffer[head+0];
          dat2 = r_buffer[head+1];
          userFunction1(dat1,dat2);
          break;
        case 2:
          dat1 = r_buffer[head+0];
          userfunction2(dat1);
        . . .
      r_state = idle;
      numdesired = numreceived = 0;
    }
    else {
      //
}}}
```

cmd can range from 0-255 (ex ‘A’ = 67)

Separate case for each user command

Default to reset on unknown command

User functions should return quickly. Typically change parameter (like pwm) or set state flag and let main do the work

Separate function for each user command. Number of arguments a function of command

get the byte from the RX register

1st byte is command byte Also determines number of bytes to follow

Queue all incoming data

Once we have all data, run the appropriate function. Data for function is collected in buffer

Reset state machine

If we don’t have the entire message wait

# State Machine Operation

## ■ Example State Machine (team 2)

Main Loop (state machine)

```
while(1) {
  switch (recieved) {
    case 'f':
      while(input(PIN_B2)) {
        driveforward(950);
        delay_ms(1);
      }
      motorstop();
      break;
    case 'm':
      while(input(PIN_B2) && input(PIN_B1)) {
        driveforward(1000);
        delay_ms(1);
      }
      motorstop();
      break;
    case 'n':
      drivebackwards(1000);
      delay_ms(250);
      printf(" nb ");
      recieved = ' ';
      motorstop();
      break;
    case 'p':
      stepperdown(100);
      break;
    case 'y':
      dispense();
      received = ‘n’
      break;
    case 'o':
      pancake();
      break;
    default:
      break;
    printf("ticks");
  }
}
```

State variable is ‘received’

‘received’ set by PC via serial comms

or set as part of state machine

Common actions called as functions

Serial Comms Interrupt

```
#int_RDA RDA_isr() {
  recieved = getc();
  putc(recieved);
  switch (recieved) {
    case 'u':
      servotime = 190;
      break;
    case 'r':
      servo3time = 0;
      break;
    default:
      break;
  }
}
```

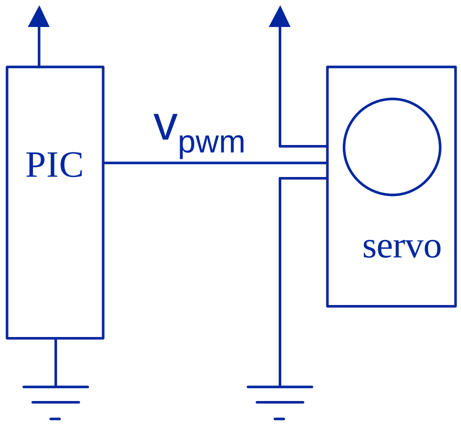
Example Function

```
void driveforward(long duty) {
  set_pwm1_duty(duty);
  output_low(PIN_D0);
  output_high(PIN_D1);
}
```

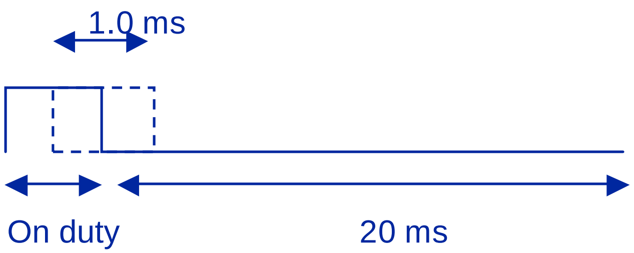
*** extracted from team 2**

# Servo Options

- Explicit Timing Loop
- PWM
- Timer Interrupt



| $t_{on}$ | angle |
|---|---|
| 1.0 ms | 0º |
| 1.5 ms | 90º |
| 2.0 ms | 180º |



# Servo Options

- **Explicit Timing Loop**
  - **Servo signal timing done via delays in main loop**
  - **Important actions done in interrupts**

```
void main() {
  set_tris_a(0);          // specify port A as outputs
  set_tris_b(0b00010000); // specify PIN B4 as input
  port_b_pullups(TRUE);   // use the PIC pull-up resistors of port B
  delay_ms(100);

  output_high(PIN_A1);    // when we start the PIC, make a LED blink
  delay_ms(500);
  output_low(PIN_A1);
  delay_ms(500);
  output_high(PIN_A1);
  delay_ms(500);
  output_low(PIN_A1);
  delay_ms(500);

  while(TRUE) {           // loop processed until the PIC powered down
    switch (state) {
      case 1:
        for(i=0;i<100;i++) {
          output_high(PIN_A3);
          delay_us(800);
          for(j=0;j<10;j++){
            delay_us(servoposition);
          }
          output_low(PIN_A3);
          delay_ms(18);
          i++;
        }
   }
}
```

LED test to verify proper operation

Apply new position (multiple cycles needed for transient)

delay an initial 800us

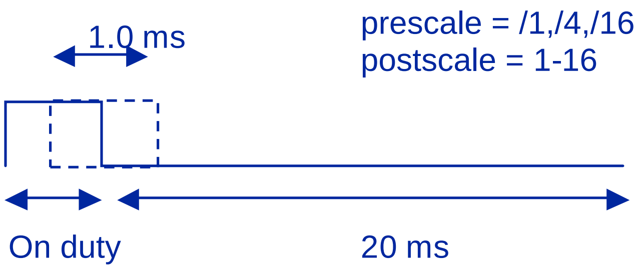
delay_us accepts 0-255 (variable). Need loop to get 1ms resolution

servoposition set by PC (in interrupt) or state machine

*** extracted from team 8**

# Servo Options

- **PWM**
  - **Simple**
  - **Dedicated hardware**



```
setup_ccp1(CCP_PWM);
setup_timer_2(T2_DIV_BY_16,78,16);
enable_interrupts(INT_RDA);
enable_interrupts(global);

void main() {
  set_tris_a(0);          // specify port A as outputs
  set_tris_b(0b00010000); // specify PIN B4 as input
  port_b_pullups(TRUE);   // use the PIC pull-up resistors of port B
  delay_ms(100);

  while(TRUE) {           // loop processed until the PIC powered down
    switch (state) {
      case 1:
        set_pwm1_duty(servoposition);
      }
   }
}
```

Timer period = 20ms

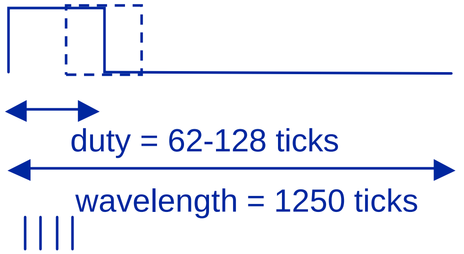
‘servoposition’ and ‘state’ set by PC
(in interrupt) or state machine

Period = 20ms
Resolution = 20ms / 255 = 78us
Effective range =
    1ms = 12 counts
    2ms = 24 counts
    => 12 out of 255

# Servo Options

- **Timer Interrupt**
  - **Better resolution**
  - **Simple state machine**

timer interrupts every 16us



duty = 62-128 ticks

wavelength = 1250 ticks

phase = # timer interrupt ticks

state machine for system

```
#int_timer2 void timer2_isr() {
  switch (state) {
    case STATE_READY:   // Wait 2 seconds, pick up pancake
      if (ticks >= 100) {
        ticks = 0;
        enter_state(STATE_HOLDING);
      }
      break;
    case STATE_HOLDING: // Wait 1 sec, flip pancake
      if (ticks >= 50) {
        ticks = 0;
        enter_state(STATE_FLIPPED);
      }
      break;
    case STATE_FLIPPED: // Wait 3 sec, pick up pancake
      if (ticks >= 150) {
        ticks = 0;
        enter_state(STATE_READY);
      }
      break;
  }

  if ((phase < duty) && (pwm_status == 0)) {
    pwm_status = 1;
    output_high(PIN_C7);
  }
  else if ((phase >= duty) && (pwm_status == 1)) {
    pwm_status = 0;
    output_low(PIN_C7);
  }
  phase++;
  if (phase == wavelength) {
    phase = 0;
    ticks++;
  }
}
```

state machine for servo

phase counts number of interrupts

duty set by PC or state machine

pwm_status tracks state (on/off)

set servo pin high (phase<duty)

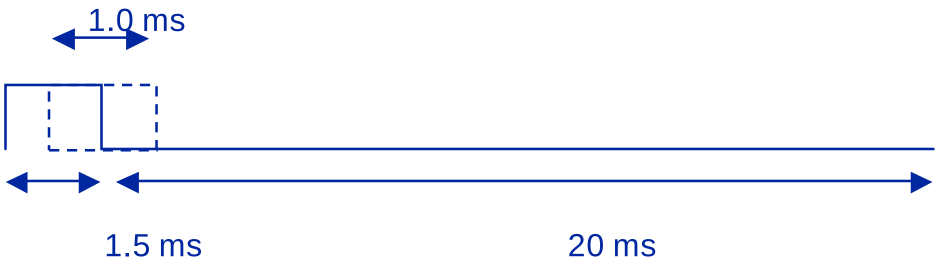
set servo pin low (phase>duty)

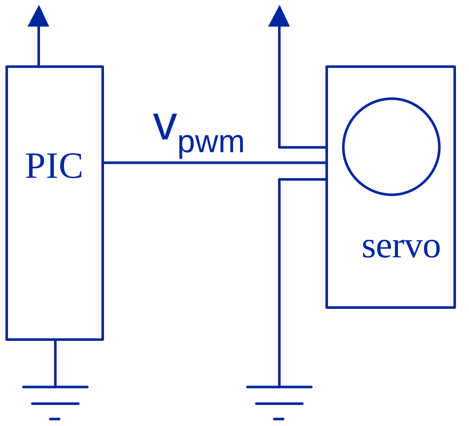
reset phase every 1250 interrupts

*** extracted from team 5**

# Servo Options

## Multiple Timer Resolution





| $t_{on}$ | angle |
|---|---|
| 1.0 ms | 0º |
| 1.5 ms | 90º |
| 2.0 ms | 180º |

```
setup_timer_2 (1, n, 8)  - produces interrupt every (8*n)us  (n from 128-255)

#int_timer2 timer2_int_handler() {
  If(nextstate==servolow) {
    output_low(servopin);
    setup_timer(longtimeout);
    nextstate=servohigh;
  }
  else {
    output_high(servopin);
    setup_timer(shorttimeout);
    nextstate=servolow;
  }
}
```

set next timer interrupt to be long, fixed (20ms), low resolution

set next timer interrupt to be short, variable (1-2ms), high resolution

Question? – can setup_timer() be called in function