Arduino Mega 2560 Light and Shade Controls

Assumptions:

- Arduino Mega or clone used –
- Shade and Light control program on separate Megas – not combined on one
- Try to maximize number of out pins and in pins on each version – lights and shades

Requirements Lights:

- LightOffTimer implementation:
- Configurable timer and switch to light mapping via serial interface and web –
- Configuration conserved across reset
- Time of Day timer functions
- All lights off or all lights on functions
  - Via switchpush counter and via mapping of dedicated switch
- Configurable to send light and switch status via serial and/or web interface
- NTP client
- Webserver
  - HTTP
- UDP – used for initiation of registration of Arduino with Raspy
  - Broadcast
  - Multicast

Arduino Mega 2560 Light and Shade Controls

Assumptions:

- Arduino Mega or clone used –
- Shade and Light control program on separate Megas – not combined on one
- Try to maximize number of out pins and in pins on each version – lights and shades

Requirements Shades:

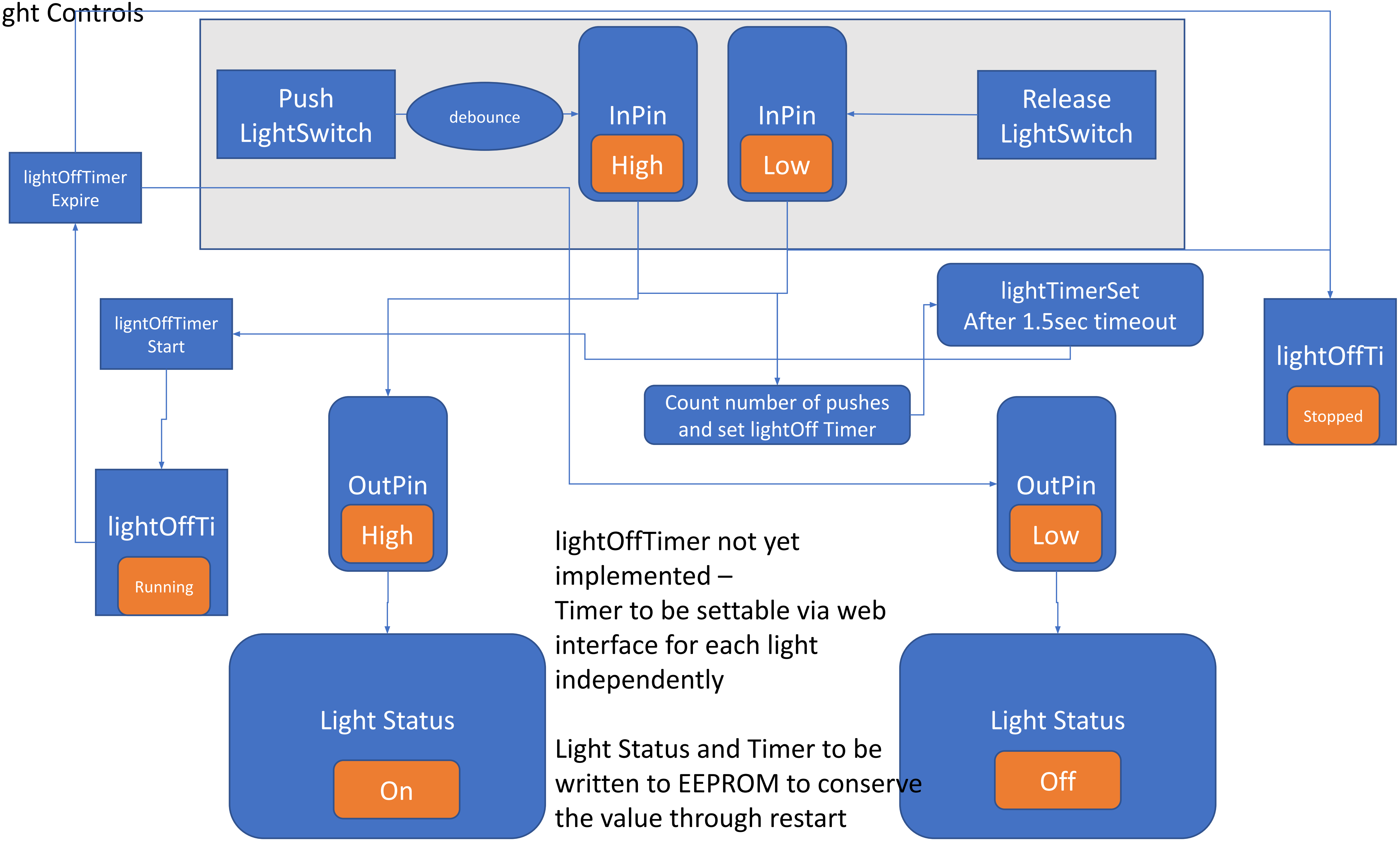
- shadeUpTimer and shadeDownTimer implementation: **critical to make sure up and down relays are never on at the same time**
- Configurable timer and switch to shade mapping via serial interface and web –
- Configuration conserved across reset
- Configurable to send shade and switch status via serial and/or web interface
- NTP client
- Webserver
    - HTTP
    - Heartbeat monitoring from Raspy – with configurable timeout – initially 30 minutes
- UDP – used for initiation registration of arduino with Raspy
    - Broadcast
    - Multicast

Arduino Mega 2560 Light and Shade Controls

Future Development:

- Arduino web interface consolidation on Raspy
  - Configuration storage in db on Raspy
  - ArdID storage on Raspy
  - Authentication controlled by Raspy through web interface
  - Tunneling out to cloud from Raspy
- Registration Procedure
- Secure communication – potentially implementation of chacha encryption if possible
  - Thoughts about this – maybe a second Arduino is needed just for the comms?
- Physical interfaces
  - Input I/O protection against overload and EMF etc.
  - Output I/O protection against overcurrent
    - Transistors?
    - Optical isolation?

Light Controls



lightOffTimer not yet implemented –
Timer to be settable via web interface for each light independently

Light Status and Timer to be written to EEPROM to conserve the value through restart

Light Controls

Startup/Setup

- check for all timer values in EEPROM – if there use those if not use hardcoded default values
    - lightTimer[numOfLights]
    - Light timer to be configurable during runtime – by webserver or serial interface or via switch as below
        - If light switch is pressed quickly [y] number of times then light timer is set to y*15minutes
- check for Light Status for each light in EEPROM and set appropriate outPin – cycling through all pins and initializing – put a delay between each so that you don’t generate too much current at the same time
- Intitialize all needed global variables
- Initialize all inPins and outPins
- Check for device ID in EEPROM – if none then starting first time – make firstTimeStart variable True
- Check for Mac Address in EEPROM – if default then make defMac variable True
- Setup Mapping of inPin[x] -> outPin[x] – many to one (I haven’t done this yet – not sure how)
    - This should be configurable – so not constants but changeable during runtime
- Read timeOfDayOn[] timeOfDayOff[] for each light – default is always on – to be made configurable by the web server or serial interface

Light Controls

Maybe below is better in the Startup/Setup phase

Main Loop

- Check if first start – if ArdID is not default then not starting first time –
  - If NOT first time
    - Go to **Program Loop**
  - If first start init NTP, Webserver
    - Look for Raspy by sending out multicast or broadcast
    - Check for message from Raspy initiating registration
    - Raspy answers with raspyID and ArdID
      - Set ardRegWithRaspy = True
    - Raspy ID is written to EEPROM
    - ArdID is written to EEPROM

Light Controls

Program Loop

- Check status of all InPin[x]
  - If status is high –
    - Check lightStatus[y] where y is the light that is manipulated by this particular inPin[x] based on the mapping
      - If lightStatus is On
        - go into routine to turn off light
      - If lightStatus is Off
        - go into routine to turn on light

- Check NTP and update clock
- Check inbound ethernet interface and HTTP
- Check connectivity to Raspy if exists and if registered – if ardRegWithRaspy = True
- Check for request for configuration page on HTTP
  - Go to function to process inbout webpage
  - Go to function to process response
- Back to start of **Program Loop**

Light Controls

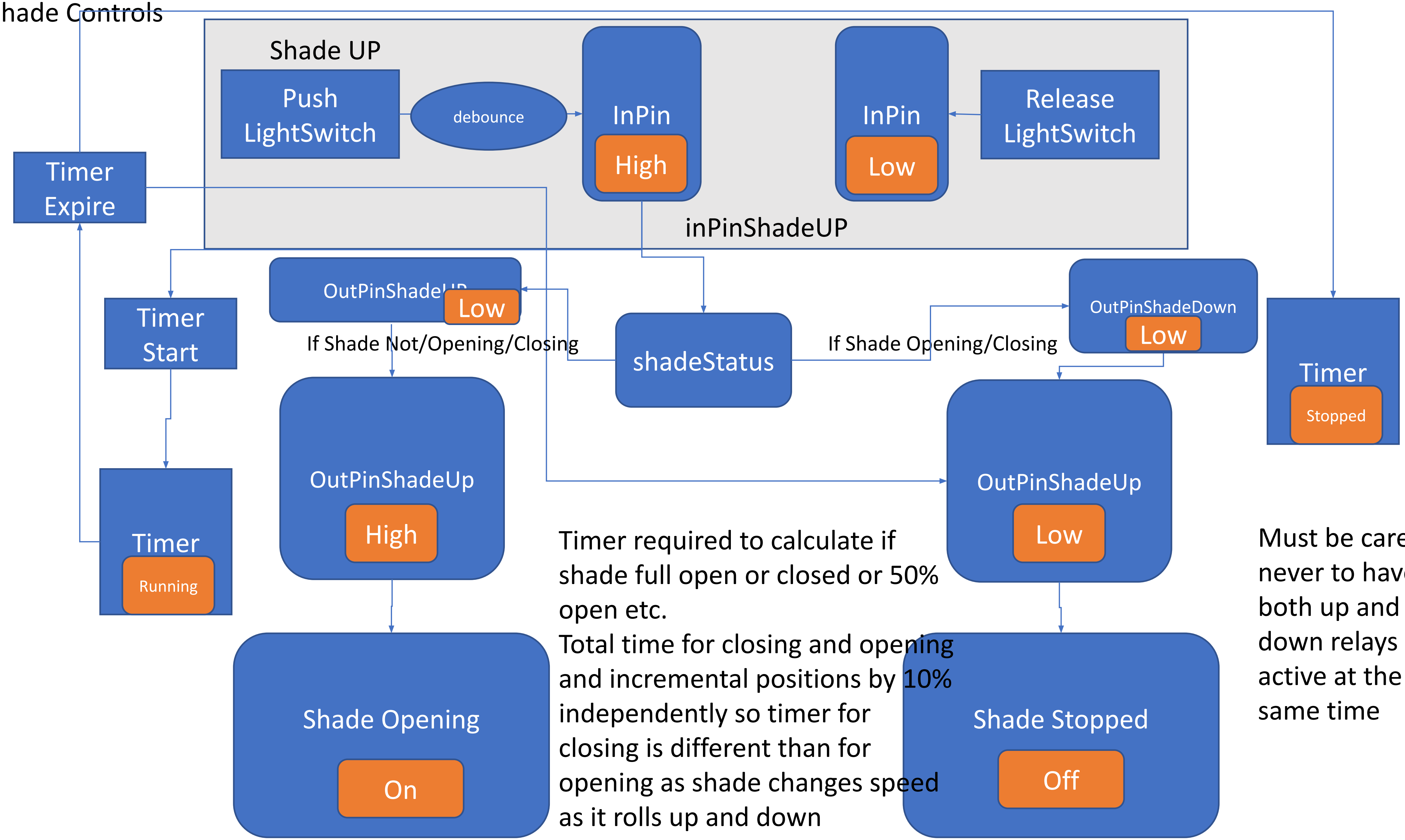
fnTurnOnLight(x)

- Debounce Timer – settable but currently 200ms
- Count number of times pressed in z time in sec (z to be configurable) 1.5sec initially thinking
  - If only once then use lightTimer[x] value as gotten when initialized
  - If more than once-
    - Start light timer based on above z*x where x = integer(minutes) or equivalent millis()
- Set outPin[x] to value to turn on light – may be high or low depending on config
- Set lightStatus[a] per config mapping

fnTurnOffLight(x)

- Debounce Timer – settable but currently 200ms
- Stop light timer
- Set outPin[x] to value to turn off light depending on config
- Set lightStatus[a] per config mapping

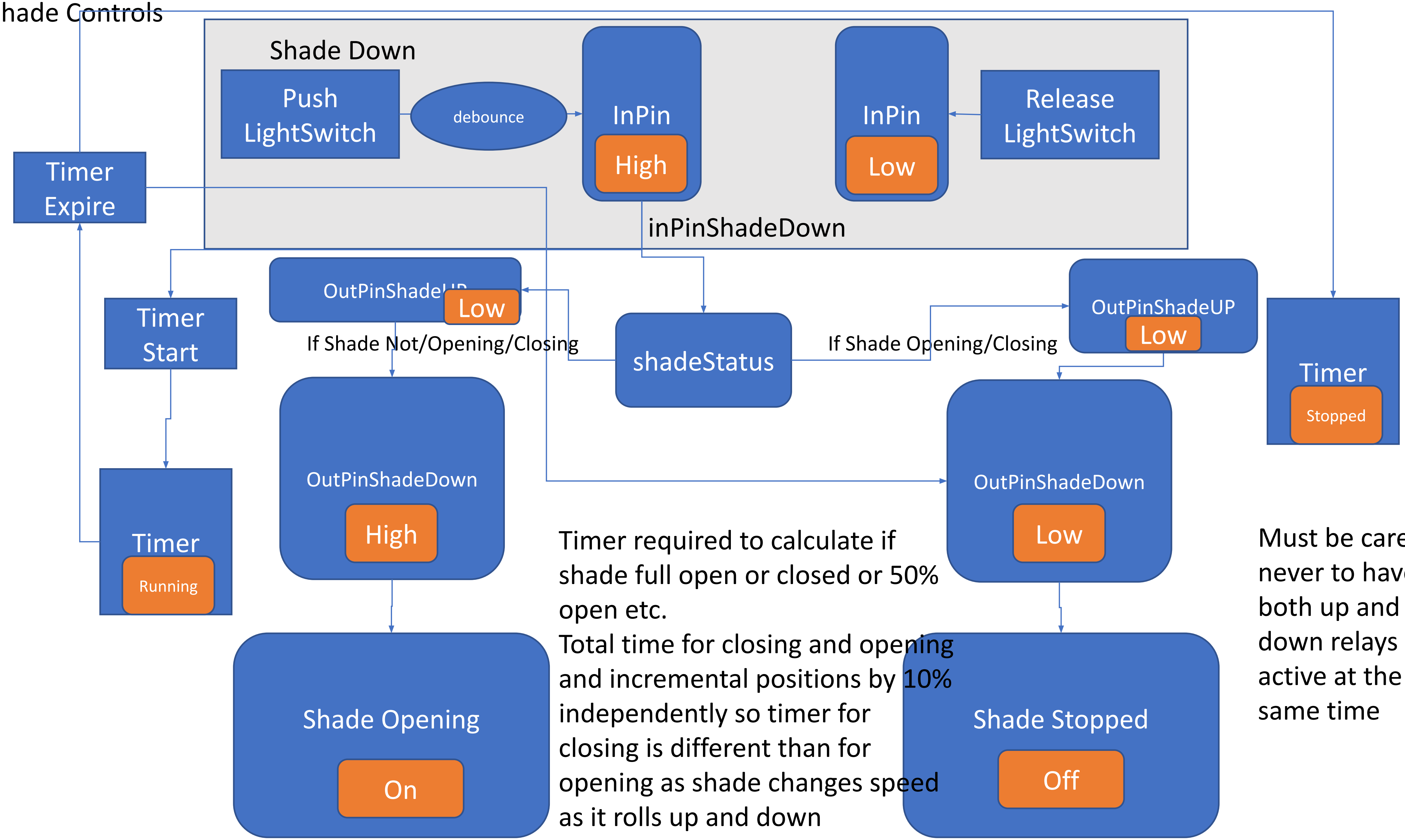
Shade Controls



Timer required to calculate if shade full open or closed or 50% open etc.
Total time for closing and opening and incremental positions by 10% independently so timer for closing is different than for opening as shade changes speed as it rolls up and down

Must be careful never to have both up and down relays active at the same time

Shade Controls



Timer required to calculate if shade full open or closed or 50% open etc.

Total time for closing and opening and incremental positions by 10% independently so timer for closing is different than for opening as shade changes speed as it rolls up and down

Must be careful never to have both up and down relays active at the same time

Light Controls

Startup/Setup

- check for all timer values in EEPROM – if there use those if not use hardcoded default values
    - shadeTimerUP[numOfShades]
    - shadeTimerDown[]
    - shadeStatus[numOfShades]
    - shade timers to be configurable during runtime –
        - By web interface or via serial interface
- check for Shade Status for each Shade in EEPROM and set appropriate outPin – cycling through all pins and initializing – put a delay between each so that you don't generate too much current at the same time
- Intitialize all needed global variables
- Initialize all inPins and outPins
- Check for device ID in EEPROM – if none then starting first time – make firstTimeStart variable True
- Check for Mac Address in EEPROM – if default then make defMac variable True
- Setup Mapping of inPin[x] -> outPin[x] – many to one (I haven't done this yet – not sure how)
    - This should be configurable – so not constants but changeable during runtime
- Read timeOfDayStatus[] for each shade– default is always half open – to be made configurable by the web server or serial interface

Light Controls

Maybe below is better in the Startup/Setup phase

Main Loop

- Check if first start – if ArdID is not default then not starting first time –
  - If NOT first time
    - Go to **Program Loop**
  - If first start init NTP, Webserver
    - Look for Raspy by sending out multicast or broadcast
    - Check for message from Raspy initiating registration
    - Raspy answers with raspyID and ArdID
      - Set ardRegWithRaspy = True
    - Raspy ID is written to EEPROM
    - ArdID is written to EEPROM

Light Controls

Program Loop

- Check status of all InPin[x]
  - If status is high –
    - Check lightStatus[y] where y is the light that is manipulated by this particular inPin[x] based on the mapping
      - If lightStatus is On
        - go into routine to turn off light
      - If lightStatus is Off
        - go into routine to turn on light
- Check NTP and update clock
- Check inbound ethernet interface and HTTP
- Check connectivity to Raspy if exists and if registered – if ardRegWithRaspy = True
- Check for request for configuration page on HTTP
  - Go to function to process inbout webpage
  - Go to function to process response

- Back to start of **Program Loop**

Light Controls

fnShadeUp(x)

- Debounce Timer – settable but currently 200ms
- Count number of times pressed in z time in sec (z to be configurable) 1.5sec initially thinking
    - If only once then use lightTimer[x] value as gotten when initialized
    - If more than once-
        - Start light timer based on above z*x where x = integer(minutes) or equivalent millis()
- Set outPin[x] to value to turn on light – may be high or low depending on config
- Set lightStatus[a] per config mapping

fnShadeDown(x)

- Debounce Timer – settable but currently 200ms
- Stop light timer
- Set outPin[x] to value to turn off light depending on config
- Set lightStatus[a] per config mapping