Transistor-Transistor Logic (TTL) Pulse setup

To synchronize time across both the Picamera system and the Data Acquisition System (Neuralynx Digital Lynx SX), we programmed Arduino Uno to send multiple TTL pulse to both the systems to mark the **START RECORDING** timestamp. The Data Acquisition System TTL Input-Output operates at 0 V - 5 V as compared to 0 V - 3.3 V operating range for the Raspberry Pi. An Arduino Uno generated the TTL square pulse at 0V low and 5V high on one of its IO pin which was fed directly into the Data Acquisition System and stepped-down to 3.3V before feeding into the 8 Raspberry Pi.

Hardware Setup

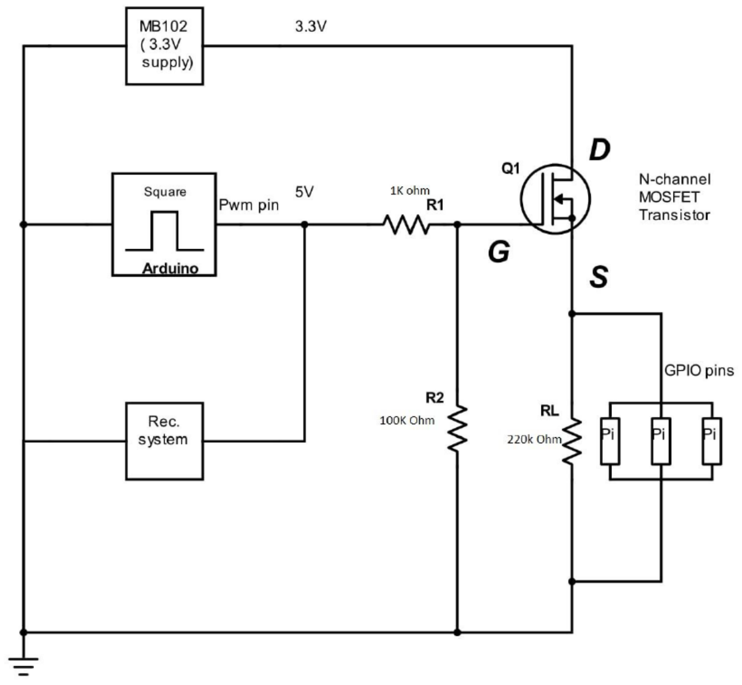
A logic level N-channel Enhancement MOSFET was used (2N7000) for stepping down the voltage from 5 V to 3.3 V to send the TTL pulse from Arduino to the Raspberry Pi. Circuit details:

- The output from the Arduino General Input Output pin (GPIO) is connected to the gate (G) of MOSFET
- MB102, a 3.3 V power supply was (https://www.petervis.com/Raspberry_PI/Breadboard_Power_Supply/YwRobot_Breadboard_Power_Supply.html) connected to the drain (D)
- source (S) is grounded.
- R2 resistor was used to discharge the gate as there is no current through the gate. A large valued resistor RL was used to get maximum voltage drop across it, and 8 Raspberry Pi were connected parallel to it.

How the circuit works

When $V_{DS} < V_{GS} – V_{t,}$ where V_t is the threshold voltage, condition is satisfied, the FET operates in the ohmic region. During the TTL ON state, the DS can be replaced by 5 ohm (2N7000 datasheet) resistor for analysis. Therefore, in the following circuit: $V_{GS} = 5 * (R2/R1+R2) => V_{GS} = 4.95$ V, $V_t = 0.8$ V (datasheet), $V_{DS} > 3.3$ V and $V_{DS} < 4.15$ V. This satisfies the $V_{DS} < V_{GS} – V_t$ equation making the FET work in the ohmic region.
For G = OFF, there was no drain source current, and for G = ON, voltage across RL and the Raspberry Pi was 3.297 V using voltage divider analysis. Thus, the Raspberry Pi GPIO input

ranges from 0 V - 3.297 V (operating range for the Raspberry Pi) for the TTL pulse. Note that the 3.3 V power supply, MB102 has rating of 700 mA at 3.3 V and our circuit draws only 0.5 mA from MB102.



Using the circuit, we were able to send signals simultaneously to 8 Raspberry Pi and the Data Acquisition System at their respective voltage ratings. The operating voltages are at their peak values; thus, it can sustain large noises.

Software Setup

Arduino Code Arduino was programmed to send pulse with the following signature: 3 second HIGH, 2 second LOW, 3 second HIGH, 2 second LOW followed by a continuous HIGH/LOW transition each of width 1 second.

```
void setup() {
   pinMode(12, OUTPUT);
   digitalWrite(12, HIGH);   // turn the LED on (HIGH is the voltage level)
   delay(3000);                      // wait for a second
```

```
  digitalWrite(12, LOW);   // turn the LED off by making the voltage LOW
  delay(2000);                      // wait for a second
  digitalWrite(12, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(3000);                      // wait for a second
  digitalWrite(12, LOW);   // turn the LED off by making the voltage LOW
  delay(2000);                      // wait for a second
}

void loop() {
  digitalWrite(12, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(12, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}
```

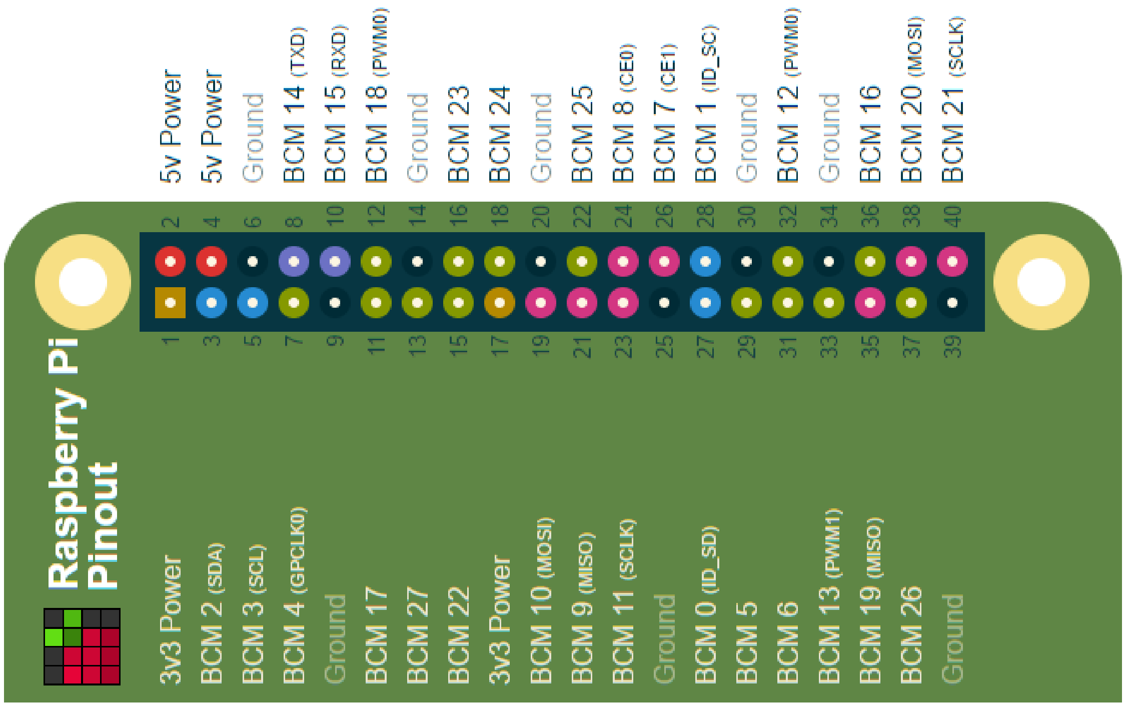
Raspberry Pi Code

TTL timestamps logging code snippet was involved in either of the main video recording code (***StartAcquisition_pigpio.py*** or ***StartAcquisition_gpio.py*** **in PicameraVideoAcquisitionScripts**). We used 'interrupts'/ 'event-detect' methods which allows detecting to events at each GPIO pin by using the following function:

```
GPIO.add_event_detect(channel,event,callback=my_callback,bouncetime=timeinmillisecs)
```

available from the library which enabled us to easily calculate the GPIO pin value and timestamp at each TTL pulse transition, removing the need to continuously poll the input data. In the add_event_detect function call, *channel* should be an input pin number (Note that this depends on the configuration type used: BCM or GPIO). The various events available are *GPIO.RISING* for the rising edge (when the signal changes from LOW to HIGH), *GPIO.FALLING* for the falling edge (when the signal changes from HIGH to LOW), and *GPIO.BOTH* for both rising and falling edges. The *my_callback* method contains the interrupt handling code, and will be executed as a separate thread. *bouncetime* is the minimum amount of time required between two events. If two events occur in succession, within the *bouncetime* limit, then the second event will be ignored.

Figure lists the individual pin (40 in number) available on the Raspberry Pi:



The table lists the GPIO pin each Pi is connected to in our setup (Note: there are multiple available pin on individual Pi. Our choice for a pin was decided randomly).

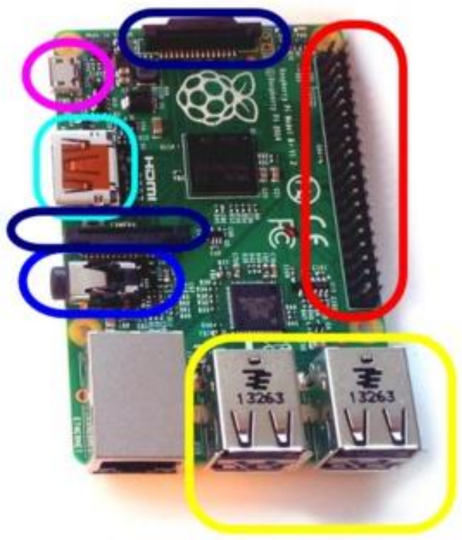
Camera Number	**Channel Number**
Cam 1	GPIO 7, BCM 4
Cam 2	GPIO 26, BCM 7
Cam 3	GPIO 33, BCM 13
Cam 4	GPIO 7, BCM 4
Cam 5	GPIO 7, BCM 4
Cam 6	GPIO 7, BCM 4
Cam 7	GPIO 7, BCM 4
Cam 8	GPIO 7, BCM 4
Cam Rec Room	GPIO 33, BCM 13

Libraries used for logging TTL timestamps on Raspberry Pi

RPi.GPIO (https://pypi.python.org/pypi/RPi.GPIO):

i. Oldest and most popular
ii. Has better version in RPIO.GPIO (which has improved hardware timed PWM suitable for LEDs and servos)
iii. time accurate to ~20 microsecs.

Force Raspberry Pi 2 Model B video output to Zmodo multi-channel system



To ensure that each Picamera system is working properly and recording videos efficiently during the behavior session, we used Zmodo multi-channel system to view the Picamera video output getting recorded and simultaneously keep an eye on animal’s behavioral states.

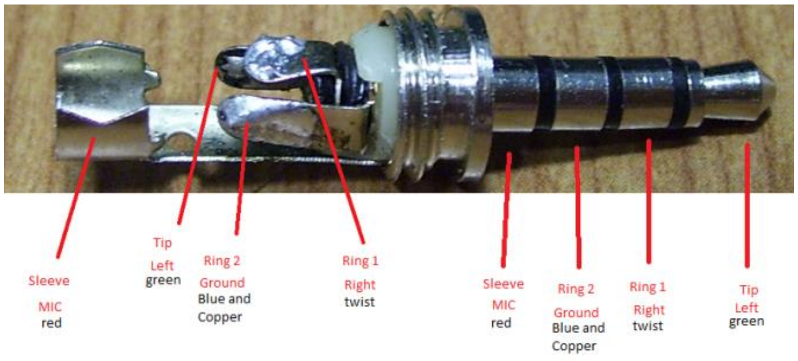
Video output from the Picamera system is driven via a four-pole AV (Audio Video) jack like the type used on many camcorders (shown in the light blue color in the figure). By default, when we boot into Raspbian OS, it outputs the video to HDMI. Since Zmodo does not have HDMI supported TVs or even HDMI cables, only option is to use the AV cables. The AV cables can be connected to composite video output of the Raspberry Pi. Rather than using traditional composite cable that one gets in the market (RCA cables with video through yellow and stereo audio through white and red), we used the BNC cables along with 4 conductor Tip Ring Ring Sleeve (TRRS).

The spatial and temporal resolution of the composite port display and the video being saved on each Picamera system are independent of one another, but the lower quality output of the composite port is sufficient for online visual monitoring of the rat behavior.

Hardware Setup to view recording on Zmodo system:

- For the Raspberry Pi end, one needs to be careful with the connections. Because of a **configuration mismatch** in the Pi, the core of the BNC cable (that goes into the ZModo system) needs to be connected to the main body, and the ground wire needs to be connected to the wing that is continuous with ring 2 of the jack.

- Connect core to the sleeve (main body) and the shield goes to the part continuous with Ring 2. One can easily test for continuity using a multimeter.



- The end which gets plugged in to ZModo has normal wiring. The core goes into the core of BNC and shield to the body of BNC.

Software Setup

- Remote login to the raspberry Pi and add the following lines in /boot/config.txt

```
# uncomment for composite PAL
sdtv_mode =2 # Different modes for NTSC & PAL
hdmi_ignore_hotplug = 1
sdtv_aspect = 1
```

- Also comment (by adding #) to the line `hdmi_force_hotplug = 1`

- Reboot the Picamera.

All the configurations are done. This should get the output on monitor connected to the Zmodo system. Note that the video should be set to either in NTSC or PAL or SECAM format according to the encoding used in the country (**PAL** for India), otherwise there will be just flicker on the monitor.

Cables from each Picamera were connected to the ZModo channels in the shown order to ensure consistency throughout.

Camera Number	**Channel Number**
Cam 1	Channel 8
Cam 2	Channel 7
Cam 3	Channel 6
Cam 4	Channel 5
Cam 5	Channel 4
Cam 6	Channel 3
Cam 7	Channel 2
Cam 8	Channel 1

Sample output: