Control Systems – By Chris Cole, BMBC Member

Control Systems are devices used to control machines accurately and repeatedly (without getting bored.) They can do wonderful and mysterious things, often with very little external input. For example, from very simple devices to control a fridge light to be only on when the door is open, to very powerful processor controllers for ships engine control - starting, stopping paralleling and reversing engines, large refrigeration compressor plants for cooling in gold mines, or big centrifugal air compressor anti-surge, flow and pressure control. P*rovided* the software, the number and logic crunching part, is good and adequately tested, the control works accurately and continuously. Depending on the systems importance, then failsafe systems also need to be considered and it is useful to have tested them before "going live."

The electronic control systems used to operate plant and equipment are referred to as Process Logic Control (PLC) and is where a dedicated "box" takes in information from the outside world, and using the "instructions" built into the box, often in the form of the software, then it sends out to the outside world, things for it to do. For example a moisture probe is connected to the controller, which works out what the electrical value from the probe means, the measured value, then compares this to a set point and decides if there is a difference, (i.e. moisture has been detected) and sends a signal to a bilge pump to start pumping out the water, till the probe no longer senses the water level in the boat.

To develop a control system :-

- Decide what it is going to do.
- Put in as much detail as possible
- Find out how it will do it, in sections if necessary, (build up the sequence,)
- Put it all together
- Test it modify and refine where it doesn't do it
- Check that it does what you originally wanted, (does it matter if it doesn't? Maybe the result is better?)
- Put it into operation.

When designing a control system, many things are considered, including why do we need any control system, how far does the automation go, and can it be done better, more accurately, repeatedly and cheeper, by any other methods? The more automatic the system becomes, the less those working around it, will understand of what is actually happening, and so in cases of problems, they will have little or no knowledge of what to do to resolve those problems, and their interference may even make the problems worse!

Arduino

The "Arduino" is the name of the family of micro-controllers, MCU, developed to teach, and be used to control all manner of things, like lights, buzzers, ticket machines, data retrieval systems and so on. They are able to take in or measure signals or inputs, and run some dedicated software instructions repeatedly, and then output signals. Others are the Basic Stamp, PICAXE or MICROCHIP.

There are other small similar devices out there like the Raspberry Pi, but these are actually micro computers, and are better suited to running multiple programmes, and are often connected to not much more than keyboards, mice and screens. Good for text, displays, number crunching, internet actions and the like.

The Arduino family is many and various, with currently the best starting point for learning the system is the UNO R3. It has many input and output pins, a USB connection for programming, and power supply connections. The MEGA has many more input and outputs, and the NANO is a condensed version of the UNO. The hardware and software is "open source." That is the information on what goes into them is freely available and openly shared. The technology is no

secret, and so many different manufacturers are out there including SparkFun and Elegoo. The software language is a combination of “C” and “C++” programming languages which are very powerful, but can be used at the learning end, to do a lot quite simply. Sharing and adapting others programmes or “sketches” is common, and a good way to learn.

The Nano is not much bigger than an old style memory stick, (+-3/4” x 1 3/4”) with 14 digital input/output pins, runs at 5 volts but can take power inputs at 6 - 20v, a micro USB socket and strangely 8 *analogue* input pins! (On essentially a digital processor!)

The UNO R3 is about the size of a credit card, has 14 digital I/O, also 5v operating voltage, but an external power jack plug socket, and printer style USB socket, but only 6 analogue input pins.

The MEGA (2560 R3 or ADK) is larger than the UNO has 54 digital I/O pins, 16 analogue input pins, and much more processing power. The ADK can interface with android phones.

Programming the Arduino

There are some example “sketches” showing how I worked the following theories, along with a brief description of what each line of the “programming code” is doing. Alan has suggested that these will be put onto the club website, for perusal and use by those who are interested!

The Arduino programmes or “sketches,” as they are commonly known, are developed on an “APP,” called the Arduino IDE (Integrated Development Environment,) that runs on Windows or MAC computers. Here you can write, test and upload “sketches.” There are many libraries of other “sketches” available from Arduino, Elegoo, or even freelancer software writers on the internet. It can also be worked on Linux, but I understand that can be more fun.
With the “APP” open you can type away to your hearts content. Getting useful “sketches” means you need to follow some conventions, of the C or C++ programming languages, as suitable to the Arduino. There is a good ‘*Arduino for Dummies*’ book, which explains a lot.
The “sketches” are usually broken to 3 sections. 1st is where the “variables” or memory boxes, or pigeon holes, are declared. 2nd is where the setup or booting instructions are set, which only runs once, at power up or after a reset. The 3rd part is the bit that runs the sketch again and again, till something goes wrong, or when powering down.
Having written or modified your “sketch,” connecting the USB lead to the Arduino, and uploading to it, the “sketch” runs right away. If it is all connected up to the external components, then the lights and buzzers should be working, as programmed!

Draft plan - for lighting controls

- How do we get a signal out of the receiver and into the Arduino
- Can we interpret this signal, correctly and repeatedly
- How do we select which set/scenario we want
- How do we define a set of LED’s/outputs
- How do we switch on a set of LED’s
- Do we need to also switch off LED’s not required
- How do we get a signal out of the Arduino

I have 3 versions of the Arduino Nano, for 3 different boats:-

1. Arduino Nano with “RC Connection to Arduino,” a 2 switch link feeding the 10 channel sequencer, for the “*Hecht*”
2. Arduino Nano with “RC Connection to Arduino + RichTugLights” for the individual light control for the “*Richardson*” tug.
3. Arduino Nano with “RC Connection to Arduino + AzizTugLights” for the lighting and radar controls for the “*Aziz*” anchor handling tug.

1. For "*Hecht*"

To make the Nano work as 2 switches when connected to a RC receiver, and output 2 independent 'switch' signals to a 10 channel sequencer, we only need to connect :-

- a servo lead onto - ground pin 4 (GND), signal pin 8 (D5)
- 6 - 20v supply to - ground pin 29 (GND), positive to pin 30 (VIN) (7 to 12v is optimal.)
- switch leads to - SW1 positive pin 15 (D12), SW2 positive pin 16 (D13) and common ground from SW1 & 2 to either of the pin 4 or 29 (GND) which connect to the "10 channel sequencer board."

The programme or "sketch" is loaded from a Windows or MAC PC, using the USB connection.

2. Arduino for the "*Richardson*" tug

The link to the RC receiver is the same as the switch version, but here the Arduino Nano is programmed to sequence internally and output an 'ON' or 'OFF' signal to each LED directly.

3. Arduino for the "*AZIZ*"

A variation I have made is that used for the AZIZ where there are over 20 LED's and 2 radars. These can be grouped into 7 stages. They are:-

- All off
- All on
- Underway, i.e. navigating with radar
- Towing
- Anchored
- Restricted operations ie divers or anchor handling, but not making way
- Restricted operations but underway.

Some lights need to be on in several scenarios, like the port and starboard navigation lights, which are on when underway, but not when at anchor.
So I could collect bunches of LED's together, so that all in a group would go on together. Such a group is the 2 fore-deck, 2 rear-deck and 6 bulkhead walkway lights.
Now we have a plan for the "matrix" of what goes on in what scenario. This forms a matrix of 6 scenarios and 9 groups.

There is a 40mA limit for an Arduino output pin, and a max total load of 200mA, so by counting up each bulb, and radar motor, in a group, based on a maximum of 20mA per LED, we can determine the need for LED drivers and ballast resistors. The bulkhead walkway lights don't need to be bright, so they were paired in series with a 330Ω resistor, so taking a lower current through each bulb. I wanted the spot lights to be as bright as possible, and based on a 7.2v supply, and 20mA load, then chose a 220Ω resister per spot. All 3 spot lights then would go on together, taking 60mA, so a transistor driver amplifier is connected between the Arduino output pin and the spot light group. Altogether the estimated current load from all LED's and the 2 radar motors came to about 260mA. Several transistor driver's would be needed.

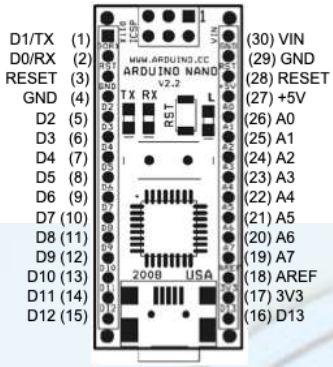
The lighting requirements are referenced from the "Victorian Recreational Boating Safety Handbook." P126 - 132

Finally

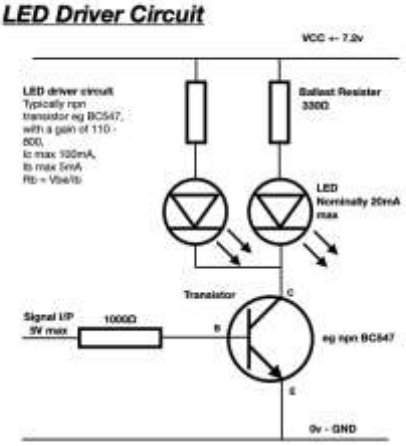
To quote the Arduino book, "the capabilities are only limited by your own imagination." The Arduino is only a small part of the range of control systems, but with a huge potential. It is not to be feared, as after all we are only controlling boats, and often at not very high speeds. So being adventurous is often rewarding.

Finally, I am happy and open to receive feedback, advise, corrections, questions and comments. I do not claim to be an expert, and always keen to learn. So please do come back to me at the pond-side or by email on these points.

Arduino Nano Pin Layout



LED Driver Circuit



The current 2019 internet prices for Nano's is in the range of £2 to £10. Original badged Arduino Nano's may be more expensive, @ £20.

An example “Sketch” for a Radio Control Connection to Arduino - “*Hecht*” Model Boat

So to get the signals from a radio control receiver to another device like an LED driver, or buzzer, we can use a simple “sketch” like the printed listing.

```
int ch1;
int selSW1=12;
int flipSW2=13;

void setup(){
  pinMode(5,INPUT);
  pinMode(selSW1,OUTPUT);
  pinMode(flipSW2,OUTPUT);}

void loop(){
  ch1=pulseIn(5,HIGH,35000);
    if(ch1>1650){digitalWrite(selSW1,HIGH);}
    if(ch1<1350){digitalWrite(flipSW2,HIGH);}
    if(ch1>=1350&&ch1<=1650){digitalWrite(selSW1,LOW);digitalWrite(flipSW2,LOW);}
  delay(100);}
```

Description

The 1st line says to setup a “pigeon hole” (memory) variable called ch1, and let it only have integers (whole numbers) stored in it. 2nd line likewise sets variable selSW1, also for integers but store in it “12” (the number 12.) 3rd line does the same for flipSW2 and put 13 in it.

The section beginning “*void setup(){ “* is the start of the one time setup instructions, that includes setting up in a pin mode digital input pin (D5) to be an Input, i.e. to look at D5 for voltage changes there. The next 2 lines do similar but as we had already said that variables selSW1 and flipSW2 are numbers 12 and 13 respectively, so the setup uses these values in the pin mode and says that digital pins D12 and D13 will become outputs, to be connected to the ‘outside world.’ This part of the programme only runs once. The end of the setup is closed with the other half of the brackets ‘*}’*

The ‘*void loop (){ ‘* will run again and again, for ever, and begins by taking variable ‘ch1’ and putting into it the resolution of the “pulseIn” formula; which is to look at the digital input D5, and every time that you see a “high” voltage step coming in there, count the time taken till the next high pulse, but don’t take longer than 35,000µs (micro seconds,) and put that time number into ‘ch1.’ Radio Control receivers work on what is called a pulse width modulation, which is a digital way of doing an analogue change. It sends out pulses that change in width, depending on the change of the position of the RC transmitter sticks. 1500µs (micro seconds,) is about mid position of the stick.
So if ‘ch1’ has got a value of greater than 1650 stored, the next line says to digitally write to the pin of the value in selSW1, 12, i.e. D12, a “high” or 5v or a logic 1. If the ‘ch1’ value was not above 1650 that line is ignored and the next line says, well if the value is less than 1350 then send to the pin of the value in flipSW2, 13, ie D13, a “high” or 5v or logic 1. However if the ‘ch1’ value wasn’t less than 1350 that line too is ignored, and the next line says that if ch1 is greater or equal to 1350, **and** less than or equal to 1650, then write to both pins relevant to selSW1 and flipSW2 a “low” or 0v or logic 0.
The delay(100) allows the system to take a deep breath for 100µs, before going round the loop again. The final ‘}’ tells the sketch to run the ‘void loop(){ ‘ again.
What happens here is that the loop has looked for a value coming into the Arduino, from the receiver, and if the pulse is above, below or in-between 2 values it changes the output values of digital pins D12 and D13. 100µs is enough for the external electronics to see these output values and respond accordingly. D12 & D13 both can be off, either on but never both on together. Pushing the transmitter stick up will put 5v on the D12 pin, then releasing to mid position, switches D12 back to 0v, and likewise pushing the stick down, puts D13 to 5v, till the stick is centred again. We can use 5v to drive an electronic logic circuit, like the “10 stage sequence LED circuit.”

This sketch only uses about 2% of the capacity of the Arduino, which shows how much more the Arduino can do, and also as we have only used 3 of the digital pins, and none of the analogues, there is plenty more possible.

Example of "RC Connection to Arduino + RichTugLights" - "*Richardson"* Model Tug

```
//RC connection to Arduino + RichTugLights
int ch2;
int ch3;
int ch4;
int buzzSW=11;
int smokeSW=12;
int selSW1=0;
int flipSW2=0;

const int ledPin = 13;       // the pin that the LED is attached to
const int lowestPin=5;
const int highestPin=10;    //only 6 outputs for 5 to 10
int pinCount=0;

// Variables will change:
int buttonPushCounter = 0;    // counter for the number of button presses
int buttonState = 0;          // current state of the button
int lastButtonState = 0;      // previous state of the button
int flipButtonState = 0;// current state of the button
int lastFlipButtonState = 0;     // previous state of the button
int ledStateofChannel = 0;
int ledState=LOW;
int ledStateF=LOW;
int counter=0;
long previousMillis=0;
long interval=250;

void setup() {
  pinMode(2,INPUT);
  pinMode(3,INPUT);
  pinMode(4,INPUT);
  pinMode(buzzSW,OUTPUT);
  pinMode(smokeSW,OUTPUT);
    Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  for(int thisPin=lowestPin;thisPin<=highestPin;thisPin++){
    pinMode(thisPin, OUTPUT);
  pinCount=highestPin-lowestPin+1;} }

void loop() {
  ch2=pulseIn(2,HIGH,35000);
  ch3=pulseIn(3,HIGH,35000);
  ch4=pulseIn(4,HIGH,35000);
   if(ch2>1650){selSW1=HIGH;}
    if(ch2<1350){flipSW2=HIGH;}
    if(ch2>=1350&&ch2<=1650){
       selSW1=LOW;flipSW2=LOW;}

    if(ch3>1650){digitalWrite(buzzSW,HIGH);}
```

```
   else{digitalWrite(buzzSW,LOW);}

   if(ch4>1650){digitalWrite(smokeSW,HIGH);}
   else{digitalWrite(smokeSW,LOW);}


  delay(100);

  // read the pushbutton input pin:
  buttonState = selSW1;

  // compare the buttonState to its previous state
  if (buttonState != lastButtonState) {          // if the state has changed, increment the counter
    if (buttonState == HIGH) {  // if the current state is HIGH then the button went from off to on:
      buttonPushCounter++;}    // Delay a little bit to avoid bouncing
    delay(50);}
  // save the current state as the last state, for next time through the loop
  lastButtonState = buttonState;

  // turns on the LED every 1pinCount button pushes
  if (buttonPushCounter % pinCount == 0) {
    digitalWrite(ledPin, HIGH);
    buttonPushCounter=0;} else {
    digitalWrite(ledPin, LOW);}

//Flash Delay
  unsigned long currentMillis = millis();
    if(currentMillis-previousMillis > interval && counter <buttonPushCounter){
      previousMillis = currentMillis;
      if(ledStateF == LOW){
        ledStateF = HIGH;}
       else{
        ledStateF = LOW;
        counter +=1;}
      digitalWrite(ledPin,ledStateF);}

     if(currentMillis-previousMillis > (interval*buttonPushCounter)){
         counter = 0;}

//myFindO/PValues
 int selectedPin = buttonPushCounter + lowestPin;
  int ledState = digitalRead(selectedPin);

 //Read Flip Button input
  flipButtonState = flipSW2;

  // compare the buttonState to its previous state
  if (flipButtonState != lastFlipButtonState) {
    // if the state has changed
    if (flipButtonState == HIGH) {

      ledState =! ledState;
      digitalWrite(selectedPin,ledState);}
    delay(50); }// Delay a little bit to avoid bouncing
  // save the current state as the last state, for next time through the loop
  lastFlipButtonState = flipButtonState;}
```

Description

The 1st block sets up all the memory cells and assigns values to some of them.
The 'void setup(){' configures the 3 input pins, the 2 switch outputs, (1 for a buzzer and 1 for a smoke unit. Both would need to drive through a LED/Relay driver.) A for/next loop then configures the individual LED outputs, looping round from 1st to last.
The 'void loop(){' runs the repeated loop, beginning with collecting the input values coming from the RC receiver. If these values are above or below set values, then memory values are updated, or the switch outputs are set for the buzzer or smoke. They only stay on while the transmitter stick is held.
The next section then looks for a change of state in memory selSW1, and moves a count along and turns on a flashing LED that flashes in accordance with the count.
The last sections then output to the relevant LED to change state if there has been a change in the flipSW2 memory.
The loop then runs again.

Example of Arduino for the "*AZIZ*" Model Tug

The Arduino "sketch" for controlling this is written in 'RC Connection to Arduino + AZIZTugLights.'

//RC connection to Arduino + AzizTugLights The text "//" means a comment follows and is ignored by the "sketch."

```
int ch2;
int selSW1=0;
const int lowestPin=4;
const int highestPin=13;              //only 10 outputs for 4 to 13
int pinCount=0;
const int selCount=7;                 //count of selection choices, 0 to 6 =7
int buttonPushCounter = 0;            // counter for the number of button presses
int buttonState = 0;                  // current state of the button
int lastButtonState = 0;              // previous state of the button
int outState=LOW;
int offArray[]{1,2,3,4,5,6,7,8,9};    //all off
int onnArray[]{1,2,3,4,5,6,7,8,9};    //all on
int navArray[]{1,0,0,4,0,6,7,8,0};    //under way
int towArray[]{1,0,3,4,0,6,7,8,0};    //towing under way
int ancArray[]{0,0,0,4,5,0,0,0,0};    //anchored
int re1Array[]{0,2,0,0,0,0,0,0,0};    //restricted manoeuvres at rest
int re2Array[]{1,2,0,4,0,6,7,8,0};    //restricted manoeuvres underway

void setup() {
        pinMode(2,INPUT);
        for(int thisPin=lowestPin;thisPin<=highestPin;thisPin++){
                pinMode(thisPin, OUTPUT);
                pinCount=highestPin-lowestPin+1;} }

void loop() {
        ch2=pulseIn(2,HIGH,35000);
        if(ch2>1650){selSW1=HIGH;}
        if(ch2>=1350&&ch2<=1650){selSW1=LOW;}
        delay(50);
        buttonState = selSW1;                   // read the pushbutton input pin:
        if (buttonState != lastButtonState) {   // compare the buttonState to its previous state
        if (buttonState == HIGH) { buttonPushCounter++;  } // if the state has changed, increment
the counter
        delay(50);      }                       // Delay a little bit to avoid bouncing
```

```
    lastButtonState = buttonState;          // save the current state as the last state, for next time through the loop

    for(int i=0;i<10;i++){                  //using arrays to set outputs
    if(buttonPushCounter==0) {if(i+1 ==offArray[i]) {digitalWrite(i+lowestPin,LOW);}}
    if(buttonPushCounter==1) {if(i+1 ==onnArray[i]) {digitalWrite(i+lowestPin,HIGH);}
else{digitalWrite(i+lowestPin,LOW);}}
    if(buttonPushCounter==2) {if(i+1==navArray[i]) {digitalWrite(i+lowestPin,HIGH);}
else{digitalWrite(i+lowestPin,LOW);}}
    if(buttonPushCounter==3) {if(i+1==towArray[i]) {digitalWrite(i+lowestPin,HIGH);}
else{digitalWrite(i+lowestPin,LOW);}}
    if(buttonPushCounter==4) {if(i+1==ancArray[i]) {digitalWrite(i+lowestPin,HIGH);}
else{digitalWrite(i+lowestPin,LOW);}}
    if(buttonPushCounter==5) {if(i+1==re1Array[i]) {digitalWrite(i+lowestPin,HIGH);}
else{digitalWrite(i+lowestPin,LOW);}}
    if(buttonPushCounter==6) {if(i+1==re2Array[i]) {digitalWrite(i+lowestPin,HIGH);}
else{digitalWrite(i+lowestPin,LOW);}}}

    if(buttonPushCounter % selCount == 0) {buttonPushCounter=0;}} // moves the count back to the start > selCount pushes.
```

Description

All the “sketches” are split to 3 sections - declarations, a setup and loop.

We need to prepare the groundwork, that can be used by the sketch later on. The 1st line ‘int ch2;’ says to setup a memory/pigeon hole called ch2, and allow it to have only integers (whole numbers..1 2 3 etc.,) put into it. Likewise line 2 sets up a memory called selSW1, for integers only, and for now put a ‘0’ into it. (Arduinos count 0 as a valid number and even the 1st step of a sequence.) Then we setup the memory lowestPin & highestPin, and put in the number 4 and 13 respectively, but won’t allow them to be changed..i.e. constant. pinCount & the selCount memories are set with 0 and a constant 7 respectively. The next 4 lines set up memories that will change but we want to start with a particular value, usually 0 or low, to allow the “sketch” to start from the same point. Then we have the arrays declared with whole numbers, in 7 rows, each with numbers in them that relate to the lighting groups we want to be influenced by that row. For example re1Array has only group 2 to work on which are the lights to show “restricted operations, while the vessel is stationary.” The arrays could have been setup as one “matrix” of 7 x 9 but I found the referencing in the “sketch” to be more confusing, so I stayed with single line arrays, with a name for each row.

Next in the ‘*void setup(){*‘ we prepare the bits that run only once at startup. 1st up is in ‘pinMode’ setting the pin which is digital i/o 2 (D2) to be an input…always.

A For/next loop now runs which beginning with the value in lowestPin (4), till the value in the highestPin (13), going up by a count of 1 each time round the loop, and puts the current value into a memory called thisPin. At each thisPin value in the loop, we setup with the pinMode, that the digital i/o with the value of thisPin is setup as an output. Here all digital i/o pins from 4 to 13 are set as output pins. So using a loop we can save typing the same rows of programme several times, and essentially do the same thing to a group. It means if we need to change the sketch we don’t have to edit every line, just 1 part, and it will change the result for each item in the group. 2 lines of text can replace, in this case 9 lines. By changing the value in highestPin to 100 the for/next loop would go round (100-4) times. Saves lots of repeated typing. The loop is contained in the pair of brackets { } and loops as long as it is between the lowest and highest value. When it gets to the highest the loop stops and the “sketch” moves on.

The pinCount actually doesn’t need to be in the loop as it only will have 1 value (13-4+1=10), but as it is there it won’t harm anything.

We now have the main loop that will run again and again, beginning with ‘*void loop(){* ‘. The 1st 3 lines measure the time delay between 2 high pulses coming in on digital i/o pin 2 (D2) from the receiver, and puts it in ch2. It waits for pulses for only 35,000µs, then if it hasn’t seen the pulses it

lets the “sketch” go on. Now if the value in ch2 is greater than 1650, it puts a high into memory selSW1. If below 1650 but above 1350, selSW1 gets set as a low. The next 4 lines check for a change in selSW1, and if it went high, then to increment a counter ‘buttonPushCounter’ (“bpc”) by 1, and remember that. Another for/next loop begins, and for every-time round the loop, it steps from 0 to 9, putting the current value into a memory called ‘i’ and then using this value in i to compare firstly the value of the buttonPushCounter memory with a number from 0 to 6. (Remember that Arduino counts 0 as a valid step, so that we have 7 steps.) If the “bpc” comparison is true, then the next If statement below it runs, and says if i+1 is the same as the value in the “ith” value of the relevant array, then digitally write (send a signal to a Di/o) to the output D(i+4) a high or 5V. The offArray doesn’t have the write high section, as it only turns off all the LED’s. The other xxxArray’s will also turn off any LED’s that the previous “bpc” value had switched on, that it doesn’t itself want on. The for/next loop will run from 0 to 9 checking or setting each of the lighting groups, according to the number existing in the current array line, then when i =9, the for/next loop exits and the main loop continues by checking if the “bpc” when divided by the value of selCount, (the total number of arrays,) has a ‘modulo’ value of 0. That is 7/7=0. If so the next line sets “bpc”back to 0, and the main loop goes round again, starting by looking at the input pulse from the receiver, then running round the for/next loops, setting the LED’s accordingly. Again and again. If ch2 doesn’t go high, when the previous loop had it as a low, then nothing changes. But as soon as the ch2 does go high, then the next “bpc” and its array row comes into play, and the LED’s do change accordingly. Every upwards blip of the transmitter stick moves the sequence on one step. Just 1 push, 7 times goes through all the scenarios.

Again the arrays could have been combined to 1 matrix, and correspondingly compacted the “if “bpc” ==“ lines, which are a bit clumsy, but that was more complex, and as I had plenty of space, memory wise in the Arduino, I left it expanded. Sometimes spreading out the lines makes reading easier, when fault finding, but it means that if you need to change the meaning and logic of the rows, then you may have to repeat the changes by editing the same in each of 7 sets of the “sketch.” There are often many ways to do the same things.