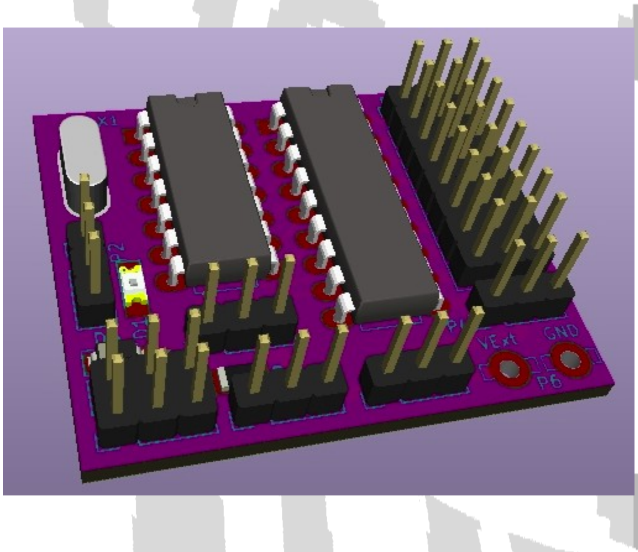
MS8-Xany



More than a simple 8-output MultiSwitch decoder for OpenAVRc

MS8-Xany MultiSwitch user manual

Table of Content

1 THIS DOCUMENT

1.1 Versions

The version of this document is the last of the ***version*** column of the following table.

Version	*Date*	*Reason for update*
0.1	20/12/2018	Creation
0.2	01/06/2020	Tulip holder for ATtiny84, diode serigraphy correction
0.3	29/03/2021	- Picture of PCB added - Correction of jumper ↓ and = position for P1 - Pinning added for USB/Serial type « FTDI » for P9 - Minor correction in command table - ATtiny84 programming with the help of an arduino UNO configured as ArduinoISP added
0.4	27/07/2021	« Pulse » mode commands added (Release MS8 V0.5)

1.2 Copyright

1.3 Disclaimer

The **OpenAVRc** team is not responsible for any damage that may result from the misuse or possible malfunction of the **OpenAVRc** transmitter, the **MS8-Xany** decoder module and/or associated software.

It is therefore up to the end user to estimate, assume the risks and comply with the legislation in force depending on the country of use.

1.4 Contents

This document describes the making of the **MS8-Xany** decoder module as well as the settings for its use with the **OpenAVRc** transmitter.

MS8-Xany is a MultiSwitch decoder module with 8 digital outputs.

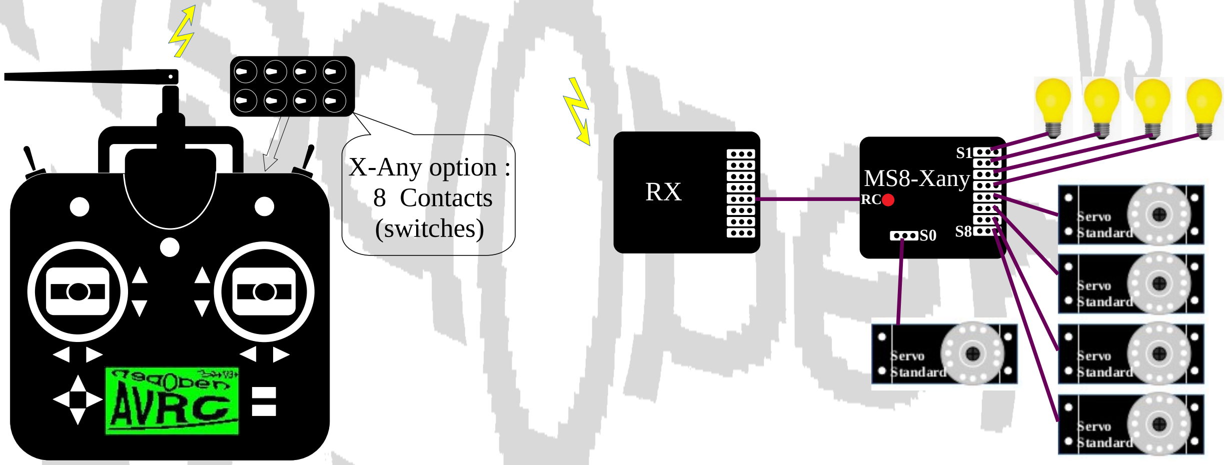
In addition to the classic MultiSwitch function, **MS8-Xany** can be set up so that its outputs drive standard servos with digital commands (0/1).

Finally, **MS8-Xany** can provide an auxiliary proportional channel for driving a standard proportional servo or an ESC.

It is also possible to command each output in « Pulse » mode.

2 PRESENTATION OF MS8-XANY

2.1 Overview

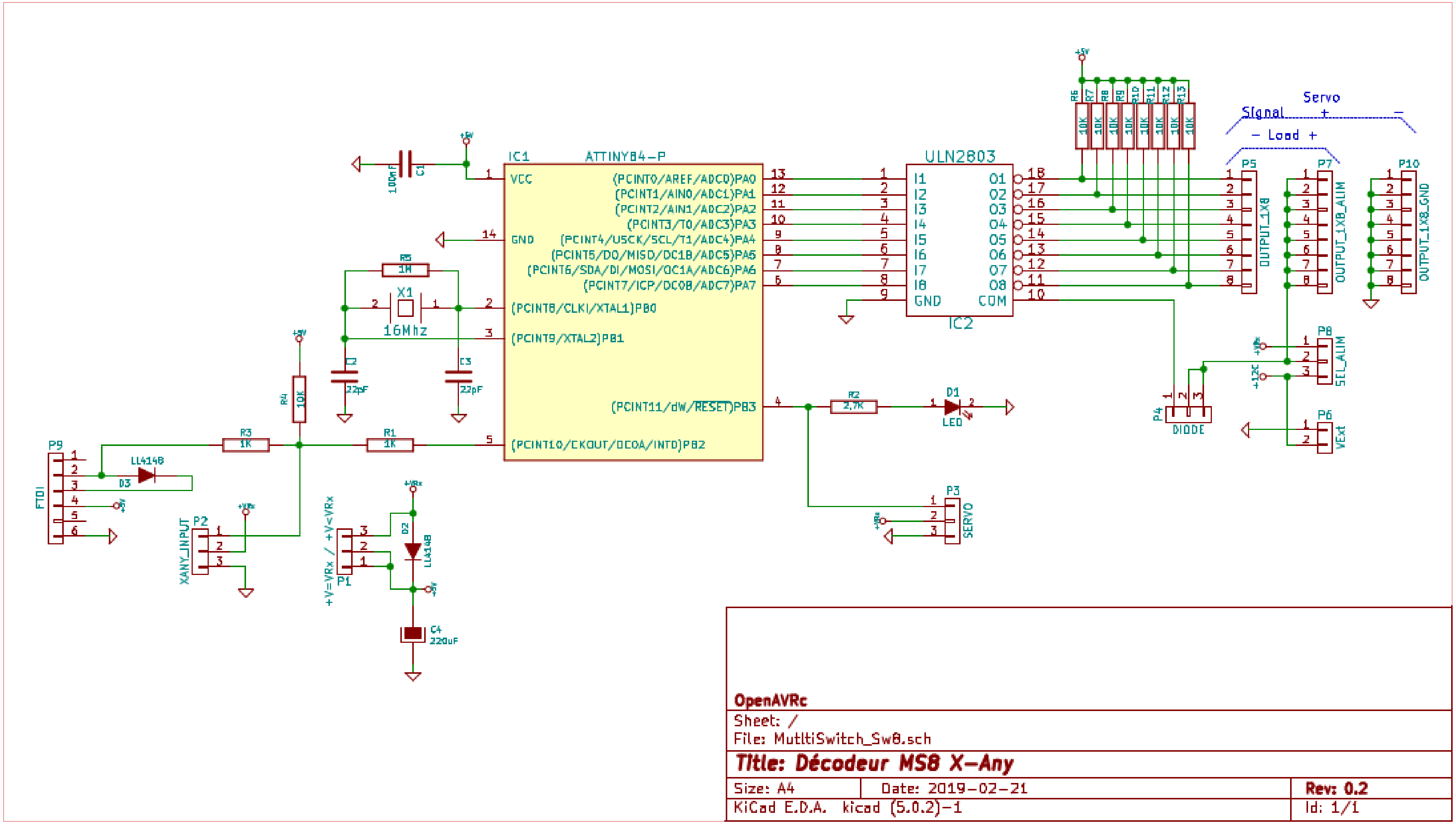


2.2 MS8-Xany decoder specifications

Specification	Value / Feature	Notes
Power supply	+3.3V to +6.6V	Set the "receiver voltage" jumper to "=" or "↓" according to the value of the voltage supplied by the receiver
X-Any Protocol	-Universal digital protocol used by OpenAVRc for remote accessories - Integrity check with 8-bit checksum - Works with all protocols, including 2.4GHz: - PPM Protocol - SPIRfMod Protocols - MultiMod Protocols	Unlike many MultiSwitches, **MS8-Xany** also works with 2.4GHz HF modules
8 Digital Outputs	- Commanded in Digital mode by transistor output	Outputs configured in "Digital" mode (MultiSwitch)
Output voltage	- Internal: receiver voltage - External: external voltage 5 to 24V	Selectable by jumper (common to 8 outputs)
Free wheeling diodes	Free wheeling diodes between the 8 outputs and + output power	Selectable by jumper (common to 8 outputs)
8 servo outputs	- Digitally driven - End positions can be configured - Duration of movement between end positions is configurable	- Outputs configured in "Servo" mode - Reversal of servo travel possible by changing extreme values - Ability to use receiver voltage - Possibility to use an external voltage (compatible with servos)
1 proportional servo output	Proportional control of a servo from 988µs to 2008µs (0 to 255 steps of 4µs)	Presence of servo control is dynamically detected by **MS8-Xany**: nothing to configure except at the OpenAVRc transmitter side
Red LED lights when signal is lost	After 1.5 seconds without correct signal	Not managed if proportional servo used (shared connection)
Failsafe	- All outputs go to 0 in case of RC signal loss - The proportional servo retains its current position	Synchronized with red LED turning on
TTL serial port	USB cable connector / FTDI TTL	For advanced configuration using a "Serial Terminal" application
Dimensions (mm)	L x l x h : 39 x 33 x 12	

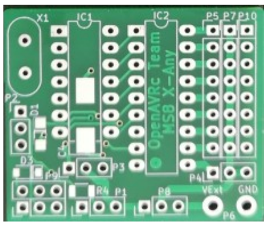
Page left intentionnally blank

3 MS8-XANY SCHEMATIC DIAGRAM



4 Making the MS8-XANY DECODER

4.1 Printed circuit board



4.2 Tulip holder for integrated circuits

4.2.1 Tulip holder for ATtiny84

ATTENTION :

The ***ATtiny84*** shall be programmed **before** mounting it on the **MS8-Xany** printed circuit board.

That's why, it is hghly recommended to mount it on a 14 points tulip holder :

It is also possible to use 2 portions (of 7 points) of a breakable tulip bar :

In both cases, check carefully the orientation of the ***ATtiny84*** before powering the **MS8-Xany**.

4.2.2 Tulip holder for ULN2803

In case of bad wiring for the outputs, the ***ULN2803*** may be destroyed.

In order to facilitate its replacement, it is highly recommended to mount it on 18 points tulip holder :

It is also possible to use 2 portions (of 9 points) of a breakable tulip bar :

In both cases, check carefully the orientation of the ***ULN2803*** before powering the **MS8-Xany**.

4.3 Uploading firmware and configuring fuses

The programming of the microcontroller ***ATtiny84*** is done in 2 phases (in this order):

1. Upload firmware
2. Configure fuses

4.3.1 *Uploading Firmware in ATtiny84*

ATTENTION :

The P9 connector (labelled FTDI) **is not** a **programming** connector, but is a **parametering** connector for **MS8-Xany** (See §Advanced Mode / Servo Control).

2 firmwares are available :

- **MS8_Xany_PWM_Vx_y.hex** : **MS8-Xany** is then driven from a ***PWM*** output of the receiver
- **MS8_Xany_CPPM_Vx_y.hex** : **MS8-Xany** is then driven from the ***CPPM*** output of the receiver

The **MS8_Xany_PWM_Vx_y.hex** ***PWM*** version will be more often used.

Using an AVR microcontroller ***ICSP programmer***, upload the **MS8_Xany_PWM_Vx_y.hex** file to the ***ATtiny84***.

For doing this, we will use an ***arduino UNO*** configured as ***ICSP programmer*** to load the ***HEX file*** in the ***ATtiny84***.

All the programming phase (**Firmware loading** + F**uses configuration**) is preformed **before** mounting the ***ATtiny84*** on its tulip holder.

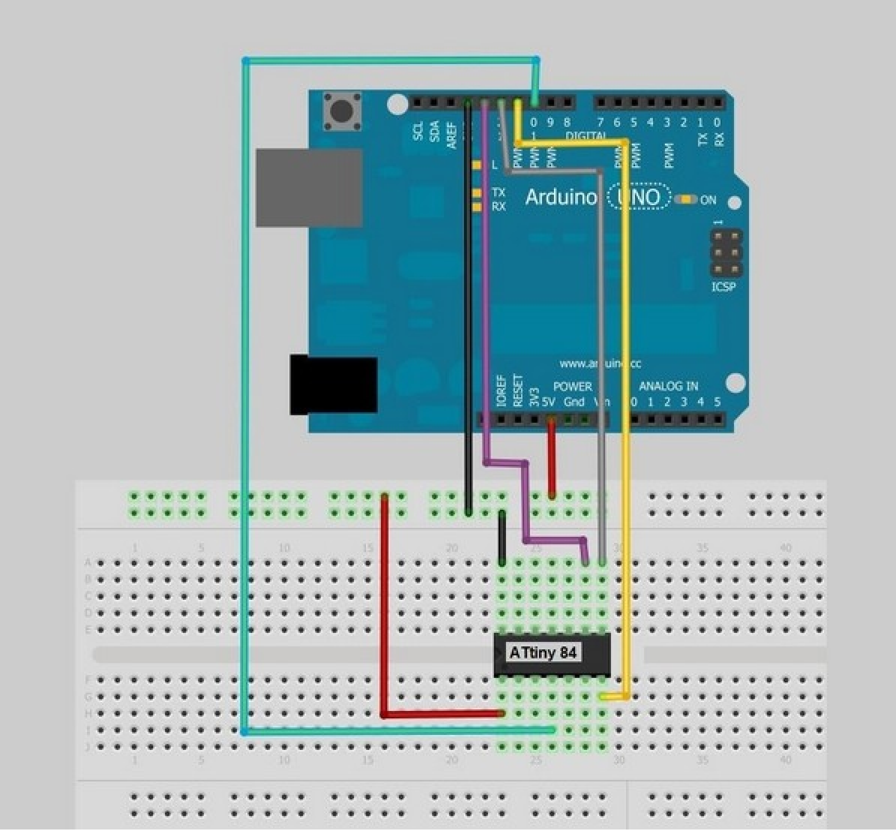
4.3.2 Arduino UNO as ICSP programmer under Windows

Connect your **arduino UNO**, and with the **arduino IDE,** load the **ArduinoISP** sketch by doing :

Files→Examples→11.ArduinoISP

Then, click on ***Upload*** : your ***arduino UNO*** behaves now as an ***ICSP programmer***.

Do the following wiring to program the ***ATtiny84*** with the ***arduino UNO*** :



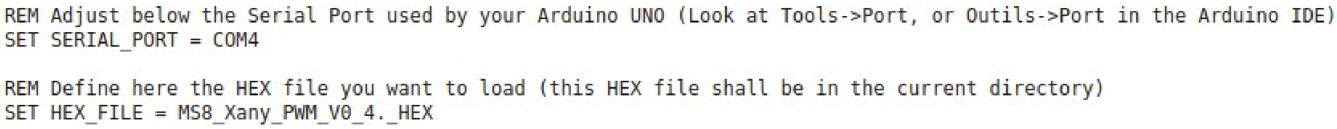
Copy the **Windows** sub-directory of the **[PROG]** directory on your PC. Open a **DOS console** (invite command) and go in this **Windows** directory.

If required, edit the **prog_ms8.bat** file with a text editor in order to redefine :

1. the right **serial port** used by the **arduino UNO : COMx**
2. the **HEX file** you want to load :

   (**MS8_Xany_PWM_Vx.y._HEX**, or **MS8_Xany_CPPM_Vx.y._HEX**)

   In case, you don't know which one to choose, take **MS8_Xany_PWM_Vx.y._HEX**.



```
REM Adjust below the Serial Port used by your Arduino UNO (Look at Tools->Port, or Outils->Port in the Arduino IDE)
SET SERIAL_PORT = COM4

REM Define here the HEX file you want to load (this HEX file shall be in the current directory)
SET HEX_FILE = MS8_Xany_PWM_V0_4._HEX
```

In the ***DOS console***, launch the **prog_ms8.bat** command and follow instructions.

The **HEX file** will be loaded and the **fuses** will be automatically programmed.

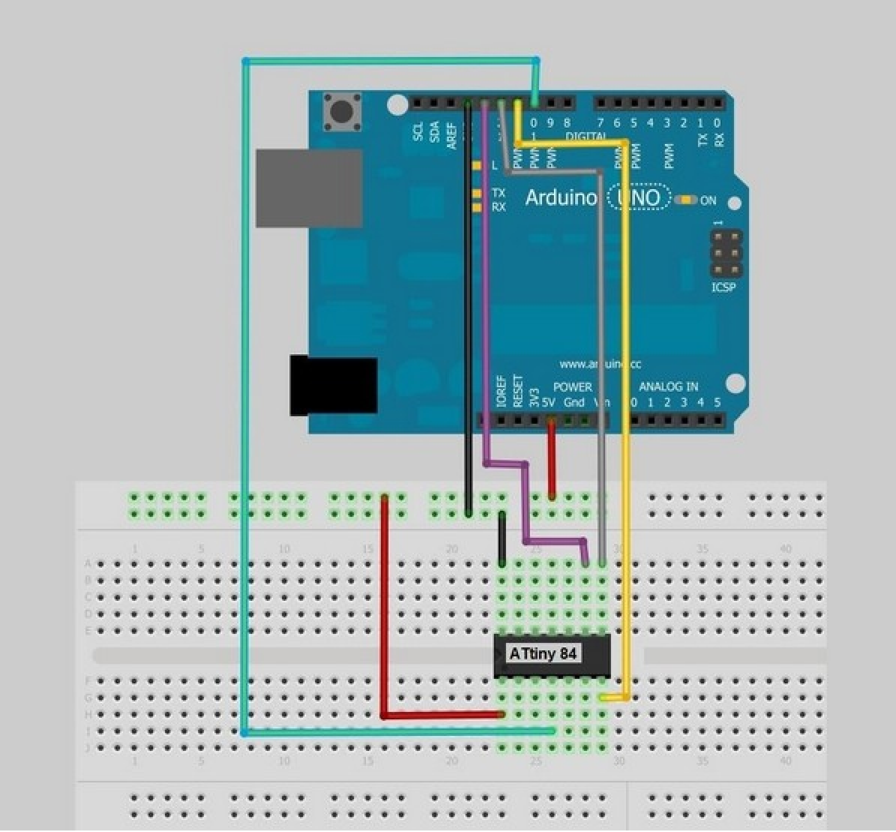
4.3.3 Arduino UNO as ICSP programmer under Linux

Connect your **arduino UNO**, and with the **arduino IDE,** load the **ArduinoISP** sketch by doing :

Files→Examples→11.ArduinoISP

Then, click on ***Upload*** : your ***arduino UNO*** behaves now as an ***ICSP programmer***.

Do the following wiring to program the ***ATtiny84*** with the ***arduino UNO*** :



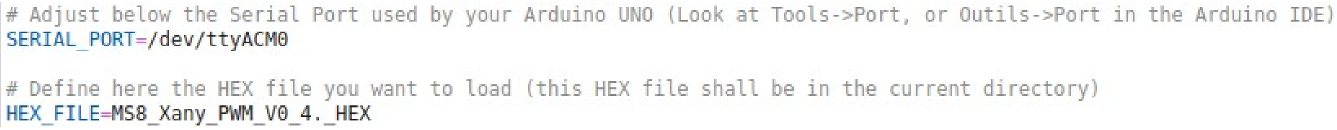
Copy the **Linux** sub-directory of the PROG directory on your PC. Open a **terminal console** and go in this **Linux** directory.

If required, edit the **prog_ms8.sh** file with a text editor in order to redefine :

3. the right **serial port** used by the **arduino UNO : for example, */dev/ttyACM0***
4. the **HEX file** you want to load :

   (**MS8_Xany_PWM_Vx.y._HEX**, or **MS8_Xany_CPPM_Vx.y._HEX**)

   In case, you don't know which one to choose, take **MS8_Xany_PWM_Vx.y._HEX**.



```
# Adjust below the Serial Port used by your Arduino UNO (Look at Tools->Port, or Outils->Port in the Arduino IDE)
SERIAL_PORT=/dev/ttyACM0

# Define here the HEX file you want to load (this HEX file shall be in the current directory)
HEX_FILE=MS8_Xany_PWM_V0_4._HEX
```

In the ***terminal console***, launch the ***./prog_ms8.sh*** command and follow instructions.

The **HEX file** will be loaded and the **fuses** will be automatically programmed.

4.3.4 Very important note about the value of the fuses

Once the ***ATtiny84*** configured (with its fuses), it is no more possible to ***directly*** re-program it with an ***ICSP programmer*** since this one needs the ***Reset*** function of the ***ATtiny84***.

This is due to the fact the ***MS8-Xany*** firmware needs the ***Reset*** pin, but configured as an output to drive the « Signal lost » led and to drive the proportional output.

If, for any reason, you need to re-program your ***ATtiny84***, it will be needed to restore the ***Reset*** function of the ***Reset*** pin.

To do that, it is mandatory to apply ***calibrated voltage (12V)*** on the ***Reset*** pin. A « ***Fuse Resetter*** » box is then needed.

After doing this, the ***ATtiny84*** becomes programmable again using an ***ICSP programmer***.

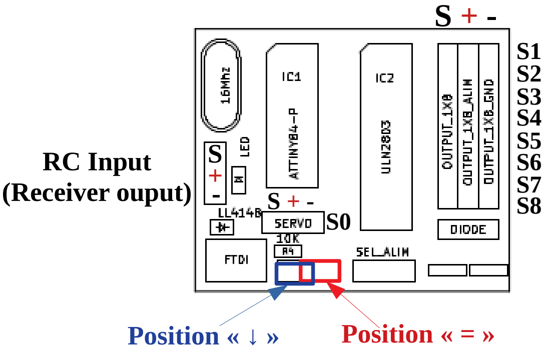
5 Usage

5.1 Connecting to a receiver

Before connecting **MS8-Xany** to the receiver, it is imperative to measure the voltage supplied by the receiver.

If the available voltage between the - and + pins of the 3-pin connector of the channel used is:

1. Lower than 5.7V, set the jumper "receiver voltage" to "="
2. Higher than 5.7V, set the jumper "receiver voltage" to "↓"



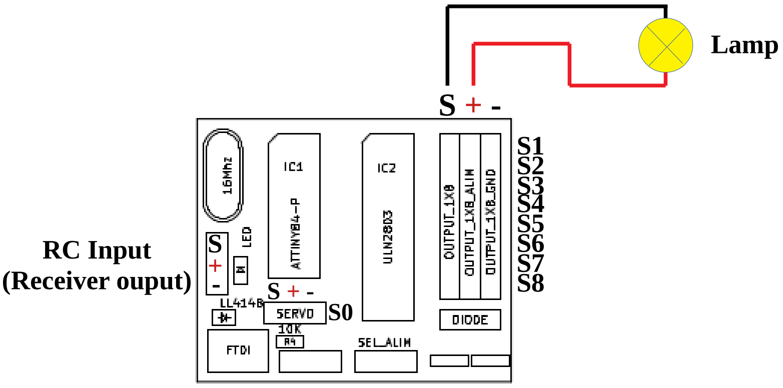
5.2 Standard Mode : MultiSwitch / Digital switch

After uploading the firmware, MultiSwitch is the default mode: the digital switching mode of the 8 **S1** to **S8** outputs. So there is nothing else to configure to operate in MultiSwitch mode.

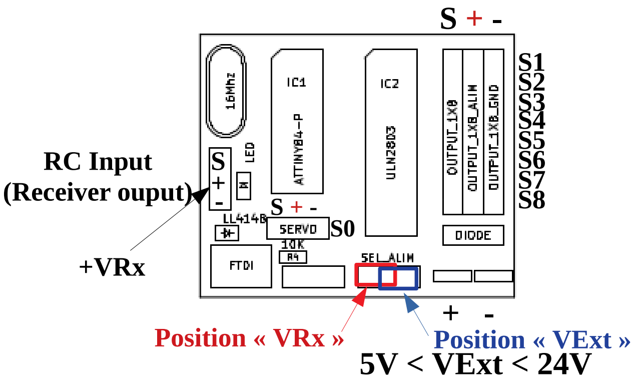
5.2.1 *Wiring the loads on the outputs*

The **MS8-Xany** is then used as a standard MultiSwitch module:

- The "loads" (like: a lamp) are connected to the **S1** to **S8** outputs between the pins "**S**" and "**+**", the 8 row points of "**-**" at the edge of the board is not used in this case.



5.2.2 Selecting the supply voltage for the S1 to S8 outputs



It is possible to supply the S1 to S8 outputs (supply to **+**) from:

1. **+VRx** voltage supplied by the receiver (Beware of the current consumption on the outputs!)

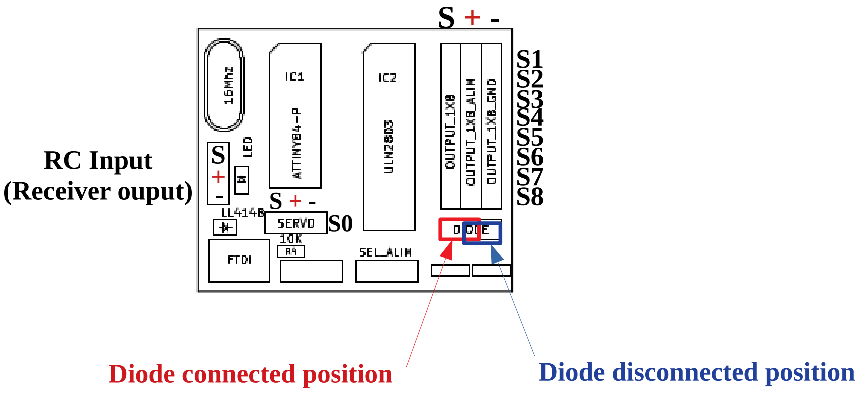
→ Power selection jumper on "**VRx**"

2. An external **VExt** voltage (**5V** to **24V**) applied to the 2-pin connector at the bottom right of the board

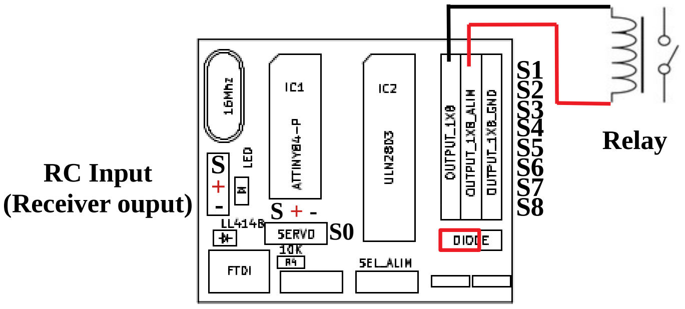
→ Power selection jumper on "**VExt**"

5.2.3 Free wheeling diodes

If the "loads" connected to the outputs are inductive (eg relays), the internal free wheeling diodes shall be connected, otherwise there is a risk to destroy the ULN2803.

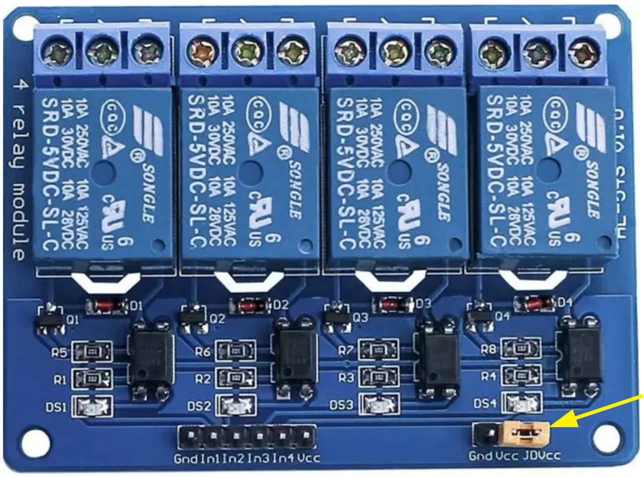


5.2.4 Direct relay control with integrated diodes in the ULN2803



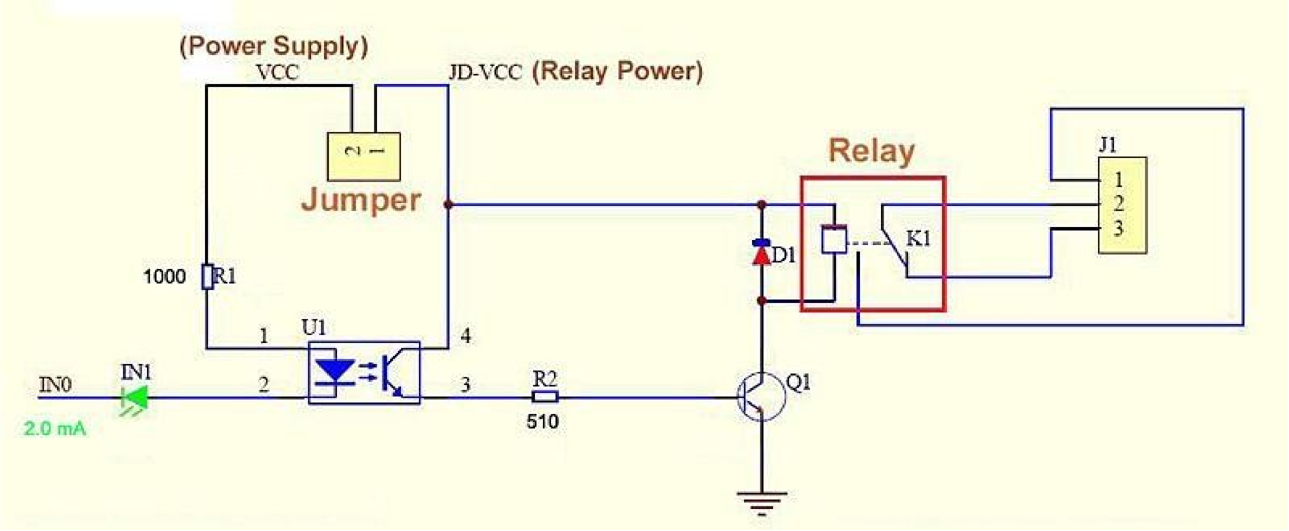
5.2.5 Controlling opto-isolated relays

There are very cheap "relay modules" often called "Arduino Relay Module". These modules include an opto-coupler allowing a total isolation between the control voltage and the supply voltage of the coils of the relays.



Yellow Jumper

The equivalent circuit diagram of one relay channel is given below:

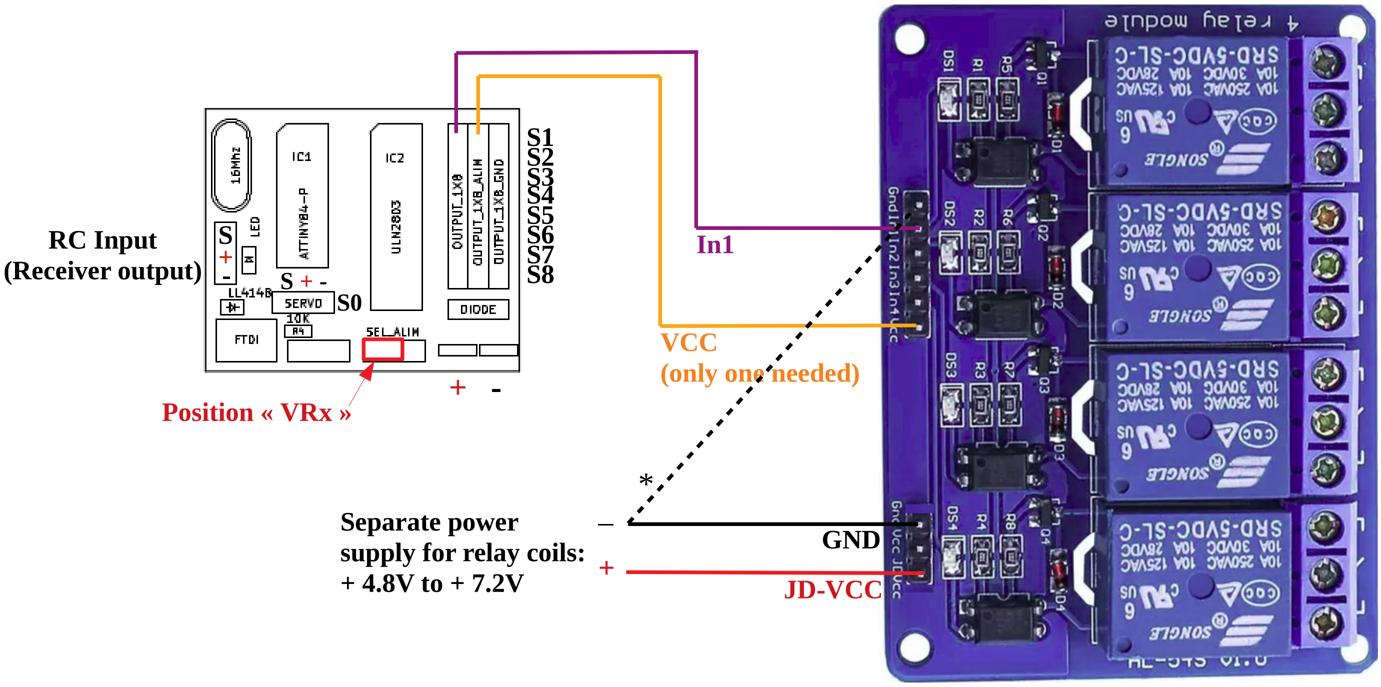


- **VCC** is the opto-coupler control voltage: accessible on the connector on the edge of the board
- **JD-VCC** is the control voltage of the relay coils: accessible on one of the jumper connector pins

Important Note:

To achieve full isolation between VCC and JD-VCC, the yellow jumper shall be removed.

5.2.6 Recommended connections for 5V opto-isolated relays



- ◆ The optocouplers are powered by the receiver voltage (total consumption: around 10 mA)
- ◆ VCC of the relay board is connected to one of the OUTPUT_1X8 supply pins
- ◆ In1 of the relay board is connected to the S1 output
- ◆ In2 of the relay board is connected to the S2 output, etc.
- ◆ Relay coils are powered by a separate power supply (+ 4.8V to + 7.2V)

→ Since the power supplies are completely isolated, the receiver voltage will not be disturbed during relay switching: no risk of losing RC control.

***** : On some "relay module" models, there is no **GND** pin near the **JD-VCC**. In this case, use the **GND** pin close to the **In1** pin on the other connector.

5.3 X-Any configuration at the OpenAVRc transmitter side

Refer to the **OpenAVRc** manual to configure the X-Any instance with the following parameters:

1. The channel number shall correspond to the channel number on which the **MS8-Xany** decoder will be connected to the receiver
2. The number of repetitions will be first set to 3 (as soon as it will work, it will be possible to reduce this value to reach the maximum reactivity allowed by your HF set).
3. Configure "Sw." with Sw.8: this will transmit the state of the 8 contacts
4. If the proportional servo will be used on **MS8-Xany**, select one of the proposed choices by

   « Prop. », this will add the transmission of the proportionnal value.

5.4 Advanced Mode / Servo Control

The **MS8-Xany** decoder has an access for advanced configurations: a TTL serial port.

It is this serial access that will allow the use of servos connected to the S1 to S8 outputs.

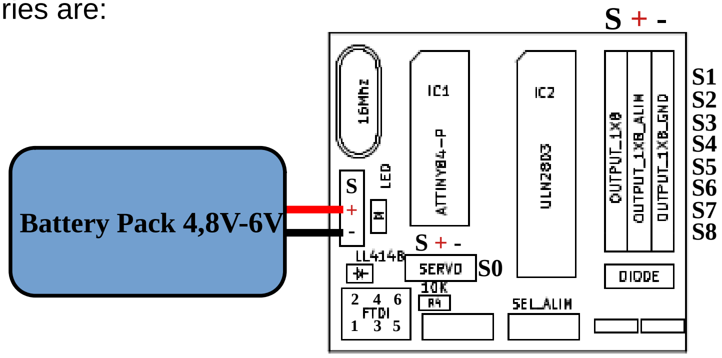
In this case, the "**+**" output voltage must be compatible with the servos!

5.4.1 Using the serial port on MS8-Xany

To access the serial port of **MS8-Xany**, you need a USB / TTL serial cable eg "FTDI" type.

The necessary pins on the USB cable / TTL series are:

- GND « FTDI » → Pin 6 of P9
- +5V « FTDI » → ***Not connected***
- TX « FTDI » → Pin 3 of P9
- RX « FTDI » → Pin 2 of P9



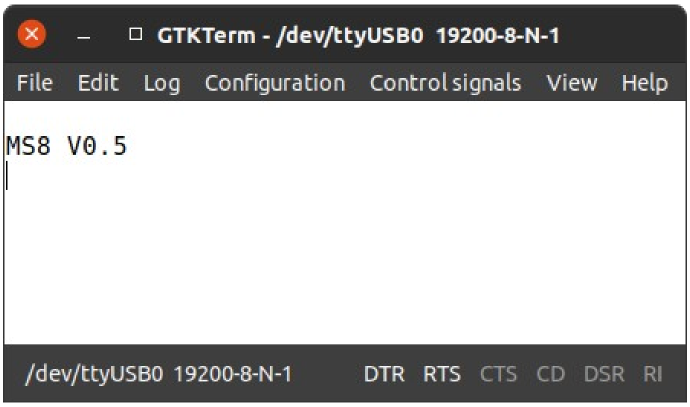
1. Connect USB side of the USB / TTL serial cable to an USB port on a PC

2. On the PC, open a Serial Terminal, for example, PuTTY, TeraTerm, HyperTerminal, GtkTerm, or CoolTerm with the following parameters: 19200 baud, 8 data bits, 1 stop bit, no parity.

Depending on the Serial Terminal, it may be necessary to enable automatic line feeds on receipt of CR / LF to have a good display.

3. Connect a ***4,8V à 6V Battery pack*** on the **MS8-Xany** RC connector : this ***pack*** will provide the power.

4. Within 3 seconds after connecting the ***Battery pack***, press the "Enter" key on your keyboard, the message "MS8 VX.Y" should appear on the Serial Terminal as shown below. If this is not the case, disconnect the ***Battery pack*** and repeat step 3 above.



```
MS8 V0.5
```

Example of connection with GtkTerm Terminal on Linux

5.4.2 *MS8-Xany command messages*

The list of messages supported by **MS8-Xany** is given in the following table:

←**Command/**→**Response**	**Action**	**Notes**
←Enter →MS8 Vx.y	If sent within the 3 seconds after power on, the decoder enters Terminal mode	If failed and 3 seconds elapsed, unplug and reconnect the 6-pin USB / TTL serial cable connector
←S0? →S0=Pos:4usStepOffset	Returns the current position in µs and the 4usStepOffset command which is the number of steps of 4µs (value between 0 and 255) to be added to 988 to have the pulse width in µs for the proportional servo	Pulse width (us) = 988 + (4usStepOffset x 4)
←S0=Pos →S0	Sets the position in µs for proportional servo	Returns ERR, if value not between 988 and 2008

←Sx?
→Sx=D;M:C
or
→Sx=S;M;Pos0;Pos1;Dur:C
	If x is between 1 and 8, returns the configuration of the output No. x and the state "C" of the associated current Command (0 or 1)
	- If the output is configured as MultiSwitch Digital output, the answer is: Sx = D;M:C
	D = Digital (Digital)
	M=command Mode (N : Normal, P : Pulse)
	- If the output is configured as a Servo output, the response is: Sx = S;M;Pos0;Pos1;Dur:C with
	S=Servo
	M=command Mode (N : Normal, P : Pulse)
	Pos0 = the position in µs for the state 0, Pos1 = the position in µs for the state 1, and Dur = the duration (in ms) of the movement of the servo between Pos0 and Pos1
		Returns ERR, if
		- Value x not between 0 and 8
		
		- Pos0 or Pos1 < 600
		- Pos0 or Pos1 > 2400
		
		Sample answers:
		
		S1=D;N:0
		
		S2=D;P:1
		
		S3=S;N;1000;2000;5000:0
		
		S4=S;P;2300;600;8500:1

←Sx=D;M
→Sx
	Sets the output x to be Digital
	M=command Mode (N : Normal, P : Pulse)
		S1=D;N
		
		S2=D;P

←Sx=S;Pos0;Pos1;Dur →Sx	Sets the output x as servo with Pos0 µs for 0, Pos1 µs for 1, and Dur ms	The real value in ms is internally recalculated by **MS8-Xany** taking into account the different resolutions / limitations and may be different when displayed.
←Sx=C →Sx	If x is between 1 and 8, "C" defines the State (0 or 1) for the output x, whether the type is either Digital or Servo	Very handy for testing using the serial access without RC.

←C? →C=Ch	Returns the channel used in CPPM mode	Command only available with the firmware in CPPM mode
←C=Ch →C	Set the used channel in CPPM mode	
←Q	Exit Terminal Mode: **MS8-Xany** can be connected to receiver	Do not forget to disconnect the USB / TTL cable!

5.4.3 Real configuration example

In the example below:

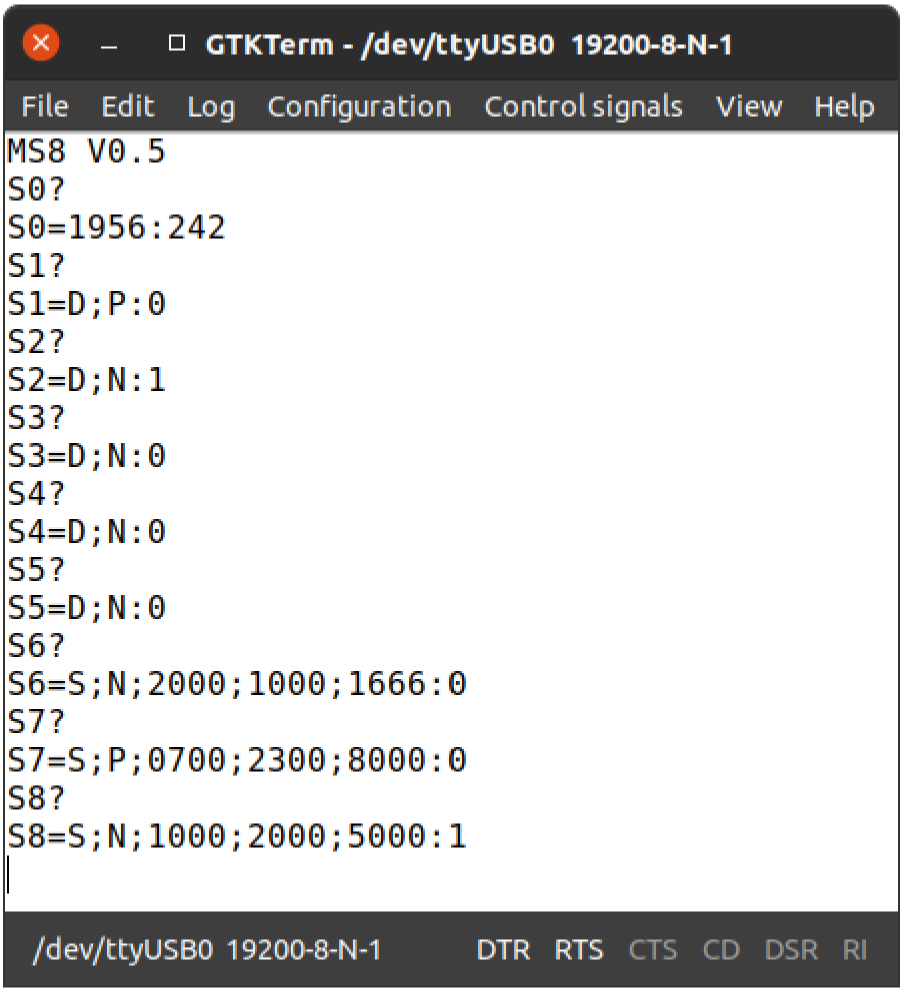
- The pulse width for the servo connected to the proportional S0 output is:

  [988 + (242 x 4)] = 1956 µs

  → the command (= the number of 4 µs steps to add to 988 µs) would be 242 for the pulse width of 1956 µs

- The outputs S1, S2, S3, S4 and S5 are Digital type (MultiSwitch) type, their states are respectively 0, 1, 0, 0 and 0. Ouput S1 is commanded in **P**ulse mode.

- The outputs S6, S7 and S8 are Servo type, their commands are respectively 0, 0 and 1. Ouput S7 is commanded in **P**ulse mode.

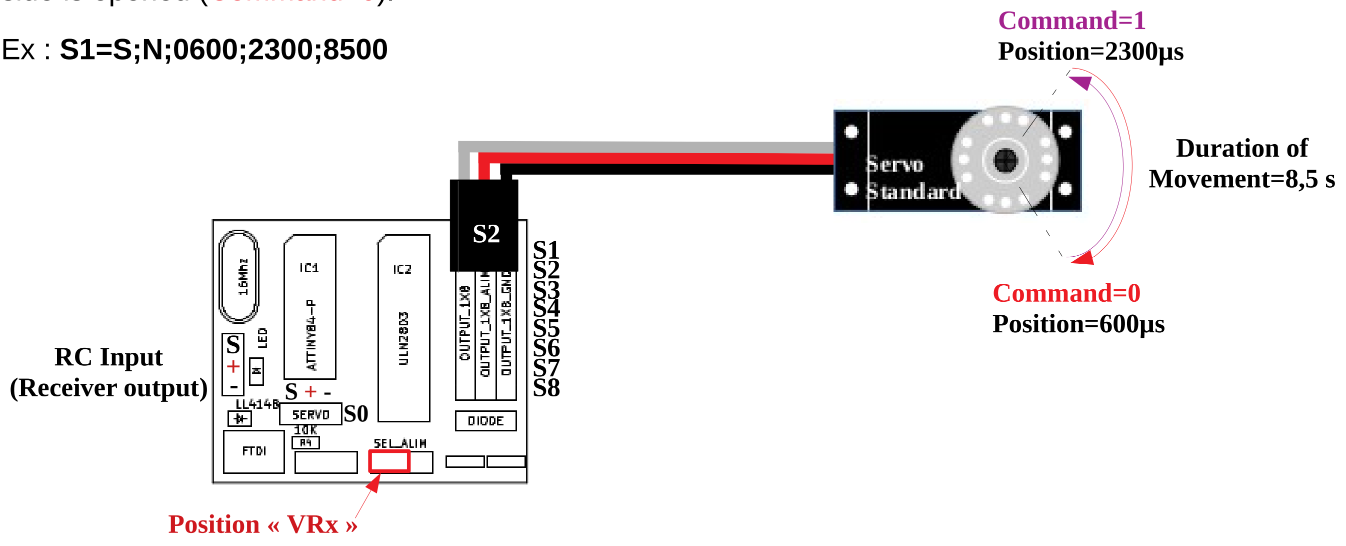


```
MS8 V0.5
S0?
S0=1956:242
S1?
S1=D;P:0
S2?
S2=D;N:1
S3?
S3=D;N:0
S4?
S4=D;N:0
S5?
S5=D;N:0
S6?
S6=S;N;2000;1000;1666:0
S7?
S7=S;P;0700;2300;8000:0
S8?
S8=S;N;1000;2000;5000:1
```

1. **Counterclockwise servo movement**

If the contact N°1 at transmitter side is closed (Command=1), the pulse of the servo connected to the output N°1 goes from 600 µs to 2300 µs (→ a movement of around 180°) in 8,5 seconds, and goes from 2300 µs to 600 µs (→ a movement of around 180°) in 8,5 seconds if the contact N°1 at transmitter side is opened (Command=0).

Ex : **S1=S;N;0600;2300;8500**

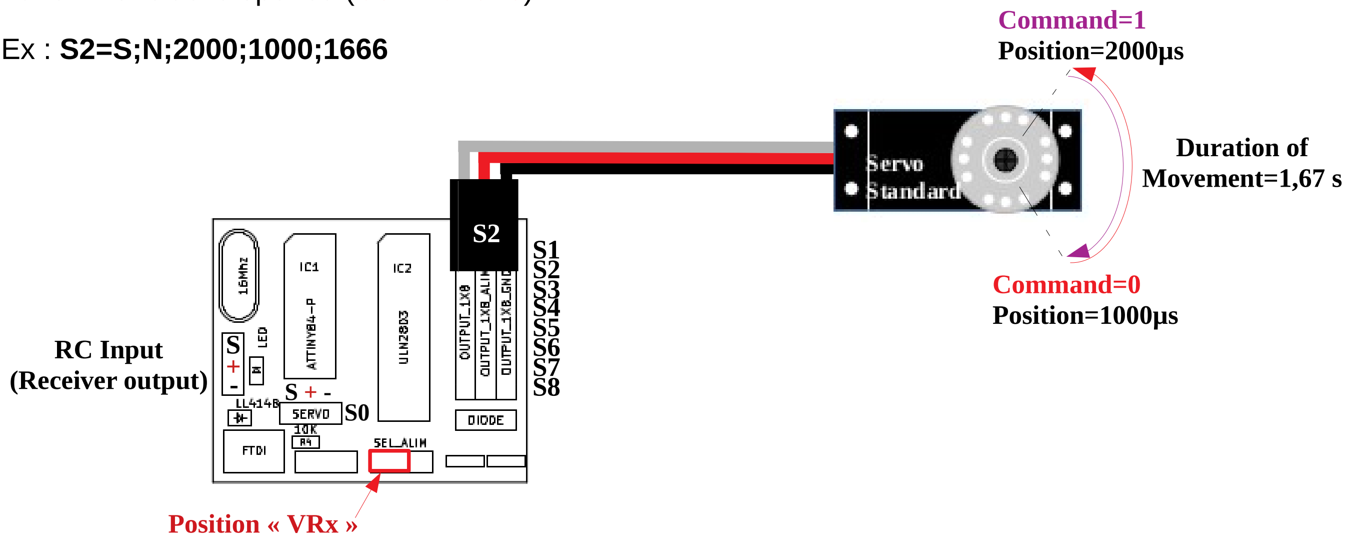


2. **Clockwise servo movement**

With **MS8-Xany**, it's possible to obtain a movement in the opposite direction : just swap the extreme positions (Pos0 and Pos1).

If the contact N°2 at transmitter side is closed (Command=1), the pulse of the servo connected to the output N°2 goes from 2000 µs to 1000 µs (→ a movement of around 100°) in 1,67 seconds, and goes from 1000 µs to 2000 µs (→ a movement of around 100°) in 1,67 seconds if the contact N°2 at transmitter side is opened (Command=0).

Ex : **S2=S;N;2000;1000;1666**



5.4.4 Normal mode and Pulse mode command

Whatever the output is configured in **D**igital type (Multiswitch) or in **S**ervo type, it is needed to define the command mode :

- **Sx=D;M** (with M equal to **N** for **N**ormal or **P** for **P**ulse)
- **Sx=S;M;1000;2000;5000** (with M equal to **N** for **N**ormal or **P** for **P**ulse)

Let’s suppose the following configuration and, at transmitter side, S1 et S2 are commanded with the contacts of push-buttons :

S1=D;N to control a foghorn

S2=D;P to control a light

In both cases, the outputs are **D**igital type (Multiswitch), but **S1** is commanded in **N**ormal mode, wheras **S2** is commanded in **P**ulse mode.

1. Case of Normal mode :

While the C1 contact is closed, the S1 output is 1 and the foghorn is enabled.

As soon as the C1 contact is released, the S1 output is 0 and the foghorn is disabled.

To summarize, the S1 output follows the status of the C1 contact.

2. Case of Pulse mode :

While the C1 contact is closed, the S1 output is 1 and the ligth is enabled.

As soon as the C2 contact is released, the S2 output remains to 1 and the light is still enabled.

To disable the light, it is neede to close again the C2 contact.

To summarize, a pulse on the C2 contact enables the S2 output, and a new pulse on the C2 contact disables the S2 output and so on.

Note :

If the type of the output is **S**ervo and the command mode is **P**ulse, a first pulse on the push-button, trigs the servo motion until the position N°1 and a second pulse trigs the servo motion to the position N°2.

Ex : **S3=S;P;1000;2000;5000**