Remote Controller Protocols and Scancodes

IR is encoded as a series of pulses and spaces, using a protocol. These protocols can encode e.g. an address (which device should respond) and a command: what it should do. The values for these are not always consistent across different devices for a given protocol.

Therefore out the output of the IR decoder is a scancode; a single u32 value. Using keymap tables this can be mapped to linux key codes.

Other things can be encoded too. Some IR protocols encode a toggle bit; this is to distinguish whether the same button is being held down, or has been released and pressed again. If has been released and pressed again, the toggle bit will invert from one IR message to the next.

Some remotes have a pointer-type device which can used to control the mouse; some air conditioning systems can have their target temperature target set in IR.

The following are the protocols the kernel knows about and also lists how scancodes are encoded for each protocol.

rc-5 (RC_PROTO_RC5)

This IR protocol uses manchester encoding to encode 14 bits. There is a detailed description here https://www.sbprojects.net/knowledge/ir/rc5.php.

The scancode encoding is *not* consistent with the lirc daemon (lircd) rc5 protocol, or the manchester BPF decoder.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\rc\[linux-master][Documentation][userspace-api][media][rc]rc-protos.rst, line 38)

Unknown directive type "flat-table".

```
.. flat-table:: rc5 bits scancode mapping
   :widths:       1 1 2

   * - rc-5 bit

     - scancode bit

     - description

   * - 1

     - none

     - Start bit, always set

   * - 1

     - 6 (inverted)

     - 2nd start bit in rc5,  re-used as 6th command bit

   * - 1

     - none

     - Toggle bit

   * - 5

     - 8 to 13

     - Address

   * - 6

     - 0 to 5

     - Command
```

There is a variant of rc5 called either rc5x or extended rc5 where there the second stop bit is the 6th commmand bit, but inverted. This is done so it the scancodes and encoding is compatible with existing schemes. This bit is stored in bit 6 of the scancode, inverted. This is done to keep it compatible with plain rc-5 where there are two start bits.

rc-5-sz (RC_PROTO_RC5_SZ)

This is much like rc-5 but one bit longer. The scancode is encoded differently.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\rc\[linux-master][Documentation][userspace-api][media][rc]rc-protos.rst, line 88)

Unknown directive type "flat-table".

```
.. flat-table:: rc-5-sz bits scancode mapping
   :widths:       1 1 2

   * - rc-5-sz bits

     - scancode bit

     - description

   * - 1

     - none

     - Start bit, always set

   * - 1

     - 13

     - Address bit

   * - 1

     - none

     - Toggle bit

   * - 6

     - 6 to 11

     - Address

   * - 6

     - 0 to 5

     - Command
```

rc-5x-20 (RC_PROTO_RC5X_20)

This rc-5 extended to encoded 20 bits. The is a 3555 microseconds space after the 8th bit.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\rc\[linux-master][Documentation][userspace-api][media][rc]rc-protos.rst, line 133)

Unknown directive type "flat-table".

```
.. flat-table:: rc-5x-20 bits scancode mapping
   :widths:       1 1 2

   * - rc-5-sz bits

     - scancode bit

     - description

   * - 1

     - none

     - Start bit, always set

   * - 1
```

```
      - 14

      - Address bit

    * - 1

      - none

      - Toggle bit

    * - 5

      - 16 to 20

      - Address

    * - 6

      - 8 to 13

      - Address

    * - 6

      - 0 to 5

      - Command
```

jvc (RC_PROTO_JVC)

The jvc protocol is much like nec, without the inverted values. It is described here https://www.sbprojects.net/knowledge/ir/jvc.php.

The scancode is a 16 bits value, where the address is the lower 8 bits and the command the higher 8 bits; this is reversed from IR order.

sony-12 (RC_PROTO_SONY12)

The sony protocol is a pulse-width encoding. There are three variants, which just differ in number of bits and scancode encoding.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\rc\[linux-master][Documentation][userspace-api][media][rc]rc-protos.rst, line 194)

Unknown directive type "flat-table".

```
.. flat-table:: sony-12 bits scancode mapping
   :widths:       1 1 2

   * - sony-12 bits

     - scancode bit

     - description

   * - 5

     - 16 to 20

     - device

   * - 7

     - 0 to 6

     - function
```

sony-15 (RC_PROTO_SONY15)

The sony protocol is a pulse-width encoding. There are three variants, which just differ in number of bits and scancode encoding.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-

master\Documentation\userspace-api\media\rc\[linux-master][Documentation][userspace-api][media][rc]rc-protos.rst, line 221)

Unknown directive type "flat-table".

```
.. flat-table:: sony-12 bits scancode mapping
   :widths:       1 1 2

   * - sony-12 bits

     - scancode bit

     - description

   * - 8

     - 16 to 23

     - device

   * - 7

     - 0 to 6

     - function
```

sony-20 (RC_PROTO_SONY20)

The sony protocol is a pulse-width encoding. There are three variants, which just differ in number of bits and scancode encoding.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\rc\[linux-master][Documentation][userspace-api][media][rc]rc-protos.rst, line 248)

Unknown directive type "flat-table".

```
.. flat-table:: sony-20 bits scancode mapping
   :widths:       1 1 2

   * - sony-20 bits

     - scancode bit

     - description

   * - 5

     - 16 to 20

     - device

   * - 7

     - 0 to 7

     - device

   * - 8

     - 8 to 15

     - extended bits
```

nec (RC_PROTO_NEC)

The nec protocol encodes an 8 bit address and an 8 bit command. It is described here https://www.sbprojects.net/knowledge/ir/nec.php. Note that the protocol sends least significant bit first.

As a check, the nec protocol sends the address and command twice; the second time it is inverted. This is done for verification.

A plain nec IR message has 16 bits; the high 8 bits are the address and the low 8 bits are the command.

nec-x (RC_PROTO_NECX)

Extended nec has a 16 bit address and a 8 bit command. This is encoded as a 24 bit value as you would expect, with the lower 8 bits

the command and the upper 16 bits the address.

nec-32 (RC_PROTO_NEC32)

nec-32 does not send an inverted address or an inverted command; the entire message, all 32 bits, are used.

For this to be decoded correctly, the second 8 bits must not be the inverted value of the first, and also the last 8 bits must not be the inverted value of the third 8 bit value.

The scancode has a somewhat unusual encoding.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\rc\[linux-master][Documentation][userspace-api][media][rc]rc-protos.rst, line 307)

Unknown directive type "flat-table".

```
.. flat-table:: nec-32 bits scancode mapping

   * - nec-32 bits

     - scancode bit

   * - First 8 bits

     - 16 to 23

   * - Second 8 bits

     - 24 to 31

   * - Third 8 bits

     - 0 to 7

   * - Fourth 8 bits

     - 8 to 15
```

sanyo (RC_PROTO_SANYO)

The sanyo protocol is like the nec protocol, but with 13 bits address rather than 8 bits. Both the address and the command are followed by their inverted versions, but these are not present in the scancodes.

Bis 8 to 20 of the scancode is the 13 bits address, and the lower 8 bits are the command.

mcir2-kbd (RC_PROTO_MCIR2_KBD)

This protocol is generated by the Microsoft MCE keyboard for keyboard events. Refer to the ir-mce_kbd-decoder.c to see how it is encoded.

mcir2-mse (RC_PROTO_MCIR2_MSE)

This protocol is generated by the Microsoft MCE keyboard for pointer events. Refer to the ir-mce_kbd-decoder.c to see how it is encoded.

rc-6-0 (RC_PROTO_RC6_0)

This is the rc-6 in mode 0. rc-6 is described here https://www.sbprojects.net/knowledge/ir/rc6.php. The scancode is the exact 16 bits as in the protocol. There is also a toggle bit.

rc-6-6a-20 (RC_PROTO_RC6_6A_20)

This is the rc-6 in mode 6a, 20 bits. rc-6 is described here https://www.sbprojects.net/knowledge/ir/rc6.php. The scancode is the exact 20 bits as in the protocol. There is also a toggle bit.

rc-6-6a-24 (RC_PROTO_RC6_6A_24)

This is the rc-6 in mode 6a, 24 bits. rc-6 is described here https://www.sbprojects.net/knowledge/ir/rc6.php. The scancode is the exact 24 bits as in the protocol. There is also a toggle bit.

rc-6-6a-32 (RC_PROTO_RC6_6A_32)

This is the rc-6 in mode 6a, 32 bits. rc-6 is described here https://www.sbprojects.net/knowledge/ir/rc6.php. The upper 16 bits are the vendor, and the lower 16 bits are the vendor-specific bits. This protocol is for the non-Microsoft MCE variant (vendor != 0x800f).

rc-6-mce (RC_PROTO_RC6_MCE)

This is the rc-6 in mode 6a, 32 bits. The upper 16 bits are the vendor, and the lower 16 bits are the vendor-specific bits. This protocol is for the Microsoft MCE variant (vendor = 0x800f). The toggle bit in the protocol itself is ignored, and the 16th bit should be takes as the toggle bit.

sharp (RC_PROTO_SHARP)

This is a protocol used by Sharp VCRs, is described here https://www.sbprojects.net/knowledge/ir/sharp.php. There is a very long (40ms) space between the normal and inverted values, and some IR receivers cannot decode this.

There is a 5 bit address and a 8 bit command. In the scancode the address is in bits 8 to 12, and the command in bits 0 to 7.

xmp (RC_PROTO_XMP)

This protocol has several versions and only version 1 is supported. Refer to the decoder (ir-xmp-decoder.c) to see how it is encoded.

cec (RC_PROTO_CEC)

This is not an IR protocol, this is a protocol over CEC. The CEC infrastructure uses rc-core for handling CEC commands, so that they can easily be remapped.

imon (RC_PROTO_IMON)

This protocol is used by Antec Veris/SoundGraph iMON remotes.

The protocol describes both button presses and pointer movements. The protocol encodes 31 bits, and the scancode is simply the 31 bits with the top bit always 0.

rc-mm-12 (RC_PROTO_RCMM12)

The rc-mm protocol is described here https://www.sbprojects.net/knowledge/ir/rcmm.php. The scancode is simply the 12 bits.

rc-mm-24 (RC_PROTO_RCMM24)

The rc-mm protocol is described here https://www.sbprojects.net/knowledge/ir/rcmm.php. The scancode is simply the 24 bits.

rc-mm-32 (RC_PROTO_RCMM32)

The rc-mm protocol is described here https://www.sbprojects.net/knowledge/ir/rcmm.php. The scancode is simply the 32 bits.

xbox-dvd (RC_PROTO_XBOX_DVD)

This protocol is used by XBox DVD Remote, which was made for the original XBox. There is no in-kernel decoder or encoder for this protocol. The usb device decodes the protocol. There is a BPF decoder available in v4l-utils.