Pronto IR Formats

Preamble

This article describes in detail the formats of IR code filing for Pronto and ProntoPro models (RU-890, RU-940, RU-970, TS-1000, TSU-2000, TSU-6000, RC-5000, RC-5000i, RC-5200, RC-9200, RAV-2000, USR-5). It is supposed to be interesting for
- those, who want to clean up his IR codes – in order to find out what all the numbers in field “IR code:” mean and for troubleshooting;
- those, who want to develop an IR code converter into other driver/configuration formats – Crestron (.IR), AMX (.IRL), Xantech (.PAL), Niles (.LIN), ADI Ocelot (.LIR), RedRat2 (.TXT), Denon RC-8000 (.RCX), ProntoNeo (.NCF), ProntoNG (.PCF) etc, as a manual of most popular and complicated IMHO IR code format.

It is advisable for you to have a notion of modulated IR signals transmission [1], but I will also coin that terms for completeness. References to some concepts can be earlier than the description of theirs, so as not to touch on same concept doubly. I am not going to mention the details of realization which is not essential for the description of the format.

Acknowledgments

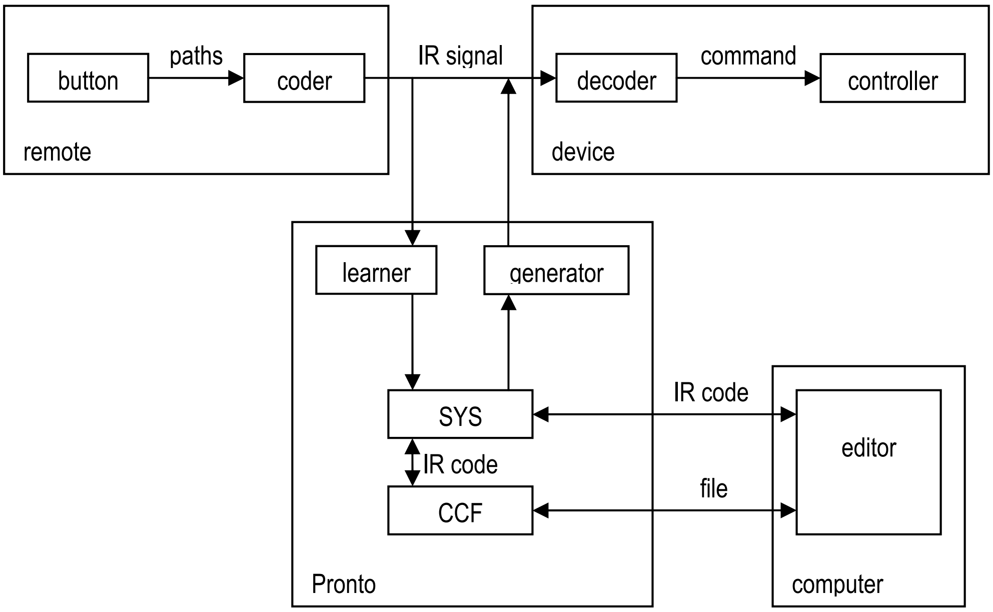
I would like to thank Daniel Tonks, Stewart Allen, Barry Gordon, Marcel Majoor, Steven Keyser, Bertrand Gillis, Loran Richardson, Bernard Barrier, Barry Shaw, Rob Crowe, Andrea Whitlock, those, who reply at forums - for help and support, my wife – for indulgence and star50fiveoh - for sober intolerance to this article’s subject :)

Warning

Standard warning about “… your own risk” is in effect.

Buttons, signals, commands and codes

In order to avoid involving in terms, I specify them precisely:

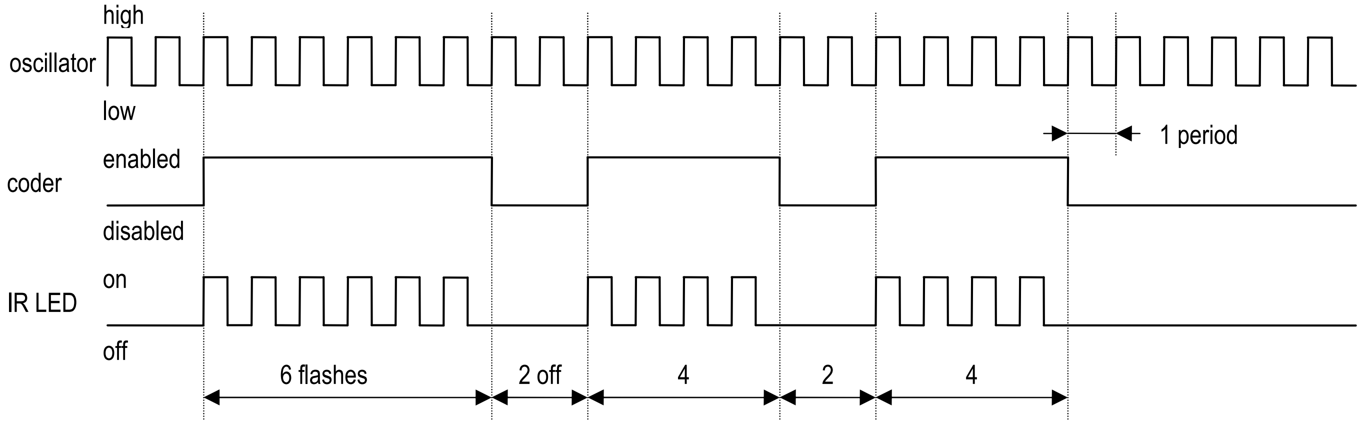


Picture 1: Interaction of buttons, IR signals, commands and IR codes

The button, IR signal, command and IR code are not corresponding mutually:
- the same button can produce different IR signals (see Toggle Codes);
- different IR signals can be recognized by decoder as same command (see Clean and Dirty Codes);
- the same IR signal can be encoded as a number of IR codes in different formats.

Modulated IR signal

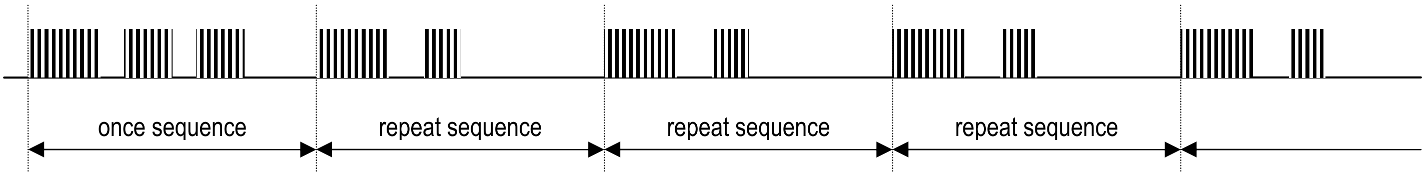
For some reasons, mainly for simplicity and interference immunity, almost all IR signals, currently used to IR translation, have the same structure. IR transmitter includes a square-wave generator (oscillator), coder and IR LED. All the time of transfer oscillator generates square impulses with fixed (for this remote) carrier frequency; depending on pressed button (and, may be, remote mode), coder forms a code of command – sequence of conventional logical 0s and 1s; IR LED flashes this command modulated by generated impulses:



Picture 2: IR LED replays modulated IR signal

Any modulated IR signal can be fully characterized via carrier frequency and the sequence of period amounts when emitting is on and off. For picture 2 this sequence is 6-2-4-2-4-...

Sometimes remote emits the signal all the time until the button such as VOLUME+ will released. Another way it flashes once per press, or first, once emitted signal is not the same that repeating one, followed it. So, it is convenient to suppose that in the general way IR signal consists of once part, the start emitting sequence and repeat part. For example, endless IR signal 6-2-4-2-4-6-6-3-3-12-6-3-3-12-…



Picture 3: Cutting IR signal into once sequence and repeat sequence

has the once sequence 6-2-4-2-4-6 and repeat sequence 6-3-3-12: 6-2-4-2-4-6 | 6-3-3-12. All currently used IR signals can be represented either as a union of once and repeat sequences or as the only once or only repeating sequence.

Usually, IR codes, describing IR signal this way, are not so long. If learned IR code is not an air conditioner code, it can be completely go in “IR code:” field at ProntoEdit.
In the most of cases carrier frequencies are nearby 35KHz, but you will certainly be lucky enough to try a remote with 175KHz, 345KHz, 455KHz or 1.1MHz.

“Clean” and “dirty” codes

[1: “Unclean” and “clean” IR commands (Working With Prontoedit: Learning & Infrared)]
It is not necessary to replay IR signal precisely, IR receiver will “identify” received IR signal as correct command, if it looks like real command adequately, say, carrier frequency and burst pairs’s lengths differ from original IR signal less than for 10%. I.e. there is precise, original IR signal for every command, may be not replay-able for Pronto, and also many IR signals that Pronto can replay and IR receiver can recognize as that command. The codes of those signals may have different lengths, and the shortest one is “clean” code, and the all other codes are “dirty” and very “dirty”. There are some reasons to prefer “clean” codes:

- IR signal of “clean” IR code is more recognizable by IR receiver
- some “dirty” IR codes can lead IR receiver astray or halt it
- “clean” code is shorter, and replay of it takes less time in macros
- you can visually check “clean” IR code’s accuracy – the lengths of every “clean” IR codes per remote are the same, except, may be, buttons like VOLUME+
- “clean” code is more convenient to analyzing, in order to guess a discrete command, absent at this remote
- using of “clean” IR codes stirs ambitions of punctual Prontoyers

The ordinal way to obtain “clean” IR code is cutting from existing one [1]. Sometimes it is useful to develop special software - generator of all concrete type IR codes – in order to “discover” all possible commands, supported by that device, including those absent at remotes.
Note: There is no difference in IR learning to Pronto standalone or to computer via Pronto, but sometimes this process is faulted due a communication error.

Toggle IR Codes

[1: Why won't my buttons work twice in a row? (Working With Prontoedit: Learning & Infrared)]

It is necessary not to mix up toggle IR *codes* and toggle IR *commands*. Toggle Command is a command with toggle function such as standby/on, see TOAD at [1]. In contrast to that, Toggle IR Code contains as minimal 1 toggle bit, see RC5 at [2].

There are many possible troubles and noises at IR transfer – sun, fluorescent lamps, interference with another IR transfer, dust, pets, household and tremor of hesitating hand, holding remote. First two troubles can be cured by signal modulation. Other problems, concerning temporary obstacle, are solve logically. Main question with it is to discriminate noise and "double click", second button press. Commonly they use two different IR signals per button – one for odd presses and another for even. As a rule, these codes differ in one or two logical bits ("toggle" bit or "parity"), and both of them encode the same command. When IR decoder receives IR signal such type, it ignores the same signals (or signals with the same "parity" bit) received twice, in order to avoid taking noise as double click.

Pronto IR learning procedure uses the only button pressing, and because of it Pronto can detect as toggle codes only codes of some predefined types, that look like known codes, for example, RC5. When Pronto replays these codes, it emits in turn odd and even IR signals for every type of predefined code format (and if existing, subformat). So the common way to emulate "long button press" in macro, repeating of the same alias, is not effective for toggle codes; it is necessary to convert this code to format `0000` to do it.

In case when Pronto learns really toggle codes as ordinal, it is necessary to learn all codes in same parity, and only the "blank" command that doing nothing (if it exists at remote) – in opposite parity, and any Pronto button must have two commands at action list – odd real command and even "blank" [1].

Pronto IR code filling formats

[1: Type of code (Working With Prontoedit: Learning & Infrared)]

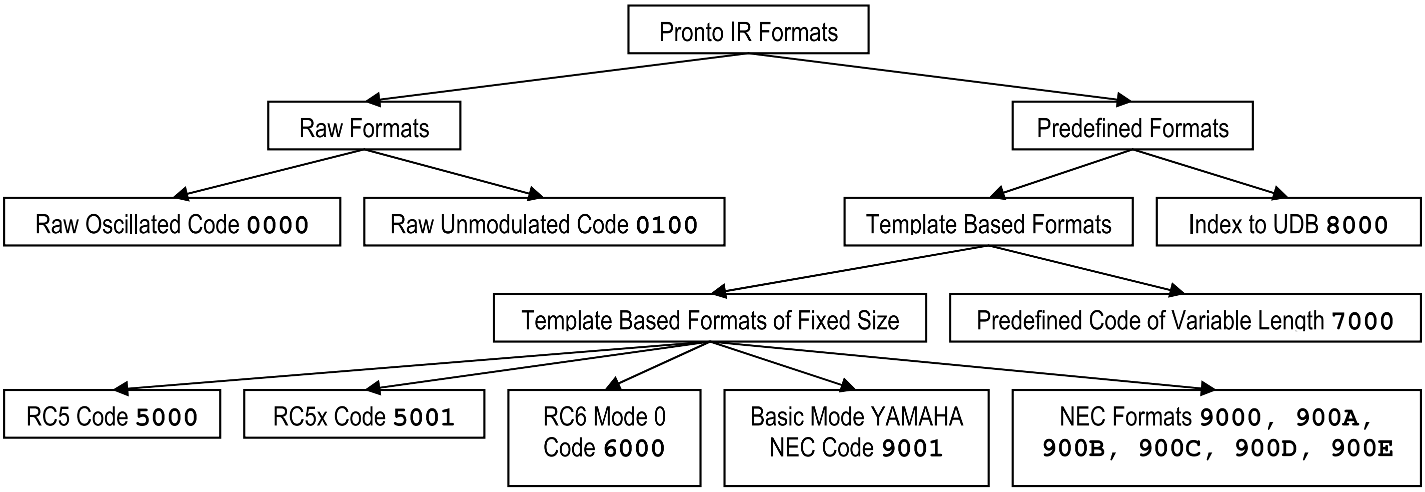
IR codes are kept in Pronto and in CCFs in bytes (as all data), but Pronto software uses two-byte hexadecimal words delimited by spaces at IR code view (editBox):

`0000 0070 0003 0002 0006 0002 0004 0002 0004 0006 0006 0003 0003 000C`,

so I will write all sizes and offsets in 16bit words and all numbers in hex, if not specified other. Also, all tables are based on current (Feb 2003) software and firmware versions.

First word in IR code - wFmtID – identifies IR format, so that its value explains how to use all other words in code. Currently the following formats are used:

`0000` – raw oscillated code
`0100` – raw unmodulated code
`5000` – RC5
`5001` – RC5x
`6000` – RC6 Mode 0
`7000` – predefined code of variable length
`8000` – index to UDB
`9000, 900A, 900B, 900C, 900D, 900E` – NEC
`9001` – basic mode YAMAHA NEC code


Pronto IR Formats
Raw Formats
Predefined Formats
Raw Oscillated Code 0000
Raw Unmodulated Code 0100
Template Based Formats
Index to UDB 8000
Template Based Formats of Fixed Size
Predefined Code of Variable Length 7000
RC5 Code 5000
RC5x Code 5001
RC6 Mode 0 Code 6000
Basic Mode YAMAHA NEC Code 9001
NEC Formats 9000, 900A, 900B, 900C, 900D, 900E

Picture 4: hierarchy of Pronto IR formats

Raw formats (wFmtID = `0000` or `0100`)

They are simple, basic, most commonly used formats. Almost all IR signals can be represented (exactly or cognitive) either in `0000` or `0100` format. There is no way to encode toggle IR codes at these formats, but separately, odd and even IR code can be converted from toggle form to `0000` format.

Raw oscillated code (wFmtID = `0000`)

This is the format that is often implied, when they say "Pronto IR format". It contains lengths while LED flashes and while LED is off in carrier periods ("burst pairs") in two optional parts - once and repeat sequences.

For example, code from pic.2 and 3 will encoded in this format as `0000 0070 0003 0002 0006 0002 0004 0002 0004 0006 0006 0003 0003 000C`, where:

offset	size	name	type	description	sample
0	1	wFmtID	word, ID	Format ID. Must be `0000` for this format	`0000`
1	1	wFrqDiv	word, positive	Carrier frequency divider	`0070`
2	1	nOnceSeq	word, length	Number of burst pairs at once sequence	`0003`
3	1	nRepeatSeq	word, length	Number of burst pairs at repeat sequence	`0002`
4	2* nOnceSeq	aOnceSeq	array of rBurstPair	Once sequence	`0006 0002 0004 0002 0004 0006`
4+2* nOnceSeq	2* nRepeatSeq	aRepeatSeq	array of rBurstPair	Repeat sequence	`0006 0003 0003 000C`

and rBurstPair consists of:

offset	size	name	type	description	sample
0	1	wLEDflash	word, positive	Amount of periods when LED flashes with carrier	`0006`
1	1	wLEDoff	word, positive	Amount of periods when LED is off	`0002`

Details:

wFmtID: word = `0000`, Format ID

wFrqDiv: word in range `0001..FFFF`, wFrqDiv = 4,145146 MHz / <*signal carrier*>. So wFrqDiv = `0001` corresponds to signal carrier ≈ 4,1 MHz, and wFrqDiv =`FFFF` ~ 63Hz. I have measured this constant indirectly and have obtained 4,1455±0,0006 MHz, which is close enough with magical number 4,145146 [2], and far enough from 4,194304 [5], but I can't locate quartz oscillator with this nominal at catalogues.

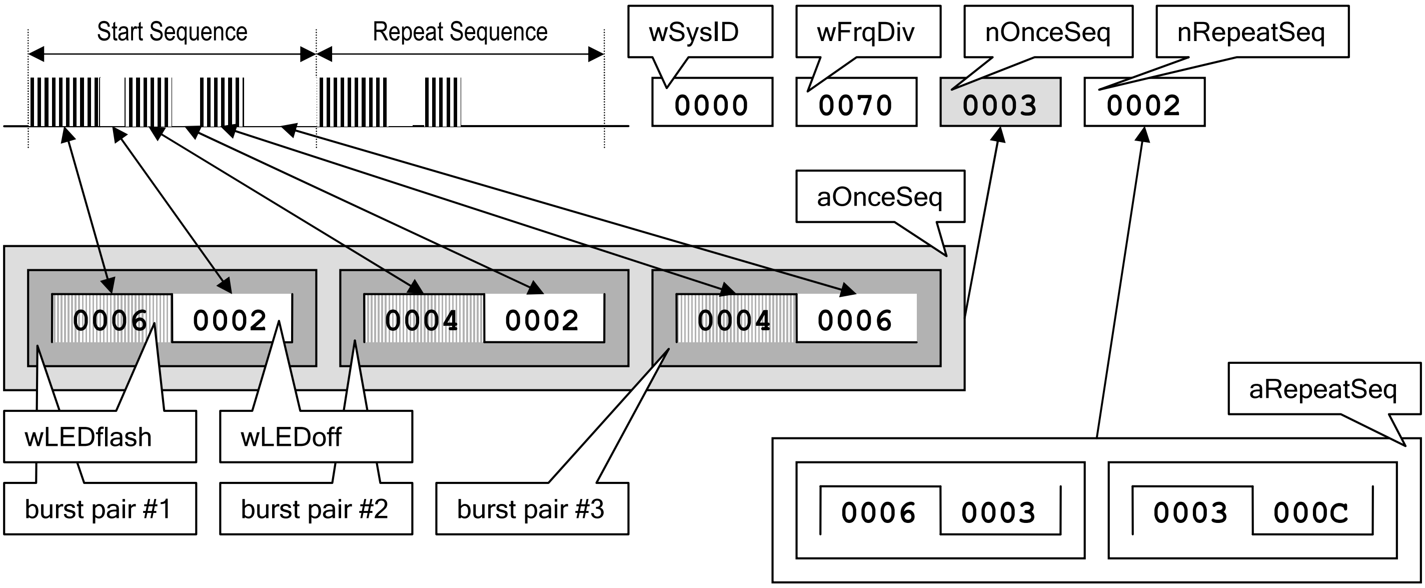
nOnceSeq: word in range `0000..0100`, is equal to amount of burst pairs at once sequence

nRepeatSeq: word in range `0000..0100`, is equal to amount of burst pairs at repeat sequence

wLEDflash: word in range `0001..FFFF`, is amount of carrier frequency periods when LED flashes every first half-period and turned off for the last half-period

wLEDoff: word in range `0001..FFFF`, is amount of carrier frequency periods when the LED is off

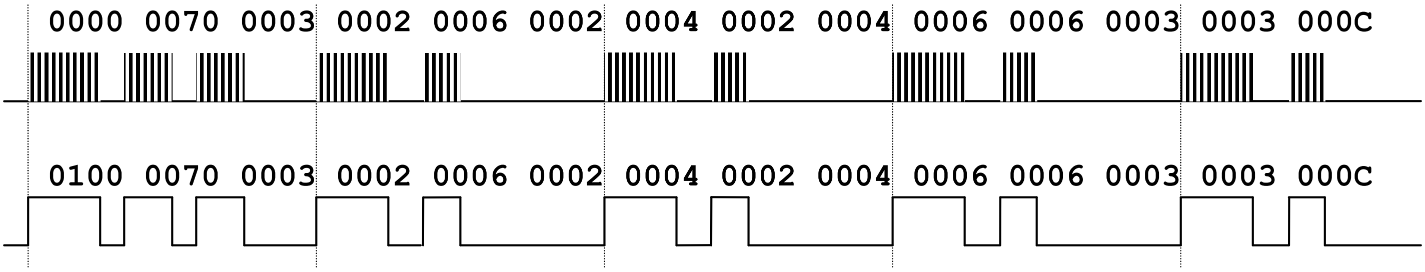
Next picture will dispel all residuary questions:



Picture 5: a correspondence between IR-signal and IR-code at format 0000

Raw unmodulated code (wFmtID = `0100`)

This format is the same as `0000`, but LED is turned on (not flashes) all the time from 1st word of burst pair, let's compare:


0000 0070 0003 0002 0006 0002 0004 0002 0004 0006 0006 0003 0003 000C
0100 0070 0003 0002 0006 0002 0004 0002 0004 0006 0006 0003 0003 000C

Picture 6: a difference between IR signal of IR codes of formats `0000` and `0100`

Differences 0100 from 0000:

offset	size	name	type	description	sample
0	1	wFmtID	word, ID	Format ID. Must be `0100` for this format	`0100`

aBurstPair consists of:

offset	size	name	type	description	sample
0	1	wLEDon	word, positive	Amount of periods when LED is on	`0006`
1	1	wLEDoff	word, positive	Amount of periods when LED is off	`0002`

This format completes format 0000 to universal description of all raw IR signals, but codes at format 0100 are rare – it is exposed to sun light and other IR noises. I have learned only 2 devices with these codes – noname drape drive and Dish Satellite Positioner. Nevertheless, sometimes it is useful to use IR signal without carrier, for example, for Sony Contol-S: you can replace an IR probe at RX-77 to ordinal jack, change 0000 to 0100 at IR codes, and your Sony device will be controlled by Control-S. Some other systems also support IR signals without carrier, like Crestron's CstmFreq value = 43.

Predefined formats (wFmtID = `5000, 5001, 6000, 7000, 8000, 9000, 9001, 900A, 900B, 900C, 900D, 900E`)

There are a number of additional, predefined IR formats, supported by some reasons. They can't describe all IR signals, every type/subtype of this format represents only IR codes with specific structure, for custom "brand". Also, we need additional data tables to replay these signals. These predefined formats are more compact than corresponding raw IR codes and, as a rule, simple and "clean". I.e., if you learn IR as `5000` or `7000` and it is not a bug, then that and all other IR signals from this remote must be codes of this type.

Predefined formats have different structure, but for the reason of compatibility with format `0000`, fields wFrqDiv, nOnceSeq, nRepeatSeq leave as dummy, so that code "looks" the same. aOnceSeq and aRepeatSeq are replaced with sCode, that consist real code info. Also, nOnceSeq and nRepeatSeq must meet the condition (nOnceSeq + nRepeatSeq) * 2 = sizeOf(sCode):

offset	size	name	type	description	sample
0	1	wFmtID	word, ID	Format ID	`5000`
1	1	wFrqDiv	word, dummy	Unused word for compatibility	`0000`
2	1	nOnceSeq	word, dummy	Dummy code length for compatibility	`0000`
3	1	nRepeatSeq	word, dummy	Dummy code length for compatibility	`0001`
4	(nOnceSeq+nRepeatSeq)*2	sCode	structure	Predefined code	`0000 0000`

Not all of the predefined formats are supported by any Pronto model – here a table of compatibility is:

wFmtID	RU-890, RU-940, TS-1000, RC-5000, RC-5000i	TSU-2000	RC-5200, RC-9200	TSU-6000, RU-970, USR-5	RAV-2000
`0000, 0100, 5000, 5001, 6000, 7000`	+	+	+	+	+
`8000`		+		+	+
`9000`			+	+	+
`9001`					+
`900A, 900B, 900C, 900D, 900E`			+	+	

Also, RC-3200 does not support all of this formats directly (except `0000`), but RC-3200 Setup converts codes **`0100, 5000, 5001`** and **`6000`** codes into format **`0000`** automatically, and this feature can be used from elsewhere!
Note: 0000 format used by RC-3200 is extended by including support of unmodulated IR signals with special value of wFrqDiv = `0001`.

When I describe these formats, I will translate them to format `0000`, in anticipation of this converter is already exist.

Template based formats (wFmtID = `5000, 5001, 6000, 7000, 9000, 9001, 900A, 900B, 900C, 900D, 900E`)

Template based formats are used for memory saving, for representing toggle IR codes, and for encoding IR signals with high carrier frequency.
Segment SYS at Pronto Firmware contains 2 tables for encoding/decoding IR codes these types:

<table>
<tr><th>dID</th><th>zSystem</th><th>zTemplate</th><th>zMask</th><th>dFrqDiv</th><th>bU1</th><th>dU2</th><th>dU3</th><th>dU4</th></tr>
<tr><td>00</td><td>rc5</td><td>|1[01]{01}[01]%11R</td><td>5000h + {4-8}{!2,9-14}</td><td>73</td><td>0</td><td>2</td><td>0</td><td>0A</td></tr>
<tr><td>01</td><td>rc6m0</td><td>|H1000{tT}[01]%16R</td><td>6000h + {7-14}{15-22}</td><td>73</td><td>0</td><td>0</td><td>0</td><td>0A</td></tr>
<tr><td>02</td><td>rc5e</td><td>|1[01]{01}[01]%5S[01]%12R</td><td>5001h + {4-8}{!2,10-15}{15-20}</td><td>73</td><td>0</td><td>3</td><td>0</td><td>0A</td></tr>
<tr><td>03</td><td>b&o</td><td>115[1234]*R?|<3[12345]*</td><td>7000h</td><td>-09</td><td>0</td><td>5</td><td>0</td><td>14</td></tr>
<tr><td>04</td><td>kenwood</td><td>s$[01]%32e|rp</td><td>7000h</td><td>-09</td><td>0</td><td>2</td><td>1</td><td>14</td></tr>
<tr><td>05</td><td>pioneer</td><td>o$[01]%32[RSTUo] |r[01rRSTUo]*</td><td>7000h</td><td>-04</td><td>0</td><td>4</td><td>1</td><td>14</td></tr>
<tr><td>06</td><td>ehrep</td><td>s[abcderst]%4[abcderst]* |s[abcderst]%4[abcderst]*</td><td>7000h</td><td>-0C</td><td>0</td><td>4</td><td>0</td><td>14</td></tr>
<tr><td>07</td><td>ehonce</td><td>s[abcderst]%4[abcderst]*</td><td>7000h</td><td>-0C</td><td>0</td><td>4</td><td>0</td><td>14</td></tr>
<tr><td>08</td><td>grundig16ac</td><td>|P{ac}[abcd]%7r</td><td>7000h</td><td>88</td><td>0</td><td>0</td><td>0</td><td>06</td></tr>
<tr><td>09</td><td>grundig16bd</td><td>|P{bd}[abcd]%7r</td><td>7000h</td><td>88</td><td>0</td><td>0</td><td>0</td><td>06</td></tr>
<tr><td>0A</td><td>thomson1</td><td>|2{12}[12]%9R</td><td>7000h</td><td>7C</td><td>0</td><td>1</td><td>0</td><td>0E</td></tr>
<tr><td>0B</td><td>thomson2</td><td>|{12}{12}[12]%9[RS]</td><td>7000h</td><td>7C</td><td>0</td><td>-1</td><td>0</td><td>0F</td></tr>
<tr><td>0C</td><td>thomson3</td><td>|[12]%4{12}[12]%7[RST]</td><td>7000h</td><td>7C</td><td>0</td><td>-1</td><td>0</td><td>10</td></tr>
<tr><td>0D</td><td>ferguson</td><td>|s%2{01}[01]%9[RS]</td><td>7000h</td><td>-0C</td><td>1</td><td>4</td><td>0</td><td>0D</td></tr>
<tr><td>0E</td><td>telefunken</td><td>|{01}{01}[01]%9[RS]</td><td>7000h</td><td>-18</td><td>1</td><td>-1</td><td>0</td><td>0E</td></tr>
<tr><td>0F</td><td>echostar</td><td>|[01]%5R</td><td>7000h</td><td>42</td><td>0</td><td>-1</td><td>0</td><td>07</td></tr>
<tr><td>10</td><td>saba</td><td>|{01}{01}[01]%9[RS]</td><td>7000h</td><td>-08</td><td>1</td><td>-1</td><td>0</td><td>0E</td></tr>
<tr><td>11</td><td>crown</td><td>|2{12}[12]%9[RS]</td><td>7000h</td><td>6D</td><td>0</td><td>2</td><td>0</td><td>0E</td></tr>
<tr><td>12</td><td>seleco</td><td>|2{12}[12]%9[RS]</td><td>7000h</td><td>63</td><td>0</td><td>2</td><td>0</td><td>0E</td></tr>
<tr><td>13</td><td>nec1a</td><td>I[01]%32F|R</td><td>900Ah + {8-1,16-9}{24-17,32-25}</td><td>6D</td><td>0</td><td>0</td><td>0</td><td>0E</td></tr>
<tr><td>14</td><td>nec1b</td><td>|I[01]%32F</td><td>900Bh + {9-2,17-10}{25-18,33-26}</td><td>6D</td><td>0</td><td>0</td><td>0</td><td>0E</td></tr>
<tr><td>15</td><td>nec1c</td><td>I[01]%32FI[01]%32F|R</td><td>900Ch + {8-1,16-9}{24-17,32-25} {42-35,50-43}{58-51,66-59}</td><td>6D</td><td>0</td><td>0</td><td>0</td><td>0E</td></tr>
<tr><td>16</td><td>nec1d</td><td>I[01]%32F|I[01]%32F</td><td>900Dh + {8-1,16-9}{24-17,32-25} {43-36,51-44}{59-52,67-60}</td><td>6D</td><td>0</td><td>0</td><td>0</td><td>0E</td></tr>
<tr><td>17</td><td>nec1e</td><td>|I[01]%32FI[01]%32F</td><td>900Eh + {9-2,17-10}{25-18,33-26} {43-36,51-44}{59-52,67-60}</td><td>6D</td><td>0</td><td>0</td><td>0</td><td>0E</td></tr>
<tr><td>18</td><td>nec2a</td><td>I[01]%32F|R</td><td>900Ah + {8-1,16-9}{24-17,32-25}</td><td>68</td><td>0</td><td>0</td><td>0</td><td>0E</td></tr>
<tr><td>19</td><td>nec2b</td><td>|I[01]%32F</td><td>900Bh + {9-2,17-10}{25-18,33-26}</td><td>68</td><td>0</td><td>0</td><td>0</td><td>0E</td></tr>
<tr><td>1A</td><td>nec2c</td><td>I[01]%32FI[01]%32F|R</td><td>900Ch + {8-1,16-9}{24-17,32-25} {42-35,50-43}{58-51,66-59}</td><td>68</td><td>0</td><td>0</td><td>0</td><td>0E</td></tr>
<tr><td>1B</td><td>nec2d</td><td>I[01]%32F|I[01]%32F</td><td>900Dh + {8-1,16-9}{24-17,32-25} {43-36,51-44}{59-52,67-60}</td><td>68</td><td>0</td><td>0</td><td>0</td><td>0E</td></tr>
<tr><td>1C</td><td>nec2e</td><td>|I[01]%32FI[01]%32F</td><td>900Eh + {9-2,17-10}{25-18,33-26} {43-36,51-44}{59-52,67-60}</td><td>68</td><td>0</td><td>0</td><td>0</td><td>0E</td></tr>
<tr><td>1D</td><td>nec</td><td>I[01]%32%F|R</td><td>9000h + {8-1}{16-9}{24-17}{32-25}</td><td>6D</td><td>0</td><td>0</td><td>0</td><td>0E</td></tr>
<tr><td>*</td><td>rc6m6a-24</td><td>|H1110{tT}0[01]%23R</td><td>6001h + {8-14}{15-22}{23-30}</td><td>73</td><td>0</td><td>?</td><td>0</td><td>?</td></tr>
<tr><td>**</td><td>rc6m6a-32</td><td>|H1110{tT}1[01]%31R</td><td>6001h + {8-22}{23-30}{31-38}</td><td>73</td><td>0</td><td>?</td><td>0</td><td>?</td></tr>
<tr><td>***</td><td>yamahanec</td><td>?I[01]%32F|R</td><td>?9001h + {8-1}{24-17}, + {!16-!9}{!32-!25}</td><td>?6D</td><td>0</td><td>0</td><td>0</td><td>?</td></tr>
</table>

dID	bCh	[i]	aBurstSeq
00	R	0	-0CA0
00	0	1	0020 -0020
00	1	2	-0020 0020
01	H	0	0060 -0020
01	T	1	0020 -0020
01	t	2	-0020 0020
01	0	3	-0010 0010
01	1	4	0010 -0010
01	R	5	-0BC0
02	R	0	-0AA0
02	S	1	-0080
02	0	2	0020 -0020
02	1	3	-0020 0020
03	R	0	005B -C422
03	5	1	005B -1BC0
03	4	2	005B -1622
03	3	3	005B -1083
03	2	4	005B -0AE4
03	1	5	005B -0546
04	p	0	0103 -AD01
04	e	1	0103 -47D5
04	s	2	0103 -1745
04	r	3	1010 -040C
04	0	4	0103 -0309
04	1	5	0103 -0103
04	s$	6	1010 -081E
05	r	0	227B -1160

dID	bCh	[i]	aBurstSeq
05	R	1	022C -8FFA
05	S	2	022C -6108
05	T	3	022C -4165
05	o	4	022C -31AF
05	U	5	022C -2C13
05	0	6	022C -0684
05	1	7	022C -022C
05	o$	8	227B -1160
06, 07	r	0	00FC -347B
06, 07	a	1	00FC -00FC
06, 07	e	2	00FC -007E
06, 07	t	3	007E -347B
06, 07	s	4	007E -0274
06, 07	b	5	007E -017A
06, 07	c	6	007E -00FC
06, 07	d	7	007E -007E
08, 09	P	0	0019 -005A 0028
08, 09	a	1	-0044 0024
08, 09	b	2	-0036 0012 -000E 0012
08, 09	c	3	-0022 0012 -0022 0012
08, 09	d	4	-000E 0012 -0036 0012
08, 09	r	5	-07FC
0A	R	0	0006 -0857
0A	2	1	0006 -011A
0A	1	2	0006 -00BB
0B	R	0	0010 -0857
0B	S	1	0010 -0642
0B	2	2	0010 -0112
0B	1	3	0010 -00B1
0C	R	0	0010 -059D
0C	S	1	0010 -0417
0C	T	2	0010 -02FB
0C	2	3	0010 -0099
0C	1	4	0010 -0044
0D	R	0	0008 -4B7B
0D	S	1	0008 -3A74
0D	0	2	0008 -0A2B
0D	1	3	0008 -06C3
0D	s	4	0008 -0515
0E	R	0	0008 -2BAE
0E	S	1	0008 -2104
0E	0	2	0008 -05CA
0E	1	3	0008 -03D9
0F	R	0	0019 -024C
0F	0	1	0019 -010A
0F	1	2	0019 -00AF
10	R	0	0009 -74F3
10	S	1	0009 -58D2
10	0	2	0009 -0F3E
10	1	3	0009 -0A2A
11, 12	R	0	0006 -0841
11, 12	S	1	0006 -065B
11, 12	2	2	0006 -011A
11, 12	1	3	0006 -00BB
13 – 1D, ***	I	0	0157 -00AB
13 - 1D, ***	F	1	0016 -05E7
13 - 1D, ***	1	2	0016 -0040
13 - 1D, ***	0	3	0016 -0015
13, 15, 18, 1A, 1D, ***	R	4	0157 -0055 0016 -0E3B
*,**	H	0	0070 -0020
*,**	T?	1	0020 -0020
*,**	t	2	-0020 0020
*,**	0	3	-0010 0010
*,**	1	4	0010 -0010
*	R	5	-0AB0
**	R	5	-09B0

At the first table:

dID – subformat, used at format `7000` and for connection with second table

zSystem – unused string, "brand"-like description of dID

zTemplate – template of logical structure of IR signal of this type (dID)

<letters> and *<numbers>* - are references as bCh to second table, where they (with dID) point out to a concrete aBurstSeq

| – delimits once sequence and repeat sequence parts

[*<chars>*] – means any of suggested characters

{*<char1><char2>*} – corresponds to toggle bit: at odd replaying of IR code of this type must be *<char1>*, at even – *<char2>*

%*<number>* - means *<number>* of duplicates of previous character/term

* - any number of any previously defined characters at [*<chars>*]

<char>$ - strange 2-byte name for char, nothing else :)

?, < - garbage that must be ignored :)

zMask – mask, correspondence between zTemplate and IR code at Fixed Size Template Based Formats. Be described at these formats. zMask is used while learning only.

Hexadecimal word is wFmtID

{*<expression>*} – means description of corresponded word from sCode

<number> - means index at forming String Code for corresponded bit at sCode word

<number1>-*<number2>* - means a range (inversed range) of these bounds – *<number1>, <number1+(-)1>, …, <number2>*

!*<number>* – means index at String Code for logical negation of corresponded bit at sCode

dFrqDiv – carrier frequency divider. "-" means high (>58KHz?) frequency, that is need to be analyzed (when learning) with another filter, I think, so "-" must be ignored.

bU1, dU2, dU3, dU4 – additional class/format characteristics. I suppose, they are used by recognition routine, or as additional info, like index of developer, who encodes special code for this format/subformat :), and must be ignored at replaying/converting.

At the second table:

dID – same as at the first table

bCh – means index (char) from zTemplate or String Code

[i] – means index to aBurstSeq from aCode (same as wCIdx) at Predefined Codes of Variable Length (`7000`)

aBurstSeq – analogue of burst pair with variable length – from 1 to 4 words. Positive words mean time when LED flashes, negative – when LED is off.

Template Based Formats of Fixed Size (wFmtID = `5000, 5001, 6000, 9000, 9001, 900A, 900B, 900C, 900D, 900E`)

Template based formats of fixed size represent strongly defined IR signals of common brands. As a rule, IR signal, leaned at this format, is the only and "clean" (but I have trouble with RC9200 + Onkyo DVD – it learns its IR codes as **`900A`** mistakenly). IR codes at this format are short, sCode can be 2, 3 or 4 words length, and every of this word means anything like “System”, “Command” or “Data” but that meanings are not essential for converting. Now, the ranges of that codes:

```
5000 0000 0000 0001 0000 0000 – 5000 0000 0000 0001 001f 007f
```

```
5001 0000 0000 0002 0000 0000 0000 0000 – 5001 0000 0000 0002 001f 007f 003f 0000
```

Note: it looks like erratum at firmware table, really zMask for dID = 02 must be 5001h + {4-8}{!2,10-15}{**16-21**}. This bug causes a real brain pain: any RC5x code can not be learned by any Pronto! But replaying of these codes works OK, because zMask is not used for replay.

```
6000 0000 0000 0001 0000 0000 – 6000 0000 0000 0001 00ff 00ff
```

Note: [2] contains description of RC6 Mode 6A as format **`6001`** with toggle bit, as all other RC-formats. Currently software represents that codes at format `0000` without toggle bit. I have inserted corresponded (obsolete?) entries * and ** to the end of firmware tables from [2] for considerations of universality:

* - subformat of **`6001`**, where first argument (Customer Code) is in range of **`0000..007f`**

```
6001 0000 0000 0002 0000 0000 0000 0000 – 6001 0000 0000 0002 007f 00ff 00ff 0000
```

** - subformat of **`6001`**, where first argument is in range of **`8000..ffff`**

```
6001 0000 0000 0002 8000 0000 0000 0000 – 6001 0000 0000 0002 ffff 00ff 00ff 0000
```

```
9000 0000 0000 0002 0000 0000 0000 0000 – 9000 0000 0000 0002 00ff 00ff 00ff 00ff
```

```
900a 0000 0000 0001 0000 0000 – 900a 0000 0000 0001 ffff ffff
```

Note: These codes represent most common (may be, after RC) IR signal type (32bits NEC). Usually it consist of 8bit device code, 8bit device code binary compliment, 8bit function code and 8bit function code binary compliment [3,4]. All Pronto models, except RC-5200, RC-9200, TSU-6000, RU-970, USR-5 must learn this type of IR signals as **`0000 006* 0022 0002 0157`** …, and it works fine. New Pronto models can replay formats **`900A`** to **`900E`**, but recognition routine is inclined to learn all NEC IR signals as format **`900A`**, that often results in fsults.

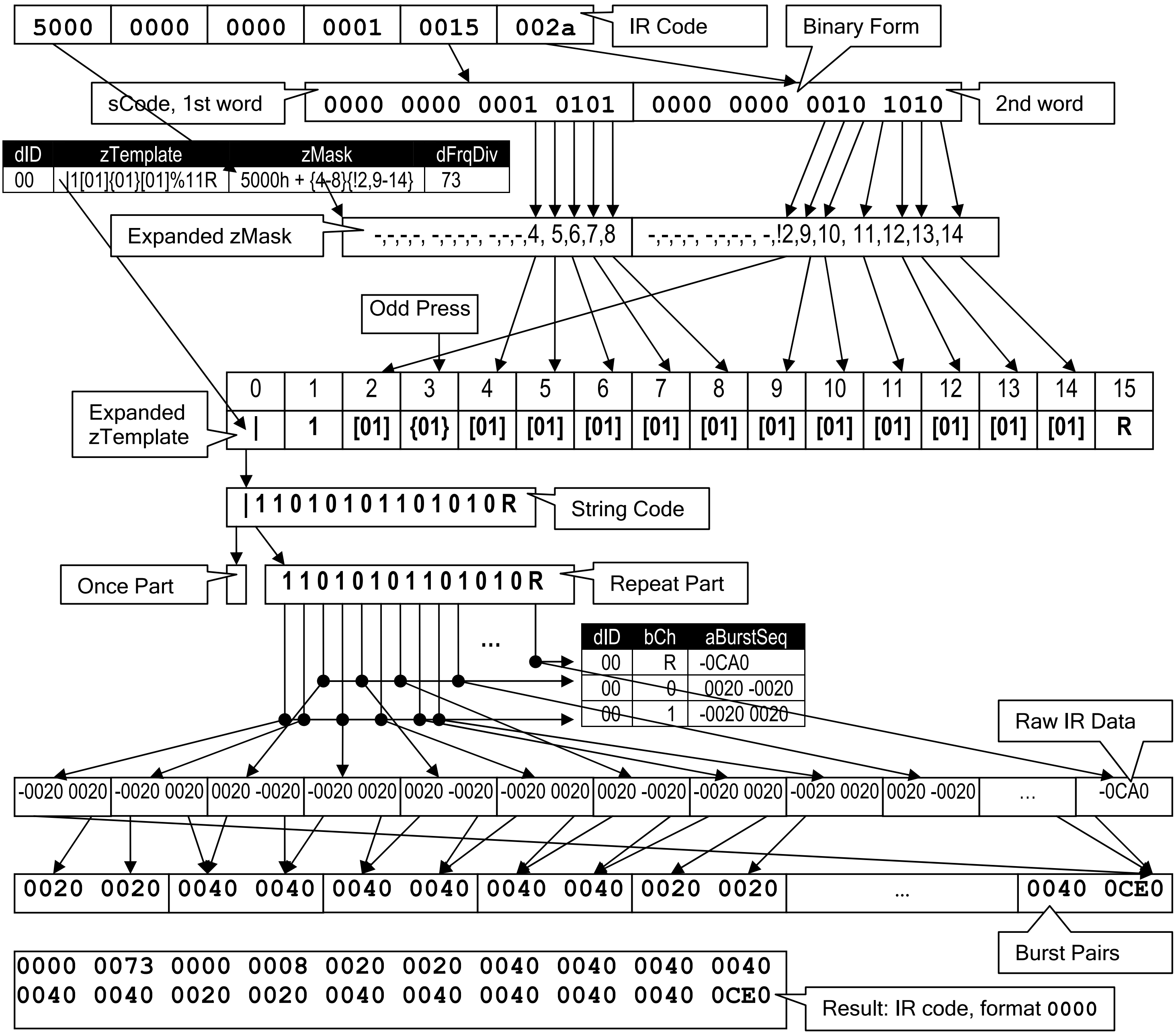
```
900b 0000 0000 0001 0000 0000 – 900b 0000 0000 0001 ffff ffff
900c 0000 0000 0002 0000 0000 0000 0000 – 900c 0000 0000 0002 ffff ffff ffff ffff
900d 0000 0000 0002 0000 0000 0000 0000 – 900d 0000 0000 0002 ffff ffff ffff ffff
900e 0000 0000 0002 0000 0000 0000 0000 – 900e 0000 0000 0002 ffff ffff ffff ffff
```

```
9001 0000 0000 0001 0000 0000 – 9001 0000 0000 0001 00ff 00ff
```

Note: format **`9001`** is supported only by RAV-2000, but absent in all tables. I can’t check it, and I hope that codes, described at [3,4] are the same, so I have inserted it to firmware tables as ***.

For converting that codes to format `0000` I offer this procedure:

1. Obtaining String Code from zTemplate, zMask and source code
2. Obtaining raw IR data by indexing from String Code
3. Forming IR code at format `0000` from raw IR data



Picture 7: Converting IR code of template based format of fixed size into format `0000`
Note: `0CE0` = -0020 (last "0" at String Code) + -0CA0 ("R") + -0020 (first "1")

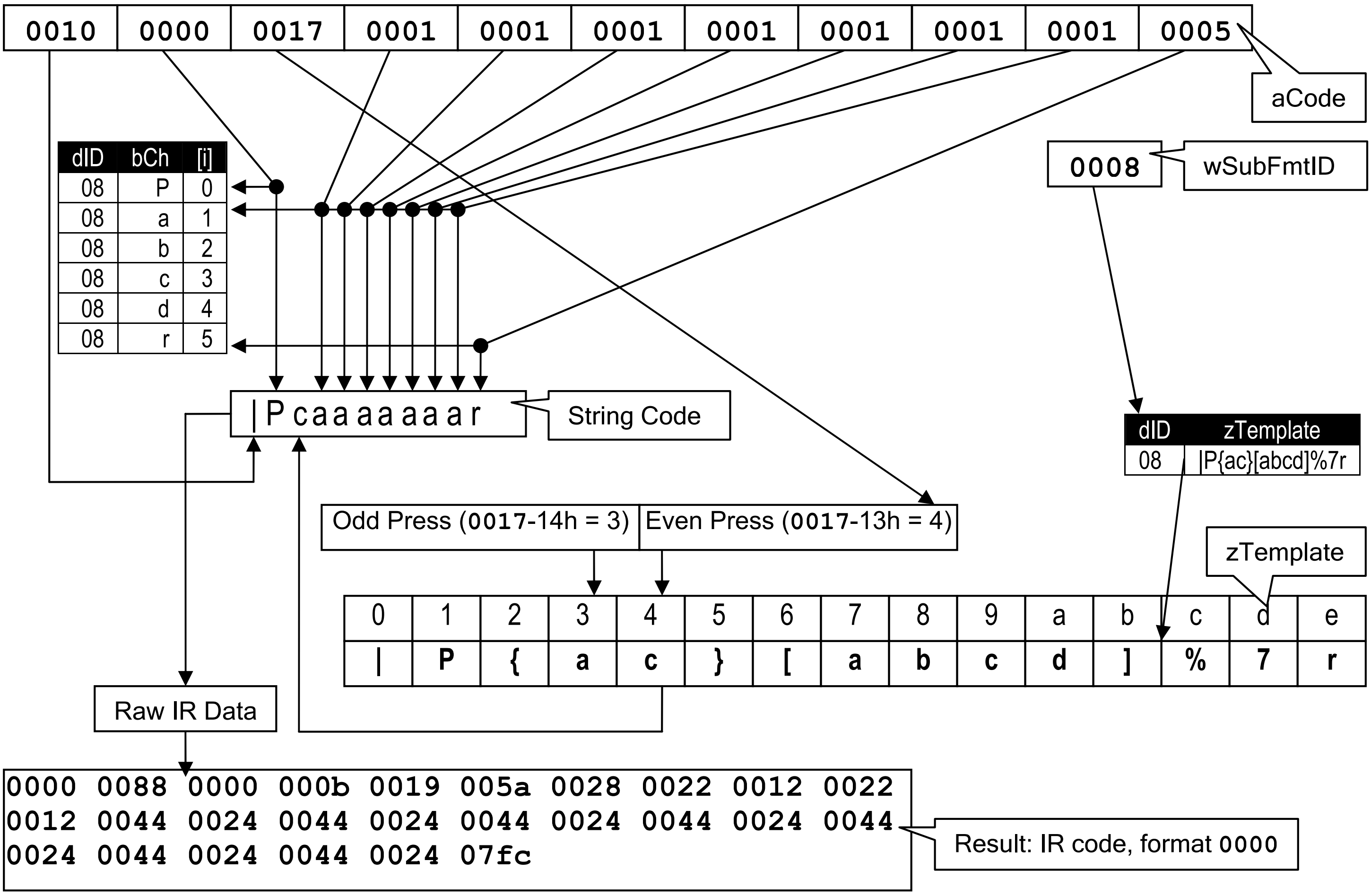
Predefined Code of Variable Length (wFmtID = `7000`)

This format is for representing IR signals that can not (yet) be easily encoded as predefined format of fixed size. The structure of this code is described on sample - Grundig `7000 0088 0000 0007 0008 000b 0010 0000 0017 0001 0001 0001 0001 0001 0001 0001 0005 0044`:

offset	size	name	type	description	sample
0	1	wFmtID	word, ID	Format ID, = 7000	`7000`
1	1	wFrqDiv	word, dummy	Unused word for compatibility	`0088`
2	1	nOnceSeq	word, dummy	Dummy code length for compatibility	`0000`
3	1	nRepeatSeq	word, dummy	Dummy code length for compatibility	`0007`
4	1	wSubFmtID	word, ID	SubFormat ID, = dID	`0008`
5	1	nCodeSeq	word, length	Length of aCode	`000b`
6	nCodeSeq	aCode	array of wCIdx	Code. Every word point a word in String Code.	`0010 0000 0017 0001 0001 0001 0001 0001 0001 0001 0005`

offset	size	name	type	description	sample
6 + nCodeSeq	0 or 1	wRest	word, dummy	Unused word for compatibility, it is present if nCodeSeq is odd	`0044`

Also, nOnceSeq and nRepeatSeq must meet the condition (nOnceSeq + nRepeatSeq) * 2 = 2 + nCodeSeq + sizeOf(wRest)
Converting procedure is the same as previous one except String Code obtaining, that I desribe now:
Character choosing to String Code depends on value of corresponded wCIdx from aCode:
- if wCIdx < `0010`, then corresponded bCh indexes from second firmware table dID bCh [i], where dID = wSubFmtID and [i] = wCIdx
- if wCIdx = `0010`, then bCh = "|"
- if wCIdx > `0010`, then this char is toggle. On odd code (of this subformat) replaying bCh is equal (wCIdx – 14h)-th char of corresponded zTemplate: bCh = zTemplate[wCIdx – 14], where first index in zTemplate is 0. On even code replaying – next char: bCh = zTemplate[wCIdx – 13]:



Picture 8: Forming String Code from IR code format `7000`

Index to UDB (wFmtID = `8000`)

Some Pronto models (RAV-2000, USR-5, TSU-2000, TSU-6000, TSU-500, TSU-3000, RU-970, RU930) supports internal IR database, others have no internal DB, but only software, and RC-3200 has no any databases:

model	DB type	file	size
TS-1000, RU-890, RU-940	IR database at ProntoEdit	rcir.mdb (Standard Jet DB)	360448
TSU-2000, TSU-6000	UDB	UDP_int.hex (internal format)	362711
RU-970	UDB	UDP_int.hex (internal format)	662090
TSU-500	UDB as part of	TSU500.dat (Standard Jet DB)	5851136
RU-930	UDB as part of	RU930.dat (Standard Jet DB)	7299072
TSU-3000	UDB	UDB_TSU3000.bin (internal format)	422622
RC-5000, RC-5000i, RC-5200, RC-9200	IR database at TSS	rcir.mdb (Standard Jet DB)	978944
RC-3200	-		
RAV-2000	UDB	ww_udp.idb (internal format)	662090
USR-5	UDB	ww_udp.idb (internal format)	661011

This format is used for recall IR codes from real UDB. UDB has clear structure – all commands in it are indexed first by Device Type (see tables); next – by Brand/Code Set and finally – by Function (depending on device type); it is really easy to custom. Accordingly, structure of this format is systematic – code contains no IR data, only function – device type, brand and device function!
For example – code of "Power On" for Amp Yamaha, code set 1 - `8000 0000 0002 0000 000a 23f0 0002 0000`:

offset	size	name	type	description	sample
0	1	wFmtID	word, ID	Format ID. Must be **8000** for this format	**8000**
1	1	wFrqDiv	word, dummy	Unused word for compartibility	**0000**
2	1	nOnceSeq	word, dummy	Dummy code length for compartibility	**0002**
3	1	nRepeatSeq	word, dummy	Dummy code length for compartibility	**0000**
4	1	wDevType	word, index	Device type (**000a** is "Amp")	**000a**
5	1	wBrandCodeSet	word, index	Brand and Code Set (**23f0** is "Yamaha-1")	**23f0**
6	1	wFunction	word, index	Function (**0002** is "Power On")	**0002**
7	1	wRest	word, dummy	Dummy - rest to fill to even number of words	**0000**

Fields wDevType, wBrandCodeSet and wFunction indexes corresponded list values from UDB:

wDevType	description
1	A. proc
2	Cable
3	CD
…	…

wFunction	description
1	Power off
2	Power on
3	Channel down
…	…

Not all of the UDB codes (format 8000) are supported by all Pronto models, also, not all of the code combinations of this format are supported by any Pronto at all – UDBs differ from model to model! It is not so hard to extract real IR codes from corresponded DB files, but it is easily to get them directly via TSS.

Please excuse my poor English. On any questions about Pronto IR – please request me to eoulianov@hotbox.ru, and I will try to answer more clearly :)

References

There are a number of articles about IR formats applied to Pronto in Internet, I will refer to nearest to www.remotecentral.com:

1. Daniel Tonks "Unofficial Philips Pronto & Marantz RC5000 FAQ", http://www.remotecentral.com
2. Marcel Majoor "Communicating with the Pronto", http://home.hccnet.nl/m.majoor
3. Barry Gordon "ProntoEdit's IR Display Format", http://the-gordons.net:8080/, http://www.remotecentral.com/features/irdisp1.htm
4. Barry Shaw, Rob Crowe, Andrea Whitlock "Yamaha extended IR codes", http://darius.mobius-soft.com/~andrea/
5. Stewart Allen, "The CCF file format", http://giantlaser.com/tonto/