The PCW Keyboard

Physically, there are three main designs of PCW keyboard:

PCW8256/8512: Grey, function keys between the main block and the number pad.
PCW9512/9512+: White, XT-style layout with function keys on the left.
PCW9256/10 : White, but with the 8256/8512 layout.Electrically, all three keyboard types are identical.

Keycaps

The keyboard layout for keyboards on dot-matrix PCWs (8256, 8512, 9256, 10) is:

Keyboards on daisywheel PCWs (9512, 9512+) have this layout:

There are a few slight differences in the keycaps (for example, "{" and "}" on the dot-matrix keyboard are replaced with "¼" and "¾" on the daisywheel). The CPMKEYS command under CP/M supports both these layouts. If a PCW has a mismatched keyboard connected (eg, an 8256 with a 9512 keyboard) the commands `CPMKEYS 8` and `CPMKEYS 9` can be used to force a particular keyboard type.

Hardware

Both keyboards use the same controller: an 8048, part number 40027.

Option links

There are three option links on the keyboard PCB:

- If LK1 is connected, the keyboard enters a self-test mode; the Shift Lock LED flickers and, rather than keypresses, the keyboard transmits test data patterns to the computer.
- If LK2 is connected, pressing Shift does not cancel Shift Lock. It also enables the use of W/A/D/X as a keyboard joystick (see below).
- LK3 does not change keyboard behaviour, except to note that this link is set. In my opinion this was a missed opportunity: if the daisywheel keyboard had come with this link set, this could have been used to detect which keyboard type was connected without the need for manual overrides.

Connection

The PCW keyboard connector is a 4-pin DIN socket. Seen from the outside of the case, it has this pinout:

Protocol

Unlike a PC keyboard, the PCW keyboard does not send scancodes. Instead, it repeatedly sends the entire keyboard state: 17 words of 12 bits each.

The information in the next two paragraphs, and the timing diagrams, have been superseded by James Ots on Hackaday: PCW Keyboard timings.

I don't know the exact timings used by the keyboard. I've guessed that they're similar to the ones used by the PC1512 keyboard, which isn't too different electrically. The PCW keyboard runs at a slower clock speed, so I have based my estimates on a 6µs quantum rather than 5µs.

By default the Data and Clock lines float high. To send a bit, the keyboard sets the Data line low or high, waits (for ~6µs), pulls Clock low, waits (again, for ~6µs) and then returns both lines to high. Before sending each word, the keyboard toggles Data twice while leaving Clock high. This may be used by the PCW's controller to maintain synchronisation.

I was reading John Elliott's keyboard protocol information, and noticed that he says that he didn't know the exact timing for the keyboard. So I thought I'd find out, and connected a PCW keyboard up to my logic analyzer.

==========

From Link above: http://www.seasip.info/Unix/Joyce/pcwkbd.html

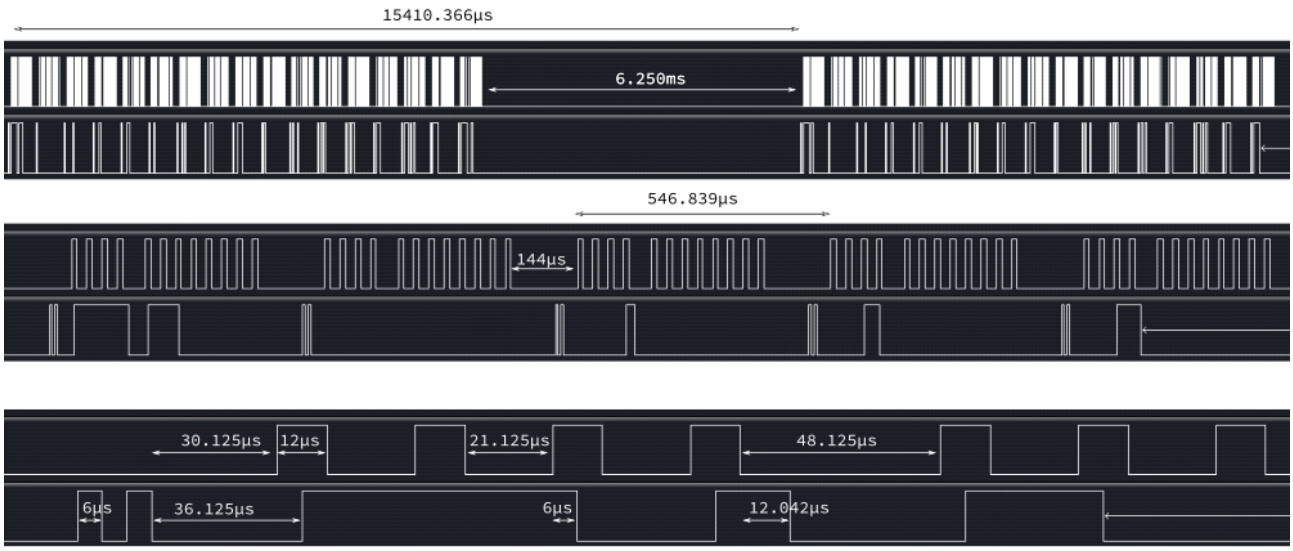
I was interested to find that although the overall idea is as he describes, the fine details of the wire protocol are actually a little different:

- John Elliott said that by default the data and clock lines float high, whereas although they are pulled high on the PCW motherboard, they are actually driven low most of the time.
- The clock signal is quite a bit faster — there's only a 21µs gap between most clock pulses, although there is a 48µs gap after the fourth clock pulse.
- The data signal is valid on the back-end of the clock signal, not at the start. (Although since his clock signal is inverted, it is actually correct that it is valid on the falling edge of the clock!)

I tried three different PCW keyboards (all from 8256s), and they all had the same timings. The values transmitted did match John Elliott's description.

I imagine the actual timing of this is fairly irrelevant though — so long as the clock signal is within certain tolerances I would guess that the gate array just clocks the signal in on the falling edge of the clock.

So here's the output from the logic analyser, with some times added. The times seemed to be fairly consistent, but I should probably have used fewer decimal places. The overall time of a transmission of the keyboard state (top chart) should be pretty accurate though, as I took the time for 50 transmissions and divided it down, and the time for one row of transmissions (second chart) was also averaged from 8 transmissions.



==========

Each 12-bit word is sent most significant bit first. The first 4 bits are an offset (0x0 to 0xF); the remaining 8 are a byte, which will be stored in PCW memory at 0x3FF0 plus the offset. As mentioned above, the full keyboard state is 17 words; the first and last words both write to offset 0xF, while the ones in between write to offsets 0, 1, 2, 3, ... , 0xE.

The meaning of the bytes in the keyboard state is:

Bits 11-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Keypad 2	Keypad 3	Keypad 6	Keypad 9	Paste	F1/F2	Keypad 0	F3/F4
1	Keypad 1	Keypad 5	Keypad 4	Keypad 8	Copy	Cut	PTR	Exit
2	[+]	½	Shift	Keypad 7	>	Return	]	Del->
3	.	?	;	<	P	[	-	=
4	,	M	K	L	I	O	9	0
5	Space	N	J	H	Y	U	7	8
6	V	B	F	G	T	R	S	6
7	X	C	D	S	W	E	3	4
8	Z	Shift Lock	A	Tab	Q	Stop	2	1
9	<-Del		J1 Fire 1	J1 fire 2	J1 right	J1 left	J1 down	J1 up
A	Alt	Keypad .	Keypad Enter	F7/F8	[-]	Cancel	Extra	F5/F6
B			J2 Fire 1	J2 fire 2	J2 right	J2 left	J2 down	J2 up
C			(KP enter)	(Space)	(KP 0)	(Exit)	(F1/F2)	(F3/F4)
C (if LK2 present)			(Shift)	(S)	(D)	(A)	(X)	(W)
D	~LK2	Shift Lock LED	(Space)	(KP 2)	(KP 3)	(KP 1)	(KP .)	(KP 5)
E	LK3	LK1 (never set)	(Shift)	(Space)	(W R P] ; > . ½)	(Q E O [L < , /)	(Z X C V B N M)	(A S D F G H J)

F	Update flag	Ticker	(Shift)	(Space)	(W R P] S F X V)	(Q E O [A D Z C)	(B N M , . / ½)	(H J K L ; < >)

Notes:

- Entries marked J1 and J2 are for joysticks. The PCW keyboard has no joystick sockets, but the controller leaves space for them in the keyboard matrix.
- Bits 5-0 of the last four bytes (0xC-0xF) are for keyboard joysticks — sets of keys that could be used directionally. The assignments correspond to the joystick entries; so bit 0 is up, bit 1 is down and so on.
- The W / A / D / X keyboard joystick is only available if link LK2 is connected.
- Bit 6 of byte 0xD returns the state of the Shift Lock LED. This is controlled by the keyboard and cannot be set by the PCW.
- The status of the three option links is reported in the top two bits of bytes 0xD and 0xE. However, if LK1 is present, the keyboard enters a self-test mode, so in normal use bit 6 of byte 0xE will never be set. For some reason the state of LK2 is inverted, so on a stock keyboard with this link disconnected, the corresponding status bit is 1.
- Bit 7 of byte 0xF is set when the keyboard is transmitting data to the host, reset when it is scanning the keys. This is the reason why a keyboard data packet is 17 bytes; the first byte sets bit 7 of byte 0xF, and the last byte resets it.
- Bit 6 of byte 0xF is toggled each time the keyboard transmits its state.

Key Matrix

The keyboard matrix is given in the service manual:

Row	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
P10	Keypad 2	Keypad 3	Keypad 6	Keypad 9	Paste	F1/F2	Keypad 0	F3/F4
P11	Keypad 1	Keypad 5	Keypad 4	Keypad 8	Copy	Cut	PTR	Exit
P12	[+]	½	Shift	Keypad 7	>	Return	]	Del->
P13	.	?	;	<	P	[	-	=
P14	,	M	K	L	I	O	9	0
P15	Space	N	J	H	Y	U	7	8
P16	V	B	F	G	T	R	S	6
P17	X	C	D	S	W	E	3	4
P24	Z	Shift Lock	A	Tab	Q	Stop	2	1
P25	<-Del	Keypad .	Keypad Enter	F7/F8	[-]	Cancel	Extra	F5/F6
P26	Alt		J1 Fire 1	J1 fire 2	J1 right	J1 left	J1 down	J1 up
P27			J2 Fire 1	J2 fire 2	J2 right	J2 left	J2 down	J2 up

Except for two lines, this matrix is identical to the first 12 bytes of the keyboard controller state returned to the PCW. Those two lines (highlighted in the table above) are deliberately swapped by the controller firmware. My guess is that initial development was done with a keyboard that did support joysticks; when these were removed the keyboard matrix was redesigned, but the controller was reprogrammed to swap the two rows to retain compatibility with the old layout.

CP/M Key Numbers

Since the PCW keyboard does not send scancodes, there are no 'key numbers' at the hardware level. CP/M assigns the following key numbers for use by programs such as SETKEYS:

- For bytes 0-8 of the key state, key *n* corresponds to bit (*n* mod 8) of byte (*n / 8*).
- For byte 9 of the key state, key 72 corresponds to bit 7.
- For byte 0xA of the key state, key *n* corresponds to bit (*n* - 73).

In expanded form:

Key number	Key
0	F3/F4
1	Keypad 0
2	F1/F2
3	Paste
4	Keypad 9
5	Keypad 6
6	Keypad 3
7	Keypad 2
8	Exit
9	PTR
10	Cut
11	Copy
12	Keypad 8
13	Keypad 4
14	Keypad 5
15	Keypad 1
16	Del->
17	]
18	Return
19	>
20	Keypad 7
21	Shift
22	½
23	[+]
24	=
25	-
26	[
27	P
28	<
29	;
30	?
31	.
32	0

33	9
34	O
35	I
36	L
37	K
38	M
39	,
40	8
41	7
42	U
43	Y
44	H
45	J
46	N
47	Space
48	6
49	S
50	R
51	T
52	G
53	F
54	B
55	V
56	4
57	3
58	E
59	W
60	S
61	D
62	C
63	X
64	1
65	2
66	Stop
67	Q
68	Tab
69	A
70	Shift Lock
71	Z
72	<-Del
73	F5/F6
74	Extra
75	Cancel

76	[-]
77	F7/F8
78	Keypad Enter
79	Keypad .
80	Alt

John Elliott 16 March 2011.