Correcting The Schematics Of Shannon's "Mind-Reading Machine"

Rainer Glaschick, Paderborn (rainer@glaschick.de)
2024-03-29

Claude E. Shannon has published the schematics of his *Mind Reading Machine*, also known as *Penny matching game*, in his publication in 1953.

Schematics in publications are seldomly error-free, as is Shannon's. The relay circuit as published was first run in a logical simulation, and finally physically created as a relay machine.

Basic test revealed that both versions in the published version did not work properly.

Two errors are quite evident and are presumed to be tiny errors in creating the drawing (`W2` instead of `W1´` and `1´` instead of `1`).

The third one was detected by observing that half of the memory was unused. No solution was found that just changes the contact designations; additional contacts have to be inserted. Two improper solutions insert a single contact, but create race conditions, that normally do not show and were not found with the simulator, where all contacts change atomically at the same time. The only proper solution found requires insertion of two contacts and to change the designation of a third one.

As Shannon's text was published three years before Hagelbarger's and does not mention the umpire and playing against his machine, the error might be detected after publication.

Logical expressions are written using `&` for *AND*, `|` for *OR*, `=` for equvalence and and `≠` for antivalence (XOR).

For a technical description of the machine see [Glaschick], which also contains timing diagrams.

Details

See also the pulse diagrams in the appendix.

1. Relay 'W2'

There is one undeniable error in the logic path for relay `W2`: It is set by `W1`, but in the schematics, it resets itself immediately by `W2`, creating a short circuit between `-` and `A` with unpredictable results, oszillating at best.

Using `W1´` instead is assumed, because then `W2` is `W1` delayed. So this change is considered stringent.

2. Writing X Memory

In the published schematics, writing to the X memory (*previous behaviour*) is done when `2´&1´`, while reading is timed by `2´&1`.

Relay `1` becomes active when a toggle switch (`TL` or `TR`) is operated, i.e. the user´s reply

evaluated. It becomes incative once the push button (PB) is pressed, i.e. the machine´s choice evaluted.

Updating the memory contents is sensible after the player´s choice.

Writing both memory bits at the same time is logical, as the cell address is the same. So it is plausibel that the extra prime is a drawing error.

Also, as S is part of the address, it is logical that writing is done during 2, while reading is done during 2´.

Furthermore, memory read (at K&1´) overlaps with write if done at 2´ instead of 2&1, which makes the whole memory obsolete as the read always delivers the current and not the previously stored value.

The original behaviour, i.e. setting the memory at the wrong time, can be selected as variant m in the emulated version.

See also next paragraphs for a deeper analysis.

3. Setting the situation relay S

In the original schematics, clock signal 2 is used to update S. Because P is copied to P1 by clock 3, they are the same when 2 ends and fixes S to always the same value as part of the address for the memory used at 1&2´ leaving half the the memory useless (detected by this effect).

Shortening the enable time to 2&3´ solves this problem with a small change (the insertion of 3´ between 2 and S). But it creates a new race condition (see below), this time for S depending on P1.

Also, when S is changed by 2, there is a race condition for the memory write X,Y via 1&2´ at transition 9 (start of 2) because S is part of the memory address.

While not a minor typographic error, this defect must be repaired for proper functioning, as tests confirmed.

No simpler solution has been found than inserting 3´ before S, despite the race condition that can be overcome by using contacts from physically the same relay.

4. Race conditions

Normal analysis of logic circutits and most simulators treat logic changes that occur simultaneously as atomic.

However, in physical systems, one change may be sufficiently earlier or later than the other to produce a small intermediate signal that may provide a different result. This is called a race condition. It may lead to either sporadic or, much better, permanent errors.

If two contacts of the same relay are concerned, the effect is often neglectible, as the time between the opening of one before closing another is short enough to have no effect. (Normally change-over contacts are of break-before-make type; special make-before-break contacts are no longer used.)

If different relays are used (or a second one in parallel to increase the number of contats) the time difference is much larger, and race conditions are more likely to cause malfunction.

When checking (quasi) simultaneous transitions, the question is if the result of one transition depends on the changes affected by another transition. In particular, logically raising and falling slopes are concerned, assuming that the latter make an input permanent (fix it), and the former change the output signal.

If both are falling slopes, there is no race, as the input is stable and the output is fixed too, shortly after the slope.

If both are rising slopes, there is also no race, as the output will be stable shortly.

If a falling slope coincides with rising slope, the input may change just before it is fixed, if the rising slope changes the output that is (part to) the other's input, giving a different result than the stable state before this event.

Thus, each falling slope must be checked if there is a rising one for the inputs to state to be changed:

Transition 8:
No conflicts, as there is no rising slope changing a signal.
Transition 9:
Rising slopes: `S` and `W2`,
Falling slopes: `MP`, `P` and `X`,`Y`
No conflict: `MP`, `P`, as these are not affected by `MP`, `P` and `S`
Conflict: `X`,`Y`; the address depends on `S`, which is changed
Transition 10:
Rising slopes: `P1` and `W1`,
Falling slopes: `MW/PW` and `W2`
No conflict: `MW/PW` is not affected by `P1` or `W1`
Conflict: `W2` depends on `W1`

No simple proper solution has yet been seen to solve race conditions; as the goal is not to make a better machine, but just to build a useable replica, no such attempt has been followed.

Tests

The (simulated) machine was tested with some test sequences that do no depend on the machine's choice.

As for the situation etc. the changes of the player's moves are used in terms of *same* or *different*, the sequences are best evaluated in these terms too.

In the following table, the possbile variants are listed. Each sequences is repeated until a stable result is obtained. Any cyclic permuations are not shown (e.g. `SDS` is the same as `SSD`, and `DDD` the same as `D`), as are the variants starting with `R`, because they will deliver effectively the same result (after some rounds).

Code	Example
S	L…
D	LR…
SD	LLRR…
SSD	LLLRRR…
SDD	LLR…

In the first three cases, the machine should win continuously; the number of wins by the player depends on the random replies given by the machine at the beginning.

In the SD case, the situations are altenatingly LSL:d and LDL:s, i.e. if the last player's choice was S, it must be D, and vice versa, i.e. the sequence SD in the situations and the sequence ds in the predictions.

The same reasoning shows that SSD cannot be caught fully, as it would require the situation sequence LSL:s, LSL:d, LDL:s; i.e. SSD in the situations and sds in the predictions.

In these last two cases, the machine wins about 2:1 in the long term.

These tests do not fully explore the machine's features, but many careless implementations do not pass these tests, in particular the last two ones.

Appendix

Problems solved in the relay version

After 10 L, followed by 10 R, the machine sticked to R and did not catch the LR (i.e. wsw) case.

The reason was a race condition in the correction for setting the relay S with the setting of relay P1. Both are controlled by timing relay 3, which used two relays in parallel, and contacts used from different relays. Moving both contacts (to control S and P1) to the same relay made the time difference small enough to be without effect.

Problems solved in the emulated version

Both, the X (previous) and Y (repeat) memories were never cleared.

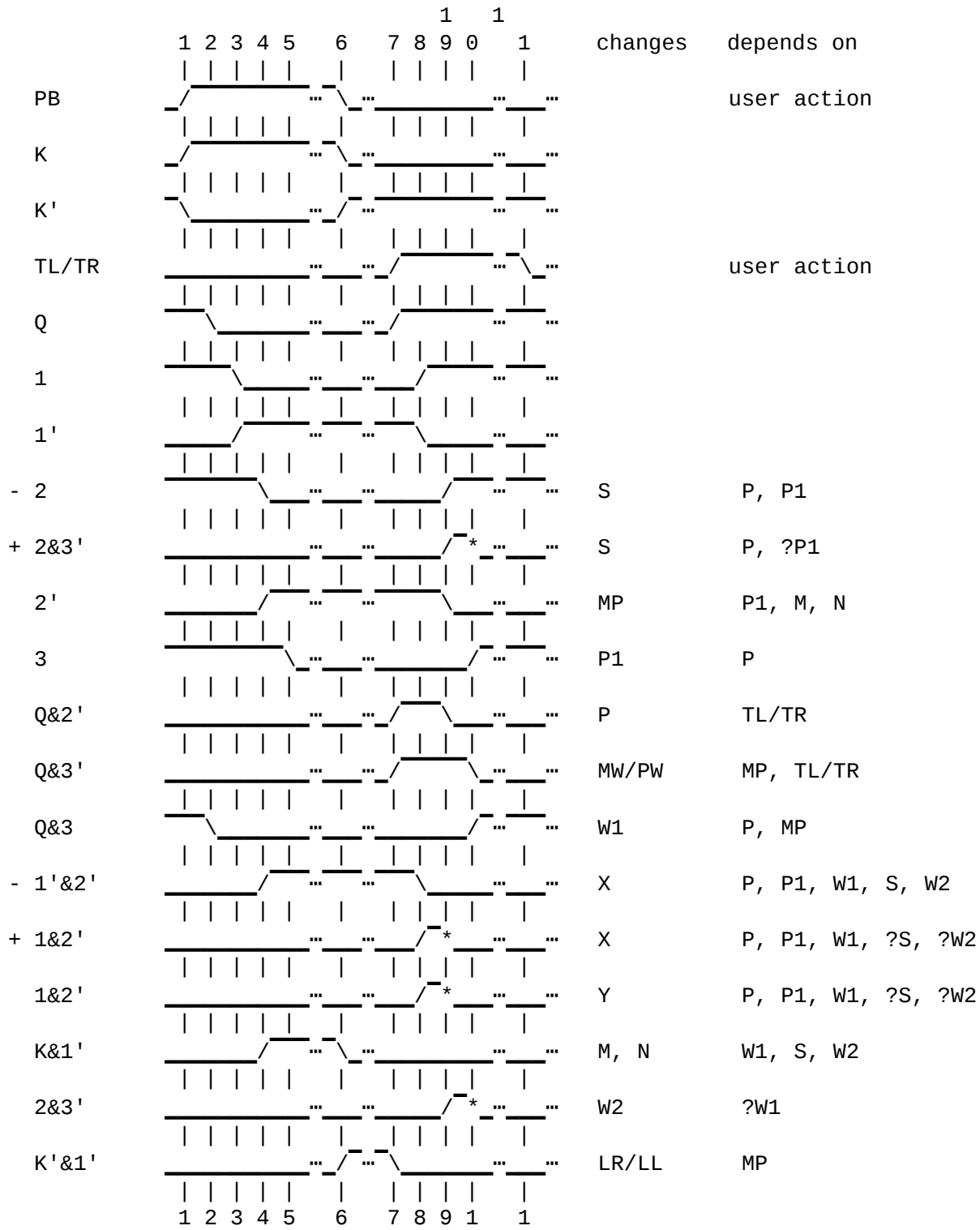
This was a structural error in the simulatation, where the result dependened on the order of the statements.

Correctly using a second set of variables for the new values and moving these en bloc solved the problem.

(The exact cause was not analyzed, as this change was necessary anyhow.)

Pulse diagram

In the following diagram, positive logic is shown, i.e. upper bars show logic true (current may flow):



Core algorithm

This is running code, currently in TUF-PL in publication syntax; will be in Python soon.

```
\ inner loop
while ?+
     \ show machine's choice and obtain player's choice
     last_p_choice = p_choice
     p_choice = players choice (m_choice)

\ player played same?
last_p_same = p_same
p_same = (last_p_choice = p_choice)

\ player wins if last machine's choice is different
last_p_win = p_win
p_win = (m_choice ~= p_choice)

\ update memory, using the last situation
ymem{situ} = (xmem{situ} = p_same)
xmem{situ} = p_same

\ new situation
situ = last_p_win, last_p_same, p_win

\ machine's choice: either a repeat or random
if ymem{situ}
    \ last is a repeat, action is logical equivalence
    m_choice = ( p_choice = xmem{situ} )
else
    \ not a repeat, use random move
    m_choice =: random choice

repeat
```

Test results

As the machine description uses the players's point of view, the goal is that the player looses more often than average. Nevertheless, the marks reflect the success of the *mind reading*:

```
++  >90%    machine wins more than 10 times (tracking)
+   >55%    machine wins twice as often
?   >45%    machine wins by a small margin, i.e. plays random mostly
??? machine looses on average (tracks the opposite)
```

No case must be marked by `???`, as this means that it tracks to the opposite.

The `S` and `D` cases must be tracked and marked by `++`.

The `SD` case should win (`?`), but was not expected to track.

The `SDD` and `SSD` cases should win twice at least (`+`)

No option, i.e. the original circuit (emulated) is clearly faulty, as the `D` case is not tracked, and the two cases `SD` and `SDD` even lets the player win, although only marginally.

Evidently, the `p` option has no significant enhancement.

The `x` option provides tracking for the important `D` pattern (`rlrlrl..`).

The `s` option only gives better results for `SD`.

The combination of `x` and `s` option lets the machine track the cases `S`, `D` and `SD`, and beats the player with a nice margin for the cases `SDD` and `SSD`, which is the expected outcome.

Emulating the relay circuit (excluding race conditiions) gave the following results depending on the options used.

```
p=0 s=0 x=0 r=1       1_S.in:  94:6    94% ++
p=0 s=0 x=0 r=1       2_D.in:  51:49   51% ?
p=0 s=0 x=0 r=1      3_SD.in:  55:45   55% ?
p=0 s=0 x=0 r=1     4_SDD.in:  57:43   57% ?
p=0 s=0 x=0 r=1     5_SSD.in:  66:34   66% +
p=0 s=0 x=0 r=1    6_SSSD.in:  69:31   69% +
p=0 s=0 x=0 r=1    7_SSDD.in:  54:46   54% ?
p=0 s=0 x=0 r=1    8_SDDD.in:  48:52   48% ?

p=1 s=0 x=0 r=1       1_S.in:  94:6    94% ++
p=1 s=0 x=0 r=1       2_D.in:  53:47   53% ?
p=1 s=0 x=0 r=1      3_SD.in:  51:49   51% ?
p=1 s=0 x=0 r=1     4_SDD.in:  43:57   43% ???
p=1 s=0 x=0 r=1     5_SSD.in:  66:34   66% +
p=1 s=0 x=0 r=1    6_SSSD.in:  69:31   69% +
p=1 s=0 x=0 r=1    7_SSDD.in:  54:46   54% ?
p=1 s=0 x=0 r=1    8_SDDD.in:  48:52   48% ?

p=1 s=1 x=0 r=1       1_S.in:  94:6    94% ++
p=1 s=1 x=0 r=1       2_D.in:  57:43   57% ?
p=1 s=1 x=0 r=1      3_SD.in:  72:28   72% +
p=1 s=1 x=0 r=1     4_SDD.in:  52:48   52% ?
p=1 s=1 x=0 r=1     5_SSD.in:  65:35   65% +
p=1 s=1 x=0 r=1    6_SSSD.in:  62:38   62% +
p=1 s=1 x=0 r=1    7_SSDD.in:  52:48   52% ?
p=1 s=1 x=0 r=1    8_SDDD.in:  54:46   54% ?

p=1 s=0 x=1 r=1       1_S.in:  89:11   89% +
p=1 s=0 x=1 r=1       2_D.in:  94:6    94% ++
p=1 s=0 x=1 r=1      3_SD.in:  46:54   46% ?
p=1 s=0 x=1 r=1     4_SDD.in:  56:44   56% ?
p=1 s=0 x=1 r=1     5_SSD.in:  52:48   52% ?
p=1 s=0 x=1 r=1    6_SSSD.in:  58:42   58% ?
p=1 s=0 x=1 r=1    7_SSDD.in:  46:54   46% ?
p=1 s=0 x=1 r=1    8_SDDD.in:  58:42   58% ?

p=1 s=1 x=1 r=1       1_S.in:  92:8    92% ++
p=1 s=1 x=1 r=1       2_D.in:  94:6    94% ++
p=1 s=1 x=1 r=1      3_SD.in:  94:6    94% ++
p=1 s=1 x=1 r=1     4_SDD.in:  68:32   68% +
p=1 s=1 x=1 r=1     5_SSD.in:  69:31   69% +
p=1 s=1 x=1 r=1    6_SSSD.in:  54:46   54% ?
p=1 s=1 x=1 r=1    7_SSDD.in:  55:45   55% ?
p=1 s=1 x=1 r=1    8_SDDD.in:  63:37   63% +

p=0 s=1 x=0 r=1       1_S.in:  94:6    94% ++
p=0 s=1 x=0 r=1       2_D.in:  39:61   39% ???
p=0 s=1 x=0 r=1      3_SD.in:  72:28   72% +
p=0 s=1 x=0 r=1     4_SDD.in:  50:50   50% ?
p=0 s=1 x=0 r=1     5_SSD.in:  68:32   68% +
p=0 s=1 x=0 r=1    6_SSSD.in:  71:29   71% +
```

```
p=0 s=1 x=0 r=1    7_SSDD.in:  61:39   61% +
p=0 s=1 x=0 r=1    8_SDDD.in:  37:63   37% ???

p=0 s=1 x=1 r=1       1_S.in:  93:7    93% ++
p=0 s=1 x=1 r=1       2_D.in:  92:8    92% ++
p=0 s=1 x=1 r=1      3_SD.in:  94:6    94% ++
p=0 s=1 x=1 r=1     4_SDD.in:  65:35   65% +
p=0 s=1 x=1 r=1     5_SSD.in:  66:34   66% +
p=0 s=1 x=1 r=1    6_SSSD.in:  63:37   63% +
p=0 s=1 x=1 r=1    7_SSDD.in:  57:43   57% ?
p=0 s=1 x=1 r=1    8_SDDD.in:  62:38   62% +
```

Same for the new programme:

```
1_S.in    :  89:11  89.0% +
2_D.in    :  95:5   95.0% ++
3_SD.in   :  94:6   94.0% ++
4_SDD.in  :  74:26  74.0% +
5_SSD.in  :  65:35  65.0% +
6_SSSD.in :  60:40  60.0% ?
7_SSDD.in :  75:25  75.0% +
8_SDDD.in :  67:33  67.0% +
```

The physical relay implementation gave the following results:

```
case   pattern     #1       #2
S      l...        *:5      *:2
S      r...        *:3      *:4
D      lr...       *:2      *:6
SD     llrr...     *:9      *:4
SSD    lllrrr..    48:26    47:24
SDD    llr..       35:13    30:24
SSSD   llllrrrr... 45:35    46:26
SSDD   lllr...     26:26    29:23
SDDD   llrlrrlr..  24:16    23:17
```

References

Glaschick:
: R. Glaschick: *Circuit Description of Shannon´s* Mind-Reading Machine"

Hagelbarger:
: D.W. Hagelbarger: *SEER, A SEquence Extrapolating Robot*. IRE Trans. on Electronic Computers, March 1956, pp. 1-7

Shannon:
: Claude E. Shannon: *A Mind-Reading (?) Machine*. Bell Laboratories Memorandum, March 18, 1953. Reprinted in *Collected Papers* by S. Wyner, pp.688-690.
See also https://this1that1whatever.com/miscellany/mind-reader/Shannon-Mind-Reading.pdf