Contents

Introduction

SIM is a simple logic simulator which can be used to simulate the operation of an electronic digital system described in terms of logic gates. SIM can be used to verify the logical correctness of a design (by exhaustive evaluation - but beware: this may not be computationally feasible even for simple designs) and to investigate potential and actual timing and initialisation problems in fragments of digital logic. It is in this second role that SIM is most useful.

SIM is a three valued logic simulator: signals may take the values 0 (definitely 0), 1 (definitely 1), and X (could be 0, could be 1, or could be between 0 and 1). The value X is 'overloaded' and does not have a consistent interpretation in all circumstances, however, the intuitive interpretation of X as 'everything that is not definitely 0 or 1' will suffice for the purposes of this explanation.

The value X arises natually as soon as a remotely realistic model of a gate is constructed. The problem is that the propagation delay of an individual gate is not known and can only be determined by measurement (even then it varies somewhat with temperature, fan-out loading, voltage level, etc). Consequently when a real circuit is designed the most that can be said of any individual gate is that its delay lies between some minimum value and some maximum value (usually the propagation delay depends also on whether the gate is turning on or turning off: TTL gates turn on about twice as slowly as they turn off; nMOS gates turn on about six times more slowly than they turn off). Because of this uncertainty in propagation delay it may not be possible to predict whether, at a given instant of time t, a signal has the value 1 or 0. In these circumstances the signal is said to have the value X at time t. In general, as a signal propagates through a logic circuit, the proportion of time that its value is uncertain increases (this is a manifestation of the law of increasing entropy). If this is allowed to continue indefinitely the signal becomes completely uncertain and no longer yields any information about the logic circuit. This is a feature three valued simulation, called pessimism, which arises because propagation delays have not been correctly modelled (can you see where the problem lies?). A pessimistic simulation result may be indicative of problems with the real hardware, but usually isn't! The best that can be done in these circumstances is to look at the problem more closely and try using minimum and maximum propagation delays that are closer together (or indeed, that are the same - equivalent to a fixed propagation delay).

SIM operates in two phases. In the first phase a time independent simulation algorithm due to Eichelberger [EICH65] is used to simulate the effect of powering up the logic circuit and to investigate the time independent effect of changing the input stimuli. In this context time independent simulation means that gates can have any delay value between 0 and infinity and that once the input stimuli have been changed no further input changes occur until the effects of the first changes have propagated through the logic to the outputs. This is an absolutely worst case, most pessimistic simulation of a logic circuit. It guarantees to find all potential races and hazards (see [EICH65]), but, of course, not all potential races and hazards will occur in a real circuit: some will be due to bizzare combinations of delays that would be impossible with real gates. What can be said, however, is that if a logic circuit can be initialised using the time independent assumptions then it can certainly be initialised using any set of time dependent assumptions. In general it is desirable that a logic circuit always initialises to some well defined state when the power is switched on (independently of what delays is component gates

possess) and so it is good design practice to ensure that it can be initialised under time independent assumptions.

As well as simulating the effect of switching on the power the first phase of the simulator can be used to investigate the effect of single or multiple input changes under the time independent assumptions (worst case, most pessimistic simulation). This is often useful for combinatorial logic or small fragments of sequential logic because a prediction of good behaviour by this phase of the simulator implies good behaviour for all possible gate delay values provided that input changes are allowed to propagate to the outputs before further input changes occur. It is almost never useful to simulate large sequential systems in this manner because there is almost always some degree of pipelining of the inputs (a second set of inputs is presented for processing before the results of the first set have propagated to the outputs) and because the correct behaviour of the system almost always depends on gates having sensible bounded delay values: infinite delays will knacker almost any sequential digital system.

The second phase of the simulator allows a logic circuit to be simulated under the assumption of bounded propagation delays and implements a simulation algorithm due to Chicoix et al [CHIC76]. The output from this phase of the simulator is a list of events and the times at which they occurred (rather than a list of circuit output values). This list can be easily postprocessed (see later) to produce a conventional timing diagram showing the relationships between any subset of the signals of the circuit. Once again the simulation is pessimistic (uncertainties grow as signals propagate through the circuit) but the degree of pessimism can be controlled (this a modelling decision to be made by the user) by specifying more or less strict delay bounds. Pessimism can be avoided altogether by running the simultor in an optimistic mode in which a random delay value, between the minimum and the maximum specified, is chosen for each gate in the circuit. This is quite a good approximation to reality in the sense that real gates have an almost fixed delay value lying somewhere between a minimum and a maximum specified by the manufacturer (or rather that's what we like to think: reality is actually rather more pathological than this), however, the power of the simulator to detect potential problems (those that may depend on funny combinations of gate delays) is greatly reduced by running it in this optimistic mode. It is important that good behaviour under these optimistic assumptions is not assumed to imply good behaviour in a real circuit except in a probabalistic sense and that simulations under more pessimistic assumptions are performed in order to investigate where potential problems may lie.

Using SIM on VAX

SIM is one of a suite of computer aided design programs known collectively as DL1. This suite can be accessed on VAX by first issuing the command DL1SETUP. Help information can be obtained by issuing the command HELP DL1, and help relating to SIM can be obtained by issuing the command HELP DL1 SIM (and from SIM itself by using the HELP subcommand - see below: Driving the simulator). Logic circuits to be simulated must be described in the ESDL notation (see separate documentation), compiled using the ESDL command, and flattened using the FLATTEN command (see HELP information and separate documentation about ESDL). SIM recognises the following gate types: AND, OR, NOT, INV (same as NOT), NAND, NOR, XOR, XNOR, DLAY (a delay element), ROM (a read only memory), and RAM (random access memory). All other elements (such as flip-flops) must be expanded in terms of these elements (a further function performed by FLATTEN: see

separate documentation about ESDL). The compiled and flattened description is input to SIM on imput stream 1. If no file name extension is specified for this input file then .FIC is assumed, this being the default extension generated by FLATTEN. A file of simulator driving commands can be specified for input stream 2. The second phase of SIM produces, as output, a file of all events which have occurred during the simulation. If no filename is specified for this file then the input file name with the default extension .TRC is used. If the second phase of SIM is not to be used then the null stream .N should be specified for output stream 1 in order to prevent the generation of a plethora of empty files by VMS.

In order to produce a timing diagram from SIM's output it is necessary to use the POSTPRO command. POSTPRO accepts the output of SIM on input stream 1 (default file name extension .TRC) and produces a timing diagram on output stream 1. If no file name is specified for this output then the name of the input file with extension .WVM (for WaVeforMs) is used. This output is suitable for listing on a line printer. When POSTPRO is invoked the user is prompted for certain data: the width of the page on which the timing diagram is to be printed (line printer paper is 80 columns wide); the names of the signals to be included in the timing diagram (in order, from left to right across the page); and whether or not the diagram is to be compressed. In general a compressed diagram is the most readable (and saves paper). Timing diagrams are compressed by removing all time periods of length greater than some specified minimum in which no signal changes occur. This minimum period is called the compression factor and is supplied by the user: 3, 4, and 5 are reasonable values to use (see also the example on page 9).

Examples:

```
SIM JCOUNT                  {JCOUNT.FIC/JCOUNT.TRC is assumed}
SIM ADDER/.N                {only the first phase of the simulator
                             is used on ADDER.FIC}
SIM COMB.EIC/.N             {very simple circuit - doesn't need
                             flattening so use compiler output .EIC}

POSTPRO JCOUNT              {produce timing diagram in JCOUNT.WVM}
POSTPRO JCOUNT/.LIS         {produce timing diagram in JCOUNT.LIS}
```

Driving the simulator

When the simulator is first invoked phase 1 is activated and all signals are given the undefined value X. Phase 2 of the simulator cannot be invoked until at least one (not necessarily successful) attempt to initialise the circuit has been made (see introductory paragraphs for details). After SIM has read the circuit specification and checked that it is a valid description the prompt -> is given and simulator subcommands may then be issued. The simplest subcommands are described below:

HELP causes a list of available subcommands to be printed at the terminal. HELP followed by a subcommand name causes a brief description of the effect of that subcommand to be printed at the terminal. HELP may be abreviated to ?.

INPUTS? causes a list of the names of circuit inputs to be printed at the terminal.

OUTPUTS? causes a list of the names of all gate outputs to be printed at the terminal. Outputs which are also circuit outputs are identified by having (cct) printed after their names.

ALL followed by a logic value (0, 1, LO, HI, X, or HALF) causes all circuit inputs to be set to the specified value. The circuit is not evaluated.

SET <name> = <logic value> causes the named circuit input (or gate output) to be set to the specified logic value. The circuit is not evaluated.

GO causes the circuit to be evaluated and the values of its output signals to be printed at the terminal (in the same order as they occur in the ESDL description of the circuit). Three sets of values are printed: the values of the outputs before the circuit is evaluated; the values of the outputs with the changing inputs set to X; and the values of the outputs with the changing inputs taking their new values.

VAL <name> {[,]<name>}* causes the values of the named outputs to be printed at the terminal.

STOP stop the simulator and return to system command level.

HOLD <name> {[,]<name>}* holds the values of the named outputs fixed. This is useful when simulating the result of a 'stuck-at' fault and occasionally when trying to initialise a sequential circuit such as a J-K master slave flip-flop (without preset or clear signals) which can only be initialised by breaking an oscillation (oscillating signals take the value X in the first phase of SIM).

FREE <name> {[,]<name>}* allow the values of the named outputs to change again. This subcommand undoes the effect of HOLD.

%S read commands from input stream 2 until a STOP command or a %S command is encountered, or until the end of input on stream 2.

These commands are sufficient to allow a circuit to be initialised and to allow its time independent behaviour to be probed. In particular a (potential) hazard is detected when the sequence of output values for some circuit output is 0X0 or 1X1. Note that the cause of the hazard is not located and that further investigation is required in order to locate it. Sequential circuits may also exhibit the sequences of output values 0XX and 1XX denoting either the presence of a race (output value depends on who wins the race - which depends on the precise assignment of delay values to gates) or the presence of a oscillation. Further analysis is required in order to distinguish between races and oscillations.

Driving the second phase of the simulator

In the second phase of simulation an initialised circuit is subjected to some changing input stimuli and the resulting events are recorded in strictly increasing time order on the output stream. An event is a triple <N,v,t> where N is the name of a gate output, v is a logic value (0, 1, or X) and t is the time at which the output N takes the value v. This event list can be postprocessed by the POSTPRO program to produce a waveform trace or timing diagram (see earlier: Using SIM on VAX).

In order to stimulate a circuit its inputs can be connected to signal traces (any unconnected inputs are left with the values set by the initialisation phase of the simulation). Traces are described in a simple notation which specifies how long the signal is to be held low, how long it is to rise for, etc. Furthermore, traces can be grouped together using brackets and repeated a number of times (or indefinitely) and any element of a trace can itself be a trace. Traces may also be named and referenced by name. Formally:

```
trace     :== {tel}*
tel       :== ( trace ) {count}
              @ tracename
              value {length}
value     :== L | R | H | F
```

The values L, R, H and F represent low (0 to 0 transition), rising (0 to 1 transition), high (1 to 1 transition) and falling (1 to 0 transition). Clearly there are constraints on the order in which L, R, H and F may occur: L may follow L or F; R may follow L or F; H may follow H or R; F may follow H or R. 'Count' is an integer specifying how many times the subtrace is to be repeated (0 and * specify repetition until the time limit is exceeded) and 'length' specifies the duration of a signal value or transition. A trace definition may span any number of lines by ending each line to be continued with %C (all characters following the %C are ignored). Comments may be inserted between any pair of syntactic tokens and start with a $ character. A comment is terminated by the next $ or newline character. Comments may be useful when standard traces are defined in a file to be read from the simulator's second input stream. Named traces are defined using the DEFINE subcommand:

DEFINE <name> = <trace> {, <name> = <trace>}*
: Thereafter the signal trace is known by the specified name. This is a particularly useful command to put in a file of commands to be read from the simulator's second input stream. Note that, within SIM, names are typed and that there can be no confusion (except in the mind of the user) between the trace called T1 and the circuit input called T1: it is quite valid to connect trace T1 to signal T1.

Signal traces are associated with inputs by using the LET subcommand:

LET <name> = <trace> {, <name> = <trace>}*
: <name> is any input name and <trace> is any signal trace (the trace can even have the same name as the signal: see DEFINE above for details).

Examples:

```
DEFINE PULSE = L10 R5 H50 F5 L10
LET A<0> = (@PULSE)10
LET A<1> = (L10 (L10 R5 H50 F5)5 L100)*
LET ACK' = H200 F L
```

It should be noted that not only must consecutive trace elements be compatible with one another but also the first element of a trace must be compatible with the signal value established by the initialisation of the circuit. If, at the start of the second phase of simulation, the first element of a trace is not compatible with the value (of the input to which it is connected) established by phase 1 then the trace is ignored, a diagnostic message is issued, and the input's value remains fixed.

Traces may also be connected to inputs conditionally: that is, when a specified set of conditions first holds a specified trace is connected to a specified input. This is accomplished using the WHEN subcommand followed by a LET subcommand. Because the LET command may specify the connection of any number of traces to inputs, any number of such connections may be made contingent upon a given condition. The form of the command is:

WHEN <condition> LET ...

<condition> is any disjunction of conjuctions of terms (as usual, conjuction has higher precedence than disjunction). Disjunction is represented by + (or) and conjunction by & (and) and the <condition> may span any number of lines provided that the line is broken immediately after a + or a &. A term is either a condition enclosed in parentheses or a simple term of the form <name>=<logic-value>. <name> is the name of any gate output or primary circuit input and must occur to the left of the = character. <logic-value> is either 0 or 1 (LO and HI may be used equivalently). When the condition first becomes true the LET statements consequent upon it are obeyed: each input specified in the LET statement first has its existing trace (if any) disconnected then has the conditional trace connected to it. For example:

```
WHEN CLK=1 & ENBL'=0 & SIGNAL=1
LET  SIGNAL = F5 (TESTPULSE)10  TESTPULSE starts from 0
```

Note here how the 'guard condition' SIGNAL=1 has been included to guarantee that, when the trace is connected to SIGNAL, SIGNAL will have the value 1 (as assumed by the trace, which starts with F5 - only compatible with 1).

Using the commands DEFINE, LET, and WHEN is is possible to create quite sophisticated sets of stimuli with which to exercise a logic circuit. Two more subcommands assist with this task:

SHOW { <name> {[,] <name>}* }
: Print out the traces which are currently connected to the specified signals and under what conditions the connection will be made (only one trace may be unconditionally connected to a signal). If no signal names are specified then all connections to signals are displayed.

DELTRACE { <name> {[,] <name]* }
: Break the connection of traces to the named signals and delete the traces (unless they are named). If no signal names are specified then all connections between traces and signals are broken and all unnamed traces are deleted.

Starting the second phase of the simulation

The second phase of the simulator can be initiated as soon as the circuit has been initialised and the relevant input stimuli have been defined. This is done by issuing the RUN command. After issuing the RUN command the user is prompted for certain data and then the simulator runs without interaction until there are no more input stimuli or until the time limit for the simulation is exceeded. Three sets of data are required by the simulator when the RUN command is issued. First, a set of default delay values is required to be used as propagation delays for all gates that do not have propagation delays specified by means of the DELAY parameter of ESDL (see separate documentation). Up to four values may be

specified: min turn on time; max turn on time; min turn off time; and max turn off time. If omitted the second delay value defaults to the first, the third to the first, and the fourth to the second (1 to 4 values may be specified). Second, a maximum simulation time must be specified. This can be between 10 and 30000 units (where these are the same units as were used for propagation delays and for the length of trace elements). Third, the user is prompted for a yes or no answer to the question 'Random delays?'. If the answer given is yes (or y or Y) then the user is further prompted for a pseudo random number seed (so that the results are repeatable) and this is used to generate a random choice of propagation delays between the minima and maxima specified for each gate (see earlier comments on optimistic simulation). If the answer given is no (or n or N) then a pessimistic simulation is performed (see earlier comments on pessimistic and optimistic simulation).

The gate model

SIM employs a very simple gate model. Gates are modelled as a pure function followed by a pure delay. Because the delay follows the function there are particularly simple definitions of turn on and turn off delays: the turn on delay of a gate is the time from between an input signal change and the rising (0 to 1 transition) output signal that it causes; the turn off delay is the time between an input signal and the falling (1 to 0) transition that it causes. Minimum times are measured between the end of the input transition and the start of the output transition and maximum times between the start of the input transition and the end of the output transition.

The extension to ROM and RAM

SIM also simulates logic circuits containing memory elements. Two kinds of memory can be used: ROM (read only memory) and RAM (random access memory). In both cases there are a number of conventions which must be observed if the simulation is to be possible. First, all memory elements must be labelled in the ESDL description. That is, the description will contain constructs such as

```
ROM1:ROM(ADDRESS<0:3>,SEL)->DATA<0:3>
```

A memory is named by its label: in the example above the label is ROM1. Both ROMs and RAMs can be initialised to a predefined pattern of bits by means of the FILL subcommand of SIM (see below), and default (in the absence of any such initialisation) to being filled with the value (0 or 1) specified in the VALUE parameter of ESDL or interactively (the user is prompted for a default memory value) if no VALUE parameter is associated with the memory. This default value is also used as the value to which the memory's outputs float when the memory is not selected. This is an unfortunate deficiency of SIM's model of memory as most (all?) real memory chips' outputs float to 'high impedance' (no effect on the outside world) when the chip is not selected.

Conventions about the positions of signals must also be obeyed. For ROMs the last input signal is the SELect signal (active when high), all other inputs are interpreted as addresses, and all outputs as data. The size of the memory is calculated appropriately as 2**(no-of-address-inputs) by no-of-outputs bits. For RAMs the last input signal is also the SELect

signal (active when high), the penultimate input signal is the Read/Write signal (low for Reading, high for Writing), and all outputs are data outputs. Of the remaining inputs a number equal to the number of outputs are interpreted as data inputs, and the remaining leftmost inputs are interpreted as addresses. For example:

```
MEM1:RAM(ADDR<0:3>,DIN<0:7>,RW,SEL)->DOUT<0:7>
```

declares a 16 byte random access memory called MEM1. ROMs and RAMs are initialised by means of the FILL subcommand:

```
FILL <memory-name> FROM <filename>
```

<memory-name> is the label of a memory element (such as MEM1 in the example above). <filename> is the name of any VAX file which contains a bit pattern to be loaded into the memory. The format of this file is as follows: the file contains one line for each word in the memory element; each line contains a contiguous string of 0s and 1s of length exactly equal to the number of bits in a word. Leading spaces and all trailing characters are ignored, as are all leading and training lines that do not contain 0s or 1s. The memory is filled in such an order that the leftmost address input (NB leftmost, not highest of lowest numbered) is the most significant bit of the word address. Suppose that in the example above MEM1 was filled from a file in which the seventh line contained the bit string 0100, then the selection of MEM1 with ADDR<0>=0, ADDR<1>=1, ADDR<2>=1, and ADDR<3>=1 (address = 0111 = 7) would cause DOUT<0> to take the value 0, DOUT<1> the value 1, DOUT<2> the value 0 and DOUT<3> the value 0.

References

[CHIC76] C Chicoix, J Pedoussat and N Giambiasi
An accurate time delay model for large digital network simulation.
13th ACM/IEEE Design Automation Conference Proceedings, pp42-47, 1976.

[EICH65] E B Eichelberger
Hazard detection in combinatorial and sequential switching circuits.
IBM Journal of Research and Development, March 1965.

A worked example

In this section an example is given of the simulation of a four bit Johnson counter. All interactions with VAX and with SIM are shown. The VAX command prompt is a $, SIM's command prompt is -> , and explanatory comments are enclosed in curly brackets {}. The ESDL description of the Johnson counter is shown below and is assumed to be stored in a file called JCOUNT.SRC:

```
UNIT JCOUNT(CK,CL)->D1,D2,D3,D4
   UNIT DFF(P,CK,D,CL)->Q,QB
      NAND(J,P,KB)->JB
      NAND(CK,JB,CL)->J
      NAND(CK,KB,J)->K
      NAND(D,K,CL)->KB
      NAND(J,QB,P)->Q
      NAND(K,Q,CL)->QB
   END

   DFF(.1,CK,D0,CL)->D1,?
   DFF(.1,CK,D1,CL)->D2,?
   DFF(.1,CK,D2,CL)->D3,?
   DFF(.1,CK,D3,CL)->D4,?
   NOT(D4)->D0
END
```

The dialogue with VAX and SIM proceeds as follows:

```
$ esdl jcount                        {compile the description}
$ flatten jcount                     {and flatten it}
$ sim jcount                         {invoke the simulator
                                      output to jcount.trc}
   SIM version 4.0 (VAX)

Circuit initialisation (Eichelberger)

-> inputs?                           {enquire the names of circuit inputs}
   CK CL                             {there are two - CK and CL}
-> all 0                             {set all inputs to 0}
-> go                                {and evaluate the circuit}
    X X X X                          {outputs with inputs uncertain}
    0 0 0 0                          {outputs with inputs 0}
-> set cl=1                          {stop asserting clear signal}
-> go
    0 0 0 0
    0 0 0 0
    0 0 0 0
-> let ck=(140 r5 h80 f5 140)20  {set up a signal trace connected to CK}
-> run                              {and start the second phase running}
Default delays 5 10 3 6              {5 10 3 6 supplied by the user}
Simulat'n time 1000                  {replied to prompt with 1000}
Random delays? y                     {choose optimistic simulation}
Random no seed 1234                  {1234 chosen as seed}
*End of simulation                   {SIM has done it in no time!}
$ postpro john                       {produce the waveform trace}
Width of page= 60                    {print output in 60 columns}
Signal  names  ck d1 d2 d3 d4        {trace clock and outputs only}
Signal  names  *                     {* to end the list}
Compr'n factor=3                     {remove all inactive periods of len<3}
```

Output from POSTPRO

On this page the header page of POSTPRO's output is shown. On the following page the first 900 or so time units of a 1000 time unit simulation of a Johnson counter are shown. This is the example that was worked on the previous page and POSTPRO's output shows the relationships between the clock signal (CK) and the four outputs (D1, D2, D3, and D4). There are, of course, many more signals that could have been traced on this timing diagram, however these five are the most interesting as they capture the behaviour of a Johnson counter graphically. Note also the advantage of a compressed trace: by deleting time periods in which nothing happens a reasonable sample of behaviour can be represented as a timing diagram on a single page. This is much more readable than the uncompressed diagram which would have spanned 15 pages! First the header page:

```
Simulation of circuit JCOUNT by SIM version 4.0 (VAX)

Circuit inputs at start of simulation:-

     CK=0
     CL=1

Signal traces attatched to circuit inputs:-

     CK=(L40 R5 H80 F5 L40)20
```

Headers occupy a complete page and are often much more complicated that the header shown above: for example when there are several conditional signal traces connected to various signals. On the next page the first 900 or so time units of the simulation are shown.

Time	CK	D1	D2	D3	D4
0	0	0	0	0	0
++++	0	0	0	0	0
40	XXXX	0	0	0	0
++++	XXXX	0	0	0	0
51	1	XXXX	0	0	0
++++	1	XXXX	0	0	0
56	1	1	0	0	0
++++	1	1	0	0	0
125	XXXX	1	0	0	0
++++	XXXX	1	0	0	0
130	0	1	0	0	0
++++	0	1	0	0	0
210	XXXX	1	0	0	0
++++	XXXX	1	0	0	0
221	1	1	XXXX	0	0
++++	1	1	XXXX	0	0
226	1	1	1	0	0
++++	1	1	1	0	0
295	XXXX	1	1	0	0
++++	XXXX	1	1	0	0
300	0	1	1	0	0
++++	0	1	1	0	0
380	XXXX	1	1	0	0
++++	XXXX	1	1	0	0
391	1	1	1	XXXX	0
++++	1	1	1	XXXX	0
396	1	1	1	1	0
++++	1	1	1	1	0
465	XXXX	1	1	1	0
++++	XXXX	1	1	1	0
470	0	1	1	1	0
++++	0	1	1	1	0
550	XXXX	1	1	1	0
++++	XXXX	1	1	1	0
561	1	1	1	1	XXXX
++++	1	1	1	1	XXXX
566	1	1	1	1	1
++++	1	1	1	1	1
635	XXXX	1	1	1	1
++++	XXXX	1	1	1	1
640	0	1	1	1	1
++++	0	1	1	1	1
720	XXXX	1	1	1	1
++++	XXXX	1	1	1	1
725	1	1	1	1	1
++++	1	1	1	1	1
736	1	XXXX	1	1	1
++++	1	XXXX	1	1	1
741	1	0	1	1	1
++++	1	0	1	1	1
805	XXXX	0	1	1	1
++++	XXXX	0	1	1	1
810	0	0	1	1	1
++++	0	0	1	1	1
890	XXXX	0	1	1	1
++++	XXXX	0	1	1	1
895	1	0	1	1	1
++++	1	0	1	1	1
906	1	0	XXXX	1	1
++++	1	0	XXXX	1	1
911	1	0	0	1	1