BLACK/PRIO	HUE	BRIGHTNESS	R G B PRIO: 0 0 0 0 (色相)	0 0 0 0 (明度)	0 0 0 1 (色相)	0 0 0 1 (明度)	1 0 0 0 (色相)	1 0 0 0 (明度)	1 0 0 1 (色相)	1 0 0 1 (明度)	0 1 0 0 (色相)	0 1 0 0 (明度)	0 1 0 1 (色相)	0 1 0 1 (明度)	1 1 0 0 (色相)	1 1 0 0 (明度)	1 1 0 1 (色相)	1 1 0 1 (明度)
0	0	0	B4 R,G,B で指定された色相	淡 灰	ブルーシアン	濃 明	赤	淡 明	赤	淡明	緑		緑		黄		黄	
0	0	1		淡 明														
0	1	0		濃 灰		濃 灰		淡 灰										
0	1	1		濃 明				濃 明										
1	0	0		淡 灰	黒	X 黒		淡 明		濃明								
1	0	1																
1	1	0		濃 明				濃 灰										
1	1	1						濃 明										

BLACK/PRIO	HUE	BRIGHTNESS	R G B PRIO: 0 0 1 0 (色相)	0 0 1 0 (明度)	0 0 1 1 (色相)	0 0 1 1 (明度)	1 0 1 0 (色相)	1 0 1 0 (明度)	1 0 1 1 (色相)	1 0 1 1 (明度)	0 1 1 0 (色相)	0 1 1 0 (明度)	0 1 1 1 (色相)	0 1 1 1 (明度)	1 1 1 0 (色相)	1 1 1 0 (明度)	1 1 1 1 (色相)	1 1 1 1 (明度)
0	0	0	青	淡 明	青	淡明	シアン		シアン		マゼンタ		マゼンタ		オレンジ	淡 明	オレンジ	淡明
0	0	1																
0	1	0		淡 灰												淡 灰		
0	1	1		濃 明												濃 明		
1	0	0		淡 明		濃 明									白	X 明	白	X 明
1	0	1																
1	1	0		濃 灰												X 灰		
1	1	1		濃 明												X 明		

〔1 表示区間に選択できる色彩の組合せ〕

〔背景表示〕

R									0								
G									0								
B									0								
PRIO									0								
**	BLACK/PRIO	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
*	BG R	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
*	BG G	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
*	BG B	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
		ブルーシアン	赤	緑	黄	青	シアン	マゼンタ	オレンジ	黒	赤	緑	黄	青	シアン	マゼンタ	白

色指定

** BRIGHTNESS	** HUE																
0	0	淡，灰								X・黒	淡，灰						X・灰
0	1	淡，明									淡，灰						
1	0	濃，灰									濃，灰						
1	1	濃，明									濃，灰						

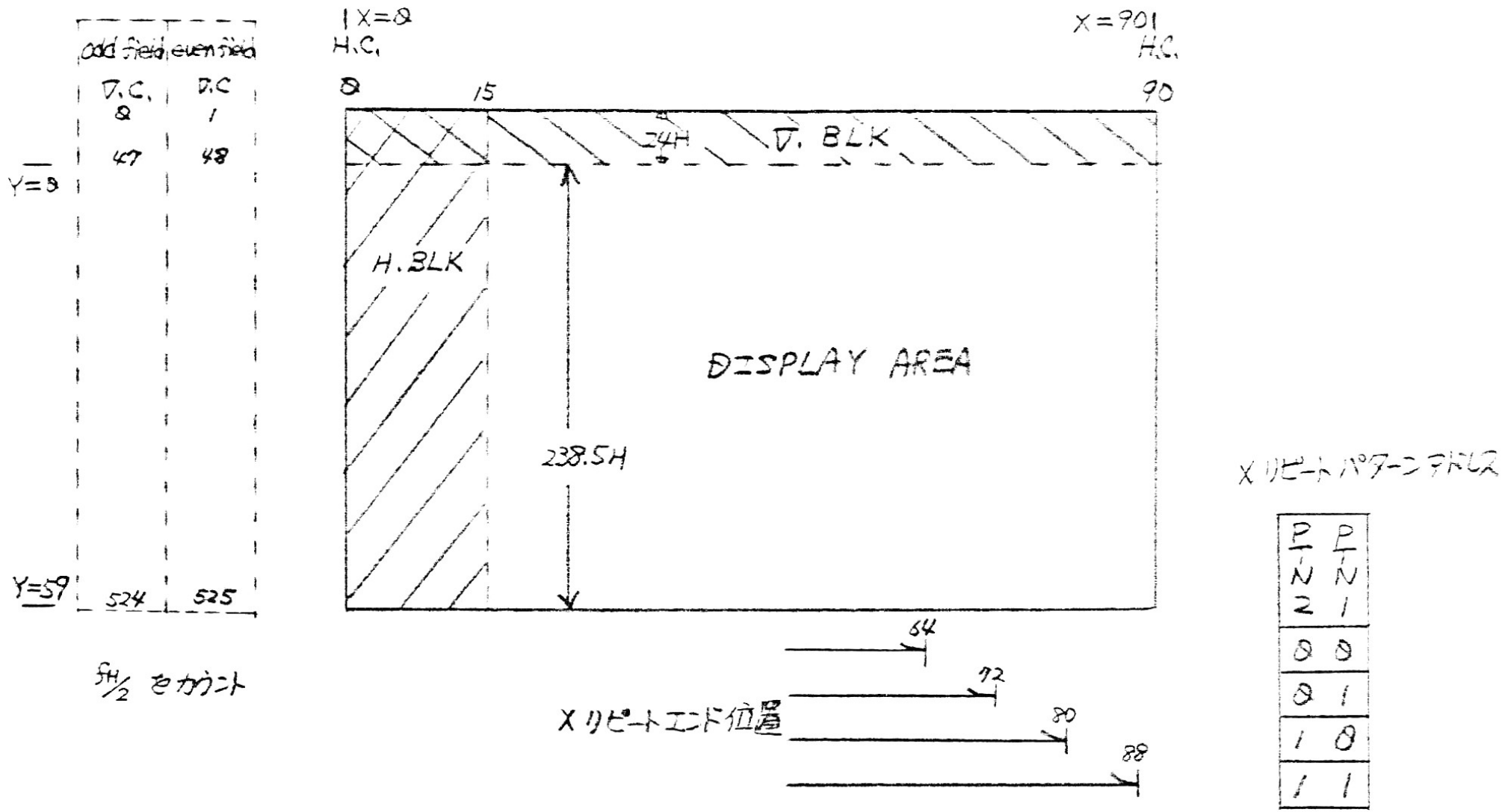
彩度(濃淡X)
明度(明灰黒)

R, G, B, PRIO 出力 All "0" のとき 背景表示

*, ** は 表示開始時に設定される モード切換ﾋﾟﾝ

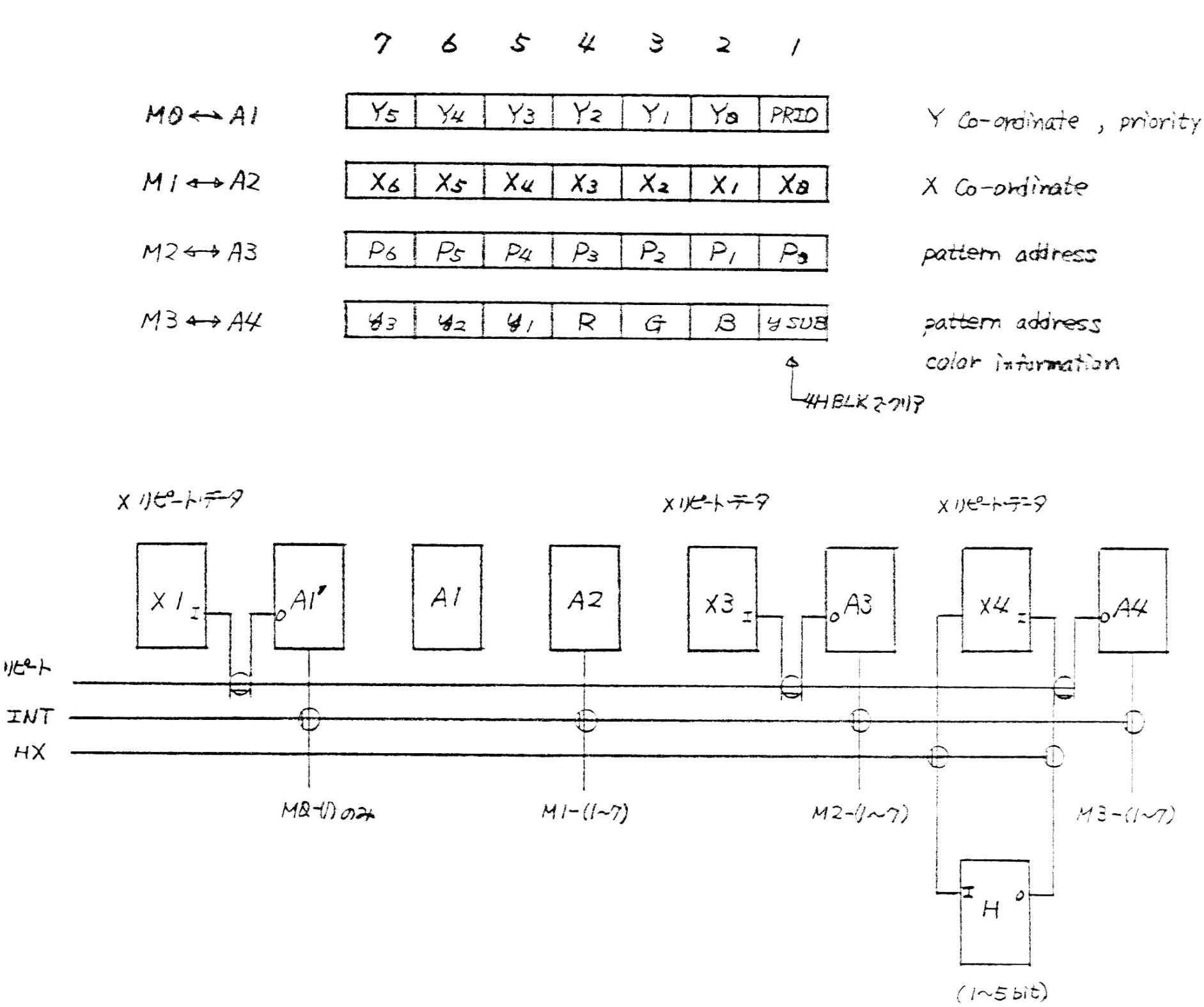
** は 図形表示にも影響を与える

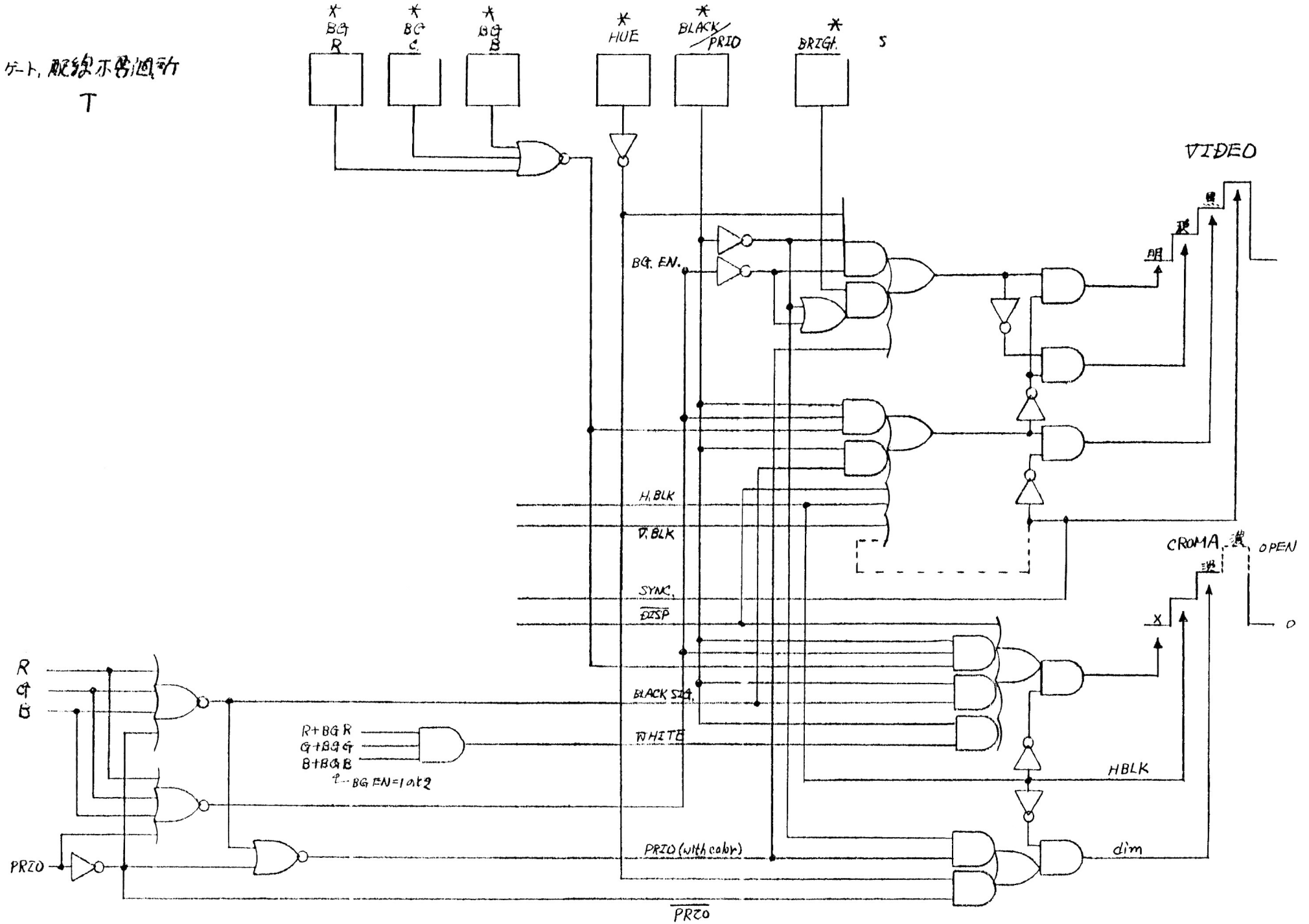
背景として"黒or白"を使用したい場合に BLACK/PRIO ﾋﾟﾝ = 1 とする



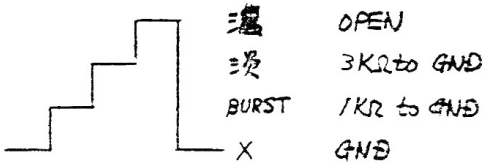
Xリピートパターンアドレス

PIN2	PIN1
0	0
0	1
1	0
1	1

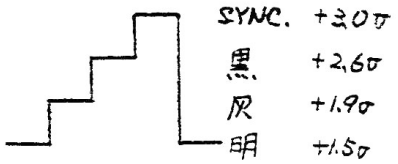

7 6 5 4 3 2 1
M0 ↔ A1
Y5 Y4 Y3 Y2 Y1 Y0 PRIO
Y Co-ordinate , priority
M1 ↔ A2
X6 X5 X4 X3 X2 X1 X0
X Co-ordinate
M2 ↔ A3
P6 P5 P4 P3 P2 P1 P0
pattern address
M3 ↔ A4
y3 y2 y1 R G B y SUB
pattern address
color information
4H BLK クリア
X リピートデータ
X リピートデータ
X リピートデータ
X1
A1'
A1
A2
X3
A3
X4
A4
リピート
INT
HX
M0-(1)のみ
M1-(1~7)
M2-(1~7)
M3-(1~7)
H
(1~5 bit)

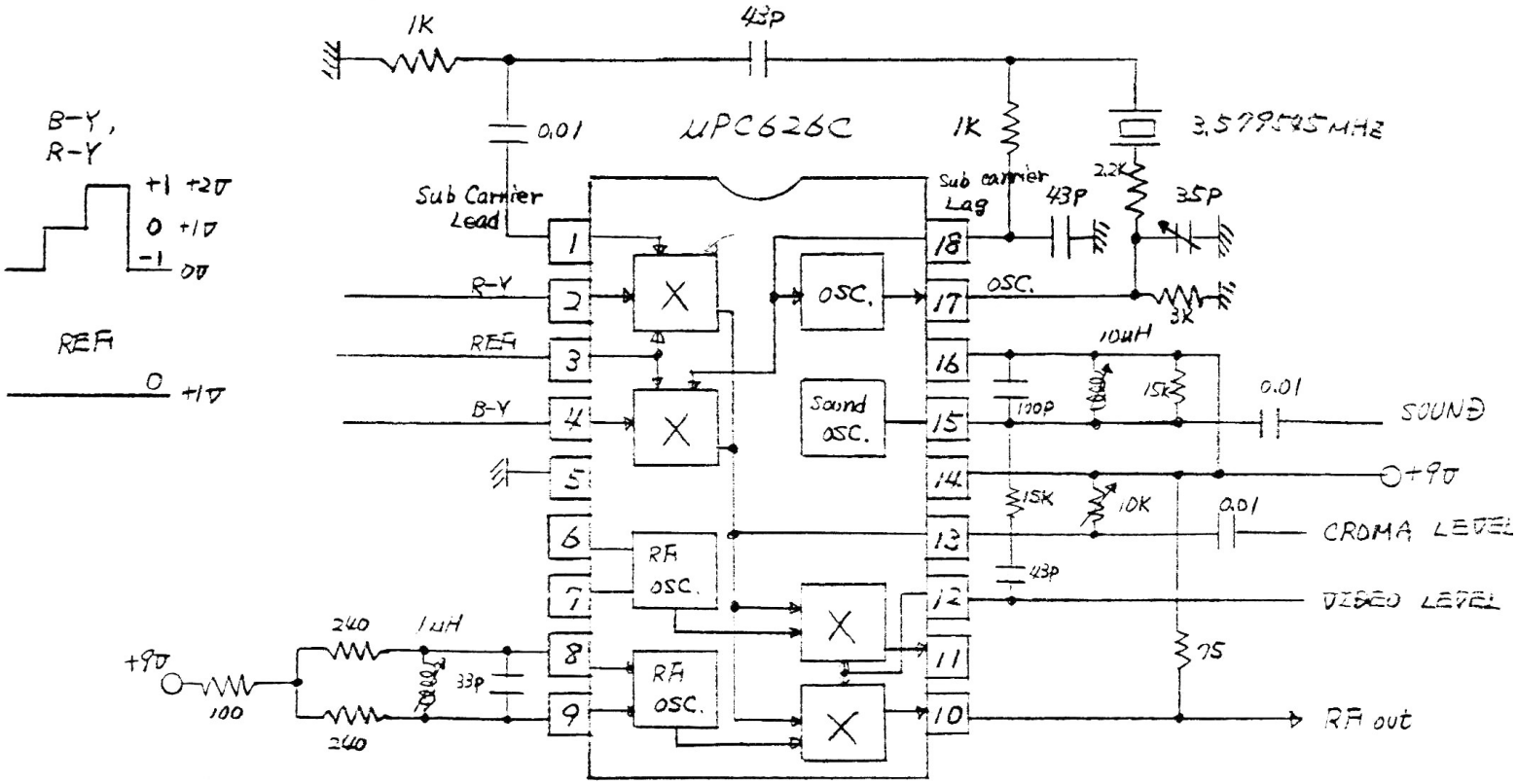

BG R
BG G
BG B
HUE
BLACK/PRIO
BRIGHT
5
VIDEO
BG. EN.
H. BLK
V. BLK
SYNC
DISP
CROMA
OPEN
0
X
R
G
B
BLACK SIG.
WHITE
R+BG R
G+BG G
B+BG B
↑ BG EN = 1 or 2
HBLK
PRIO
PRIO (with color)
dim
PRIO

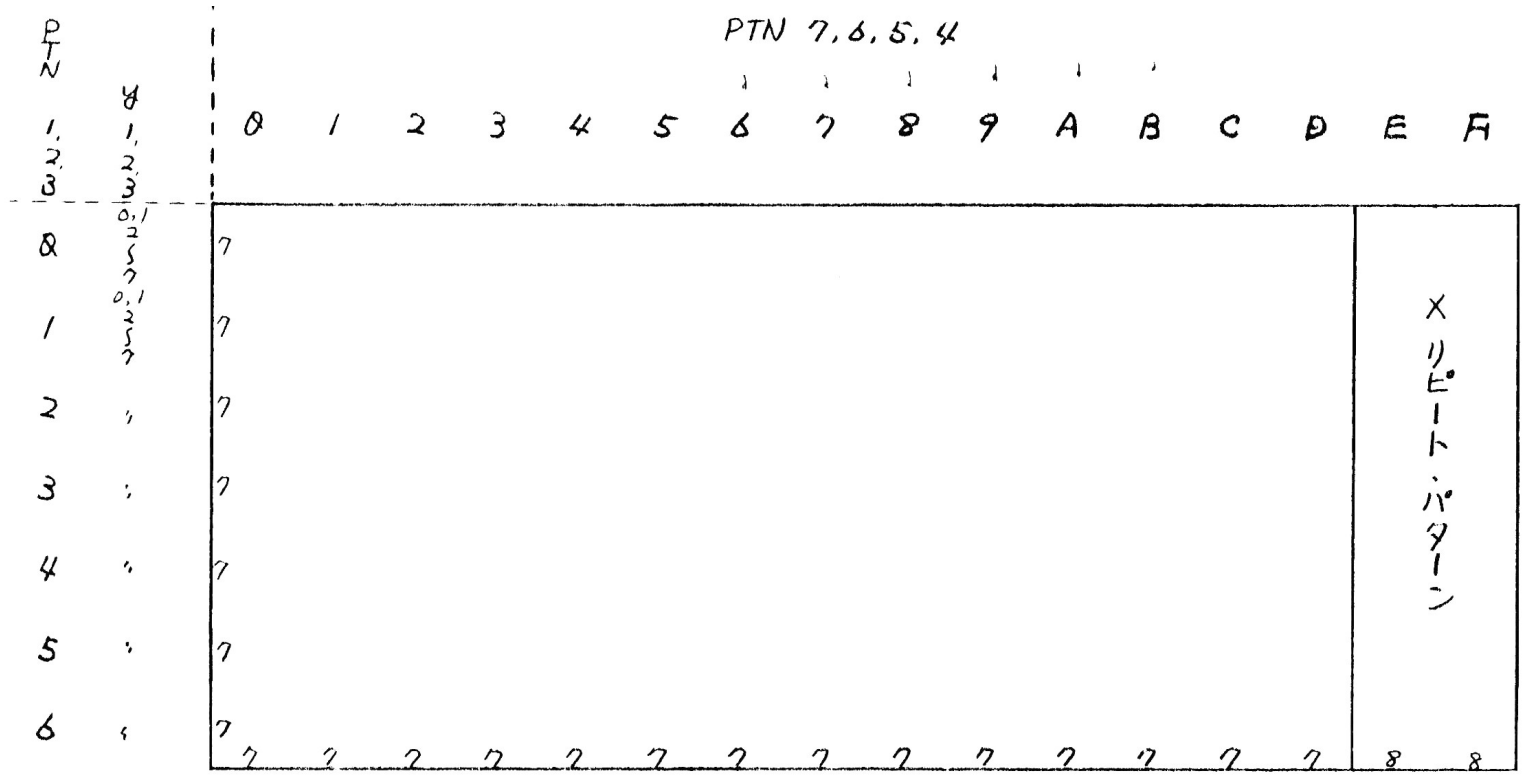
CROMA LEVEL

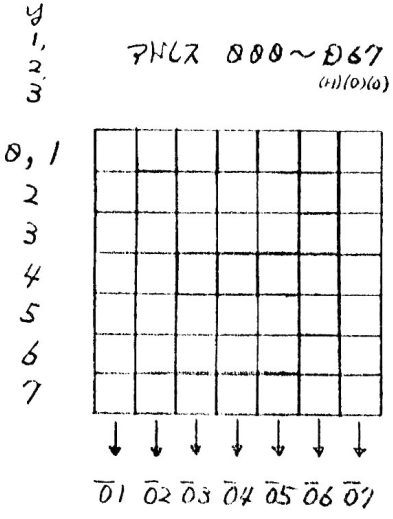


VIDEO LEVEL

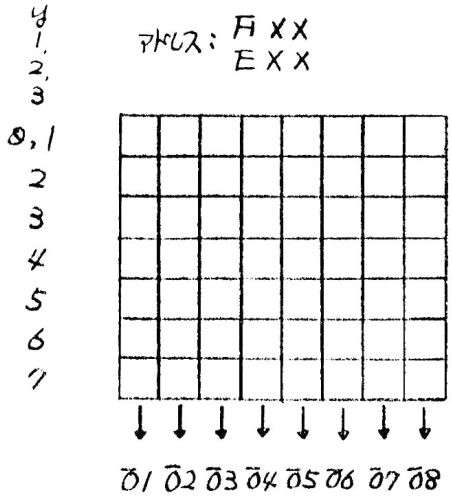








通常文字図形パターン（7×7ドット構成）
98 パターン max.



X方向リピート原形パターン（8×7ドット構成）
14 パターン max.

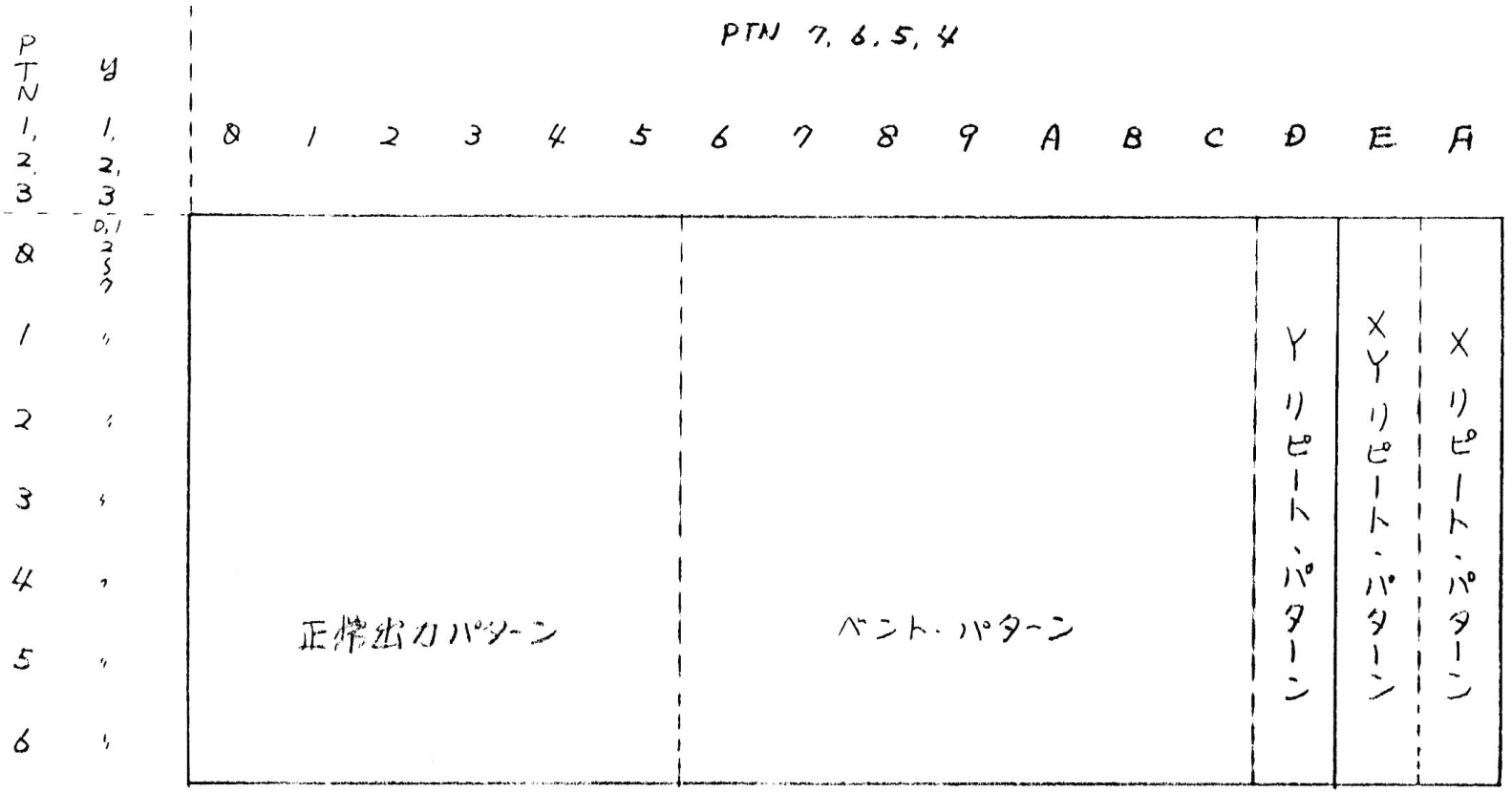
(1) X方向リピートパターンはキャラクタROMによって上図の14パターンに固定

(2) Y方向リピートパターン数，ROMアドレスはソフトによって設定する。
（X，Y両方向へのリピートパターン定義も可）

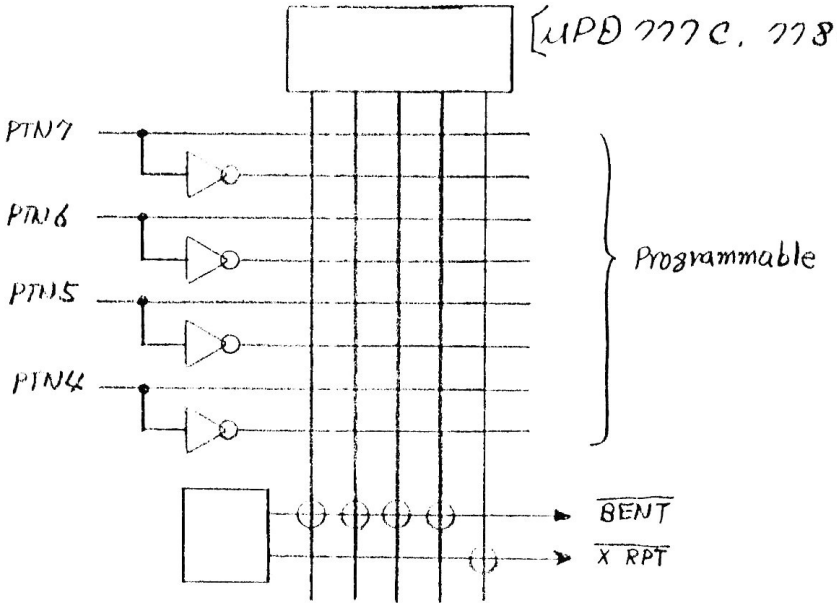
(3) ベント・パターン数，ROMアドレスはデコーダーハードを変更する事によって設定変更する。

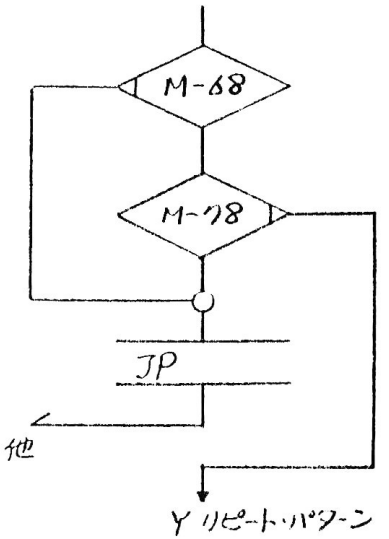
(4) Xリピート・パターンは1走査線について1種類も可能

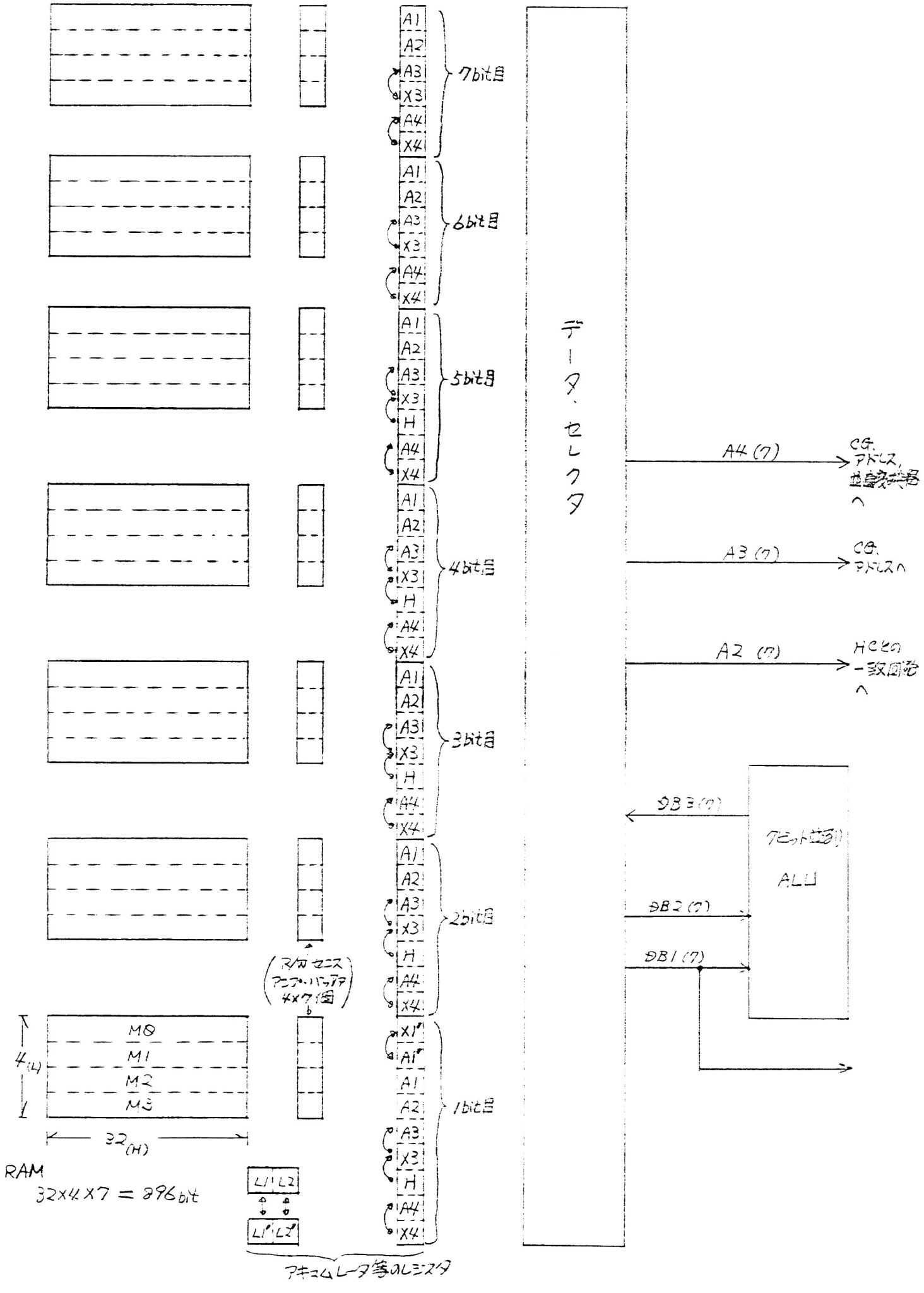
〔キャラクタ・ジェネレーター ROM 構成〕



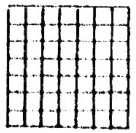
〔μPD777C, 778C の各種パターンのアドレス割当〕



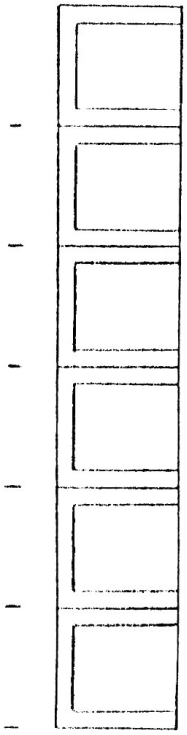



A1
A2
A3
X3
A4
X4
7bit目
6bit目
5bit目
H
4bit目
3bit目
2bit目
1bit目
データ、セレクタ
A4 (7)
CG、アドレス、
へ
A3 (7)
CG、アドレスへ
A2 (7)
HCとの一致回路へ
DB3(7)
DB2(7)
DB1(7)
アビット並列
ALU
(アドレスセンス アンプ・バッファ 4×7個)
M0
M1
M2
M3
4(H)
32(H)
RAM
32×4×7 = 896bit
L1 L2
L1' L2'
アキュムレータ等のレジスタ

◎ Yリピート・パターン 例

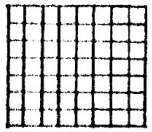


原形パターン

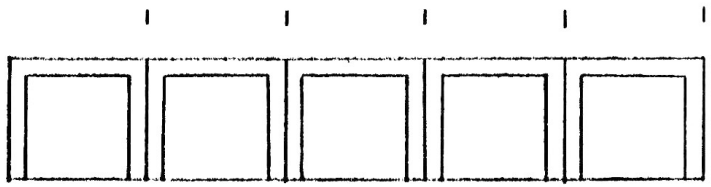


リピート スタート Y座標
リピート エンド Y座標
} ヒモ ソフトにて 任意指定可

◎ Xリピート・パターン 例

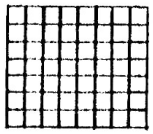


原形パターン

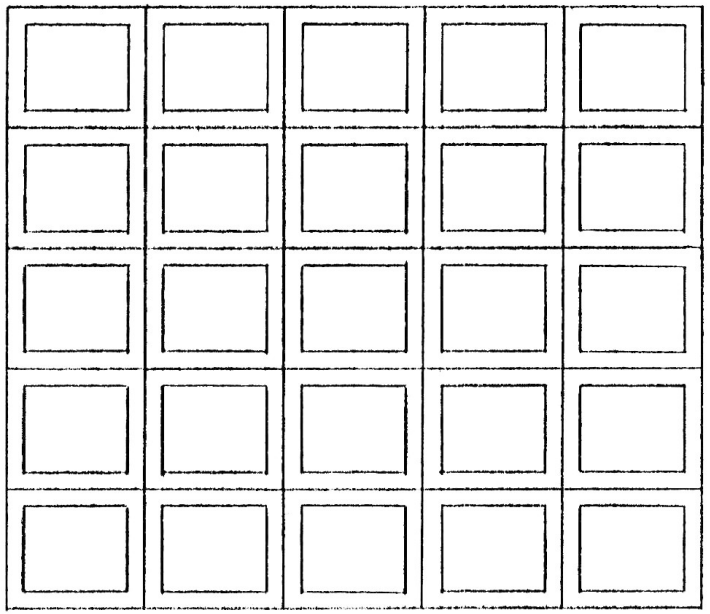


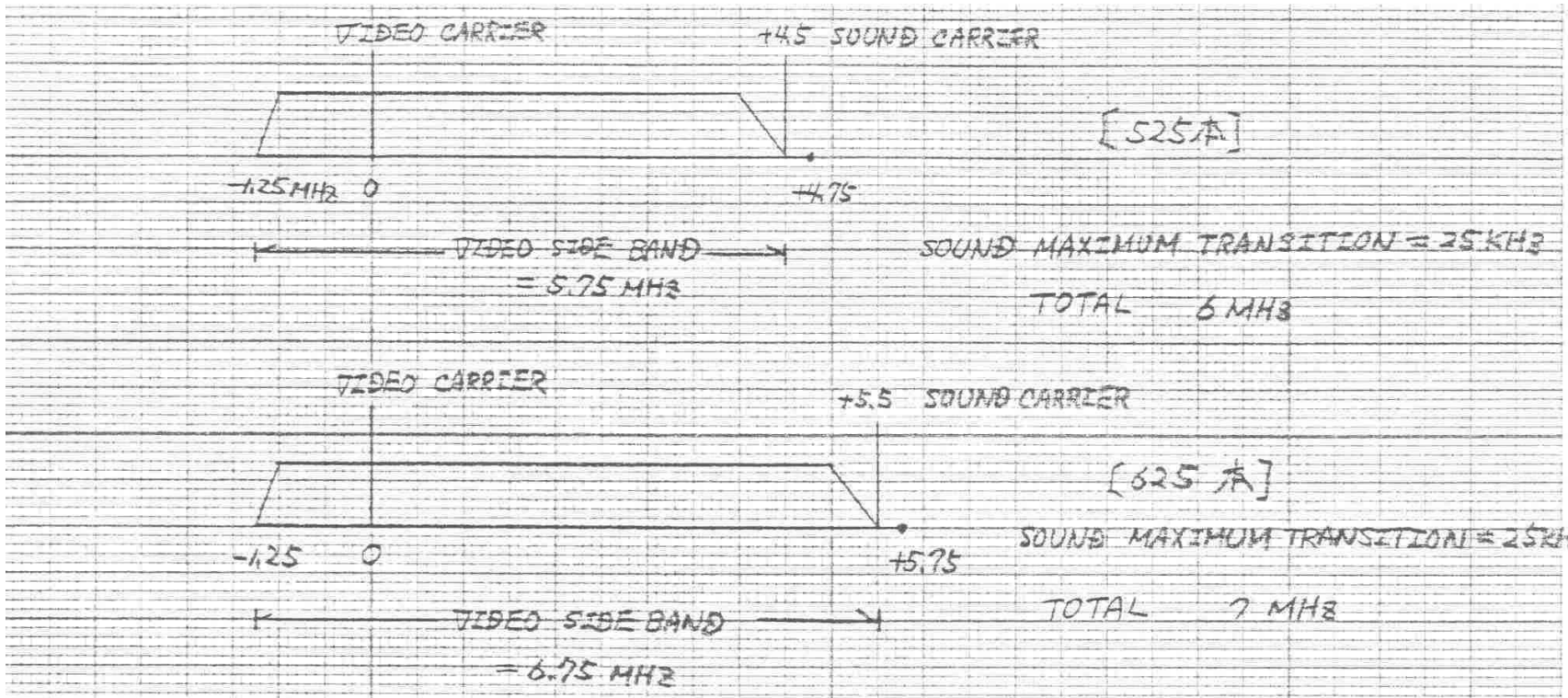
リピート・スタート X座標 は ソフトにて指定
リピート・エンド X座標 は パターン・アドレス によって固定

◎ XY リピート・パターン 例



原形パターン





*

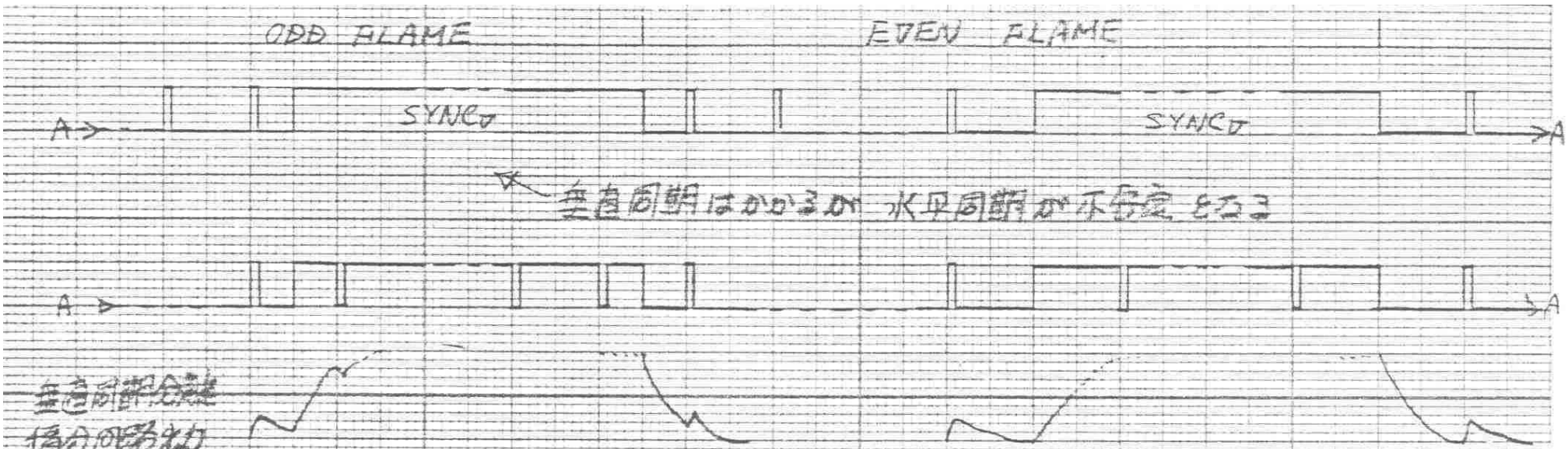
CHANNEL	1	2	3	4	5	6	7*	8*	9	10	11	12	
f(MHz)	90~	96~	102~	170~	176~	182~	188~	192~	198~	204~	210~	216~	JAPAN
	96	102	109	176	182	188	194	198	204	210	216	222	

* *

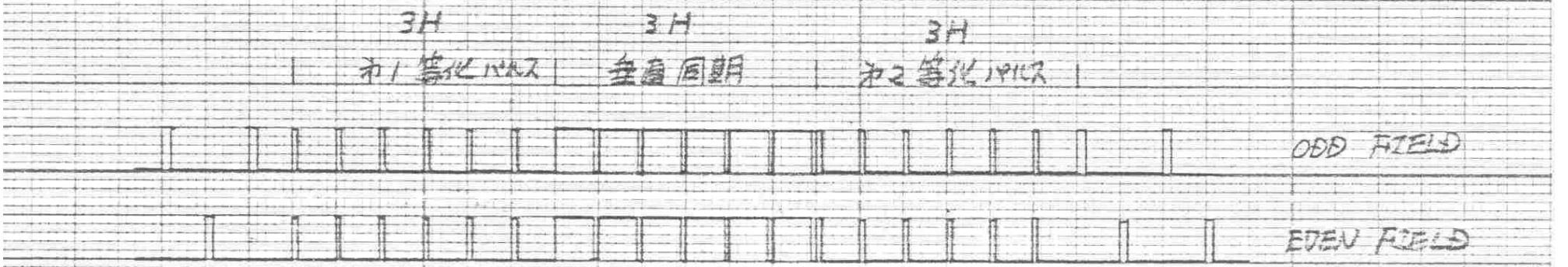
CHANNEL	2	3	4	5	6	7	8	9	10	11	12	13	
f(MHz)	54~	60~	66~	76~	82~	174~	180~	186~	192~	198~	204	210~	U.S.A.
	60	66	72	82	88	180	186	192	198	204	~210	216	

*

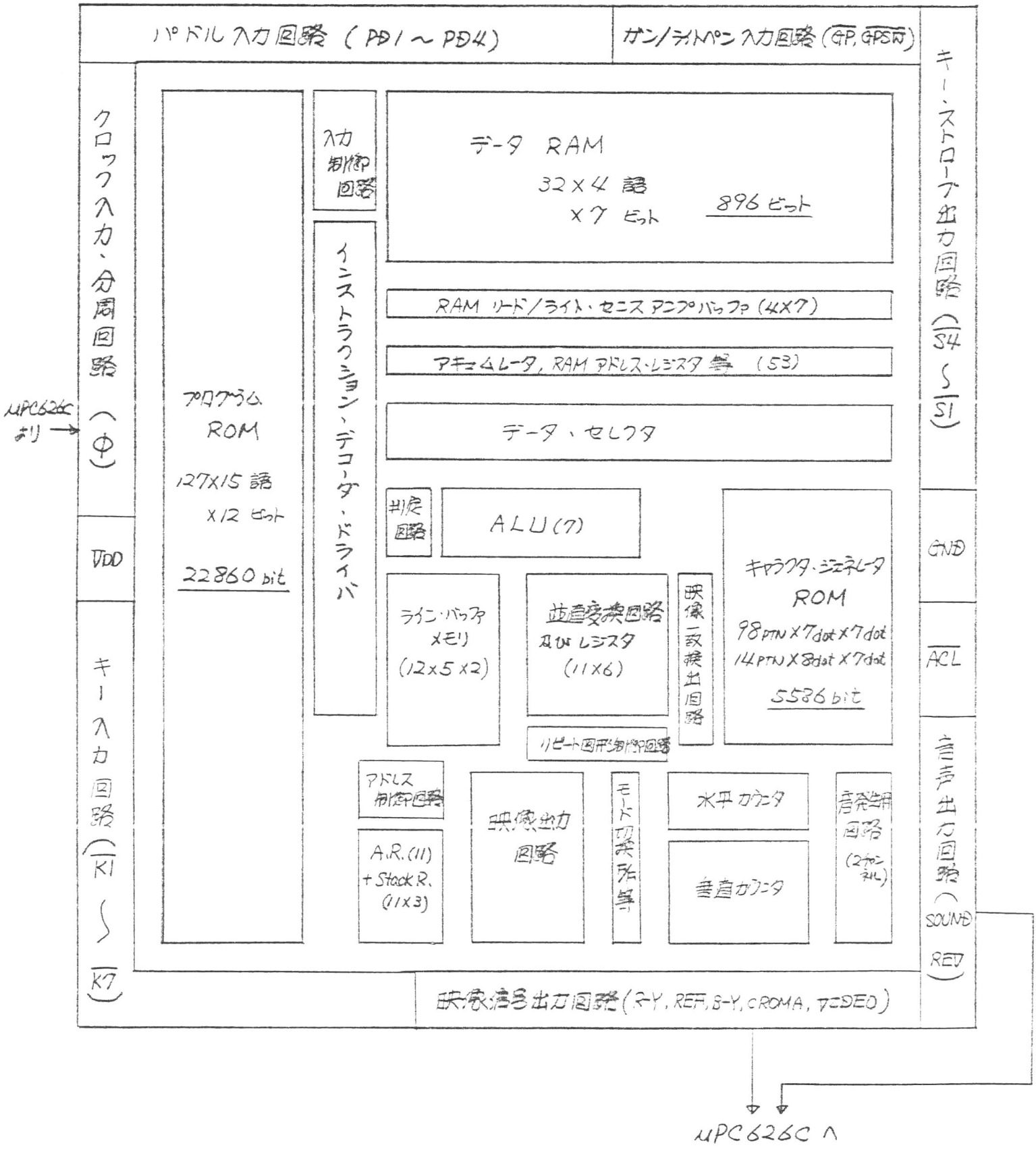
CHANNEL	E-1	2	3	4	5	6	7	8	9	10	E-11	
f(MHz)	40~	47~	54~	61~	174~	181~	188~	195~	202~	209~	216~	EUROPE
	47	54	61	68	181	188	195	202	209	216	223	



SYNC H と逆位相の信号を入れる事によって 水平同期を安定にする事ができるが 垂直同期分離出力は各フィールドによって異なってくる。



等価パルス群を垂直同期信号の前後に挿入する事によって上記欠点を除く事ができる。

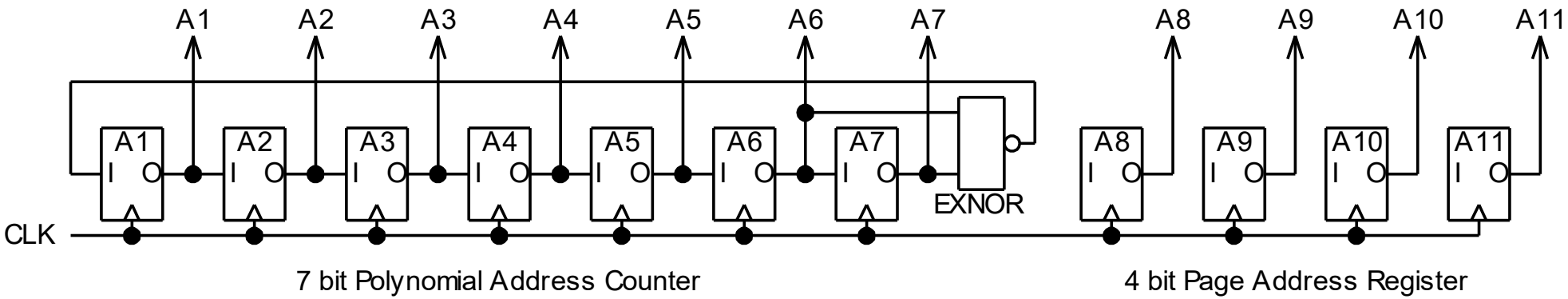


Right after architectural design, I made this block diagram (targeting 28 pin DIP (Dual In-line Package) pondering over abstract of the mask layout. In the middle of logic design, LSI test pins (Input pins of CH1, CH2, CH3, CH4, CH6 and output pins of R(I-3), G(I-2)) were added. Refer to "Test Pin Function" page more in details.

Going through the history, μPD777 was finally encapsulated in 42 pin plastic DIP and shipped.

7 bit Polynomial Address Counter Implemented

(A) Block Diagram

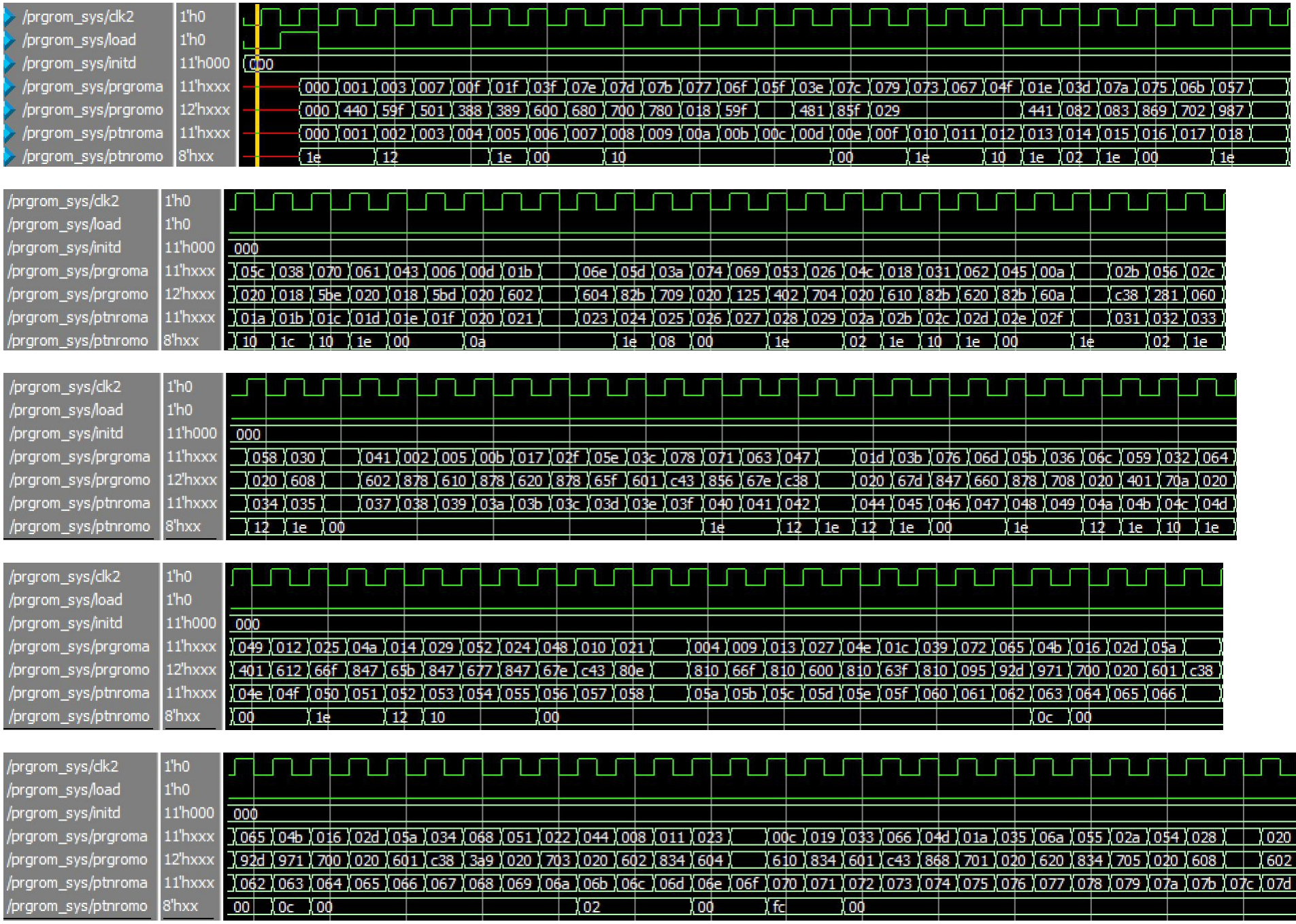


(B) Address Sequence

ADDR	HEX	ADDR	HEX	ADDR	HEX	ADDR	HEX	ADDR	HEX	ADDR	HEX	ADDR	HEX	ADDR	HEX
000 0000	00	111 0011	73	000 1101	0D	001 0101	15	111 0001	71	010 0101	25	011 1001	39	000 1100	0C
000 0001	01	110 0111	67	001 1011	1B	010 1011	2B	110 0011	63	100 1010	4A	111 0010	72	001 1001	19
000 0011	03	100 1111	4F	011 0111	37	101 0110	56	100 0111	47	001 0100	14	110 0101	65	011 0011	33
000 0111	07	001 1110	1E	110 1110	6E	010 1100	2C	000 1110	0E	010 1001	29	100 1011	4B	110 0110	66
000 1111	0F	011 1101	3D	101 1101	5D	101 1000	58	001 1101	1D	101 0010	52	001 0110	16	100 1101	4D
001 1111	1F	111 1010	7A	011 1010	3A	011 0000	30	011 1011	3B	010 0100	24	010 1101	2D	001 1010	1A
011 1111	3F	111 0101	75	111 0100	74	110 0000	60	111 0110	76	100 1000	48	101 1010	5A	011 0101	35
111 1110	7E	110 1011	6B	110 1001	69	100 0001	41	110 1101	6D	001 0000	10	011 0100	34	110 1010	6A
111 1101	7D	101 0111	57	101 0011	53	000 0010	02	101 1011	5B	010 0001	21	110 1000	68	101 0101	55
111 1011	7B	010 1110	2E	010 0110	26	000 0101	05	011 0110	36	100 0010	42	101 0001	51	010 1010	2A
111 0111	77	101 1100	5C	100 1100	4C	000 1011	0B	110 1100	6C	000 0100	04	010 0010	22	101 0100	54
110 1111	6F	011 1000	38	001 1000	18	001 0111	17	101 1001	59	000 1001	09	100 0100	44	010 1000	28
101 1111	5F	111 0000	70	011 0001	31	010 1111	2F	011 0010	32	001 0011	13	000 1000	08	101 0000	50
011 1110	3E	110 0001	61	110 0010	62	101 1110	5E	110 0100	64	010 0111	27	001 0001	11	010 0000	20
111 1100	7C	100 0011	43	100 0101	45	011 1100	3C	100 1001	49	100 1110	4E	010 0011	23	100 0000	40
111 1001	79	000 0110	06	000 1010	0A	111 1000	78	001 0010	12	001 1100	1C	100 0110	46		

Address Sequence (Available addresses = 2^7 (128) – 1 = 127)

(C) Verilog HDL Simulation Result



- The forth row exhibits the address generated by simulation which is the same as (B) Address Sequence.
- The fifth row exhibits µPD778 Program ROM codes and the seventh row exhibits µPD778 Pattern ROM codes.
- To verify, refer to "µPD778 Program ROM Contents" and "µPD778 Pattern ROM Contents".

(C-2) Verilog HDL Simulation Files

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Testbench for Address Counter + Program ROM + Pattern ROM
//  prgrom_sys.v
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
`timescale  1ns / 1ns

module   prgrom_sys;

// regs/wires/integers
reg      clk2, load;
reg      [10:0]initd;
wire     [10:0]prgroma;
wire     [12:1]prgromo;
wire     [10:0]ptnroma;
wire     [7:0]ptnromo;

// Simulation target
prgrom  prgrom(clk2, load, initd, prgroma, prgromo, ptnroma, ptnromo);

//--------------------
// Simulation vector
//--------------------
initial
begin
      clk2        = 1'b0;
      load        = 1'b0;
      initd[10:0] = 11'b0;
#50   load        = 1'b1;
#50   load        = 1'b0;

#6400  $finish;
end

// 20MHz clock generator
always  #25   clk2 = ~clk2;

endmodule
```

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Address Counter + Program ROM + Pattern ROM
//  prgrom.v
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
module  prgrom(clk2, load, initd, prgroma, prgromo, ptnroma, ptnromo);

// I/O definition
input   clk2;               // Phi2 clock                (H/L)
input   load;               // Load initial value        (H)
input   [10:0] initd;       // Initial data to be loaded (H/L)

output  [10:0] prgroma;     // Program ROM address       (H/L)
output  [12:1] prgromo;     // Program ROM output        (H/L)
output  [10:0] ptnroma;     // Pattern ROM address       (H/L)
output  [7:0]  ptnromo;     // Pattern ROM output        (H/L)

// Regs for random logic
wire    clk2, load;
wire    [10:0] initd;
reg     feedb;
reg     [10:0] prgroma;
wire    [11:0] prgromo;
reg     [10:0] ptnroma;
wire    [7:0]  ptnromo;

//---------------
// Random logic
//---------------

always @*
begin
  feedb = ~(prgroma[5] ^ prgroma[6]);
end

//---------------
// DFF
//---------------
always @ (posedge clk2)
begin
  if (load)
  begin
    prgroma[10:0] <= initd[10:0];
    ptnroma[10:0] <= initd[10:0];
  end
  else
  begin
    prgroma[6:1]  <= prgroma[5:0];
    prgroma[0]    <= feedb;
    ptnroma[10:0] <= ptnroma[10:0] + 1;
  end
end

//---------------
// ROM
//---------------
prgrom_mod  prgrom_1 (.prgromo(prgromo), .prgroma(prgroma));
ptnrom_mod  ptnrom_1 (.ptnromo(ptnromo), .ptnroma(ptnroma));

endmodule
```

µPD777 Instruction Table

ROM Out 11100 21098	ROM Out 0000000 7654321	Mnemonic	Judge	Hex Coding (N=0)	Function
00000	0000000	NOP	--	000	No Operation
	0000100	GPL	J	004	Skip if (Gun Port Latch) = 1
	0001000	H→NRM	--	008	Move H[5:1] to Line Buffer Register[5:1]
00000	0011000	H↔X	--	018	H[5:1]↔X4[5:1], 0→X4[7:6], 0→X3[7:1], 0→X1'[1], 0→A1'[1], L[2:1]↔L'[2:1]
	0100000	SRE	--	020	Subroutine End, Pop down address stack
	0101000	N→STB	--	028	Shift STB[4:1], N→STB[1]

ROM Out 11100 21098	ROM Out 0000 7654	ROM Out 000 321	Mnemonic	Judge	Hex Coding	Function
		001	4H BLK	J	049	Skip if (4H Horizontal Blank) = 1
00000	1001	010	VBLK	J	04A	Skip if (Vertical Blank) = 1, 0→M[[18:00],[3]][1]
		100	GPSW/	J	04C	Skip if (GP&SW/ input) = 1

ROM Out 11100 21098	ROM Out 0000000 7654321	Mnemonic	Hex Coding	Function
	1010100	A→MA	054	Move (A4[7:1],A3[7:1],A2[7:1],A1[7:1]) to M[H[5:1]][28:1]
00000	1011000	MA→A	058	Move M[H[5:1]][28:1] to (A4[7:1],A3[7:1],A2[7:1],A1[7:1])
	1011100	MA↔A	05C	Exchange (A4[7:1],A3[7:1],A2[7:1],A1[7:1]) and M[H[5:1]][28:1]

ROM Out 1110000 2109876	ROM Out 0 5	ROM Out 00 43	ROM Out 00 21	Mnemonic	Judge	Hex Coding	Function
0000011	0	00	00	SRE+1	--	060	Subroutine End, Pop down address stack, Skip

ROM Out					Mnemonic	Judge	Hex Coding	Function
11100 21098	0 7	00 65	00 43	00 21				
00000	0	11	00	00	PD1	J	030	Skip if (PD1 input) = 1
			01		PD2		034	Skip if (PD2 input) = 1
			10		PD3		038	Skip if (PD3 input) = 1
			11		PD4		03C	Skip if (PD4 input) = 1
	1		00		PD1	J/	070	Skip if (PD1 input) = 0
			01		PD2		074	Skip if (PD2 input) = 0
			10		PD3		078	Skip if (PD3 input) = 0
			11		PD4		07C	Skip if (PD4 input) = 0

ROM Out		Mnemonic	Judge	Hex Coding (K=0)	Function
11100 21098	0000000 7654321				
00001	KKKKKKK	M-K	BOJ	080	Skip if (M[H[5:1],L[2:1]][7:1]-K[7:1]) makes borrow

ROM Out				Mnemonic	Judge	Hex Coding (N=K=0)	Function
1110 2109	0 8	00 76	00000 54321				
0001	0	NN	KKKKK	M+K→M, N→L	CAJ	100	M[H[5:1],L[2:1]][7:1]+K[7:1]→M[H[5:1],L[2:1]][7:1], Skip if carry, N→L[2:1]
	1			M-K→M, N→L	BOJ	180	M[H[5:1],L[2:1]][7:1]-K[7:1]→M[H[5:1],L[2:1]][7:1], Skip if borrow, N→L[2:1]

ROM Out						Mnemonic	Judge	Hex Coding (N=0)	Function
1110 2109	00 87	0 6	0 5	00 43	00 21				
0010	00	0	0	00	NN	A1·A1, N→L	EQJ	200	Skip if (A1[7:1]·A1[7:1]) makes zero, N→L[2:1]
		1				A1·A1, N→L	EQJ/	220	Skip if (A1[7:1]·A1[7:1]) makes non zero, N→L[2:1]
		0		10		A1=A1, N→L	EQJ	208	Skip if (A1[7:1]-A1[7:1]) makes zero, N→L[2:1]
		1				A1=A1, N→L	EQJ/	228	Skip if (A1[7:1]-A1[7:1]) makes non zero, N→L[2:1]
		0		11		A1-A1, N→L	BOJ	20C	Skip if (A1[7:1]-A1[7:1]) makes borrow, N→L[2:1]
		1				A1-A1, N→L	BOJ/	22C	Skip if (A1[7:1]-A1[7:1]) makes non borrow, N→L[2:1]
		0	1	00		A1·A2, N→L	EQJ	210	Skip if (A1[7:1]·A2[7:1]) makes zero, N→L[2:1]
		1				A1·A2, N→L	EQJ/	230	Skip if (A1[7:1]·A2[7:1]) makes non zero, N→L[2:1]
		0		10		A1=A2, N→L	EQJ	218	Skip if (A1[7:1]-A2[7:1]) makes zero, N→L[2:1]
		1				A1=A2, N→L	EQJ/	238	Skip if (A1[7:1]-A2[7:1]) makes non zero, N→L[2:1]
		0		11		A1-A2, N→L	BOJ	21C	Skip if (A1[7:1]-A2[7:1]) makes borrow, N→L[2:1]
		1				A1-A2, N→L	BOJ/	23C	Skip if (A1[7:1]-A2[7:1]) makes non borrow, N→L[2:1]
	01	0	0	00		A2·A1, N→L	EQJ	240	Skip if (A2[7:1]·A1[7:1]) makes zero, N→L[2:1]
		1				A2·A1, N→L	EQJ/	260	Skip if (A2[7:1]·A1[7:1]) makes non zero, N→L[2:1]
		0		10		A2=A1, N→L	EQJ	248	Skip if (A2[7:1]-A1[7:1]) makes zero, N→L[2:1]
		1				A2=A1, N→L	EQJ/	268	Skip if (A2[7:1]-A1[7:1]) makes non zero, N→L[2:1]
		0		11		A2-A1, N→L	BOJ	24C	Skip if (A2[7:1]-A1[7:1]) makes borrow, N→L[2:1]
		1				A2-A1, N→L	BOJ/	26C	Skip if (A2[7:1]-A1[7:1]) makes non borrow, N→L[2:1]
		0	1	00		A2·A2, N→L	EQJ	250	Skip if (A2[7:1]·A2[7:1]) makes zero, N→L[2:1]
		1				A2·A2, N→L	EQJ/	270	Skip if (A2[7:1]·A2[7:1]) makes non zero, N→L[2:1]
		0		10		A2=A2, N→L	EQJ	258	Skip if (A2[7:1]-A2[7:1]) makes zero, N→L[2:1]
		1				A2=A2, N→L	EQJ/	278	Skip if (A2[7:1]-A2[7:1]) makes non zero, N→L[2:1]
		0		11		A2-A2, N→L	BOJ	25C	Skip if (A2[7:1]-A2[7:1]) makes borrow, N→L[2:1]
		1				A2-A2, N→L	BOJ/	27C	Skip if (A2[7:1]-A2[7:1]) makes non borrow, N→L[2:1]

ROM Out						Mnemonic	Judge	Hex Coding (N=0)	Function
1110 2109	00 87	0 6	0 5	00 43	00 21				
0010	10	0	0	00	NN	M·A1, N→L	EQJ	280	Skip if (M[H[5:1],L[2:1]][7:1]·A1[7:1]) makes zero, N→L[2:1]
		1				M·A1, N→L	EQJ/	2A0	Skip if (M[H[5:1],L[2:1]][7:1]·A1[7:1]) makes non zero, N→L[2:1]
		0		10		M=A1, N→L	EQJ	288	Skip if (M[H[5:1],L[2:1]][7:1]-A1[7:1]) makes zero, N→L[2:1]
		1				M=A1, N→L	EQJ/	2A8	Skip if (M[H[5:1],L[2:1]][7:1]-A1[7:1]) makes non zero, N→L[2:1]
		0		11		M-A1, N→L	BOJ	28C	Skip if (M[H[5:1],L[2:1]][7:1]-A1[7:1]) makes borrow, N→L[2:1]
		1				M-A1, N→L	BOJ/	2AC	Skip if (M[H[5:1],L[2:1]][7:1]-A1[7:1]) makes non borrow, N→L[2:1]
		0	1	00		M·A2, N→L	EQJ	290	Skip if (M[H[5:1],L[2:1]][7:1]·A2[7:1]) makes zero, N→L[2:1]
		1				M·A2, N→L	EQJ/	2B0	Skip if (M[H[5:1],L[2:1]][7:1]·A2[7:1]) makes non zero, N→L[2:1]
		0		10		M=A2, N→L	EQJ	298	Skip if (M[H[5:1],L[2:1]][7:1]-A2[7:1]) makes zero, N→L[2:1]
		1				M=A2, N→L	EQJ/	2B8	Skip if (M[H[5:1],L[2:1]][7:1]-A2[7:1]) makes non zero, N→L[2:1]
		0		11		M-A2, N→L	BOJ	29C	Skip if (M[H[5:1],L[2:1]][7:1]-A2[7:1]) makes borrow, N→L[2:1]
		1				M-A2, N→L	BOJ/	2BC	Skip if (M[H[5:1],L[2:1]][7:1]-A2[7:1]) makes non borrow, N→L[2:1]
	11	0	0	00		H·A1, N→L	EQJ	2C0	Skip if (H[5:1]·A1[5:1]) makes zero, N→L[2:1]
		1				H·A1, N→L	EQJ/	2E0	Skip if (H[5:1]·A1[5:1]) makes non zero, N→L[2:1]
		0		10		H=A1, N→L	EQJ	2C8	Skip if (H[5:1]-A1[5:1]) makes zero, N→L[2:1]
		1				H=A1, N→L	EQJ/	2E8	Skip if (H[5:1]-A1[5:1]) makes non zero, N→L[2:1]
		0		11		H-A1, N→L	BOJ	2CC	Skip if (H[5:1]-A1[5:1]) makes borrow, N→L[2:1]
		1				H-A1, N→L	BOJ/	2EC	Skip if (H[5:1]-A1[5:1]) makes non borrow, N→L[2:1]
		0	1	00		H·A2, N→L	EQJ	2D0	Skip if (H[5:1]·A2[5:1]) makes zero, N→L[2:1]
		1				H·A2, N→L	EQJ/	2F0	Skip if (H[5:1]·A2[5:1]) makes non zero, N→L[2:1]
		0		10		H=A2, N→L	EQJ	2D8	Skip if (H[5:1]-A2[5:1]) makes zero, N→L[2:1]
		1				H=A2, N→L	EQJ/	2F8	Skip if (H[5:1]-A2[5:1]) makes non zero, N→L[2:1]
		0		11		H-A2, N→L	BOJ	2DC	Skip if (H[5:1]-A2[5:1]) makes borrow, N→L[2:1]
		1				H-A2, N→L	BOJ/	2FC	Skip if (H[5:1]-A2[5:1]) makes non borrow, N→L[2:1]

ROM Out				Mnemonic	Judge	Hex Coding (N=0)	Function
1110 2109	00 87	0000 6543	00 21				
0011	00	0000	NN	N→L	--	300	N→L[2:1]
		0100	NN	A2→A1, N→L	--	310	Move A2[7:1] to A1[7:1], N→L[2:1]
		0010	00	A1→FLS, 0→L	--	308	Move A1[7:1] to FLS[7:1], 0→L[2:1]
			01	A1→FRS, 1→L	--	309	Move A1[7:1] to FRS[7:1], 1→L[2:1]
			1N	A1→MODE, 1N→L	--	30A	Move A1[7:1] to MODE[7:1], 1N→L[2:1]
		0110	NN	A1→RS, N→L	--	318	Right shift A1[7:1], 0→A1[7], N→L[2:1]
	01	0000	NN	A1→A2, N→L	--	340	Move A1[7:1] to A2[7:1], N→L[2:1]
		0010	00	A2→FLS, 0→L	--	348	Move A2[7:1] to FLS[7:1], 0→L[2:1]
			01	A2→FRS, 1→L	--	349	Move A2[7:1] to FRS[7:1], 1→L[2:1]
			1N	A2→MODE, 1N→L	--	34A	Move A2[7:1] to MODE[7:1], 1N→L[2:1]
		0110	NN	A2→RS, N→L	--	358	Right shift A2[7:1], 0→A2[7], N→L[2:1]
	10	0010	00	M→FLS, 0→L	--	388	Move M[H[5:1],L[2:1]][7:1] to FLS[7:1], 0→L[2:1]
			01	M→FRS, 1→L	--	389	Move M[H[5:1],L[2:1]][7:1] to FRS[7:1], 1→L[2:1]
			1N	M→MODE, 1N→L	--	38A	Move M[H[5:1],L[2:1]][7:1] to MODE[7:1], 1N→L[2:1]
		0110	NN	M→RS, N→L	--	398	Right shift M[H[5:1],L[2:1]][7:1], 0→M[H[5:1],L[2:1]][7], N→L[2:1]

ROM Out				Mnemonic	Judge	Hex Coding (N=0)	Function
11100 21098	000 765	00 43	00 21				
00110	010	00	NN	A1·A1→A1, N→L	--	320	AND A1[7:1] and A1[7:1], store to A1[7:1], N→L[2:1]
		01		A1+A1→A1, N→L	CAJ	324	Add A1[7:1] and A1[7:1], store to A1[7:1], N→L[2:1]
		10		A1vA1→A1, N→L	--	328	OR A1[7:1] and A1[7:1], store to A1[7:1], N→L[2:1]
		11		A1-A1→A1, N→L	BOJ	32C	Subtract A1[7:1] and A1[7:1], store to A1[7:1], Skip if borrow, N→L[2:1]
	011	00		A1·A2→A1, N→L	--	330	AND A1[7:1] and A2[7:1], store to A1[7:1], N→L[2:1]
		01		A1+A2→A1, N→L	CAJ	334	Add A1[7:1] and A2[7:1], store to A1[7:1], N→L[2:1]
		10		A1vA2→A1, N→L	--	338	OR A1[7:1] and A2[7:1], store to A1[7:1], N→L[2:1]
		11		A1-A2→A1, N→L	BOJ	33C	Subtract A1[7:1] and A2[7:1], store to A1[7:1], Skip if borrow, N→L[2:1]
	110	00		A2·A1→A2, N→L	--	360	AND A2[7:1] and A1[7:1], store to A2[7:1], N→L[2:1]
		01		A2+A1→A2, N→L	CAJ	364	Add A2[7:1] and A1[7:1], store to A2[7:1], N→L[2:1]
		10		A2vA1→A2, N→L	--	368	OR A2[7:1] and A1[7:1], store to A2[7:1], N→L[2:1]
		11		A2-A1→A2, N→L	BOJ	36C	Subtract A2[7:1] and A1[7:1], store to A2[7:1], Skip if borrow, N→L[2:1]
	111	00		A2·A2→A2, N→L	--	370	AND A2[7:1] and A2[7:1], store to A2[7:1], N→L[2:1]
		01		A2+A2→A2, N→L	CAJ	374	Add A2[7:1] and A2[7:1], store to A2[7:1], N→L[2:1]
		10		A2vA2→A2, N→L	--	378	OR A2[7:1] and A2[7:1], store to A2[7:1], N→L[2:1]
		11		A2-A2→A2, N→L	BOJ	37C	Subtract A2[7:1] and A2[7:1], store to A2[7:1], Skip if borrow, N→L[2:1]

ROM Out			Mnemonic	Hex Coding (N=0)	Function
1110000 2109876	000 543	00 21			
0011100	000	NN	A1→M, N→L	380	Move A1[7:1] to M[H[5:1],L[2:1]][7:1], N→L[2:1]
	100		A2→M, N→L	390	Move A2[7:1] to M[H[5:1],L[2:1]][7:1], N→L[2:1]
	001		M↔A1, N→L	384	Exchange M[H[5:1],L[2:1]][7:1] and A1[7:1], N→L[2:1]
	101		M↔A2, N→L	394	Exchange M[H[5:1],L[2:1]][7:1] and A2[7:1], N→L[2:1]
	011		M→A1, N→L	38C	Move M[H[5:1],L[2:1]][7:1] to A1[7:1], N→L[2:1]
	111		M→A2, N→L	39C	Move M[H[5:1],L[2:1]][7:1] to A2[7:1], N→L[2:1]

ROM Out				Mnemonic	Judge	Hex Coding (N=0)	Function
11100 21098	000 765	00 43	00 21				
00111	011	00	NN	M·A2→M, N→L	--	3B0	AND M[H[5:1],L[2:1]][7:1] and A2[7:1], store to M[H[5:1],L[2:1]][7:1], N→L[2:1]
		01		M+A2→M, N→L	CAJ	3B4	Add M[H[5:1],L[2:1]][7:1] and A2[7:1], store to M[H[5:1],L[2:1]][7:1], N→L[2:1] Skip if carry
		10		MvA2→M, N→L	--	3B8	OR M[H[5:1],L[2:1]][7:1] and A2[7:1], store to M[H[5:1],L[2:1]][7:1], N→L[2:1]
		11		M-A2→M, N→L	BOJ	3BC	Subtract M[H[5:1],L[2:1]][7:1] and A2[7:1], store to M[H[5:1],L[2:1]][7:1], N→L[2:1] Skip if borrow
	010	00		M·A1→M, N→L	--	3A0	AND M[H[5:1],L[2:1]][7:1] and A1[7:1], store to M[H[5:1],L[2:1]][7:1], N→L[2:1]
		01		M+A1→M, N→L	CAJ	3A4	Add M[H[5:1],L[2:1]][7:1] and A1[7:1], store to M[H[5:1],L[2:1]][7:1], N→L[2:1] Skip if carry
		10		MvA1→M, N→L	--	3A8	OR M[H[5:1],L[2:1]][7:1] and A1[7:1], store to M[H[5:1],L[2:1]][7:1], N→L[2:1]
		11		M-A1→M, N→L	BOJ	3AC	Subtract M[H[5:1],L[2:1]][7:1] and A1[7:1], store to M[H[5:1],L[2:1]][7:1], N→L[2:1] Skip if borrow

ROM Out			Mnemonic	Hex Coding (N=0)	Function
1110000 2109876	000 543	00 21			
0011110	000	NN	A1→H, N→L	3C0	Move A1[5:1] to H[5:1], N→L[2:1]
	100		A2→H, N→L	3D0	Move A2[5:1] to H[5:1], N→L[2:1]
	011		H→A1, N→L	3CC	Move H[5:1] to A1[5:1], 0→A1[7:6], N→L[2:1]
	111		H→A2, N→L	3DC	Move H[5:1] to A2[5:1], 0→A2[7:6], N→L[2:1]

ROM Out				Mnemonic	Judge	Hex Coding (N=0)	Function
11100 21098	000 765	00 43	00 21				
00111	110	00	NN	H·A1→H, N→L	--	3E0	AND H[5:1] and A1[5:1], store to H[5:1], N→L[2:1]
		01		H+A1→H, N→L	CAJ	3E4	Add H[5:1] and A1[5:1], store to H[5:1], N→L[2:1]
		10		HvA1→H, N→L	--	3E8	OR H[5:1] and A1[5:1], store to H[5:1], N→L[2:1]
		11		H-A1→H, N→L	BOJ	3EC	Subtract H[5:1] and A1[5:1], store to H[5:1], Skip if borrow, N→L[2:1]
	111	00		H·A2→H, N→L	--	3F0	AND H[5:1] and A2[5:1], store to H[5:1], N→L[2:1]
		01		H+A2→H, N→L	CAJ	3F4	Add H[5:1] and A2[5:1], store to H[5:1], N→L[2:1]
		10		HvA2→H, N→L	--	3F8	OR H[5:1] and A2[5:1], store to H[5:1], N→L[2:1]
		11		H-A2→H, N→L	BOJ	3FC	Subtract H[5:1] and A2[5:1], store to H[5:1], Skip if borrow, N→L[2:1]

ROM Out			Mnemonic	Hex Coding (D=G=K=S=N=0)	Function
11100 21098	000000 765432	0 1			
01000	000000	N	N→A11	400	N→A[11]
	000001		JPM, 0→L, N→A11	402	Jump to (000,M[H[5:1],L[2:1]][5:1],1N),0→L[2:1], N→A[11]
	1DGKS0		D→D, G→G, K→K, S→S, N→A11	440	Set D to DISP, G to GPE, K to KIE, S to SME, N→A[11]

ROM Out				Mnemonic	Judge	Hex Coding (K=0)	Function
11100 21098	0 7	0 6	00000 54321				
01001	0	0	KKKKK	H-K→H	BOJ	480	H[5:1]-K[5:1]→H[5:1], Skip if borrow
	1			H+K→H	CAJ	4C0	H[5:1]+K[5:1]→H[5:1], Skip if carry

ROM Out			Mnemonic	Hex Coding (K=0)	Function
111 210	00 98	0000000 7654321			
010	10	KKKKKKK	K→M	500	When (KIE=0)&(SME=0), Store K[7:1] to M[H[5:1],L[2:1]][7:1]
					When (KIE=1), Store KIN[7:1] to M[H[5:1],L[2:1]][7:1]
					When (SME=1), Store HCL[7:1] to M[H[5:1],L[2:1]][7:1]
	11		K→L,H	580	Store K[7:6] to L[2:1] and K[5:1] to H[5:1]
011	00		K→A1	600	Store K[7:1] to A1[7:1]
	01		K→A2	680	Store K[7:1] to A2[7:1]
	10		K→A3	700	Store K[7:1] to A3[7:1]
	11		K→A4	780	Store K[7:1] to A4[7:1]

ROM Out	Mnemonic	Hex Coding (K=0)	Function
111000000000 210987654321			
10KKKKKKKKK	JP KKK	800	Move K[10:1] to A[10:1], Jump to A[11:1]
11KKKKKKKKK	JS KKK	C00	Move K[10:1] to A[10:1], 0 to A11, Jump to A[11:1], Push next A[11:1] up to ROM address stack

- Page address A[11] is set by “0→A11”, “1→A11”, or “JS”.
- Maximum subroutine nesting level is three.

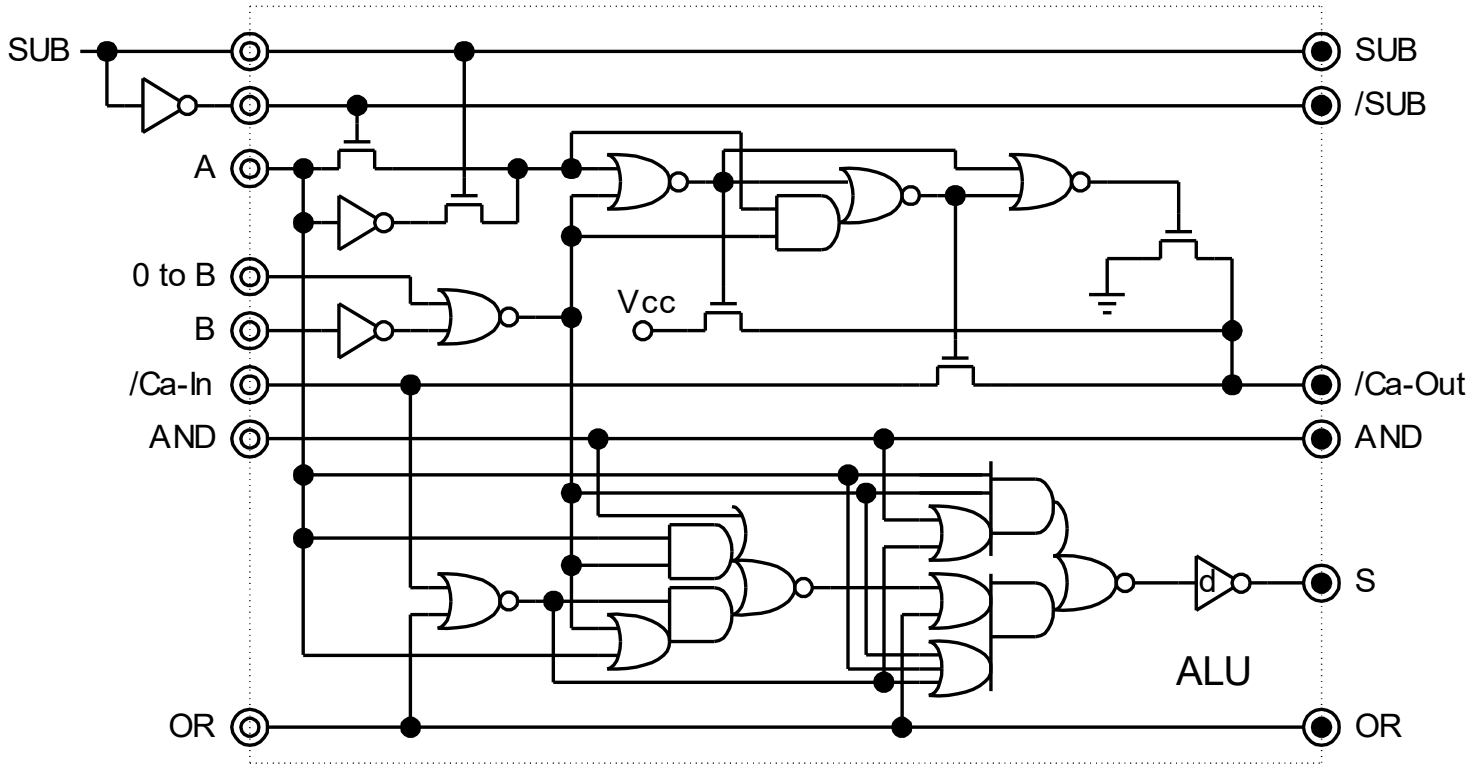
µPD777 ALU (Arithmetic Logic Unit)

(A) Truth Table

	Control signals				Input data			Output data	
	SUB	**AND**	**OR**	**0→B**	**A**	**B**	**/Ca-In**	**S**	**/Ca-Out**
Addition	0	0	0	0	0	0	1	0	1
							0	1	1
					0	1	1	1	1
							0	0	0
					1	0	1	1	1
							0	0	0
					1	1	1	0	0
							0	1	0
Subtraction	1	0	0		0	0	1	0	1
							0	1	0
					0	1	1	1	0
							0	0	0
					1	0	1	1	1
							0	0	1
					1	1	1	0	1
							0	1	0
AND	0	1	0		0	0	1	0	1
					0	1		0	
					1	0		0	
					1	1		1	
OR	0	0	1		0	0	1	0	1
					0	1		1	
					1	0		1	
					1	1		1	
Addition	0	0	0	1	0	0	1	0	1
					1			1	
Subtraction	1	0	0		0			0	
					1			1	
AND	0	1	0		0			0	
					1			0	
OR	0	0	1		0			0	
					1			1	

- Same value of S (Sum) is output regardless of addition and subtraction.
- Carry output differs between addition and subtraction.
- ALU works as an adder if ALU control signals (“Subtraction”, “AND”, and “OR”) are not asserted.
- “0→B” functions as ALU disable.

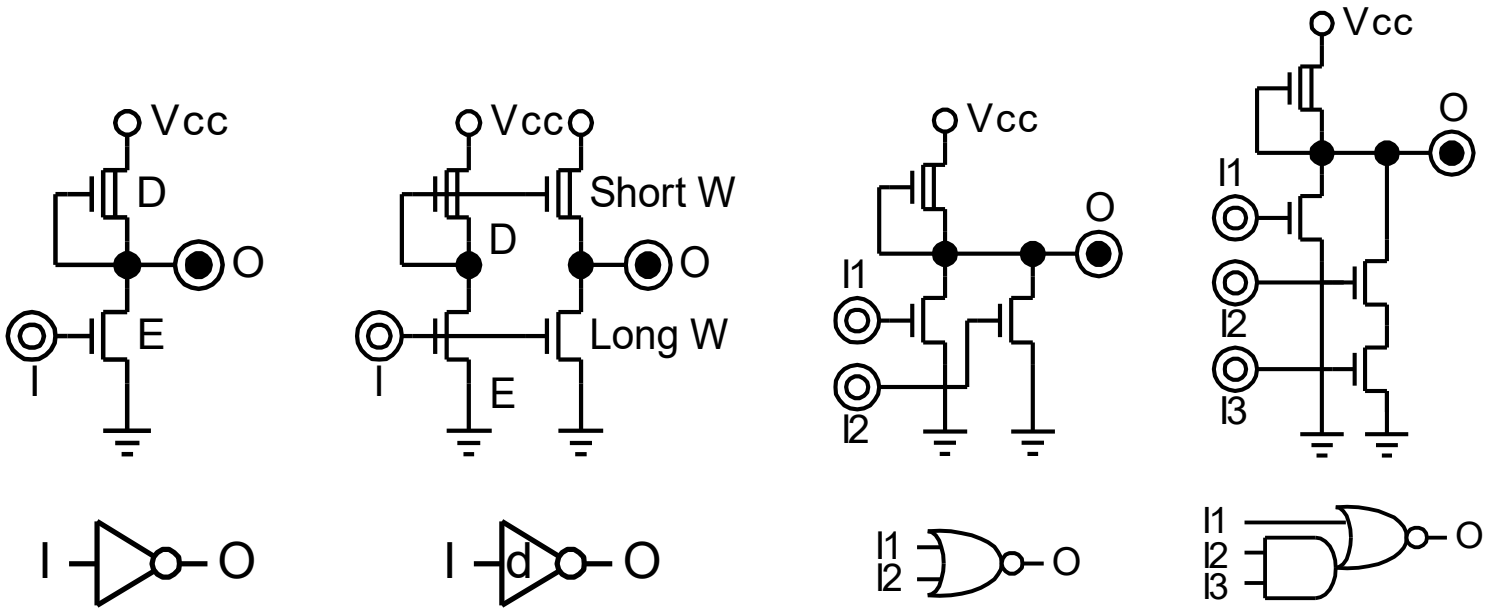
(B) Logic



One bit ALU (Arithmetic Logic Unit; Enclosed by dashed lines) implemented on μPD777

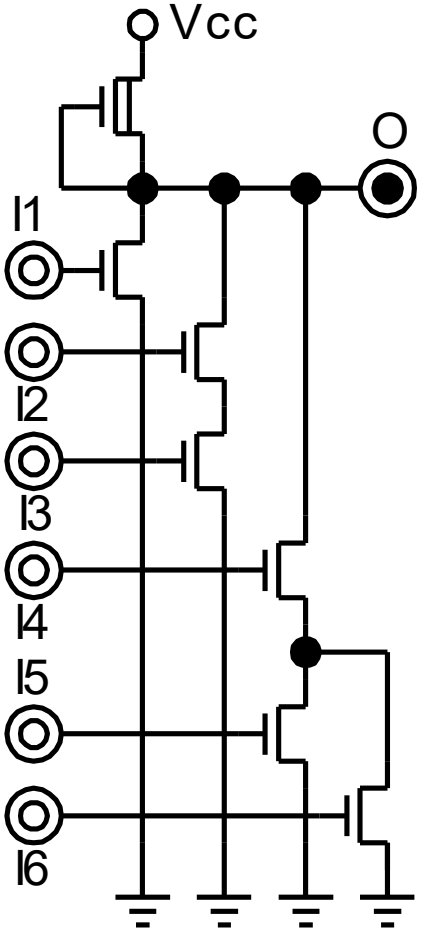
The carry propagation method applied to the ALU does not accumulate delay by addition and subtraction (The delay converges inside one bit ALU not affected by the operation and the result). Therefore, the operation speed is amazingly fast.

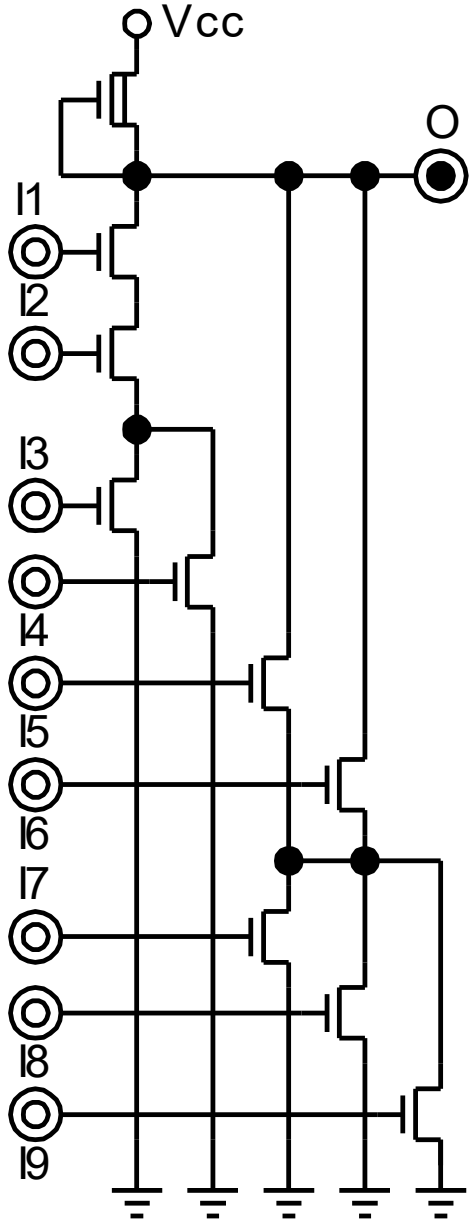
(C) MOS Transistor Interconnects vs. Logic Symbol

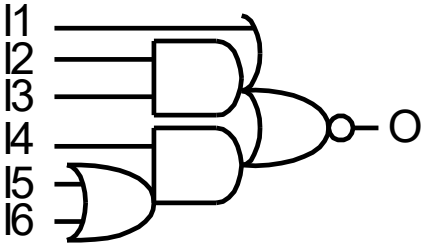


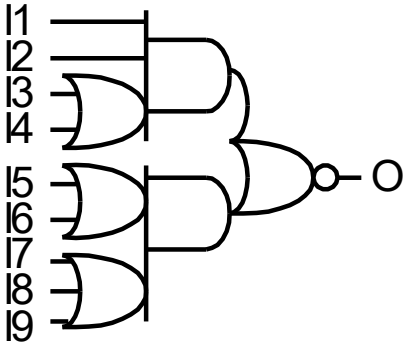
Inverter | Inverted driver | NOR2 | AND2 to NOR2

D: Depletion type silicon gate MOS transistor
E: Enhancement type silicon gate MOS transistor



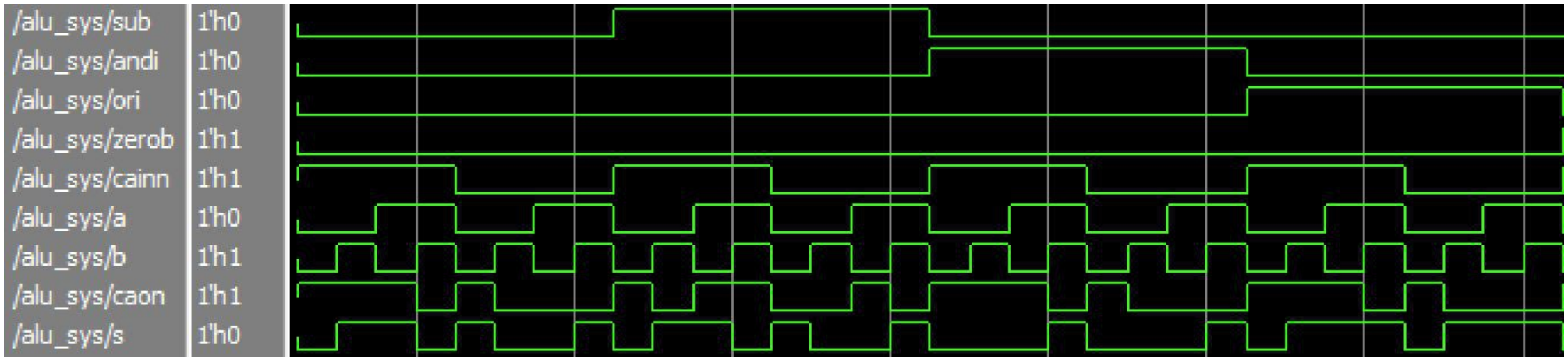




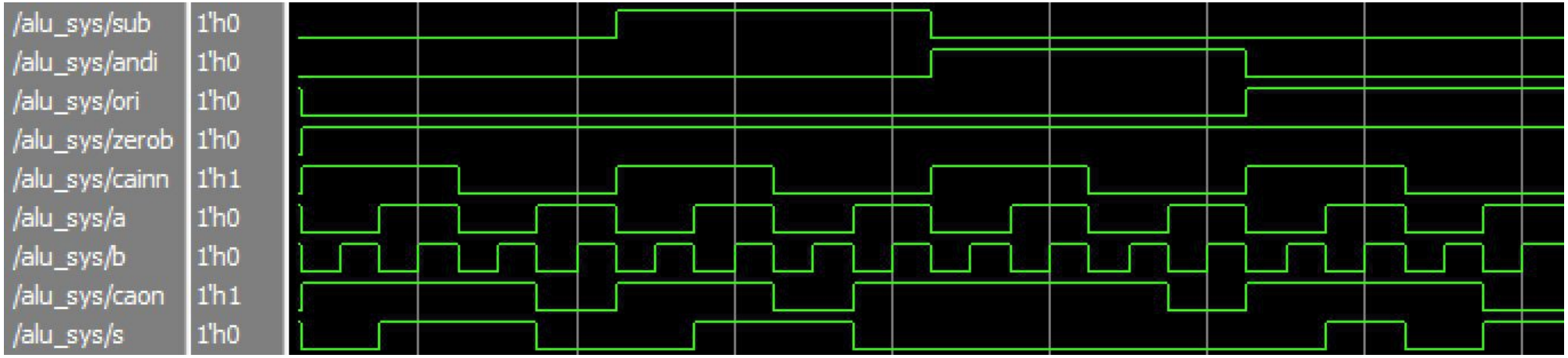


Combinational Logic used in ALU

(D) Verilog HDL Simulation Result



0→B (/alu_sys/zerob) = L



0→B (/alu_sys/zerob) = H (Since /Ca-in (/alu_sys/cainn) is always high in actual application, /Ca-out (/alu_sys/caon) is always high.)

Simulation wave form above exhibits the same as (A) Truth Table.

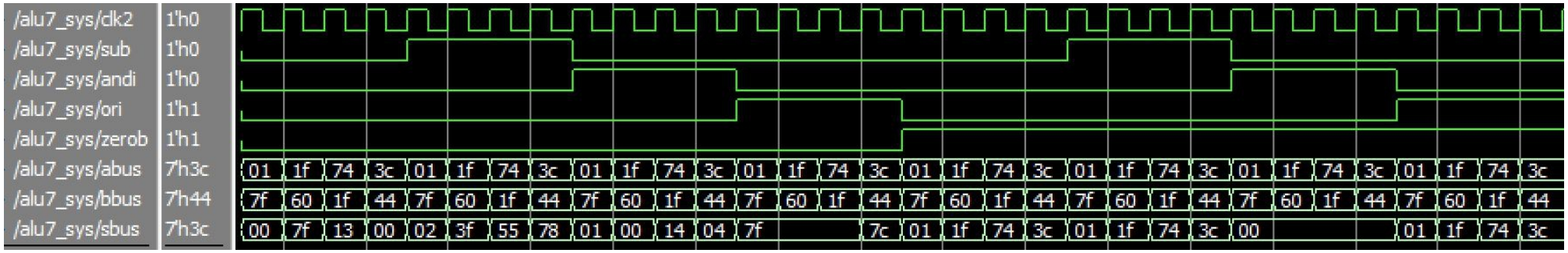
(D-2) Verilog HDL Simulation Files

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Testbench for ALU
//  alu_sys.v
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
`timescale  1ns / 1ns

module   alu_sys;

reg      clk;
reg      sub, andi, ori, zerob, cainn, a, b;
wire     caon, s;

integer  i1, i2, i3;

// Simulation target
alu      alu(sub, andi, ori, zerob, cainn, a, b, caon, s);

initial
begin
  clk = 1'b1;

// Signal generation
  for(i3 = 0;  i3 < 2;  i3 = i3 + 1)
  begin
    for(i2 = 0;  i2 < 4;  i2 = i2 + 1)
    begin
      for(i1 = 0;  i1 < 8;  i1 = i1 + 1)
      begin
#50     case (i1)
          3'b000 : begin cainn = 1'b1; a = 1'b0; b = 1'b0; end
          3'b001 : begin cainn = 1'b1; a = 1'b0; b = 1'b1; end
          3'b010 : begin cainn = 1'b1; a = 1'b1; b = 1'b0; end
          3'b011 : begin cainn = 1'b1; a = 1'b1; b = 1'b1; end
          3'b100 : begin cainn = 1'b0; a = 1'b0; b = 1'b0; end
          3'b101 : begin cainn = 1'b0; a = 1'b0; b = 1'b1; end
          3'b110 : begin cainn = 1'b0; a = 1'b1; b = 1'b0; end
          3'b111 : begin cainn = 1'b0; a = 1'b1; b = 1'b1; end
        endcase
        case (i2)
          2'b00  : begin sub = 1'b0; andi = 1'b0; ori = 1'b0; end
          2'b01  : begin sub = 1'b1; andi = 1'b0; ori = 1'b0; end
          2'b10  : begin sub = 1'b0; andi = 1'b1; ori = 1'b0; end
          2'b11  : begin sub = 1'b0; andi = 1'b0; ori = 1'b1; end
        endcase
        case (i3)
          1'b0   : zerob = 1'b0;
          1'b1   : zerob = 1'b1;
        endcase
      end
    end
  end
#100  $finish;
end

// 20MHz clock generator
always  #25   clk = ~clk;

endmodule
```

```
//~~~~~~~~~~~~~~~~~~~~~~
// 1 bit ALU
//  alu.v
//~~~~~~~~~~~~~~~~~~~~~~
module  alu(sub, andi, ori, zerob, cainn, a, b, caon, s);

// I/O definition
input   sub;            // Subtraction  (H)
input   andi;           // AND          (H)
input   ori;            // OR           (H)
input   zerob;          // 0 --> B      (H)
input   cainn;          // Carry in     (L)
input   a;              // A            (H)
input   b;              // B            (H)

output  caon;           // Carry out    (L)
output  s;              // Sum          (H)

// Regs for random logic
wire  sub, andi, ori, zerob, cainn, a, b;
reg   caon, s;
reg   g1, g2, g3, g4, g5, g6, bb;

//---------------
// Random logic
//---------------
always @*
begin
  g1 = (sub) ? (~a) : (a);
  bb = ~(zerob | ~b);
  g5 = ~(cainn | ori);
  g2 = ~(g1 | bb);
  g3 = ~((g1 & bb) | g2);
  g4 = ~(g2 | g3);
  g6 = ~(((a | bb) & g5) | (a & bb) | andi);
  s  = (((a | bb| g5) & (ori | g6)) | ((andi | g5) & a & bb));
end

always @*
begin
  casex ({g2, g3, g4})
    3'b1xx : caon = 1'b1;
    3'bx1x : caon = cainn;
    3'bxx1 : caon = 1'b0;
  endcase
end

endmodule
```

7 bit Parallel ALU Implemented on μPD777

(A) Verilog HDL Simulation Result



- (“abus[7:1]” + “bbus[7:1]”) = “sbus[7:1]” (Addition) when “sub” = 0, “andi” = 0, and “ori” = 0.
- (“abus[7:1]” - “bbus[7:1]”) = “sbus[7:1]” (Subtraction) when “sub” = 1, “andi” = 0, and “ori” = 0.
- (“abus[7:1]” & “bbus[7:1]”) = “sbus[7:1]” (Bitwise AND) when “sub” = 0, “andi” = 1, and “ori” = 0.
- (“abus[7:1]” | “bbus[7:1]”) = “sbus[7:1]” (Bitwise OR) when “sub” = 0, “andi” = 0, and “ori” = 1.
- When “zerob” = 1, “bbus[7:1]” becomes zero.

(A-2) Verilog HDL Simulation Files

```
//~~~~~~~~~~~~~~~~~~~~~
// Testbench for ALU7
//  alu7_sys.v
//~~~~~~~~~~~~~~~~~~~~~
`timescale  1ns / 1ns

module   alu7_sys;

// regs/wires/integers
reg   clk2, sub, andi, ori, zerob;
reg   [7:1]abus;
reg   [7:1]bbus;
wire  [7:1]sbus;

integer  i1, i2, i3;

// Simulation target
alu7  alu7(clk2, sub, andi, ori, zerob, abus, bbus, sbus);

//--------------------
// Simulation vector
//--------------------

initial
begin
  clk2 = 1'b1;

// Signal generation
  for(i3 = 0;  i3 < 2;  i3 = i3 + 1)
  begin
    for(i2 = 0;  i2 < 4;  i2 = i2 + 1)
    begin
      for(i1 = 0;  i1 < 4;  i1 = i1 + 1)
      begin
#50     case (i1)
          2'b00 : begin abus[7:1] = 7'h01; bbus[7:1] = 7'h7f; end
          2'b01 : begin abus[7:1] = 7'h1f; bbus[7:1] = 7'h60; end
          2'b10 : begin abus[7:1] = 7'h74; bbus[7:1] = 7'h1f; end
          2'b11 : begin abus[7:1] = 7'h3C; bbus[7:1] = 7'h44; end
        endcase
        case (i2)
          2'b00 : begin sub = 1'b0; andi = 1'b0; ori = 1'b0; end
          2'b01 : begin sub = 1'b1; andi = 1'b0; ori = 1'b0; end
          2'b10 : begin sub = 1'b0; andi = 1'b1; ori = 1'b0; end
          2'b11 : begin sub = 1'b0; andi = 1'b0; ori = 1'b1; end
        endcase
        case (i3)
          1'b0   : zerob = 1'b0;
          1'b1   : zerob = 1'b1;
        endcase
      end
    end
  end
#100  $finish;
end

// 20MHz clock generator
always  #25   clk2 = ~clk2;

endmodule
```

```
//~~~~~~~~~~~~~~~~~~
// 7 bit ALU  ALU7
//  alu7.v
//~~~~~~~~~~~~~~~~~~
module  alu7(clk2, sub, andi, ori, zerob, abus, bbus, sbus);

// I/O definition
input   clk2;       // Clock Phi2   (H/L)
input   sub;        // Subtraction  (H)
input   andi;       // AND          (H)
input   ori;        // OR           (H)
input   zerob;      // 0 --> B      (H)
input   [7:1]abus;  // A            (H)
input   [7:1]bbus;  // B            (H)

output  [7:1]sbus;  // Sum          (H)

// Regs for random logic
wire  clk2, sub, andi, ori, zerob;
wire  [7:1]abus;
wire  [7:1]bbus;

wire  [7:1]sbus;
wire  [7:1]caon;
wire  cain_0;

//---------------
// Random logic
//---------------
alu  alu_7(.sub(sub), .andi(andi), .ori(ori), .zerob(zerob), .cainn(caon[6]), .a(abus[7]),
.b(bbus[7]), .caon(caon[7]), .s(sbus[7]));
alu  alu_6(.sub(sub), .andi(andi), .ori(ori), .zerob(zerob), .cainn(caon[5]), .a(abus[6]),
.b(bbus[6]), .caon(caon[6]), .s(sbus[6]));
alu  alu_5(.sub(sub), .andi(andi), .ori(ori), .zerob(zerob), .cainn(caon[4]), .a(abus[5]),
.b(bbus[5]), .caon(caon[5]), .s(sbus[5]));
alu  alu_4(.sub(sub), .andi(andi), .ori(ori), .zerob(zerob), .cainn(caon[3]), .a(abus[4]),
.b(bbus[4]), .caon(caon[4]), .s(sbus[4]));
alu  alu_3(.sub(sub), .andi(andi), .ori(ori), .zerob(zerob), .cainn(caon[2]), .a(abus[3]),
.b(bbus[3]), .caon(caon[3]), .s(sbus[3]));
alu  alu_2(.sub(sub), .andi(andi), .ori(ori), .zerob(zerob), .cainn(caon[1]), .a(abus[2]),
.b(bbus[2]), .caon(caon[2]), .s(sbus[2]));
alu  alu_1(.sub(sub), .andi(andi), .ori(ori), .zerob(zerob), .cainn(cain_0),  .a(abus[1]),
.b(bbus[1]), .caon(caon[1]), .s(sbus[1]));

assign  cain_0 = 1'b1;

endmodule

//~~~~~~~~~~~~~~~~~~~~
// 1 bit ALU module
//~~~~~~~~~~~~~~~~~~~~
module  alu(sub, andi, ori, zerob, cainn, a, b, caon, s);

// I/O definition
input   sub;    // Subtraction  (H)
input   andi;   // AND          (H)
input   ori;    // OR           (H)
input   zerob;  // 0 --> B      (H)
input   cainn;  // Carry in     (L)
input   a;      // A            (H)
input   b;      // B            (H)
```

```
output  caon;   // Carry out    (L)
output  s;      // Sum          (H)

// Regs for random logic
wire  sub, andi, ori, zerob, cainn, a, b;
reg   caon, s;
reg   g1, g2, g3, g4, g5, g6, bb;

//---------------
// Random logic
//---------------
always @*
begin
  g1 = (sub) ? (~a) : (a);
  bb = ~(zerob | ~b);
  g5 = ~(cainn | ori);
  g2 = ~(g1 | bb);
  g3 = ~((g1 & bb) | g2);
  g4 = ~(g2 | g3);
  g6 = ~(((a | bb) & g5) | (a & bb) | andi);
  s  = (((a | bb| g5) & (ori | g6)) | ((andi | g5) & a & bb));
end

always @*
begin
  casex ({g2, g3, g4})
    3'b1xx : caon = 1'b1;
    3'bx1x : caon = cainn;
    3'bxx1 : caon = 1'b0;
  endcase
end

endmodule
```

Physical Structure of Program ROM

Memory Cells ()																	A[11:7]	Out
								---									1dh --- 00h	①
								---									1dh --- 00h	②
								---									1dh --- 00h	③
								---									1dh --- 00h	④
								---									1dh --- 00h	⑤
								---									1dh --- 00h	⑥
								---									1dh --- 00h	⑦
								---									1dh --- 00h	⑧
								---									1dh --- 00h	⑨
								---									1dh --- 00h	⑩
								---									1dh --- 00h	⑪
								---									1dh --- 00h	⑫
3Fh	3Eh	3Dh	3Ch	3Bh	3Ah	39h	38h	---	07h	06h	05h	04h	03h	02h	01h	00h		
A[6:1]																		

Physically, 128 x 15 x 12 = 23,040 bits

Logically, 127 x 15 x 12 = 22,860 bits (A[7:1] = 7Fh is not present. Refer to “7 bit polynomial address counter”)

Physical Structure of Pattern ROM

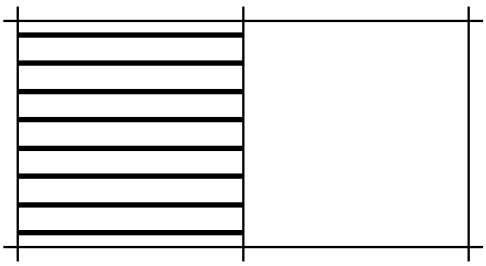
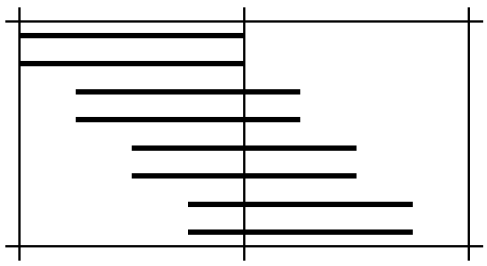
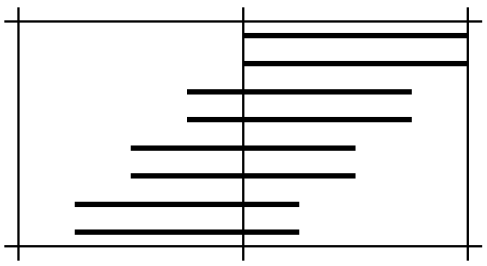
A3[3:1]	y'[3:1]																			
110	00x																	---		
	010																	---		
	011																	---		
	100																	---		
	101																	---		
	110																	---		
	111																	---		
101	111																	---		
	110																	---		
	101																	---		
	100																	---		
	011																	---		
	010																	---		
	00x																	---		
100	00x																	---		
	010																	---		
	011																	---		
	100																	---		
	101																	---		
	110																	---		
	111																	---		
011 - 010	---																	---		
001	111																	---		
	110																	---		
	101																	---		
	100																	---		
	011																	---		
	010																	---		
	00x																	---		
000	00x																	---		
	010																	---		
	011																	---		
	100																	---		
	101																	---		
	110																	---		
	111																	---		
		O[1]																**O[2] - O[7]**	**O[8]**	
		A3[7:4]																----	**A3[7:4]**	
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	----	1110	1111

Usage of Pattern ROM

A3[7:1] (PTN[7:1])	**Patterns**	**Types**	**Matrix** (X x Y)
00h ~ 06h	42 (7 x 6)	Normal	7 x 7
08h ~ 0Eh			
10h ~ 16h			
18h ~ 1Eh			
20h ~ 26h			
28h ~ 2Eh			
30h ~ 36h	49 (7 x 7)	(Bent)	
38h ~ 3Eh			
40h ~ 46h			
48h ~ 4Eh			
50h ~ 56h			
58h ~ 5Eh			
60h ~ 66h			
68h ~ 6Eh	7	Y Repeat	
70h ~ 76h	7	XY Repeat	8 x 7
78h ~ 7Eh	7	X Repeat	
Grand Total	112 (7 x 16)		

"A3[7:1] = x7h & xFh" are not implemented.

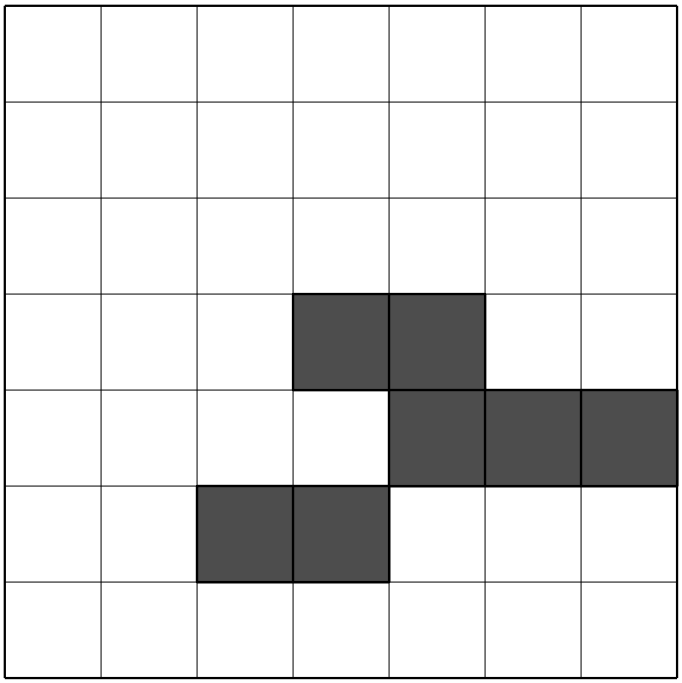
Normal & Bent Patterns

Normal Patterns	**Bent Patterns**	
	PTN[1] = 0	PTN[1] = 1
		

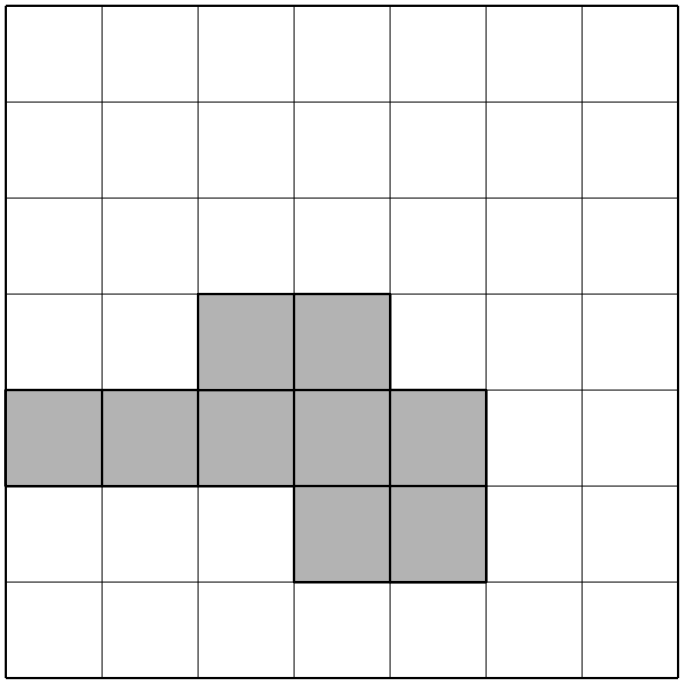
Bent pattern provides more attractive slanted figures by shifting display start timing in each scan line. It results in four times higher horizontal resolution in one X coordinate.

See the actual example at next page.

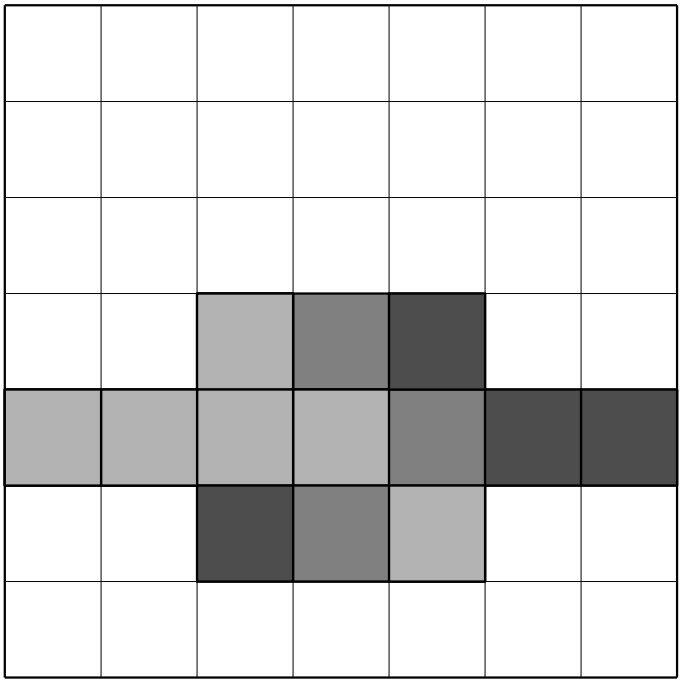
Example of Bent Pattern



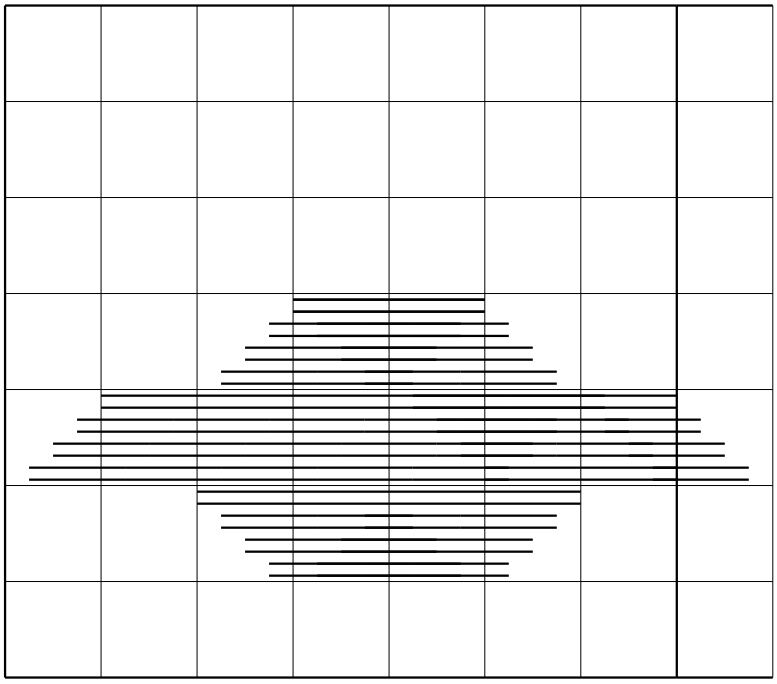
PTN[1] = 0 (For example, PTN = 60h)



PTN[1] = 1 (For example, PTN = 61h)

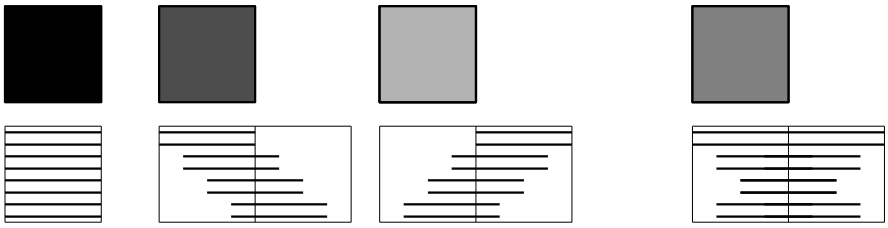


Overwrapped PTN = 60h & 61h



Overwrapped PTN = 60h & 61h (Bent display result)

UFO



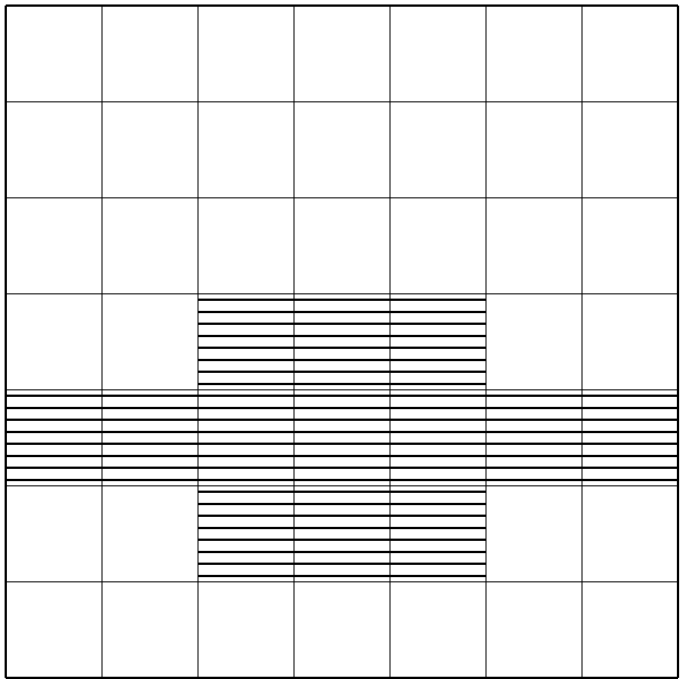
PTN[1] = 0 PTN[1] = 1 Overwrapped PTN[1] =0 & 1

Normal Bent pattern

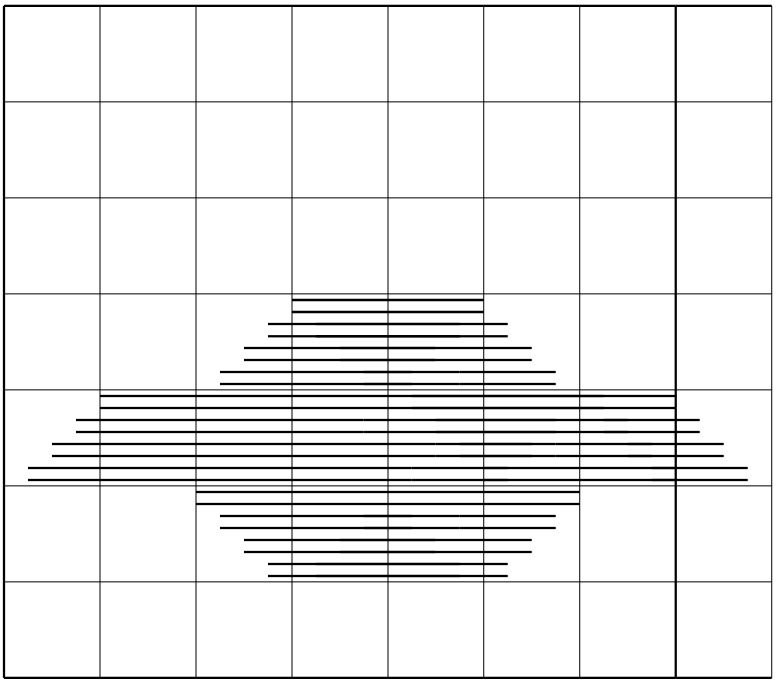
X & Y coordinate are the same in this example.

Bent Display Effect

UFO



Overwrapped PTN = 60h & 61h (Normal; No bent)



Overwrapped PTN = 60h & 61h (Bent display result)

Data RAM & Register Files

32 x 4 x 7 (896 bits)

RAM Address				Registers		
H[5:1]			L[2:1]			
00h - 1Fh	MA[7]	M0[7]	00h	A1[7]		
		M1[7]	01h	A2[7]		
		M2[7]	02h	A3[7]	X3[7]	
		M3[7]	03h	A4[7]	X4[7]	
	MA[6]	M0[6]	00h	A1[6]		
		M1[6]	01h	A2[6]		
		M2[6]	02h	A3[6]	X3[6]	
		M3[6]	03h	A4[6]	X4[6]	
	MA[5]	M0[5]	00h	A1[5]		
		M1[5]	01h	A2[5]		
		M2[5]	02h	A3[5]	X3[5]	
		M3[5]	03h	A4[5]	X4[5]	H[5]
	MA[4]	M0[4]	00h	A1[4]		
		M1[4]	01h	A2[4]		
		M2[4]	02h	A3[4]	X3[4]	
		M3[4]	03h	A4[4]	X4[4]	H[4]
	MA[3]	M0[3]	00h	A1[3]		
		M1[3]	01h	A2[3]		
		M2[3]	02h	A3[3]	X3[3]	
		M3[3]	03h	A4[3]	X4[3]	H[3]
	MA[2]	M0[2]	00h	A1[2]		
		M1[2]	01h	A2[2]		
		M2[2]	02h	A3[2]	X3[2]	
		M3[2]	03h	A4[2]	X4[2]	H[2]
	MA[1]	M0[1]	00h	A1[1]		
				A1'[1]	X1'[1]	
		M1[1]	01h	A2[1]		
		M2[1]	02h	A3[1]	X3[1]	
		M3[1]	03h	A4[1]	X4[1]	H[1]

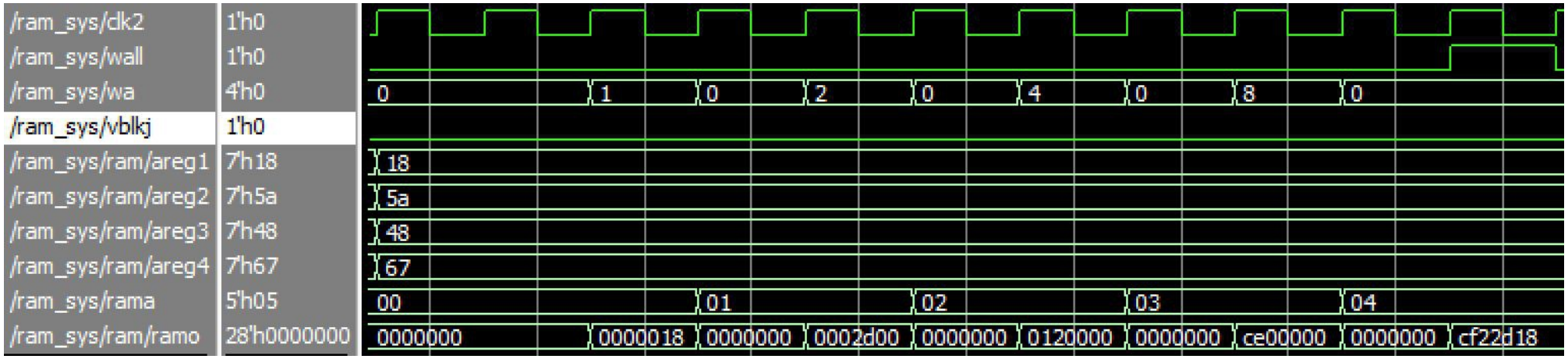
(A) Usage of Data RAM

L[2:1]	Registers	Function	
		H[5:1] = 00h to 18h	H[5:1] = 19h - 1Fh
00h	A1[7:1]	Y[6:1], PRIO	General purpose flags and counters (See next page)
01h	A2[7:1]	X[7:1]	
02h	A3[7:1]	PTN[7:1]	
03h	A4[7:1]	y[3:1], R, G, B, ySUB	

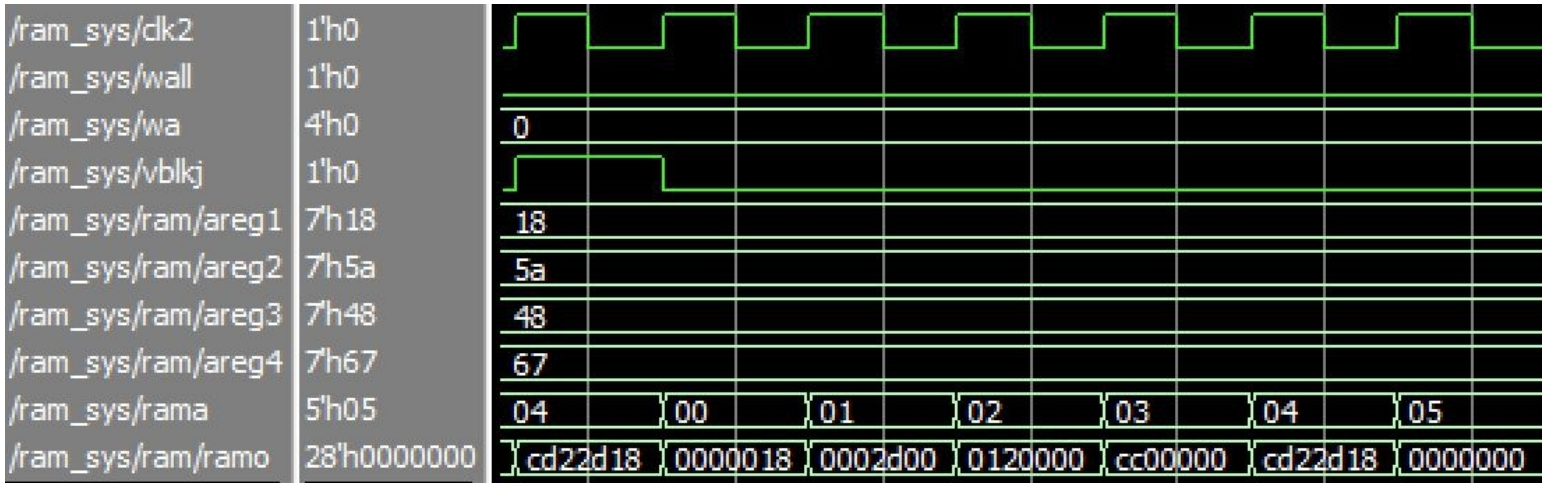
(B) Address Assignment of General Purpose Flags and Counters on Data RAM

L[2:1]	H[5:1]	Coding K (K→L, H)	Flags & Counters						
			[7]	[6]	[5]	[4]	[3]	[2]	[1]
0h	19h	19h							
1h		39h							
2h		59h							
3h		79h							
0h	1Ah	1Ah							
1h		3Ah							
2h		5Ah							
3h		7Ah							
0h	1Bh	1Bh							
1h		3Bh							
2h		5Bh							
3h		7Bh							
0h	1Ch	1Ch							
1h		3Ch							
2h		5Ch							
3h		7Ch							
0h	1Dh	1Dh							
1h		3Dh							
2h		5Dh							
3h		7Dh							
0h	1Eh	1Eh							
1h		3Eh							
2h		5Eh							
3h		7Eh							
0h	1Fh	1Fh							
1h		3Fh							
2h		5Fh							
3h		7Fh							

(C) Verilog HDL Simulation Result



- Zero clear to RAM (not displayed)
- Initialize A1, A2, A3, and A4 (already done)
- 7 bit write to upper (67h), upper middle (48h), lower middle (5ah), and lower (18) RAM bit location updating address 0 through 3
- 28 bit simultaneous write (67h, 48h, 5ah, 18h; cf22d18h) to RAM address 4
- Read back all the data written (RAM address 0 through 4)



- "vblkj" clears "ysub" bits from address H[5:1] = 00h to 18h, L[2:1] = 03h in one clock period.

Compare two simulation result above. "ce0000" at address 3 becomes "cc00000" and "cf22d18" at address 4 becomes "cd22d18".

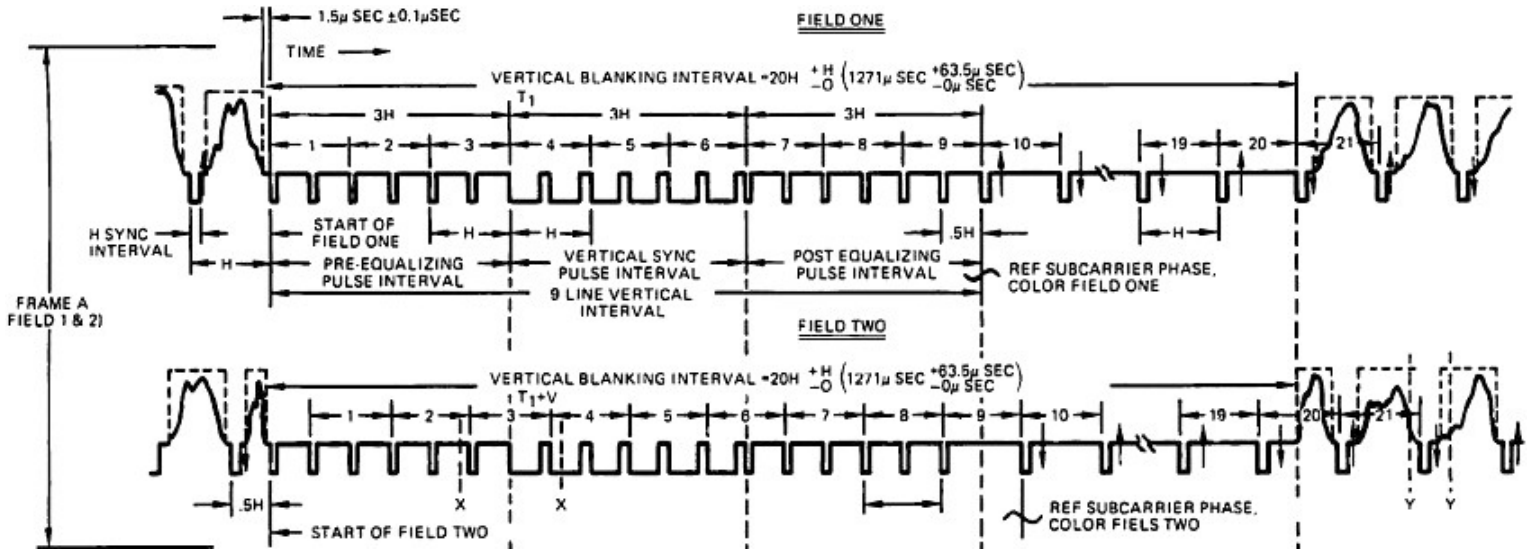
(C-2) Verilog HDL Simulation Files

```
//~~~~~~~~~~~~~~~~~~~~
// Testbench for RAM
//  ram_sys.v
//~~~~~~~~~~~~~~~~~~~~
`timescale  1ns / 1ns

module   ram_sys;

// regs/wires/integers
reg  clk2, load, wall, vblkj;
reg  [6:0]inita1;
reg  [6:0]inita2;
reg  [6:0]inita3;
reg  [6:0]inita4;
reg  [4:1]wa;
reg  [4:0]rama;

integer  i;

// Simulation target
ram  ram(clk2, load, inita1, inita2, inita3, inita4, wa, wall, vblkj, rama);

//--------------------
// Simulation vector
//--------------------
initial
begin
      clk2        = 1'b0;
      load        = 1'b0;
      inita1[6:0] = 7'h0;
      inita2[6:0] = 7'h0;
      inita3[6:0] = 7'h0;
      inita4[6:0] = 7'h0;
      rama[4:0]   = 5'h00;
      wa[4:1]     = 4'h0;
      wall        = 1'b0;
      vblkj       = 1'b0;
#25   load        = 1'b1;
#50   load        = 1'b0;

// 0 to RAM
for (i = 0; i < 33; i = i + 1)
begin
      rama[4:0]   = rama[4:0] + 1;
#50   wall        = 1'b1;
#50   wall        = 1'b0;
end

// 7 bit write x 4, 28 bit write
      rama[4:0]   = 5'h00;
      inita1[6:0] = 7'h18;
      inita2[6:0] = 7'h5A;
      inita3[6:0] = 7'h48;
      inita4[6:0] = 7'h67;
#50   load        = 1'b1;
#50   load        = 1'b0;

for (i = 1; i < 5; i = i + 1)
begin
#50   wa[i]       = 1'h1;
```

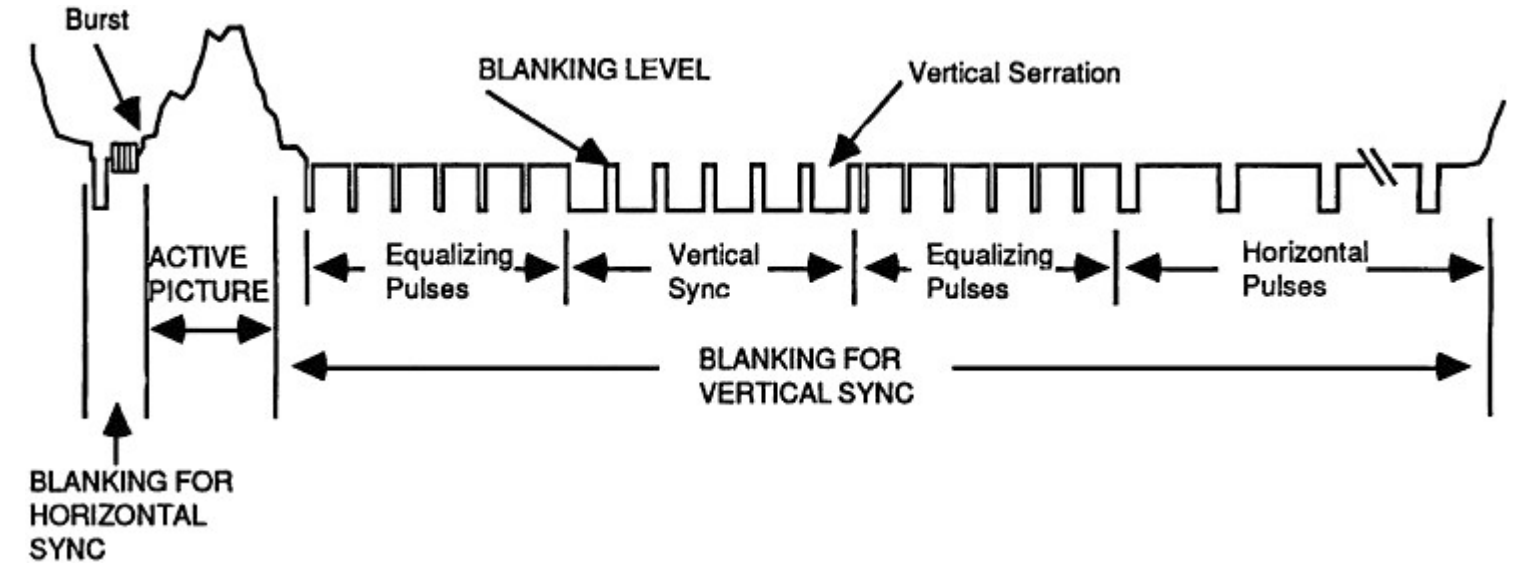
```
#50   wa[i]       = 1'h0;
      rama[4:0]   = rama[4:0] + 1;
end

#50   wall        = 1'b1;
#50   wall        = 1'b0;

#50   vblkj       = 1'b1;
#50   vblkj       = 1'b0;

      rama[4:0]   = 5'h00;

// RAM read
for (i = 0; i < 5; i = i + 1)
begin
#50    rama[4:0]   = rama[4:0] + 1;
end

#100  $finish;
end

// 20MHz clock generator
always  #25   clk2 = ~clk2;

endmodule


//~~~~~~~~~~
// RAM
//  ram.v
//~~~~~~~~~~
module  ram(clk2, load, inita1, inita2, inita3, inita4, wa, wall, vblkj, rama);

// I/O definition
input   clk2;          // Phi2 clock                          (H/L)
input   load;          // Load initial value                  (H)
input   [6:0] inita1;  // Initial data for A1                 (H/L)
input   [6:0] inita2;  // Initial data for A2                 (H/L)
input   [6:0] inita3;  // Initial data for A3                 (H/L)
input   [6:0] inita4;  // Initial data for A4                 (H/L)
input   [4:1] wa;      // Write strobe for each A4, A3, A2, A1  (H)
input   wall;          // Write strobe for all A4, A3, A2, A1   (H)
input   vblkj;         // Clear all ysubs to zero             (H)
input   [4:0] rama;    // RAM address                         (H/L)

// Regs for random logic
reg     [6:0]   areg1;  // A1
reg     [6:0]   areg2;  // A2
reg     [6:0]   areg3;  // A3
reg     [6:0]   areg4;  // A4
reg     [27:0]  ramd;   // RAM data input
wire    [27:0]  ramo;   // RAM data output

//---------------
// DFF
//---------------
always @ (posedge clk2)
begin
  if (load)
  begin
    areg1[6:0]  <= inita1[6:0];
    areg2[6:0]  <= inita2[6:0];
```

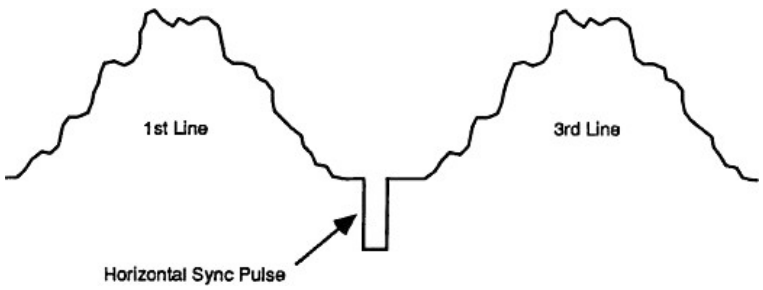
```
    areg3[6:0]  <= inita3[6:0];
    areg4[6:0]  <= inita4[6:0];
  end
end

always @(*)
begin
  ramd[27:0]  = {areg4[6:0], areg3[6:0], areg2[6:0], areg1[6:0]};
end

//---------------
// RAM
//---------------
ram_mod  ram_1 (.clk2(clk2), .vblkj(vblkj), .wall(wall), .wa(wa), .rama(rama),
                .ramd(ramd), .ramo(ramo));

endmodule
```

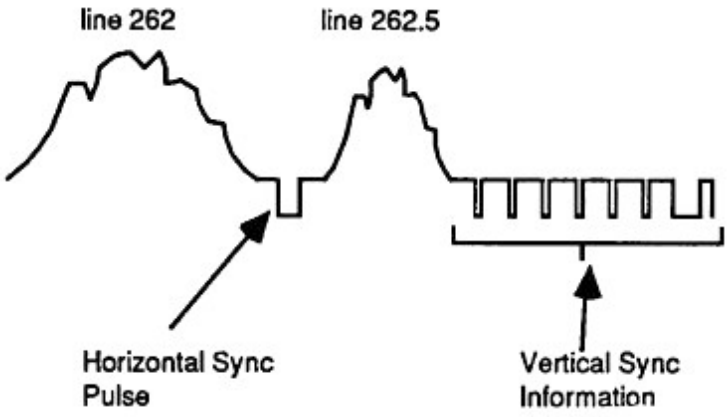
NTSC (National Television System Committee) Television



Interace, Equalizing Pulse Interval, Vertical SYNC, Vertical Blank, Horizontal SYNC







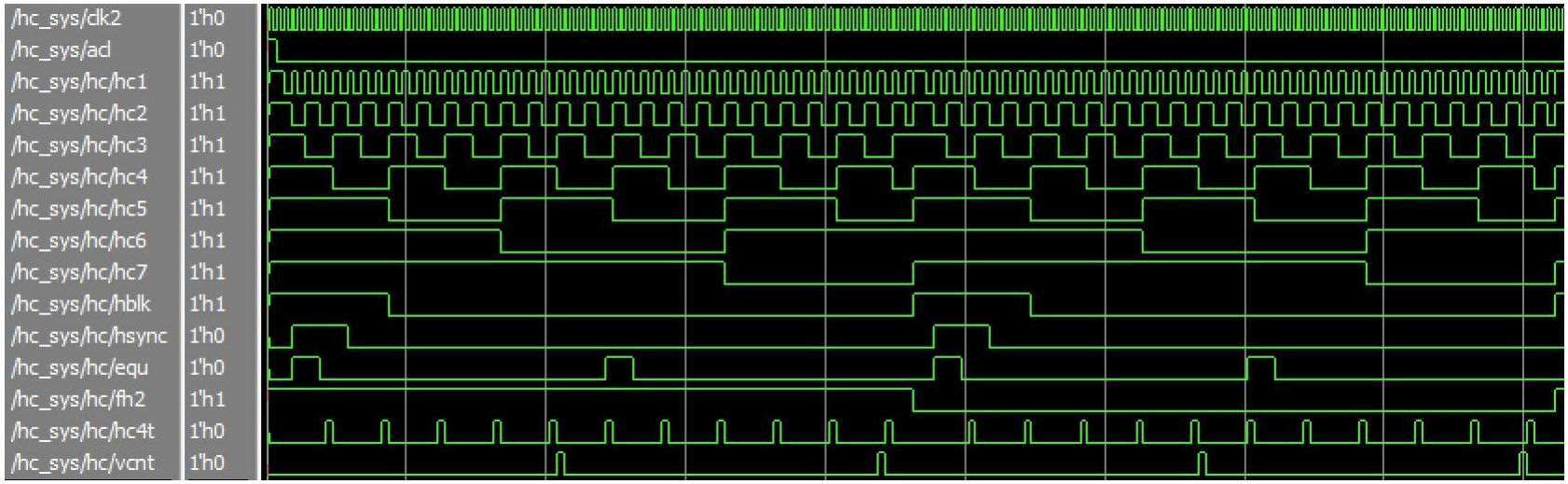
See page 11 & 12 as well.

Horizontal Counter (HC) Sequence

(A) Truth Table

<table>
<tr><th rowspan="2">HC[7:1]</th><th rowspan="2">X</th><th colspan="4">Timing Signals</th></tr>
<tr><th>HC4T</th><th>HBLK</th><th>H SYNC</th><th>EQUAL H/</th></tr>
<tr><td>0000000</td><td>0</td><td rowspan="6">0</td><td rowspan="7">1</td><td rowspan="2">0</td><td rowspan="2">0</td></tr>
<tr><td>0000001</td><td>1</td></tr>
<tr><td>0000010</td><td>2</td><td rowspan="6">1</td><td rowspan="3">1</td></tr>
<tr><td>0000011</td><td>3</td></tr>
<tr><td>0000100</td><td>4</td></tr>
<tr><td>0000101</td><td>5</td><td rowspan="6">1</td></tr>
<tr><td>0000110</td><td>6</td><td rowspan="15">0</td></tr>
<tr><td>0000111</td><td>7</td><td>1</td></tr>
<tr><td>0001000</td><td>8</td><td rowspan="4">0</td><td rowspan="14">0</td></tr>
<tr><td>0001001</td><td>9</td></tr>
<tr><td>--</td><td>--</td></tr>
<tr><td>0001110</td><td>14</td></tr>
<tr><td>0001111</td><td>15</td><td>1</td><td rowspan="13">0</td></tr>
<tr><td>0010000</td><td>16</td><td rowspan="2">0</td></tr>
<tr><td>0010001</td><td>17</td></tr>
<tr><td>--</td><td>--</td><td rowspan="19">1
every
8
clocks</td></tr>
<tr><td>0100111</td><td>39</td></tr>
<tr><td>0101000</td><td>40</td></tr>
<tr><td>--</td><td>--</td></tr>
<tr><td>0101110</td><td>46</td><td rowspan="12">0</td></tr>
<tr><td>0101111</td><td>47</td><td rowspan="4">1</td></tr>
<tr><td>--</td><td>--</td></tr>
<tr><td>0110010</td><td>50</td></tr>
<tr><td>0110011</td><td>51</td><td rowspan="8">0</td></tr>
<tr><td>--</td><td>--</td></tr>
<tr><td>1010101</td><td>85</td></tr>
<tr><td>1010110</td><td>86</td></tr>
<tr><td>--</td><td>--</td></tr>
<tr><td>1011010</td><td>90</td></tr>
<tr><td>0000000</td><td>0</td><td>1</td><td></td><td></td></tr>
<tr><td>--</td><td></td><td>--</td><td>--</td><td>--</td></tr>
<tr><td>1011010</td><td>90</td><td>0</td><td rowspan="2">0</td><td rowspan="2">0</td></tr>
<tr><td>0000000</td><td>0</td><td>1</td></tr>
</table>

(B) Verilog HDL Simulation Result



- Two horizontal periods (2H)
- "hc[7:1]" are negative outputs (high level = 0, low level = 1) of horizontal counter.
- The timing of "hblk" (horizontal blank), "hsync" (horizontal sync), "equ" (equalization pulse), "fh2" (horizontal frequency / 2), "hc4t" (pulse every 4 clocks), and "vcnt" (count pulse for vertical counter) are exhibited.

The result is same as (A) Truth Table.

(B-2) Verilog HDL Simulation Files

```
//~~~~~~~~~~~~~~~~~~~~
// Testbench for HC
//  hc_sys.v
//~~~~~~~~~~~~~~~~~~~~
`timescale  1ns / 1ns

module   hc_sys;

// regs/wires/integers
reg  clk2, acl;

integer  i;

// Simulation target
hc  hc(clk2, acl);

//--------------------
// Simulation vector
//--------------------
initial
begin
      clk2 = 1'b0;
      acl  = 1'b1;
#25   acl  = 1'b1;
#50   acl  = 1'b0;

#9250  $finish;
end

// 20MHz clock generator
always  #25   clk2 = ~clk2;

endmodule


//~~~~~~~~
// HC
//  hc.v
//~~~~~~~~
module  hc(clk2, acl);

// I/O definition
input   clk2;   // Phi2 clock    (H/L)
input   acl;    // ACL           (H)

// Regs for registers
reg     acl1d;  // ACL 1d        (H)
reg     vcnt ;  // VC count      (H)
reg     hc1;    // HC1n          (L)
reg     hc2;    // HC2n          (L)
reg     hc3;    // HC3n          (L)
reg     hc4;    // HC4n          (L)
reg     hc5;    // HC5n          (L)
reg     hc6;    // HC6n          (L)
reg     hc7;    // HC7n          (L)
reg     fh2;    // fH/2          (H)
reg     hblk;   // HBLANK        (H)
reg     hsync;  // HSYNC         (H)
reg     equ;    // EQUALIZATION  (H)
```

```
// Regs for random logic
reg     hc3t;
reg     hc4t;
reg     hc5t;
reg     hc6t;
reg     hc7t;
reg     fh2t;
reg     hblkt;
reg     synct1;
reg     synct2;
reg     synct;
reg     equt1;
reg     equt2;
reg     equt3;
reg     equt;
reg     vcntt1;
reg     vcntt2;
reg     vcntt;
reg     aclequ;

//------------
// Registers
//------------
always @ (posedge(clk2))
begin
  acl1d <= (acl | fh2t);
end

always @ (posedge(clk2) or posedge(aclequ))
begin
  if (aclequ)  vcnt <= 0;
  if (vcntt)   vcnt <= ~vcnt;
end

always @ (posedge(clk2) or posedge(acl1d) or posedge(acl) )
begin
  if (acl)
  begin
    fh2   <= 1;
  end
  if (acl1d)
  begin
    hc1   <= 1;
    hc2   <= 1;
    hc3   <= 1;
    hc4   <= 1;
    hc5   <= 1;
    hc6   <= 1;
    hc7   <= 1;
    hblk  <= 1;
    hsync <= 0;
    equ   <= 0;
  end
  else
  begin
    hc1 <= ~hc1;
    if (~hc1)   hc2   <= ~hc2;
    if (hc3t)   hc3   <= ~hc3;
    if (hc4t)   hc4   <= ~hc4;
    if (hc5t)   hc5   <= ~hc5;
    if (hc6t)   hc6   <= ~hc6;
    if (hc7t)   hc7   <= ~hc7;
```

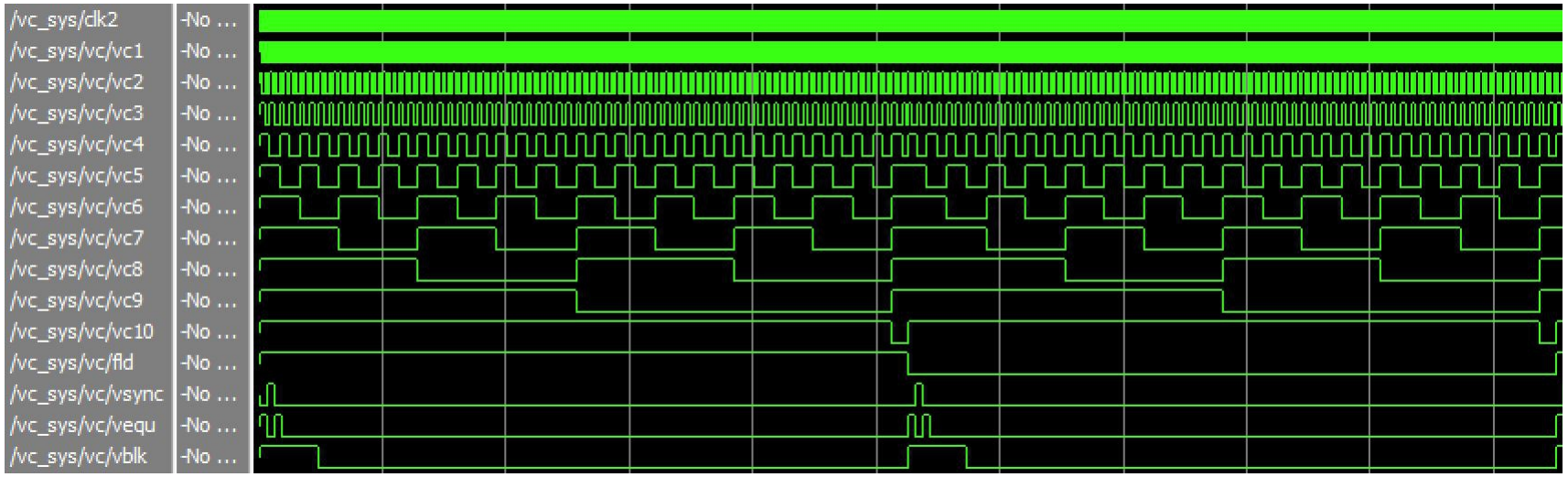
```
    if (fh2t)   fh2   <= ~fh2;
    if (hblkt)  hblk  <= ~hblk;
    if (synct)  hsync <= ~hsync;
    if (equt)   equ   <= ~equ;
  end
end

//---------------
// Random logic
//---------------
always @(*)
begin
  hc3t    = ~(hc1 | hc2);
  hc4t    = ~(hc1 | hc2 | hc3);
  hc5t    = ~(hc1 | hc2 | hc3 | hc4);
  hc6t    = ~(hc1 | hc2 | hc3 | hc4 | hc5);
  hc7t    = ~(hc1 | hc2 | hc3 | hc4 | hc5 | hc6);
  fh2t    = ~(~hc1 | hc2 | ~hc3 | hc4 | hc5 | ~hc6 | hc7);
  hblkt   = ~(hc1 | hc2 | hc3 | hc4 | ~hc5 | ~hc6 | ~hc7);
  synct1  = ~(hc1 | ~hc2 | ~hc3 | ~hc4 | ~hc5 | ~hc6 | ~hc7);
  synct2  = ~(hc1 | ~hc2 | ~hc3 | hc4 | ~hc5 | ~hc6 | ~hc7);
  synct   = (synct1 | synct2);
  equt1   = ~(hc1 | ~hc2 | hc3 | ~hc4 | ~hc5 | ~hc6 | ~hc7);
  equt2   = ~(~hc1 | hc2 | hc3 | hc4 | ~hc5 | hc6 | ~hc7);
  equt3   = ~(~hc1 | hc2 | ~hc3 | ~hc4 | hc5 | hc6 | ~hc7);
  equt    = (synct1 | equt1 | equt2 | equt3);
  vcntt1  = ~(hc1 | hc2 | hc3 | ~hc4 | ~hc5 | hc6 | ~hc7);
  vcntt2  = ~(hc1 | ~hc2 | hc3 | ~hc4 | hc5 | ~hc6 | hc7);
  vcntt   = (vcntt1 | vcntt2);
  aclequ  = (acl | ~equ);
end

endmodule
```

Vertical Counter (VC) Sequence

(A) Truth table

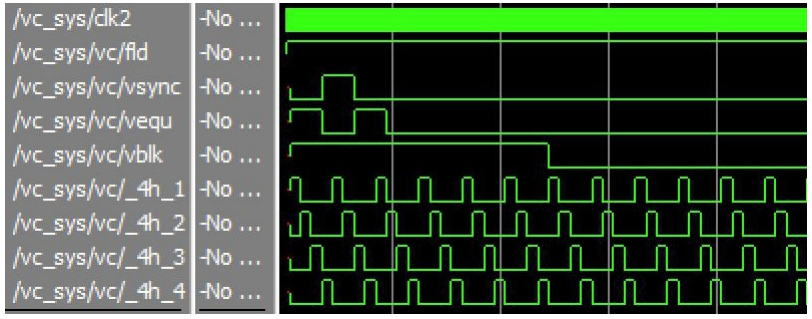
VC[10:1]	Dec	Timing Signals			
		Field	V Blank	EQUAL V	V SYNC
0000000000	0	0	1	1	0
--					
0000000101	5				
0000000110	6				1
--					
0000001011	11				
0000001100	12				0
--					
0000010001	17				
0000010010	18			0	
--					
0000101111	47				
0000110000	48		0		
--					
1000001100	524				
0000000001	1	1	1	1	0
--					
0000000110	6				
0000000111	7				1
--					
0000001100	12				
0000001101	13				0
--					
0000010010	18			0	
0000010011	19				
--					
0000110000	48				
0000110001	49		0		
--					
1000001101	525				
0000000000	0	0	1	1	0
--					

(B) Verilog HDL Simulation Result



- Two vertical periods (2V)
- "vc[10:1]" are negative outputs (high level = 0, low level = 1) of vertical counter.
- The timing of "vblk" (vertical blank), "vsync" (vertical sync), "fld" (odd/even field), "vequ" (vertical equalization), "vc[10:1]" (vertical counter), and "vcntn" (count pulse for vertical counter) are exhibited.

The result is same as (A) Truth Table.



"_4h_1" is used for 4HBLK judge. "_4h_1", "_4h_2", "_4h_3", and "_4h_4" are used as bent pattern shift positioning signals combining with ϕ1, ϕ2, ϕ3, and ϕ4.

(B-2) Verilog HDL Simulation Files

```
//~~~~~~~~~~~~~~~~~~~~
// Testbench for VC
//  vc_sys.v
//~~~~~~~~~~~~~~~~~~~~
`timescale  1ns / 1ns

module   vc_sys;

// regs/wires/integers
reg  clk2, acl, vcntn;

integer  i;

// Simulation target
vc  vc(clk2, acl, vcntn);

//--------------------
// Simulation vector
//--------------------
initial
begin
      clk2  = 1'b0;
      acl   = 1'b1;
      vcntn = 1'b1;
#25   acl   = 1'b1;
#50   acl   = 1'b0;

for (i = 0; i < 2160; i = i + 1)
begin
#50   vcntn = ~vcntn;
end

#100  $finish;
end

// 20MHz clock generator
always  #25   clk2 = ~clk2;

endmodule


//~~~~~~~~
// VC
//  vc.v
//~~~~~~~~
module  vc(clk2, acl, vcntn, fld, vsync, vequ, vblk, _4h_1, _4h_2, _4h_3, _4h_4);

// I/O definition
input   clk2;   // Phi2 clock      (H/L)
input   acl;    // Auto CLear      (H)
input   vcntn;  // VC count        (L)

output  fld;    // Odd/Even Field  (H)
output  vsync;  // VSYNC           (H)
output  vequ;   // VEQUALIZATION   (H)
output  vblk;   // VBLANK          (H)
output  _4h_1;  // 4h 1st line     (H)
output  _4h_2;  // 4h 2nd line     (H)
output  _4h_3;  // 4h 3rd line     (H)
output  _4h_4;  // 4h 4th line     (H)
```

```
// Regs for registers
reg     vccl;   // VC clear        (H)
reg     vc1;    // VC1n            (L)
reg     vc2;    // VC2n            (L)
reg     vc3;    // VC3n            (L)
reg     vc4;    // VC4n            (L)
reg     vc5;    // VC5n            (L)
reg     vc6;    // VC6n            (L)
reg     vc7;    // VC7n            (L)
reg     vc8;    // VC8n            (L)
reg     vc9;    // VC9n            (L)
reg     vc10;   // VC10n           (L)
reg     fld;    // Field           (H/L)
reg     vsync;  // VSYNC           (H)
reg     vequ;   // VEQUALIZATION   (H)
reg     vblk;   // VBLANK          (H)

// Regs for random logic
reg     vc2t;     // vc2  toggle   (H)
reg     vc3t;     // vc3  toggle   (H)
reg     vc4t;     // vc4  toggle   (H)
reg     vc5t;     // vc5  toggle   (H)
reg     vc6t;     // vc6  toggle   (H)
reg     vc7t;     // vc7  toggle   (H)
reg     vc8t;     // vc8  toggle   (H)
reg     vc9t;     // vc9  toggle   (H)
reg     vc10t;    // vc10 toggle   (H)
reg     fldt1;    //
reg     fldt2;    //
reg     fldt;     // fld toggle    (H)
reg     vsynct1;  //
reg     vsynct2;  //
reg     vsynct3;  //
reg     vsynct4;  //
reg     vsynct;   // vsync toggle  (H)
reg     vequt1;   //
reg     vequt2;   //
reg     vequt;    // vequ toggle   (H)
reg     vblkt1;   //
reg     vblkt2;   //
reg     vblkt;    // vblk toggle   (H)
reg     _4h_1;    // 4h 1st line   (H)
reg     _4h_2;    // 4h 2nd line   (H)
reg     _4h_3;    // 4h 3rd line   (H)
reg     _4h_4;    // 4h 4th line   (H)

//------------
// Registers
//------------
always @ (posedge(clk2))
begin
  vccl  <=  (fldt | acl);
end

always @ (posedge(clk2) or posedge(acl) or posedge(vccl))
begin
  if (acl)
  begin
    fld   <= 1;
    vc1   <= 1;
  end
```

```
  else
    if (vccl)
    begin
      vc2    <= 1;
      vc3    <= 1;
      vc4    <= 1;
      vc5    <= 1;
      vc6    <= 1;
      vc7    <= 1;
      vc8    <= 1;
      vc9    <= 1;
      vc10   <= 1;
      vsync  <= 0;
      vequ   <= 1;
      vblk   <= 1;
    end
  else
  begin
    if (fldt)    fld    <= ~fld;
    if (vcntn)   vc1    <= ~vc1;
    if (vc2t)    vc2    <= ~vc2;
    if (vc3t)    vc3    <= ~vc3;
    if (vc4t)    vc4    <= ~vc4;
    if (vc5t)    vc5    <= ~vc5;
    if (vc6t)    vc6    <= ~vc6;
    if (vc7t)    vc7    <= ~vc7;
    if (vc8t)    vc8    <= ~vc8;
    if (vc9t)    vc9    <= ~vc9;
    if (vc10t)   vc10   <= ~vc10;
    if (vsynct)  vsync  <= ~vsync;
    if (vequt)   vequ   <= ~vequ;
    if (vblkt)   vblk   <= ~vblk;
  end
end

//---------------
// Random logic
//---------------
always @(*)
begin
  vc2t    = ~(vcntn | vc1);
  vc3t    = ~(vcntn | vc1 | vc2);
  vc4t    = ~(vcntn | vc1 | vc2 | vc3);
  vc5t    = ~(vcntn | vc1 | vc2 | vc3 | vc4);
  vc6t    = ~(vcntn | vc1 | vc2 | vc3 | vc4 | vc5);
  vc7t    = ~(vcntn | vc1 | vc2 | vc3 | vc4 | vc5 | vc6);
  vc8t    = ~(vcntn | vc1 | vc2 | vc3 | vc4 | vc5 | vc6 | vc7);
  vc9t    = ~(vcntn | vc1 | vc2 | vc3 | vc4 | vc5 | vc6 | vc7 | vc8);
  vc10t   = ~(vcntn | vc1 | vc2 | vc3 | vc4 | vc5 | vc6 | vc7 | vc8 | vc9);
  fldt1   = ~(vcntn | ~vc1 | ~vc2 | vc3 | vc4 | ~vc5 | ~vc6 | ~vc7 | ~vc8 | ~vc9 | vc10);
  fldt2   = ~(vcntn | vc1 | ~vc2 | vc3 | vc4 | ~vc5 | ~vc6 | ~vc7 | ~vc8 | ~vc9 | vc10);
  fldt    = (fld) ? (fldt1) : (fldt2);
  vsynct1 = ~(~fld | vcntn | vc1 | ~vc2 | vc3 | ~vc4 | ~vc5 | ~vc6 | ~vc7 | ~vc8 | ~vc9 | ~vc10);
  vsynct2 = ~(~fld | vcntn | vc1 | vc2 | ~vc3 | vc4 | ~vc5 | ~vc6 | ~vc7 | ~vc8 | ~vc9 | ~vc10);
  vsynct3 = ~(fld | vcntn | ~vc1 | ~vc2 | vc3 | vc4 | ~vc5 | ~vc6 | ~vc7 | ~vc8 | ~vc9 | ~vc10);
  vsynct4 = ~(fld | vcntn | ~vc1 | vc2 | vc3 | ~vc4 | ~vc5 | ~vc6 | ~vc7 | ~vc8 | ~vc9 | ~vc10);
  vsynct  = (vsynct1 | vsynct2 | vsynct3 | vsynct4);
  vequt1  = ~(~fld | vcntn | vc1 | ~vc2 | ~vc3 | ~vc4 | vc5 | ~vc6 | ~vc7 | ~vc8 | ~vc9 | ~vc10);
  vequt2  = ~(fld | vcntn | ~vc1 | vc2 | ~vc3 | ~vc4 | vc5 | ~vc6 | ~vc7 | ~vc8 | ~vc9 | ~vc10);
  vequt   = (vsynct | vequt1 | vequt2);
  vblkt1  = ~(vcntn | vc1 | vc2 | vc3 | vc4 | ~vc5 | vc6 | ~vc7 | ~vc8 | ~vc9 | ~vc10);
  vblkt2  = ~(vcntn | ~vc1 | ~vc2 | ~vc3 | ~vc4 | vc5 | vc6 | ~vc7 | ~vc8 | ~vc9 | ~vc10);
```

```
  vblkt   = (fld) ? (vblkt1) : (vblkt2);
  _4h_1   = ~(~vc2 | ~vc3);
  _4h_2   = ~(vc2 | ~vc3);
  _4h_3   = ~(~vc2 | vc3);
  _4h_4   = ~(vc2 | vc3);
end

endmodule
```

Test Pin Function

<table>
<tr><th>CH2</th><th>CH1</th><th>Function</th></tr>
<tr><td>0</td><td>0</td><td>Normal operation</td></tr>
<tr><td>0</td><td>1</td><td>Program ROM code feed from I/O pins
<table>
<tr><th></th><th colspan="12">Program ROM</th></tr>
<tr><td>I/O Pin</td><td>K1/</td><td>K2/</td><td>K3/</td><td>K4/</td><td>K5/</td><td>K6/</td><td>K7/</td><td>GP</td><td>PD4</td><td>PD3</td><td>PD2</td><td>PD1</td></tr>
<tr><td>Bit</td><td>12</td><td>11</td><td>10</td><td>09</td><td>08</td><td>07</td><td>06</td><td>05</td><td>04</td><td>03</td><td>02</td><td>01</td></tr>
</table></td></tr>
<tr><td>1</td><td>0</td><td>Program ROM code dump to I/O pins
<table>
<tr><th></th><th colspan="12">Program ROM</th></tr>
<tr><td>Bit</td><td>12</td><td>11</td><td>10</td><td>09</td><td>08</td><td>07</td><td>06</td><td>05</td><td>04</td><td>03</td><td>02</td><td>01</td></tr>
<tr><td>I/O Pin</td><td>K1/</td><td>K2/</td><td>K3/</td><td>K4/</td><td>K5/</td><td>K6/</td><td>K7/</td><td>GP</td><td>PD4</td><td>PD3</td><td>PD2</td><td>PD1</td></tr>
</table></td></tr>
<tr><td>1</td><td>1</td><td>Initialization of on-chip prescaler</td></tr>
</table>

CH3	Function
0	Normal operation
1	- Force horizontal blank always off. - Force vertical counter counts up every clock.

<table>
<tr><th>CH6</th><th>CH4</th><th>CH3</th><th>CH2</th><th>CH1</th><th>Function</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>Normal operation</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td rowspan="3">0</td><td rowspan="3">1</td><td>Data RAM dump to I/O pins
<table><tr><th></th><th colspan="7">RAM</th></tr><tr><th>Bit</th><td>I7</td><td>I6</td><td>I5</td><td>I4</td><td>I3</td><td>I2</td><td>I1</td></tr><tr><th>I/O Pin</th><td>S4/</td><td>S3/</td><td>S2/</td><td>S1/</td><td>R</td><td>G</td><td>B</td></tr></table>B = CROMA</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>Pattern ROM dump to I/O pins
<table><tr><th></th><th colspan="3">ROM</th></tr><tr><th>Bit</th><td colspan="3">000-7-6-5-4-3-2-1-0</td></tr><tr><th>I/O Pin</th><td>R</td><td>G</td><td>B</td></tr></table>B = CROMA, 11 bit serial out</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Data RAM dump to I/O pins
<table><tr><th></th><th colspan="7">RAM</th></tr><tr><th>Bit</th><td>I7</td><td>I6</td><td>I5</td><td>BENT/</td><td>I3</td><td>I2</td><td>I1</td></tr><tr><th>I/O Pin</th><td>S4/</td><td>S3/</td><td>S2/</td><td>S1/</td><td>R</td><td>G</td><td>B</td></tr></table>B = CROMA</td></tr>
</table>

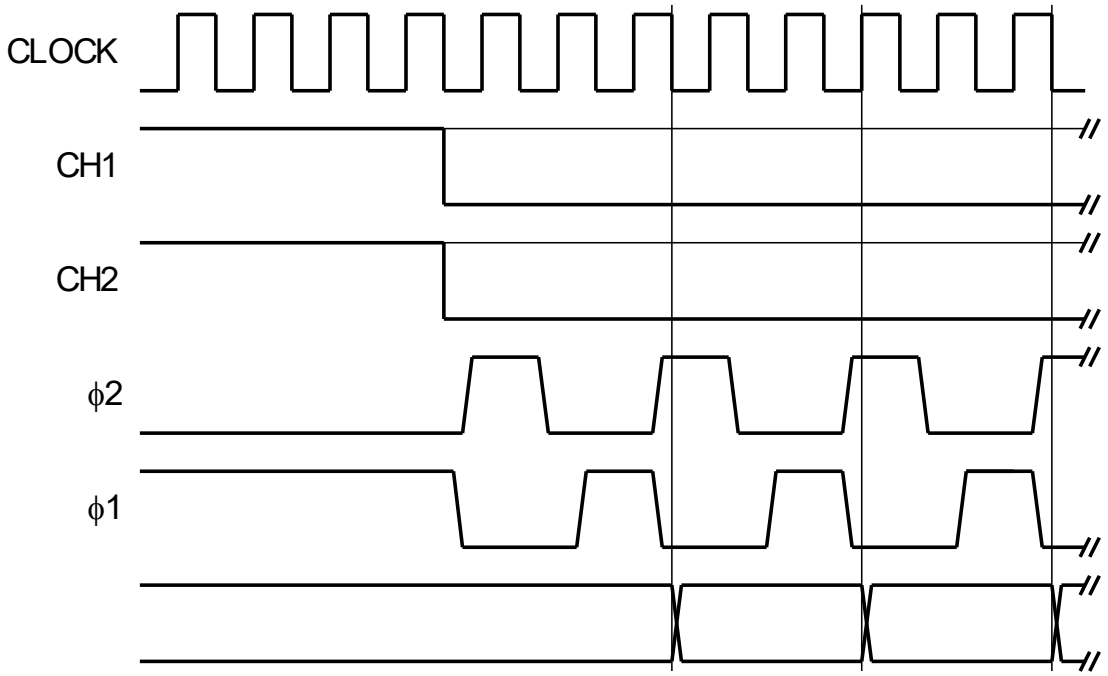
Other combinations of CH[6]+CH[4:1] are not specifically defined.

(A) Data RAM address and Pattern ROM address are specified by feeding program ROM code externally.

(B) Taking advantage of μPD777 instructions by feeding the program ROM code through I/O pins (CH[2:1] = 01) as well as syncing the timing for each purpose, all the dump of program & pattern ROM code and R/W test of data RAM and register files are achieved.

(C) These functions were originally implemented for purely LSI test purpose. Therefore, the function of pattern ROM code feed through I/O pins was not implemented.

On-chip Prescaler



On-chip prescaler generates two on-chip clocks which one clock cycle consists of ϕ1 and ϕ2 every five external CLOCKs.

The condition of CH1=CH2=1 initializes the prescaler (minimum 4 CLOCKs period) and the timing of releasing from CH1=CH2=1 determines the timing relationship between CLOCK and on-chip clock to ease LSI test.

	NTSC (National Television System Committee)	**PAL** (Phase Alternating Line)
Lines / Frame	525 / 60	625 / 50
Horizontal Frequency	15.734 KHz	15.625 KHz
Vertical Frequency	60 Hz	50 Hz
Color Sub-carrier Frequency	3.579545 MHz	4.433618 MHz

In case of connecting to NTSC (National Television System Committee) television, the color sub-carrier which frequency is 3.579545 MHz is provided to the CLOCK input. The prescaler scales down the frequency to 1/2.5 which is 1.431818 MHz.
Maximum X co-ordinate can be calculated by 1431818 Hz divided by horizontal frequency (15734 Hz). It becomes 91.
16 X co-ordinates are occupied by horizontal blank period.

Accordingly, horizontal displayable dots of μPD777 becomes 75.

fSub / fH / 2.5 = 91
91 (1H) – 16 (Horizontal blank) = 75

The total number of scan lines is 525. Vertical blank occupies 48 scan lines in it. Displayable scan lines becomes 477. μPD777 specifies 8 scan lines (4 scan lines for each odd & even field judging "4 Horizontal blank" by firmware) per minimum unit of Y co-ordinate.

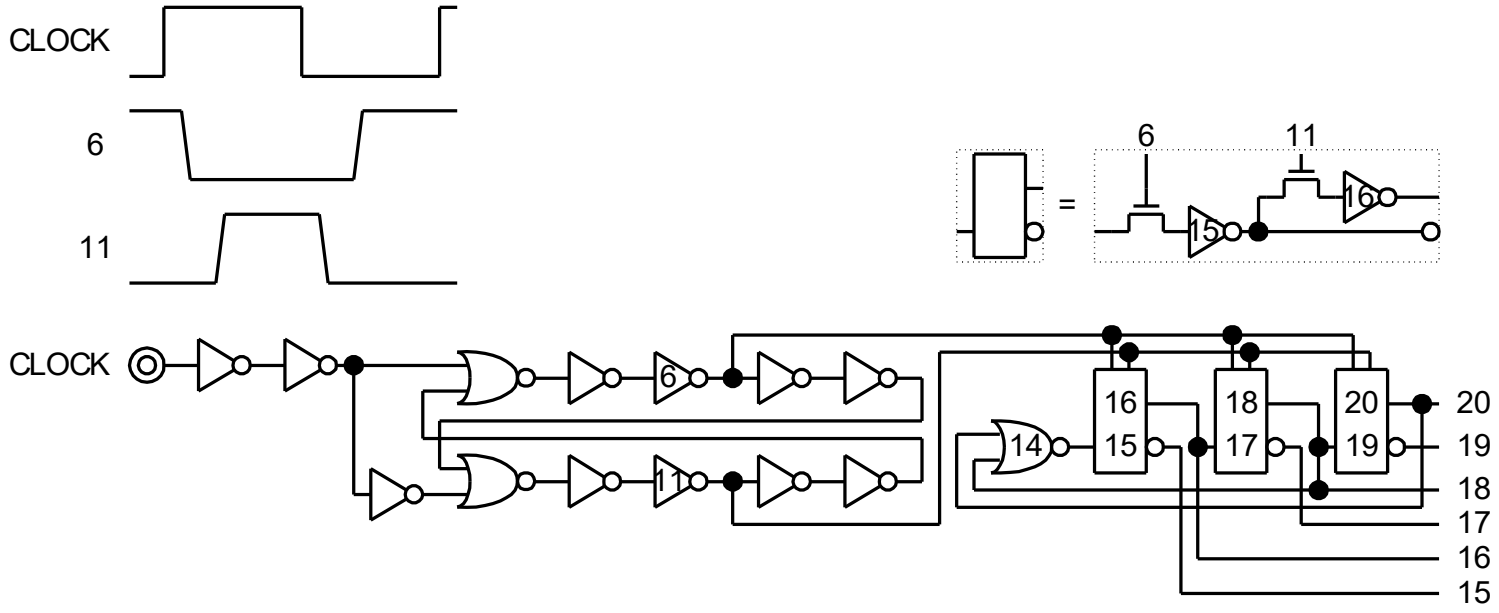
(525 – 48) / 8 = 60 Accordingly, the maximum Y co-ordinate becomes 60.

Implementation of 7 bits processor and registers on μPD777 is derived from the scale of X-Y co-ordinate system.

The co-ordinate allocation definition of X-75 & Y-60 realizes bunch of square (not rectangular) dots display on television which aspect ratio is 4:3 as well. (See Page 3)

µPD777 On-chip Clock Generator with Prescaler Function

(A) Logic



(1) 3 bit delayed flip-flops with NOR2 consists of a polynomial counter that produces 5 different values of "6, 4, 0, 1, 3" repeatedly.

(2) The polynomial counter enters an infinite loop of "6, 4, 0, 1, 3" and generates stable Φ[4:1] based upon the value.

(3) Instruction cycle (on-chip clock cycle) is defined as a cycle of rising edge of Φ2 to next rising edge of Φ2.

(4) One instruction cycle is equivalent of CLOCK (3.579545MHz) level transition of "0-1-0-1-0" or "1-0-1-0-1".

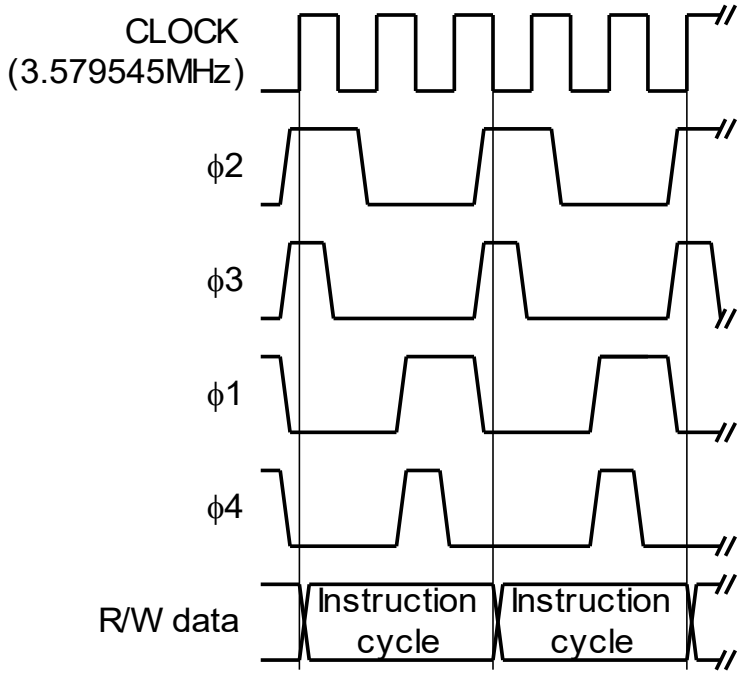
(5) Two instruction cycles (on-chip clocks) are equivalent of five CLOCKs as function of a prescalar implemented.

(6) On-chip logic is working at 1.431818MHz (one instruction cycle).

(B) Clock Timing

15	16	17	18	19	20	Φ2	Φ3	Φ1	Φ4
x	**0**	x	**1**	x	x	**1**	0	0	0
0	x	**1**	x	x	x	**1**	0	0	0
1	x	x	x	**1**	x	0	0	**1**	0
x	x	x	**1**	x	**0**	0	0	**1**	0
x	**0**	**0**	x	x	x	0	**1**	0	0
0	**0**	x	x	x	x	0	**1**	0	0
x	x	x	x	**1**	**1**	0	0	0	**1**
x	x	x	**1**	**1**	x	0	0	0	**1**

Truth table of clock generation logic

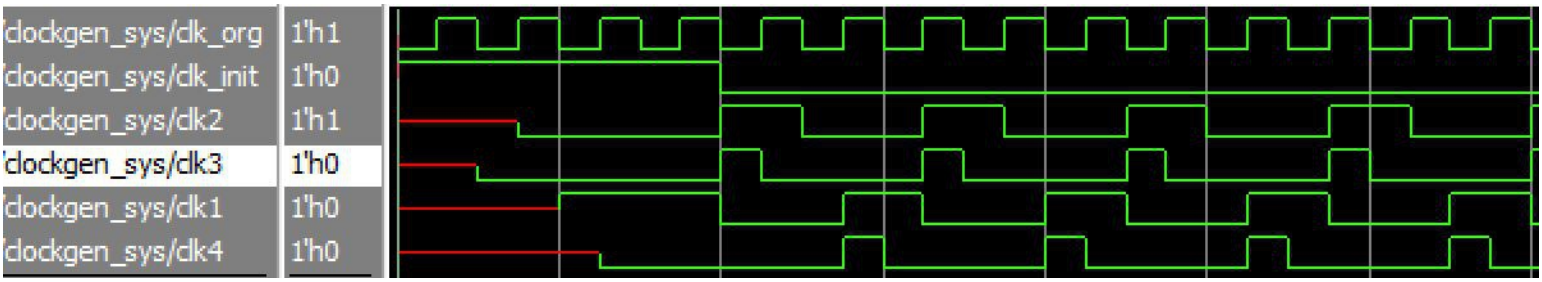


On-chip clock (Φ2, Φ3, Φ1, Φ4) waveform

(C) Truth Table

CLOCK	clock		Polynomial counter								On-chip clock				Polynomial counter sequence
	6	11	14	15	16	17	18	19	20	[20,18,16]	Φ2	Φ3	Φ1	Φ4	
0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	<Any sequence possible>
1	0	1	1	1	0	1	0	1	0	0	0	0	**1**	0	
0	1	0	1	0	0	1	0	1	0		**1**	**1**	0	0	
1	0	1	1	0	1	1	0	1	0	1	**1**	0	0	0	
0	1	0	1	0	1	0	0	1	0		0	0	0	0	
1	0	1	0	0	1	0	1	1	0	3	0	0	**1**	**1**	
0	1	0	0	1	1	0	1	0	0		0	0	**1**	0	
1	0	1	0	1	0	0	1	0	1	**6**	**1**	**1**	0	0	<Sequence fixed> 2 on-chip clocks = 5 external CLOCKs
0	1	0	0	1	0	1	1	0	1		**1**	0	0	0	
1	0	1	0	1	0	1	0	0	1	**4**	0	0	0	0	
0	1	0	0	1	0	1	0	1	1		0	0	**1**	**1**	
1	0	1	1	1	0	1	0	1	0	**0**	0	0	**1**	0	
0	1	0	1	0	0	1	0	1	0		**1**	**1**	0	0	
1	0	1	1	0	1	1	0	1	0	**1**	**1**	0	0	0	
0	1	0	1	0	1	0	0	1	0		0	0	0	0	
1	0	1	0	0	1	0	1	1	0	**3**	0	0	**1**	**1**	
0	1	0	0	1	1	0	1	0	0		0	0	**1**	0	
1	0	1	0	1	0	0	1	0	1	**6**	**1**	**1**	0	0	<Sequence fixed> 2 on-chip clocks = 5 external CLOCKs
0	1	0	0	1	0	1	1	0	1		**1**	0	0	0	
1	0	1	0	1	0	1	0	0	1	**4**	0	0	0	0	
0	1	0	0	1	0	1	0	1	1		0	0	**1**	**1**	
1	0	1	1	1	0	1	0	1	0	**0**	0	0	**1**	0	
0	1	0	1	0	0	1	0	1	0		**1**	**1**	0	0	
1	0	1	1	0	1	1	0	1	0	**1**	**1**	0	0	0	
0	1	0	1	0	1	0	0	1	0		0	0	0	0	
1	0	1	0	0	1	0	1	1	0	**3**	0	0	**1**	**1**	
0	1	0	0	1	1	0	1	0	0		0	0	**1**	0	

(D) Verilog HDL Simulation Result



Simulation wave form above exhibits the same as (B) Clock Timing & (C) Truth Table.

(D-2) Verilog HDL Simulation Files

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Testbench for Clock Generator
//  clockgen_sys.v
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
`timescale  1ns / 1ns

module   clockgen_sys;

// regs/wires/integers
reg      clk_org, clk_init;
wire     clk1, clk2, clk3, clk4;

// Simulation target
clockgen  clockgen(clk_org, clk_init, clk1, clk2, clk3, clk4);

//--------------------
// Simulation vector
//--------------------
initial
begin
      clk_org  = 1'b0;
      clk_init = 1'b1;
#200  clk_init = 1'b0;

#550  $finish;
end

// 20MHz clock generator
always  #25   clk_org = ~clk_org;

endmodule
```

```
//~~~~~~~~~~~~~~~~~~~~~~
// Clock Generator
//  clockgen.v
//~~~~~~~~~~~~~~~~~~~~~~
module  clockgen(clk_org, clk_init, clk1, clk2, clk3, clk4);

// I/O definition
input   clk_org;         // Original clock  (H/L)
input   clk_init;        // Clock initialization (H)

output  clk1;            // Phi 1 (clock)   (H/L)
output  clk2;            // Phi 2 (clock)   (H/L)
output  clk3;            // Phi 3 (clock)   (H/L)
output  clk4;            // Phi 4 (clock)   (H/L)

// Regs for random logic
wire    clk_org, clk_init;
reg     clk1, clk2, clk3, clk4;
reg     g6, g11, g14, g15, g16, g17, g18, g19, g20;

//---------------
// Random logic
//---------------
always @*
begin
  g6   = ~clk_org;
  g11  = ~g6;
  g14  = ~(g18 | g20 | clk_init);
  clk1 = ((g18 & ~g20) | (g15 & g19));
  clk2 = ~(~((~g16 & g18) | (~g15 & g17)) | clk1 );
  clk3 = ~((g16 | g17) & (g15 | g16));
  clk4 = ((g18 & g19) | (g19 & g20));
end

//---------------
// DFF
//---------------
always @ (posedge g6)
begin
  g15  <= ~g14;
  g17  <= ~g16;
  g19  <= ~g18;
end

always @ (posedge g11)
begin
  g16  <= ~g15;
  g18  <= ~g17;
  g20  <= ~g19;
end

endmodule
```

X Y Coordinate

Coordinate	Registers	Range
X	X[7:1]	0 - 90
Y	Y[6:1], y[3:1]	0 – 59, 0 - 7

MODE Registers

MODE[7:1]	Name	Function		
			1	0
7	REV	Reverberated sound effect	Enabled	Disabled
6	BRIGHTNESS	Brightness	High	Low
5	HUE	Hue	Dense	Thin
4	BLACK/PRIO	Black&White	Enabled	Color only
3	BG R	Background color Red bit	See color combination table (page 2)	
2	BG G	Background color Green bit		
1	BG B	Background color Blue bit		

ySUB

y[3:1]	ySUB	y'[3:1] = y[3:1] - ySUB
000	0	000
001	0	001
010	0	010
011	0	011
100	0	100
101	0	101
110	0	110
111	0	111
000	1	111
001	1	000
010	1	001
011	1	010
100	1	011
101	1	100
110	1	101
111	1	110

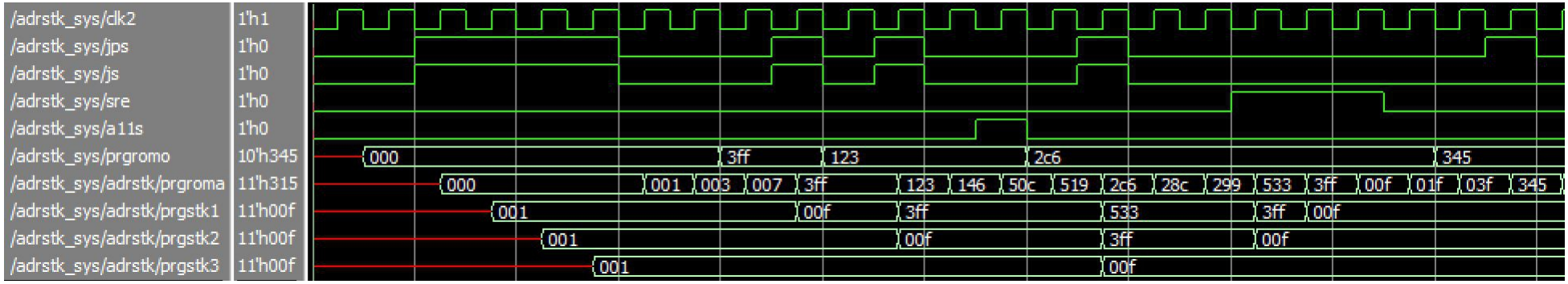
y'[3:1] is provided to pattern ROM address.

Program ROM Address Counter and Stack Registers

11 bits address counter contents can be stored at three levels of address stack registers as return addresses when jump subroutine instruction (JS) is executed.

The return addresses stored in the stack registers is poped down and returned to address counter when subroutine end instruction (SRE) is executed.

(A) Verilog HDL Simulation Result



- ACL asserts JS instruction forcing program ROM outputs to zero to initialize address counter and address stack registers.
- “a11s” sets A11 to LSB of “prgromo” (Switch program ROM address bank).
- SRE (“sre”) pops down address stack register contents (“prgstk3”, “prgstk2”, and “prgstk1”) to address counter (“prgroma”).
- JMP (“jps”) sets “prgromo” to address counter.
- JS (“js” and “jps”) sets “prgromo” to address counter, stacks next address to “prgstk1”, and pushes up to “prgstk2” and “prgstk3”.

(A-2) Verilog HDL Simulation Files

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Testbench for ROM address counter/stack
//  adrstk_sys.v
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
`timescale  1ns / 1ns

module   adrstk_sys;

// regs/wires/integers
reg      clk2, jps, js, sre, a11s;
reg      [10:1]prgromo;

// Simulation target
adrstk  adrstk(clk2, jps, js, sre, a11s, prgromo);

//--------------------
// Simulation vector
//--------------------
initial
begin
      clk2          = 1'b0;
      jps           = 1'b0;
      js            = 1'b0;
      sre           = 1'b0;
      a11s          = 1'b0;
#50   prgromo[10:1] = 10'h0;
#50   js            = 1'b1;
      jps           = 1'b1;
#200  js            = 1'b0;
      jps           = 1'b0;
#100
      prgromo[10:1] = 10'h3ff;
#50   js            = 1'b1;
      jps           = 1'b1;
#50   js            = 1'b0;
      jps           = 1'b0;
      prgromo[10:1] = 10'h123;
#50   js            = 1'b1;
      jps           = 1'b1;
#50   js            = 1'b0;
      jps           = 1'b0;
#50   a11s          = 1'b1;
#50   a11s          = 1'b0;
      prgromo[10:1] = 10'h2c6;
#50   js            = 1'b1;
      jps           = 1'b1;
#50   js            = 1'b0;
      jps           = 1'b0;
#50
#50   sre           = 1'b1;
#50   sre           = 1'b1;
#50   sre           = 1'b1;
#50   sre           = 1'b0;
#50   prgromo[10:1] = 10'h345;
#50   jps           = 1'b1;
#50   jps           = 1'b0;

#100  $finish;
end
```

```
// 20MHz clock generator
always  #25   clk2 = ~clk2;

endmodule


//~~~~~~~~~~~~~~~~~~~~~~
// Address Counter
//  adrstk.v
//~~~~~~~~~~~~~~~~~~~~~~
module  adrstk(clk2, jps, js, sre, a11s, prgromo);

// I/O definition
input   clk2;           // Phi2 clock             (H/L)
input   jps;            // JP or JS (JumP or JS)  (H)
input   js;             // JS (Jump Subroutine)   (H)
input   sre;            // SRE (SubRoutine End)   (H)
input   a11s;           // A11 set                (H)
input   [10:1]prgromo;  // Program ROM output     (H/L)

// Wires for inputs
wire    clk2, jps, js, sre, a11s;
wire    [10:1]prgromo;

// Regs for random logic
reg     feedb;

// Regs for registers
reg     [11:1]prgroma;
reg     [11:1]prgstk1;
reg     [11:1]prgstk2;
reg     [11:1]prgstk3;

//---------------
// Random logic
//---------------
always @*
begin
  feedb = ~(prgroma[6] ^ prgroma[7]);
end

//---------------
// Registers
//---------------
always @(posedge clk2)
begin
case ({a11s, jps, js, sre})
  4'b1000  :
  begin
    prgroma[11]   <= prgromo[1];
    prgroma[7:2]  <= prgroma[6:1];
    prgroma[1]    <= feedb;
  end
  4'b0100  : prgroma[10:1] <= prgromo[10:1];
  4'b0110  :
  begin
    prgroma[11]   <= 1'b0;
    prgroma[10:1] <= prgromo[10:1];
    prgstk1[11:8] <= prgroma[11:8];
    prgstk1[7:2]  <= prgroma[6:1];
    prgstk1[1]    <= feedb;
```

```
    prgstk2[11:1] <= prgstk1[11:1];
    prgstk3[11:1] <= prgstk2[11:1];
  end
  4'b0001  :
  begin
    prgstk2[11:1] <= prgstk3[11:1];
    prgstk1[11:1] <= prgstk2[11:1];
    prgroma[11:1] <= prgstk1[11:1];
  end
  4'b0000  :
  begin
    prgroma[7:2]  <= prgroma[6:1];
    prgroma[1]    <= feedb;
  end
endcase
end

endmodule
```

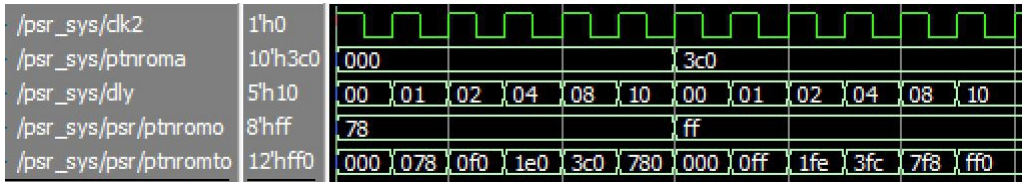
Pattern ROM Parallel–Serial Shift Registers

8 bits of pattern ROM contents are read in parallel and converted to serial signal by parallel-serial conversion registers.

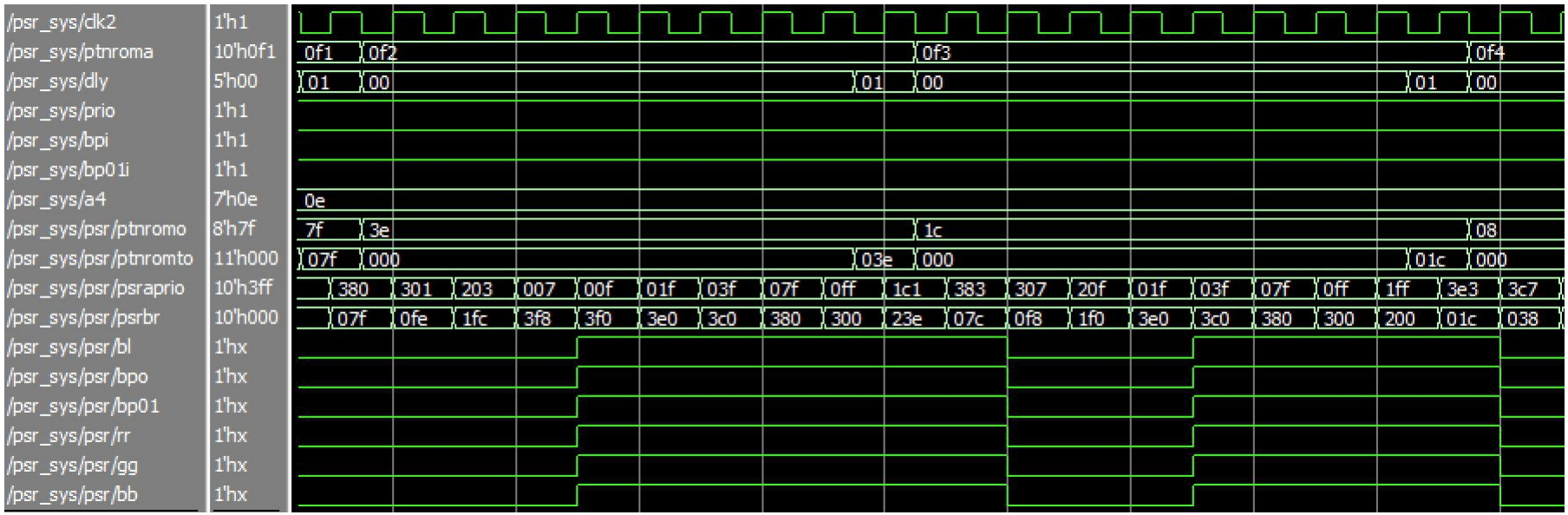
To allow pattern overllaping up to five patterns including the patterns with same X co-ordinate;
- Pattern position allocation logic controlled by dly[4:0] is implemented.
- The number of stages is 11 (7 + 5 - 1).

(A) Verilog HDL Simulation Result

(1) Pattern position allocation logic

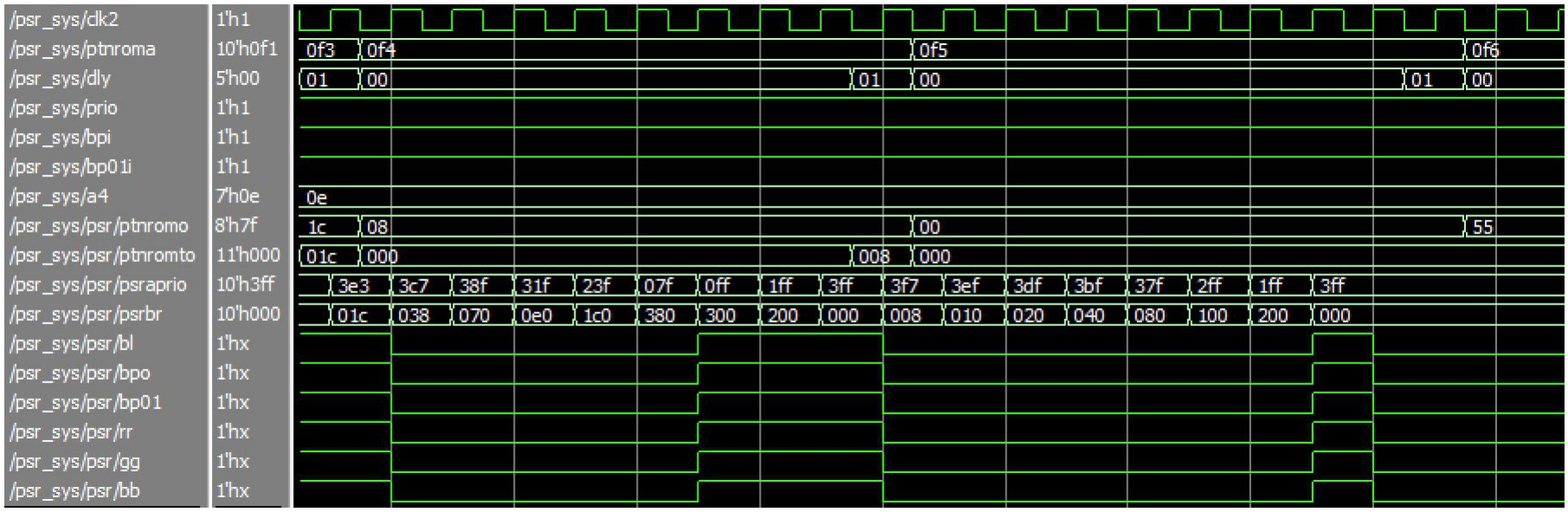


(2) µPD778 "Baseball" Pattern ROM

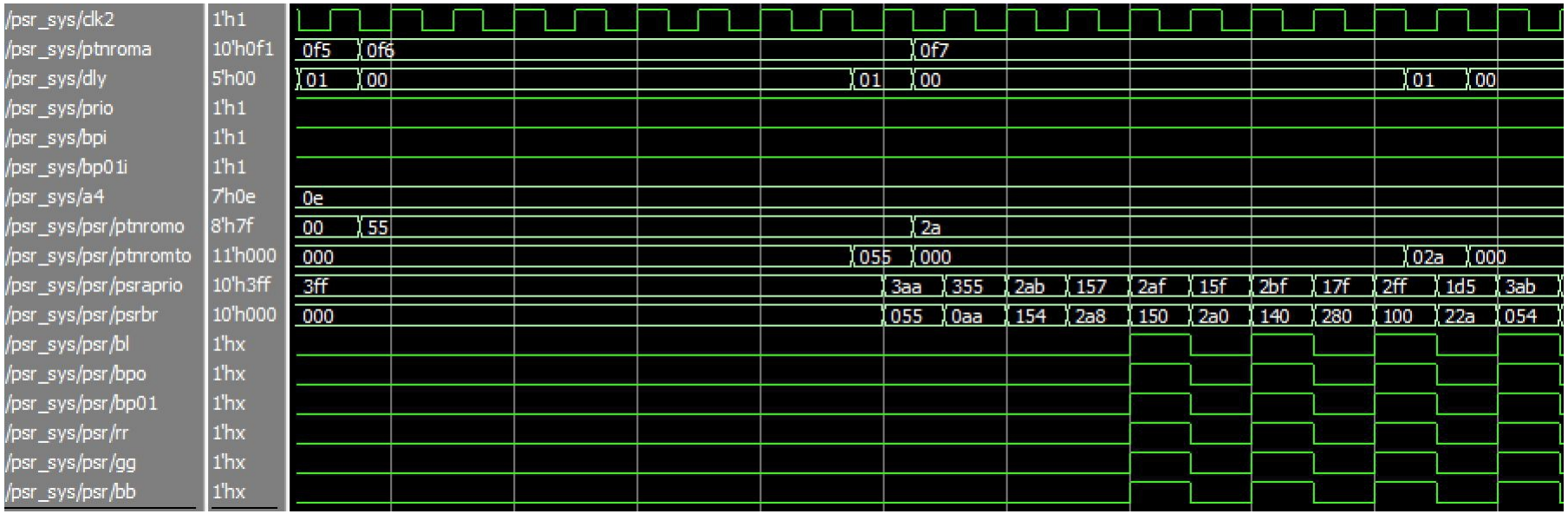


PTN address = 0f1h (y=1; 7fh), 0f2h (y=2; 3eh)

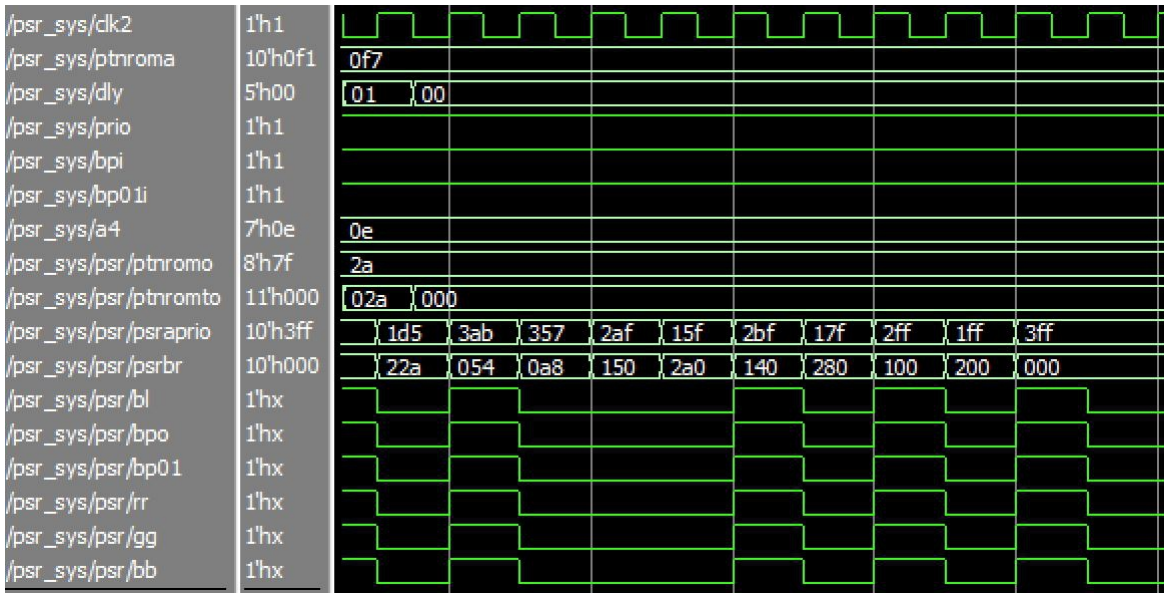
- "ptnroma" is pattern ROM address.
- "ptnromo" is pattern ROM output.
- "dly" is load timing with delay 0.
- Lower 6 rows ("bl", "bpo", "bp01", "rr", "gg", and ""bb") are serial output signals. In this simulation, they are the same.



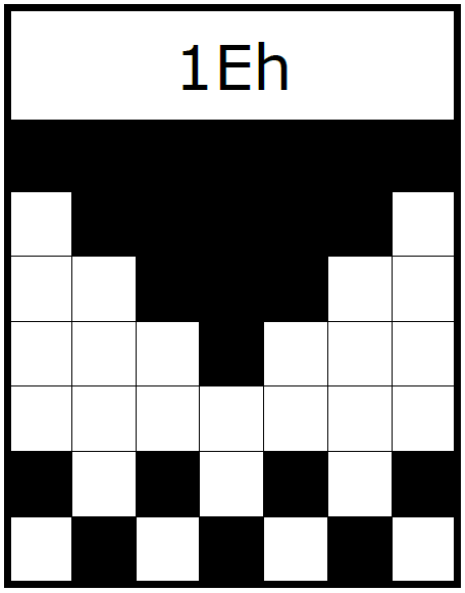
PTN address = 0f3h (y=3; 1ch), 0f4h (y=4; 08h)



PTN address = 0f5h (y=5; 00h), 0f6h (y=6; 55h)

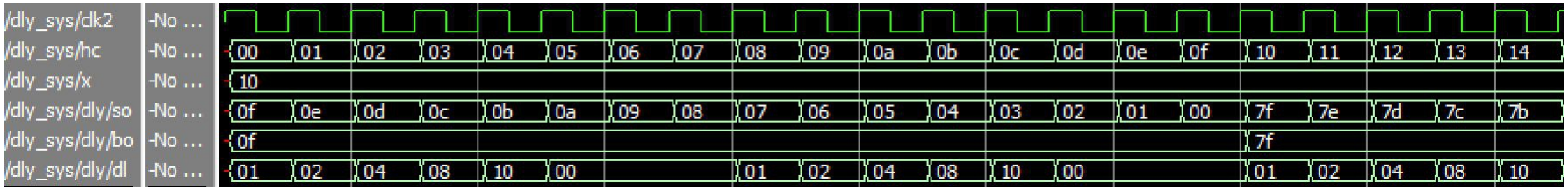


PTN address = 0f7h (y=7; 2ah)



Pattern applied

(3) "dly[4:0]" generation logic



- The case that 5 sprites are positioned at the same X coordinate (10h) is shown.
- They can be overlapped and displayed cooperating with pattern position allocation logic (see (1)).

(A-2) Verilog HDL Simulation Files

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Testbench for Parallel Serial Register
//  psr_sys.v
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
`timescale  1ns / 1ns

module  psr_sys;

// regs/wires/integers
reg     clk2;
reg     [9:0]ptnroma;
reg     [4:0]dly;
reg     prio, bpi, bp01i;
reg     [7:1]a4;

// Simulation target
psr  psr(clk2, ptnroma, dly, prio, bpi, bp01i, a4);

//--------------------
// Simulation vector
//--------------------

initial
begin
      clk2 = 1'b1;
// suppressed, 0 to 4 bit shift
#25   ptnroma[9:0] = 10'h001;  // ptnromo=78h (7 x 7)
      dly[4:0]     = 5'h00;    // No load
#50   dly[4:0]     = 5'h01;    // Load ptnromto 0 bit shift
#50   dly[4:0]     = 5'h02;    // Load ptnromto 1 bit shift
#50   dly[4:0]     = 5'h04;    // Load ptnromto 2 bit shift
#50   dly[4:0]     = 5'h08;    // Load ptnromto 3 bit shift
#50   dly[4:0]     = 5'h10;    // Load ptnromto 4 bit shift
#50   ptnroma[9:0] = 10'h3c0;  // ptnromo=ffh (8 x 7)
      dly[4:0]     = 5'h00;    // No load
#50   dly[4:0]     = 5'h01;    // Load ptnromto 0 bit shift
#50   dly[4:0]     = 5'h02;    // Load ptnromto 1 bit shift
#50   dly[4:0]     = 5'h04;    // Load ptnromto 2 bit shift
#50   dly[4:0]     = 5'h08;    // Load ptnromto 3 bit shift
#50   dly[4:0]     = 5'h10;    // Load ptnromto 4 bit shift

// Para-seri register 0 bit shift
#50   dly[4:0]     = 5'h00;    // No load
      prio         = 1'b1;
      bpi          = 1'b1;
      bp01i        = 1'b1;
      a4[7:1]      = 7'h0e;    // R = G = B = 1
      ptnroma[9:0] = 10'h0f1;  // H=0fh, L=0h, y=1, Out=ffh
#400  dly[4:0]     = 5'h01;    // Load 7fh 0 bit shift
#50   dly[4:0]     = 5'h00;    // No load
      ptnroma[9:0] = 10'h0f2;  // y=2, Out=7fh
#400  dly[4:0]     = 5'h01;    // Load 3eh 0 bit shift
#50   dly[4:0]     = 5'h00;    // No load
      ptnroma[9:0] = 10'h0f3;  // y=3, Out=3ef
#400  dly[4:0]     = 5'h01;    // Load 1ch 0 bit shift
#50   dly[4:0]     = 5'h00;    // No load
      ptnroma[9:0] = 10'h0f4;  // y=4, Out=1ch
#400  dly[4:0]     = 5'h01;    // Load 08h 0 bit shift
#50   dly[4:0]     = 5'h00;    // No load
```

```
      ptnroma[9:0] = 10'h0f5;  // y=5, Out=08h
#400  dly[4:0]     = 5'h01;    // Load 00h 0 bit shift
#50   dly[4:0]     = 5'h00;    // No load
      ptnroma[9:0] = 10'h0f6;  // y=6, Out=00h
#400  dly[4:0]     = 5'h01;    // Load 55h 0 bit shift
#50   dly[4:0]     = 5'h00;    // No load
      ptnroma[9:0] = 10'h0f7;  // y=7, Out=55h
#400  dly[4:0]     = 5'h01;    // Load 2ah 0 bit shift
#50   dly[4:0]     = 5'h00;    // No load
#450

#100  $finish;
end

// 20MHz clock generator
always  #25   clk2 = ~clk2;

endmodule


//~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Parallel Serial Register
//  psr.v
//~~~~~~~~~~~~~~~~~~~~~~~~~~~
module  psr(clk2, ptnroma, dly, prio, bpi, bp01i, a4);

// I/O definition
input   clk2;             // Clock Phi2                       (H/L)
input   [9:0]  ptnroma;   // Pattern ROM address              (H)
input   [4:0]  dly;       // Delay                            (H)
input   prio;             // PRIO                             (H)
input   bpi;              // Bent pattern                     (H)
input   bp01i;            // Bent even/odd                    (H)
input   [7:1]  a4;        // A4[4]=R, A4[3]=G, A4[2]=B        (H)

// Regs for random logic
wire    [4:0]  dly;
wire    [7:0]  ptnromo;   // Pattern ROM output               (H)
reg     [12:2] ptnromto;  // Pattern ROM transmission output  (H)
reg     [10:1] psraprio;  // Para-seri register Prio          (H)
reg     [10:1] psrabp;    // Para-seri register BP            (H)
reg     [10:1] psrabp01;  // Para-seri register BP0/1         (H)
reg     [10:1] psrbr;     // Para-seri register R             (H)
reg     [10:1] psrbg;     // Para-seri register G             (H)
reg     [10:1] psrbb;     // Para-seri register B             (H)
reg     [11:1] ptnp;      // Prio vs Pattern                  (H)
reg     bl, bpo, bp01, rr, gg, bb;
wire    prio, bpi, bp01i;
wire    vcc, gnd;

//---------------
// Random logic
//---------------
always @(*)
begin
  case (dly[4:0])
    5'b00001 : ptnromto[12:2] = {gnd, gnd, gnd, gnd, ptnromo[7:0]};
    5'b00010 : ptnromto[12:2] = {gnd, gnd, gnd, ptnromo[7:0], gnd};
    5'b00100 : ptnromto[12:2] = {gnd, gnd, ptnromo[7:0], gnd, gnd};
    5'b01000 : ptnromto[12:2] = {gnd, ptnromo[7:0], gnd, gnd, gnd};
    5'b10000 : ptnromto[12:2] = {ptnromo[7:0], gnd, gnd, gnd, gnd};
    default  : ptnromto[12:2] = 11'h000;
```

```
  endcase

  ptnp[11] = ~(psraprio[10] & ptnromto[12]);
  ptnp[10] = ~(psraprio[9]  & ptnromto[11]);
  ptnp[9]  = ~(psraprio[8]  & ptnromto[10]);
  ptnp[8]  = ~(psraprio[7]  & ptnromto[9]);
  ptnp[7]  = ~(psraprio[6]  & ptnromto[8]);
  ptnp[6]  = ~(psraprio[5]  & ptnromto[7]);
  ptnp[5]  = ~(psraprio[4]  & ptnromto[6]);
  ptnp[4]  = ~(psraprio[3]  & ptnromto[5]);
  ptnp[3]  = ~(psraprio[2]  & ptnromto[4]);
  ptnp[2]  = ~(psraprio[1]  & ptnromto[3]);
  ptnp[1]  = ~ptnromto[2];
end

assign  vcc = 1'b1;
assign  gnd = 1'b0;

//------------
// Registers
//----------    --
always @(posedge (clk2))
begin
  bl           <=  (~psraprio[10] | (ptnromto[12] & prio));
  psraprio[10] <= ~(~psraprio[9]  | (ptnromto[11] & prio));
  psraprio[9]  <= ~(~psraprio[8]  | (ptnromto[10] & prio));
  psraprio[8]  <= ~(~psraprio[7]  | (ptnromto[9]  & prio));
  psraprio[7]  <= ~(~psraprio[6]  | (ptnromto[8]  & prio));
  psraprio[6]  <= ~(~psraprio[5]  | (ptnromto[7]  & prio));
  psraprio[5]  <= ~(~psraprio[4]  | (ptnromto[6]  & prio));
  psraprio[4]  <= ~(~psraprio[3]  | (ptnromto[5]  & prio));
  psraprio[3]  <= ~(~psraprio[2]  | (ptnromto[4]  & prio));
  psraprio[2]  <= ~(~psraprio[1]  | (ptnromto[3]  & prio));
  psraprio[1]  <= ~(ptnromto[2] & prio);
  bpo          <=  (~psrabp[10]   | (ptnromto[12] & bpi));
  psrabp[10]   <= ~(~psrabp[9]    | (ptnromto[11] & bpi));
  psrabp[9]    <= ~(~psrabp[8]    | (ptnromto[10] & bpi));
  psrabp[8]    <= ~(~psrabp[7]    | (ptnromto[9]  & bpi));
  psrabp[7]    <= ~(~psrabp[6]    | (ptnromto[8]  & bpi));
  psrabp[6]    <= ~(~psrabp[5]    | (ptnromto[7]  & bpi));
  psrabp[5]    <= ~(~psrabp[4]    | (ptnromto[6]  & bpi));
  psrabp[4]    <= ~(~psrabp[3]    | (ptnromto[5]  & bpi));
  psrabp[3]    <= ~(~psrabp[2]    | (ptnromto[4]  & bpi));
  psrabp[2]    <= ~(~psrabp[1]    | (ptnromto[3]  & bpi));
  psrabp[1]    <= ~(ptnromto[2] & bpi);
  bp01         <=  (~psrabp01[10] | (ptnromto[12] & bp01i));
  psrabp01[10] <= ~(~psrabp01[9]  | (ptnromto[11] & bp01i));
  psrabp01[9]  <= ~(~psrabp01[8]  | (ptnromto[10] & bp01i));
  psrabp01[8]  <= ~(~psrabp01[7]  | (ptnromto[9]  & bp01i));
  psrabp01[7]  <= ~(~psrabp01[6]  | (ptnromto[8]  & bp01i));
  psrabp01[6]  <= ~(~psrabp01[5]  | (ptnromto[7]  & bp01i));
  psrabp01[5]  <= ~(~psrabp01[4]  | (ptnromto[6]  & bp01i));
  psrabp01[4]  <= ~(~psrabp01[3]  | (ptnromto[5]  & bp01i));
  psrabp01[3]  <= ~(~psrabp01[2]  | (ptnromto[4]  & bp01i));
  psrabp01[2]  <= ~(~psrabp01[1]  | (ptnromto[3]  & bp01i));
  psrabp01[1]  <= ~(ptnromto[2] & bp01i);
  rr           <= (ptnp[11]) ? (psrbr[10]) : (a4[4]);
  psrbr[10]    <= (ptnp[10]) ? (psrbr[9])  : (a4[4]);
  psrbr[9]     <= (ptnp[9])  ? (psrbr[8])  : (a4[4]);
  psrbr[8]     <= (ptnp[8])  ? (psrbr[7])  : (a4[4]);
  psrbr[7]     <= (ptnp[7])  ? (psrbr[6])  : (a4[4]);
  psrbr[6]     <= (ptnp[6])  ? (psrbr[5])  : (a4[4]);
```

```
  psrbr[5]     <= (ptnp[5])  ? (psrbr[4])  : (a4[4]);
  psrbr[4]     <= (ptnp[4])  ? (psrbr[3])  : (a4[4]);
  psrbr[3]     <= (ptnp[3])  ? (psrbr[2])  : (a4[4]);
  psrbr[2]     <= (ptnp[2])  ? (psrbr[1])  : (a4[4]);
  psrbr[1]     <= (ptnp[1])  ? (gnd)       : (a4[4]);
  gg           <= (ptnp[11]) ? (psrbg[10]) : (a4[3]);
  psrbg[10]    <= (ptnp[10]) ? (psrbg[9])  : (a4[3]);
  psrbg[9]     <= (ptnp[9])  ? (psrbg[8])  : (a4[3]);
  psrbg[8]     <= (ptnp[8])  ? (psrbg[7])  : (a4[3]);
  psrbg[7]     <= (ptnp[7])  ? (psrbg[6])  : (a4[3]);
  psrbg[6]     <= (ptnp[6])  ? (psrbg[5])  : (a4[3]);
  psrbg[5]     <= (ptnp[5])  ? (psrbg[4])  : (a4[3]);
  psrbg[4]     <= (ptnp[4])  ? (psrbg[3])  : (a4[3]);
  psrbg[3]     <= (ptnp[3])  ? (psrbg[2])  : (a4[3]);
  psrbg[2]     <= (ptnp[2])  ? (psrbg[1])  : (a4[3]);
  psrbg[1]     <= (ptnp[1])  ? (gnd)       : (a4[3]);
  bb           <= (ptnp[11]) ? (psrbb[10]) : (a4[2]);
  psrbb[10]    <= (ptnp[10]) ? (psrbb[9])  : (a4[2]);
  psrbb[9]     <= (ptnp[9])  ? (psrbb[8])  : (a4[2]);
  psrbb[8]     <= (ptnp[8])  ? (psrbb[7])  : (a4[2]);
  psrbb[7]     <= (ptnp[7])  ? (psrbb[6])  : (a4[2]);
  psrbb[6]     <= (ptnp[6])  ? (psrbb[5])  : (a4[2]);
  psrbb[5]     <= (ptnp[5])  ? (psrbb[4])  : (a4[2]);
  psrbb[4]     <= (ptnp[4])  ? (psrbb[3])  : (a4[2]);
  psrbb[3]     <= (ptnp[3])  ? (psrbb[2])  : (a4[2]);
  psrbb[2]     <= (ptnp[2])  ? (psrbb[1])  : (a4[2]);
  psrbb[1]     <= (ptnp[1])  ? (gnd)       : (a4[2]);
end

//--------------
// Pattern ROM
//--------------
ptnrom_mod  ptnrom_1 (.ptnromo(ptnromo), .ptnroma(ptnroma));

endmodule
```

Refer to a PDF version of “ptnrom_mod.v”.

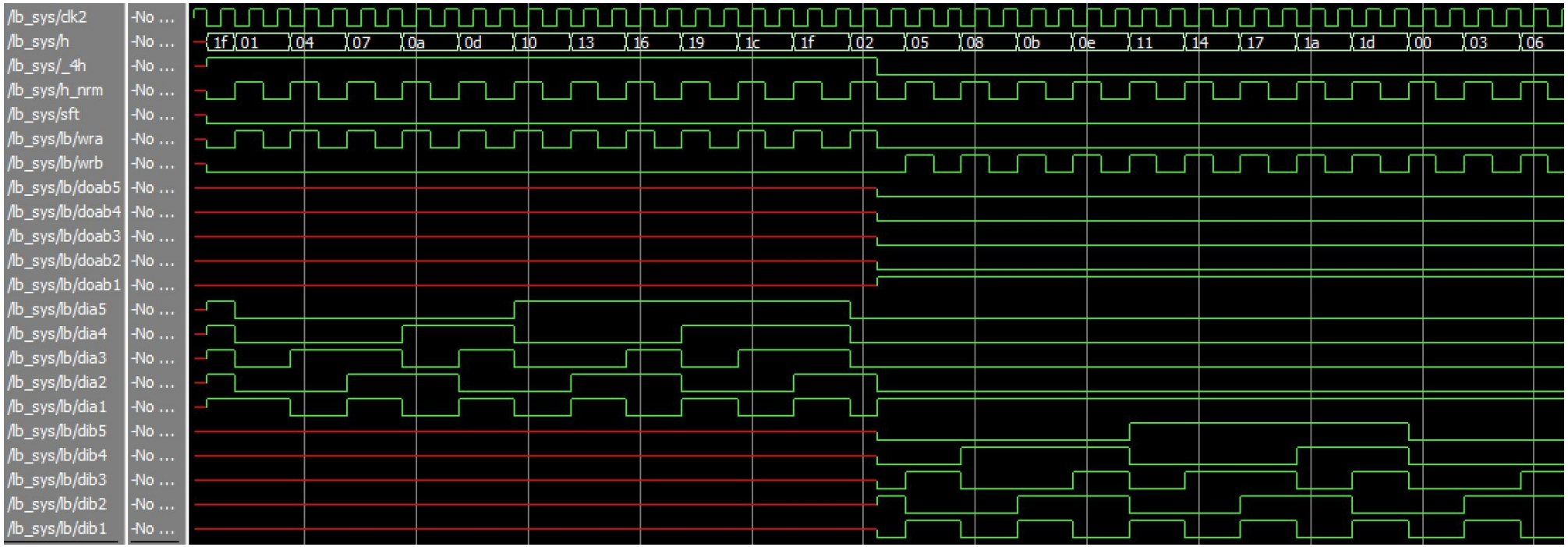
Line Buffer Registers

Line buffer registers consist of two register blocks (“lba” and “lbb”) of 12 stages by 5 bits register files which store upper 5 bits of pattern ROM address (H[5:1]).

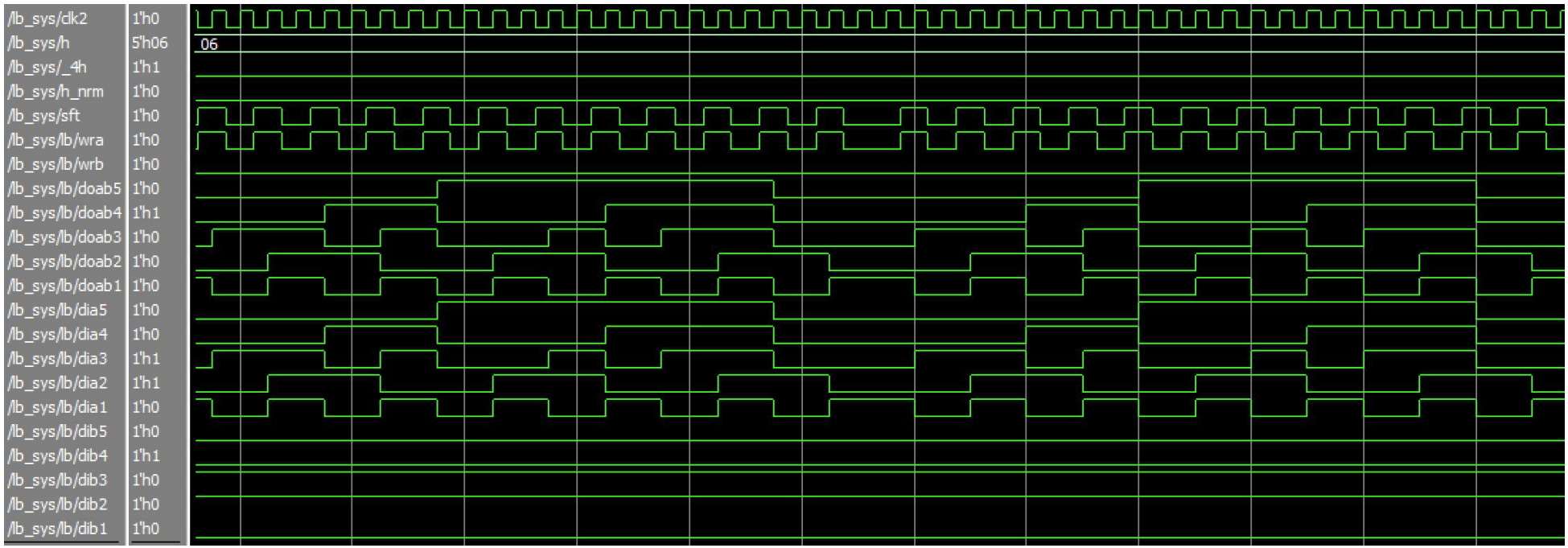
“H→NRM” instruction writes H[5:1] into one of “lba” and “lbb” without obstructing normal display.
When “lba” is in use for display, H[5:1] is written into “lbb”.
When “lbb” is in use for display, H[5:1] is written into “lba”.

Up to 12 sprites' H[5:1] must be written by software in 4H (4 scan lines display time; 360 clocks) into line buffer.
This line buffer resolves strict time constraint as well as provides the capability of up to 12 different sprites display in one scan line.

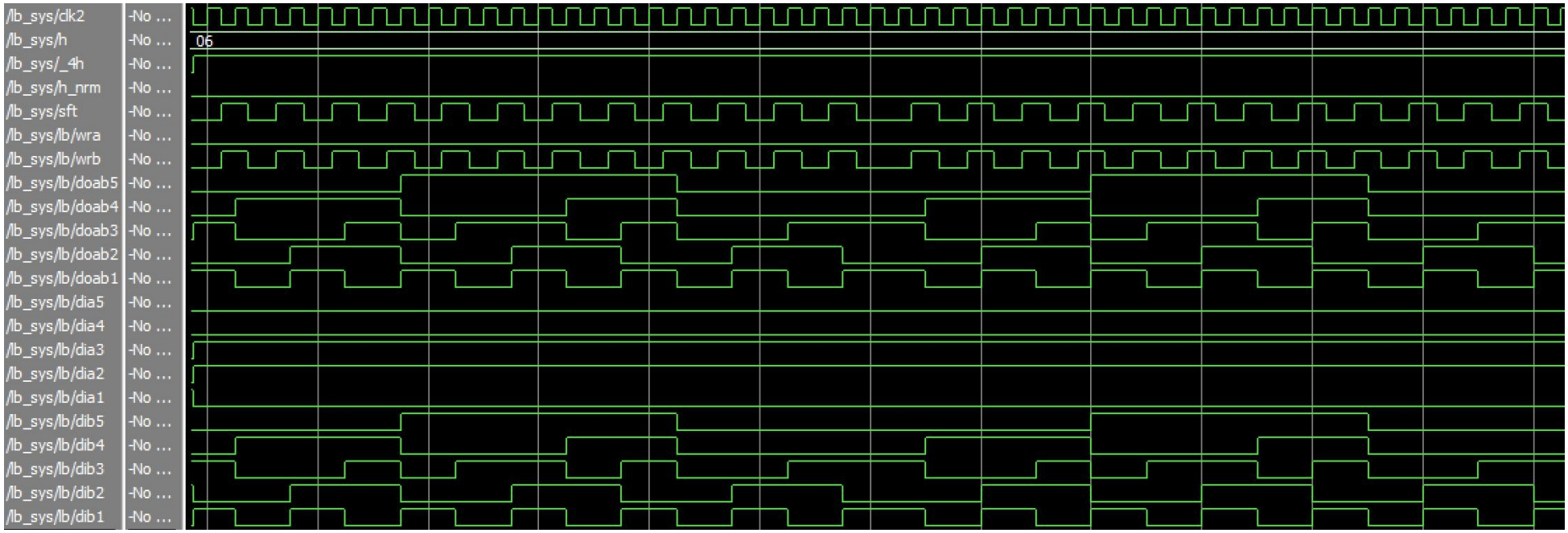
(A) Verilog HDL Simulation Result



- “H→NRM” instruction writes “01h, 04h, 07h, 0ah, 0dh, 10h, 13h, 16h, 19h, 1ch, 1fh, 02h” into “lba”.
- “H→NRM” instruction writes “05h, 08h, 0bh, 0eh, 11h, 14h, 17h, 1ah, 1dh, 00h, 03h, 06h” into “lbb”.



Reading H[5:1]s from "lba" for display in realtime.



Reading H[5:1]s from "lbb" for display in realtime.

(A-2) Verilog HDL Simulation Files

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Testbench for Line Buffer
//  lb_sys.v
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~
`timescale  1ns / 1ns

module  lb_sys;

// regs/wires/integers
reg      clk2;
reg      [4:0]  h;
reg      _4h, h_nrm, sft;

integer  i;

// Simulation target
lb  lb(clk2, _4h, h, h_nrm, sft);

//--------------------
// Simulation vector
//--------------------

initial
begin
      clk2 = 1'b1;
// suppressed, 0 to 4 bit shift
#25   h[4:0] = 5'h1f;
      h_nrm  = 1'b0;
      sft    = 1'b0;

// h_nrm to line buffer A (_4h = 1)
      _4h    = 1'b1;
for(i = 1;  i < 37;  i = i + 3)
begin
#50   h_nrm  = 1'b1;
      h[4:0] = i;
#50   h_nrm  = 1'b0;
end

// h_nrm to line buffer B (_4h = 0)
      _4h    = 1'b0;
for(i = 2;  i < 38;  i = i + 3)
begin
#50   h_nrm  = 1'b1;
      h[4:0] = i + 3;
#50   h_nrm  = 1'b0;
end

// H from line buffer B (_4h = 0)
#50   _4h    = 1'b0;
for(i = 0;  i < 12;  i = i + 1)
begin
#50   sft    = 1'b1;
#50   sft    = 1'b0;
end

// H from line buffer B (_4h = 0)
#50   _4h    = 1'b0;
for(i = 0;  i < 12;  i = i + 1)
begin
```

```
#50   sft    = 1'b1;
#50   sft    = 1'b0;
end

// H from line buffer A (_4h = 1)
#50   _4h    = 1'b1;
for(i = 0;  i < 12;  i = i + 1)
begin
#50   sft    = 1'b1;
#50   sft    = 1'b0;
end

// H from line buffer A (_4h = 1)
#50   _4h    = 1'b1;
for(i = 0;  i < 12;  i = i + 1)
begin
#50   sft    = 1'b1;
#50   sft    = 1'b0;
end

#100  $finish;
end

// 20MHz clock generator
always  #25   clk2 = ~clk2;

endmodule


//~~~~~~~~~~~~~~~
// Line Buffer
//  lb.v
//~~~~~~~~~~~~~~~
module  lb(clk2, _4h, h, h_nrm, sft);

// I/O definition
input   clk2;      // Clock Phi2                       (H/L)
input   _4h;       // 4 Horizontal period timing (VC4) (H)
input   [4:0] h;   // Pattern RAM address (H)          (H)
input   h_nrm;     // H to NRM instruction             (H)
input   sft;       // Shift timing                     (H)

// regs/wires
reg     doab5, doab4, doab3, doab2, doab1;
reg     dia5, dia4, dia3, dia2, dia1;
reg     dib5, dib4, dib3, dib2, dib1;
reg     wra, wrb;

wire    doa5, doa4, doa3, doa2, doa1;
wire    dob5, dob4, dob3, dob2, dob1;

//---------------
// Random logic
//---------------
lbr_mod  lbra5  (.clk2(clk2), .wr(wra), .di(dia5), .do(doa5));
lbr_mod  lbra4  (.clk2(clk2), .wr(wra), .di(dia4), .do(doa4));
lbr_mod  lbra3  (.clk2(clk2), .wr(wra), .di(dia3), .do(doa3));
lbr_mod  lbra2  (.clk2(clk2), .wr(wra), .di(dia2), .do(doa2));
lbr_mod  lbra1  (.clk2(clk2), .wr(wra), .di(dia1), .do(doa1));
lbr_mod  lbrb5  (.clk2(clk2), .wr(wrb), .di(dib5), .do(dob5));
lbr_mod  lbrb4  (.clk2(clk2), .wr(wrb), .di(dib4), .do(dob4));
lbr_mod  lbrb3  (.clk2(clk2), .wr(wrb), .di(dib3), .do(dob3));
```

```
lbr_mod  lbrb2  (.clk2(clk2), .wr(wrb), .di(dib2), .do(dob2));
lbr_mod  lbrb1  (.clk2(clk2), .wr(wrb), .di(dib1), .do(dob1));

always @(*)
begin
  doab5 = (_4h) ? (dob5)  : (doa5);
  doab4 = (_4h) ? (dob4)  : (doa4);
  doab3 = (_4h) ? (dob3)  : (doa3);
  doab2 = (_4h) ? (dob2)  : (doa2);
  doab1 = (_4h) ? (dob1)  : (doa1);
  wra   = (_4h) ? (h_nrm) : (sft);
  wrb   = (_4h) ? (sft)   : (h_nrm);
  dia5  = (_4h) ? (h[4])  : (doab5);
  dia4  = (_4h) ? (h[3])  : (doab4);
  dia3  = (_4h) ? (h[2])  : (doab3);
  dia2  = (_4h) ? (h[1])  : (doab2);
  dia1  = (_4h) ? (h[0])  : (doab1);
  dib5  = (_4h) ? (doab5) : (h[4]);
  dib4  = (_4h) ? (doab4) : (h[3]);
  dib3  = (_4h) ? (doab3) : (h[2]);
  dib2  = (_4h) ? (doab2) : (h[1]);
  dib1  = (_4h) ? (doab1) : (h[0]);
end

//------------
// Registers
//----------    --
//always @(posedge (clk2))
//begin

//end

endmodule

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Line Buffer Register Module
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
module  lbr_mod(clk2, wr, di, do);

// I/O definition
input   clk2;        // Clock Phi2    (H/L)
input   wr;          // Write strobe  (H)
input   di;          // Data input    (H)
output  do;          // Data output   (H)

// regs/wires
reg     [11:0] lbr;  // 12 bit Line buffer register  (H)
reg     do;

//------------
// Registers
//------------
always @(posedge (clk2))
begin
  lbr[11] <= (wr) ? (lbr[10]) : (lbr[11]);
  lbr[10] <= (wr) ? (lbr[9])  : (lbr[10]);
  lbr[9]  <= (wr) ? (lbr[8])  : (lbr[9]);
  lbr[8]  <= (wr) ? (lbr[7])  : (lbr[8]);
  lbr[7]  <= (wr) ? (lbr[6])  : (lbr[7]);
  lbr[6]  <= (wr) ? (lbr[5])  : (lbr[6]);
  lbr[5]  <= (wr) ? (lbr[4])  : (lbr[5]);
  lbr[4]  <= (wr) ? (lbr[3])  : (lbr[4]);
```

```
  lbr[3]  <= (wr) ? (lbr[2])  : (lbr[3]);
  lbr[2]  <= (wr) ? (lbr[1])  : (lbr[2]);
  lbr[1]  <= (wr) ? (lbr[0])  : (lbr[1]);
  lbr[0]  <= (wr) ? (di)      : (lbr[0]);
end

//---------------
// Random logic
//---------------
always @(*)
begin
  do = lbr[11];
end

endmodule
```

Sound Generator

Sound suppression is performed by setting 01h to sound frequency registers (FLS[7:1] & FRS[7:1]).
01→M
M→FLS, 0→L or M→FRS, 1→L

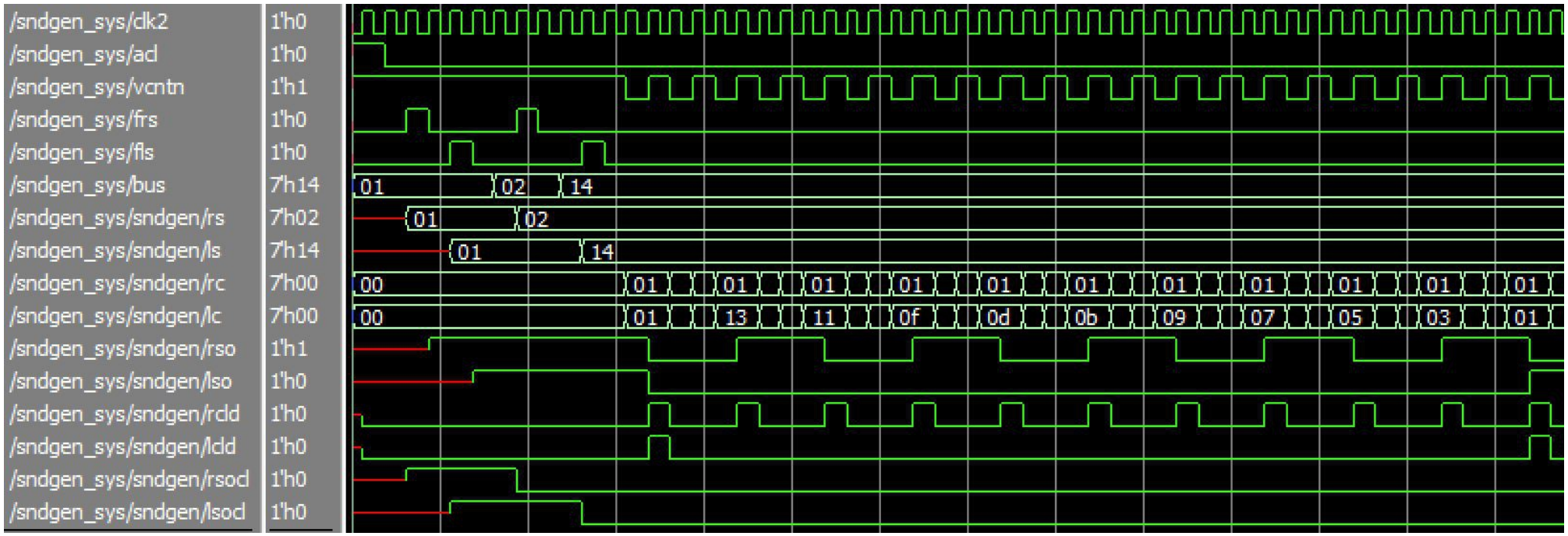
A counter which initial value is FLS[7:1] or FRS[7:1] down-counts every horizontal period (15.734 KHz). Two sets of sound down-counters are implemented and the sounds generated are mixed together. Therefore, SOUND pin outputs monaural sound superimposed the two sound sources.

(A) FLS/FRS vs. Frequency

FLS[7:1] or FRS[7:1]	Frequency (Hz)
01h (01d)	0 (No sound)
02h (02d)	15,734
03h (03d)	7,867
--	
0Bh (11d)	1,573
--	
24h (36d)	787
--	
3Ch (60d)	450
--	
7fh (127d)	125

Sound can be reverberated in conformity to REV input when REV mode (MODE[7]) is set to 1.

(B) Verilog HDL Simulation Result



- The cycle of count pulse (“vcntn”) is shortened.
- “01h” is set to “rs” and “ls” (sound is suppressed).
- “02h” is set to “rs” and “14h” is set to “ls”.
- Right channel sound output, “rso”, altered every two count pulses (15,734KHz).
- Left channel sound output, “lso”, altered every 20 count pulses.

(D-2) Verilog HDL Simulation Files

```
//~~~~~~~~~~~~~~~~~~~~~~
// Testbench for SNDGEN
//  sndgen_sys.v
//~~~~~~~~~~~~~~~~~~~~~~
`timescale  1ns / 1ns

module   sndgen_sys;

// regs/wires/integers
reg  clk2, acl, vcntn, frs, fls;
reg  [7:1]bus;

integer  i;

// Simulation target
sndgen  sndgen(clk2, acl, vcntn, frs, fls, bus);

//--------------------
// Simulation vector
//--------------------
initial
begin
      clk2     = 1'b0;
      acl      = 1'b1;
      frs      = 1'b0;
      fls      = 1'b0;
      vcntn    = 1'b1;
      bus[7:1] = 7'h1;    // Sound suppression
#25   acl      = 1'b1;
#50   acl      = 1'b0;
#50   frs      = 1'b1;
#50   frs      = 1'b0;
#50   fls      = 1'b1;
#50   fls      = 1'b0;
#50   bus[7:1] = 7'h02;   // 15,734Hz (max f)
#50   frs      = 1'b1;
#50   frs      = 1'b0;
#50   bus[7:1] = 7'h14;   // Hit
#50   fls      = 1'b1;
#50   fls      = 1'b0;

for (i = 0; i < 1000; i = i + 1)
begin
#50   vcntn = ~vcntn;
end

#100  $finish;
end

// 20MHz clock generator
always  #25   clk2 = ~clk2;

endmodule


//~~~~~~~~~~~~
// SNDGEN
//  sndgen.v
//~~~~~~~~~~~~
```

```
module  sndgen(clk2, acl, vcntn, frs, fls, bus);

// I/O definition
input   clk2;      // Phi2 clock          (H/L)
input   acl;       // Auto CLear          (H)
input   vcntn;     // VC count            (L)
input   frs;       // Right data load     (L)
input   fls;       // Left data load      (L)
input   [7:1]bus;  // Data bus            (L)

// Regs for registers
reg     [7:1]rs;   // Right data          (H)
reg     [7:1]ls;   // Left data           (H)
reg     [7:1]rc;   // Right count         (H)
reg     [7:1]lc;   // Left count          (H)
reg     rso;       // Right sound output  (H)
reg     lso;       // Left sound output   (H)
reg     rcld;      // Right count load    (H)
reg     lcld;      // Left count load     (H)

// Regs for random logic
reg     rc2t;
reg     rc3t;
reg     rc4t;
reg     rc5t;
reg     rc6t;
reg     rc7t;
reg     lc2t;
reg     lc3t;
reg     lc4t;
reg     lc5t;
reg     lc6t;
reg     lc7t;
reg     rsot;
reg     lsot;
reg     rsocl;
reg     lsocl;

//------------
// Registers
//------------
always @ (posedge(clk2))
begin
  rcld <= rsot;
  lcld <= lsot;
  if (frs)  rs[7:1] <= bus[7:1];
  if (fls)  ls[7:1] <= bus[7:1];
  if (rsocl)  rso <= 1'b1;
  if (lsocl)  lso <= 1'b1;
end

always @ (posedge(clk2) or posedge(acl))
begin
  if (acl)
  begin
    rc[7:1] <= 7'h00;
    lc[7:1] <= 7'h00;
  end
  else
  begin
    if (~vcntn)
    begin
```

```
      rc[1] <= ~rc[1];
      lc[1] <= ~lc[1];
    end
    if (rc2t)  rc[2]   <= ~rc[2];
    if (rc3t)  rc[3]   <= ~rc[3];
    if (rc4t)  rc[4]   <= ~rc[4];
    if (rc5t)  rc[5]   <= ~rc[5];
    if (rc6t)  rc[6]   <= ~rc[6];
    if (rc7t)  rc[7]   <= ~rc[7];
    if (lc2t)  lc[2]   <= ~lc[2];
    if (lc3t)  lc[3]   <= ~lc[3];
    if (lc4t)  lc[4]   <= ~lc[4];
    if (lc5t)  lc[5]   <= ~lc[5];
    if (lc6t)  lc[6]   <= ~lc[6];
    if (lc7t)  lc[7]   <= ~lc[7];
    if (rsot)  rso     <= ~rso;
    if (lsot)  lso     <= ~lso;
    if (rcld)  rc[7:1] <= rs[7:1];
    if (lcld)  lc[7:1] <= ls[7:1];
  end
end

//---------------
// Random logic
//---------------
always @(*)
begin
  rc2t  = ~(vcntn | rc[1]);
  rc3t  = ~(vcntn | rc[1] | rc[2]);
  rc4t  = ~(vcntn | rc[1] | rc[2] | rc[3]);
  rc5t  = ~(vcntn | rc[1] | rc[2] | rc[3] | rc[4]);
  rc6t  = ~(vcntn | rc[1] | rc[2] | rc[3] | rc[4] | rc[5]);
  rc7t  = ~(vcntn | rc[1] | rc[2] | rc[3] | rc[4] | rc[5] | rc[6]);
  lc2t  = ~(vcntn | lc[1]);
  lc3t  = ~(vcntn | lc[1] | lc[2]);
  lc4t  = ~(vcntn | lc[1] | lc[2] | lc[3]);
  lc5t  = ~(vcntn | lc[1] | lc[2] | lc[3] | lc[4]);
  lc6t  = ~(vcntn | lc[1] | lc[2] | lc[3] | lc[4] | lc[5]);
  lc7t  = ~(vcntn | lc[1] | lc[2] | lc[3] | lc[4] | lc[5] | lc[6]);
  rsot  = ~(vcntn | ~rc[1] | rc[2] | rc[3] | rc[4] | rc[5] | rc[6] | rc[7]);
  lsot  = ~(vcntn | ~lc[1] | lc[2] | lc[3] | lc[4] | lc[5] | lc[6] | lc[7]);
  rsocl = ~(~rs[1] | rs[2] | rs[3] | rs[4] | rs[5] | rs[6] | rs[7]);
  lsocl = ~(~ls[1] | ls[2] | ls[3] | ls[4] | ls[5] | ls[6] | ls[7]);
end

endmodule
```

Sequence of Entire Program ROM Code Dump (Abstract)

11 bit program ROM address can be specified by feeding ROM code of "JP xxx" and "1→A11" through 12 I/O pins under CH[2:1] = 1. Under CH[2:1] = 2, the program ROM code appears on the 12 I/O pins updating the lower 7 bit address counter contents from 000h to 040h automatically (See "7 bit polynomial address counter implemented"). "JP xxx" specifies lower 10 bit addresses and "1→A11" sets A11 (MSB (Most Significant Bit) of ROM page address) to 1. Each instruction takes one clock and then the ROM code updated is output one clock later. Needs (129x16 + 1) on-chip clocks.

Beforehand, put prescaler initialization sequence (gray zone) below along with feeding AC/.

Feeding instruction codes and dumping ROM code must be synchronized with on-chip clock, not CLOCK (See "On-chip Prescaler").

CLOCK	On-chip clocks			AC/	Instruction	ROM Address	CH		I/O Pin											
	ϕ2	ϕ1	Cycle				2	1	K1/	K2/	K3/	K4/	K5/	K6/	K7/	GP/	PD4	PD3	PD2	PD1
1	0	1	x	0	NOP	xxx xxxx xxxx	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0																				
1																				
0																				
1																				
0																				
1																				
0		0					0													
1	1		1																	
0	0																			
1																				
0		1		1																
1																				
0	1	0	1	1	JP 000	xxx xxxx xxxx	0	1	1	0	0	0	0	0	0	0	0	0	0	0
1	0																			
0																				
1		1																		
0		0																		
1	1		1		000 0000 0000		1	0	x	x	x	x	x	x	x	x	x	x	x	x
0	0																			
1																				
0		1																		
1		0																		

On-chip Clocks	Instruction	ROM Address	CH		I/O Pin											
			2	1	K1/	K2/	K3/	K4/	K5/	K6/	K7/	GP/	PD4	PD3	PD2	PD1
1	JP 000	xxx xxxx xxxx	0	1	1	0	0	0	0	0	0	0	0	0	0	0
1		000 0000 0000	1	0	x	x	x	x	x	x	x	x	x	x	x	x
127		000 0000 0001 - 000 0100 0000 000 0000 0000			*	*	*	*	*	*	*	*	*	*	*	*
1	JP 080	000 0000 0001	0	1	1	0	0	0	1	0	0	0	0	0	0	0
1		000 1000 0000	1	0	x	x	x	x	x	x	x	x	x	x	x	x
127		000 1000 0001 - 000 1100 0000 000 1000 0000			*	*	*	*	*	*	*	*	*	*	*	*
1	JP 100	000 1000 0001	0	1	1	0	0	1	0	0	0	0	0	0	0	0
1		001 0000 0000	1	0	x	x	x	x	x	x	x	x	x	x	x	x
127		001 0000 0001 - 001 0100 0000 001 0000 0000			*	*	*	*	*	*	*	*	*	*	*	*
1	JP 180	001 0000 0001	0	1	1	0	0	1	1	0	0	0	0	0	0	0
1		001 1000 0000	1	0	x	x	x	x	x	x	x	x	x	x	x	x
127		001 1000 0001 - 001 1100 0000 001 1000 0000			*	*	*	*	*	*	*	*	*	*	*	*
1	JP 200	001 0000 0001	0	1	1	0	1	0	0	0	0	0	0	0	0	0
1		010 0000 0000	1	0	x	x	x	x	x	x	x	x	x	x	x	x
127		010 0000 0001 - 010 0100 0000 010 0000 0000			*	*	*	*	*	*	*	*	*	*	*	*
1	JP 280	010 0000 0001	0	1	1	0	1	0	1	0	0	0	0	0	0	0
1		010 1000 0000	1	0	x	x	x	x	x	x	x	x	x	x	x	x
127		010 1000 0001 - 010 1100 0000 010 1000 0000			*	*	*	*	*	*	*	*	*	*	*	*
1	JP 300	010 0000 0001	0	1	1	0	1	1	0	0	0	0	0	0	0	0
1		011 0000 0000	1	0	x	x	x	x	x	x	x	x	x	x	x	x
127		011 0000 0001 - 011 0100 0000 011 0000 0000			*	*	*	*	*	*	*	*	*	*	*	*
1	JP 380	011 0000 0001	0	1	1	0	1	1	1	0	0	0	0	0	0	0
1		011 1000 0000	1	0	x	x	x	x	x	x	x	x	x	x	x	x
127		011 1000 0001 - 011 1100 0000 011 1000 0000			*	*	*	*	*	*	*	*	*	*	*	*

(Continued)

On-chip Clocks	Instruction	ROM Address	CH		I/O Pin											
			2	1	K1/	K2/	K3/	K4/	K5/	K6/	K7/	GP/	PD4	PD3	PD2	PD1
1	1→A11	011 1000 0001	0	1	0	1	0	0	0	0	0	0	0	0	0	1
1	JP 000	111 1000 0011			1	0	0	0	0	0	0	0	0	0	0	0
1		100 0000 0000	1	0	x	x	x	x	x	x	x	x	x	x	x	x
127		100 0000 0001 - 100 0100 0000 100 0000 0000			*	*	*	*	*	*	*	*	*	*	*	*
1	JP 080	100 0000 0001	0	1	1	0	0	0	1	0	0	0	0	0	0	0
1		100 1000 0000	1	0	x	x	x	x	x	x	x	x	x	x	x	x
127		100 1000 0001 - 100 1100 0000 100 1000 0000			*	*	*	*	*	*	*	*	*	*	*	*
1	JP 100	100 1000 0001	0	1	1	0	0	1	0	0	0	0	0	0	0	0
1		101 0000 0000	1	0	x	x	x	x	x	x	x	x	x	x	x	x
127		101 0000 0001 - 101 0100 0000 101 0000 0000			*	*	*	*	*	*	*	*	*	*	*	*
1	JP 180	101 0000 0001	0	1	1	0	0	1	1	0	0	0	0	0	0	0
1		101 1000 0000	1	0	x	x	x	x	x	x	x	x	x	x	x	x
127		101 1000 0001 - 101 1100 0000 101 1000 0000			*	*	*	*	*	*	*	*	*	*	*	*
1	JP 200	101 0000 0001	0	1	1	0	1	0	0	0	0	0	0	0	0	0
1		110 0000 0000	1	0	x	x	x	x	x	x	x	x	x	x	x	x
127		110 0000 0001 - 110 0100 0000 110 0000 0000			*	*	*	*	*	*	*	*	*	*	*	*
1	JP 280	110 0000 0001	0	1	1	0	1	0	1	0	0	0	0	0	0	0
1		110 1000 0000	1	0	x	x	x	x	x	x	x	x	x	x	x	x
127		110 1000 0001 - 110 1100 0000 110 1000 0000			*	*	*	*	*	*	*	*	*	*	*	*
1	JP 300	110 0000 0001	0	1	1	0	1	1	0	0	0	0	0	0	0	0
1		111 0000 0000	1	0	x	x	x	x	x	x	x	x	x	x	x	x
127		111 0000 0001 - 111 0100 0000 111 0000 0000			*	*	*	*	*	*	*	*	*	*	*	*
1	JP 380	111 0000 0001	0	1	1	0	1	1	1	0	0	0	0	0	0	0
1		111 1000 0000	1	0	x	x	x	x	x	x	x	x	x	x	x	x
127		111 1000 0001 - 111 1100 0000 111 1000 0000			*	*	*	*	*	*	*	*	*	*	*	*

This function was originally implemented for purely LSI test purpose, not for replacing on-chip ROM code by external ROM code generator (emulator).

Sequence of Entire Pattern ROM Code Dump (Abstract)

Pattern ROM addresses, PTN[7:1](A3[7:1]) & y[3:1](A4[7:5]), are specified by feeding ROM code (NN→A3, NN→A4) through I/O pins. Pattern ROM outputs are stored in 11 bit shift registers when CH6=1. The shift register contents shift every clock and output through "R" output pin, bit by bit serially. The serial output format is leading zeros first and bit positions "7-6-5-4-3-2-1-0" next. One word dump takes 11 clocks per address. Needs ((16 x 7) x 11 + 7) on-chip clocks.

Put a 10kΩ pull-up resistor to R output pin because of open-drain.

Beforehand, put prescaler initialization sequence (gray zone) below along with feeding AC/.

Feeding instruction codes and dumping ROM code must be synchronized with on-chip clock, not CLOCK (See "On-chip Prescaler").

CLOCK	On-chip clocks			AC/	Instruction	I/O Pin (ROM Code)												CH6	CH4	CH3	CH2	CH1	R (A4[4])	PRIO	MODE	Pattern Register		Out
	ϕ2	ϕ1	Cycle			K1/	K2/	K3/	K4/	K5/	K6/	K7/	GP/	PD4	PD3	PD2	PD1									PTN (A3[7:1])	y (A4[7:5])	R
1	0	1	x	0	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	x	x	x	xx	x	x
0																												
1																												
0																					1							
1																												
0																												
1																												
0																					0							
1	1	0	1	1																								
0	0																											
1																												
0		1																										
1																												
0	1	0	1	1	00→A1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1						
1	0																											
0																												
1		1																										
0																												
1	1	0	1	1	00→A2	0	1	1	0	1	0	0	0	0	0	0	0	0	0	1	0	1						
0	0																											
1																												
0		1																										
1		0																										

On-chip Clocks	Instruction	I/O Pin (ROM Code)												CH6	CH4	CH3	CH2	CH1	R (A4[4])	PRIO	MODE	Pattern Register		Out
		K1/	K2/	K3/	K4/	K5/	K6/	K7/	GP/	PD4	PD3	PD2	PD1									PTN (A3[7:1])	y (A4[7:5])	R
1	00→A1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	x	x	x	xx	x	x
	00→A2	0	1	1	0	1	0	0	0	0	0	0	0											
	00→A3	0	1	1	1	0	0	0	0	0	0	0	0											
	18→A4	0	1	1	1	1	0	0	1	1	0	0	0									00	3	000-7-6-5-4-3-2-1-0 1 bit serial MSB first
	40→L,H	0	1	0	1	1	1	0	0	0	0	0	0						1	0	0			
	A→MA	0	0	0	0	0	1	0	1	0	1	0	0											
	M→MODE, 3→L	0	0	1	1	1	0	0	0	1	0	1	1											
	NOP	0	0	0	0	0	0	0	0	0	0	0	0	1										
9	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0										
1	28→A4	0	1	1	1	1	0	1	0	1	0	0	0	0										000-7-6-5-4-3-2-1-0 1 bit serial MSB first
1	NOP	0	0	0	0	0	0	0	0	0	0	0	0	1									2	
9	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0										
1	38→A4	0	1	1	1	1	0	1	1	1	0	0	0	0										000-7-6-5-4-3-2-1-0 1 bit serial MSB first
1	NOP	0	0	0	0	0	0	0	0	0	0	0	0	1										
9	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0									3	
1	48→A4	0	1	1	1	1	1	0	0	1	0	0	0	0										000-7-6-5-4-3-2-1-0 1 bit serial MSB first
1	NOP	0	0	0	0	0	0	0	0	0	0	0	0	1										
9	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0									4	
1	58→A4	0	1	1	1	1	1	0	1	1	0	0	0	0										000-7-6-5-4-3-2-1-0 1 bit serial MSB first
1	NOP	0	0	0	0	0	0	0	0	0	0	0	0	1									5	
9	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0										
1	68→A4	0	1	1	1	1	1	1	0	1	0	0	0	0										000-7-6-5-4-3-2-1-0 1 bit serial MSB first
1	NOP	0	0	0	0	0	0	0	0	0	0	0	0	1									6	
9	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0										
1	78→A4	0	1	1	1	1	1	1	1	1	0	0	0	0										000-7-6-5-4-3-2-1-0 1 bit serial MSB first
1	NOP	0	0	0	0	0	0	0	0	0	0	0	0	1									7	
8	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0										
1	01→A3	0	1	1	1	0	0	0	0	0	0	0	1	0										000-7-6-5-4-3-2-1-0 1 bit serial MSB first
	18→A4	0	1	1	1	1	0	0	1	1	0	0	0	0								01	1	
	NOP	0	0	0	0	0	0	0	0	0	0	0	0	1										
9	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0										
1	28→A4	0	1	1	1	1	0	1	0	1	0	0	0	0										000-7-6-5-4-3-2-1-0 1 bit serial MSB first
1	NOP	0	0	0	0	0	0	0	0	0	0	0	0	1									2	
9	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0										
1	38→A4	0	1	1	1	1	0	1	1	1	0	0	0	0										000-7-6-5-4-3-2-1-0 1 bit serial MSB first
1	NOP	0	0	0	0	0	0	0	0	0	0	0	0	1									3	
9	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0										
1	48→A4	0	1	1	1	1	1	0	0	1	0	0	0	0										000-7-6-5-4-3-2-1-0 1 bit serial MSB first
1	NOP	0	0	0	0	0	0	0	0	0	0	0	0	1									4	
Repeat																								

Repeat from PTN[7:1]=01 & y=5 to PTN[7:1]=7E & y=4 and concatenate next page sequence.

(Continued)

Repeat																							
9	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0								5	000-7-6-5-4-3-2-1-0 1 bit serial MSB first
1	68→A4	0	1	1	1	1	1	1	0	1	0	0	0	0									
1	NOP	0	0	0	0	0	0	0	0	0	0	0	0	1								6	000-7-6-5-4-3-2-1-0 1 bit serial MSB first
9	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0									
1	78→A4	0	1	1	1	1	1	1	1	1	0	0	0	0							7E		
1	NOP	0	0	0	0	0	0	0	0	0	0	0	0	1								7	000-7-6-5-4-3-2-1-0 1 bit serial MSB first
8	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0									
1	7F→A3	0	1	1	1	0	1	1	1	1	1	1	1	0									
	18→A4	0	1	1	1	1	0	0	1	1	0	0	0	0									
	NOP	0	0	0	0	0	0	0	0	0	0	0	0	1								1	000-7-6-5-4-3-2-1-0 1 bit serial MSB first
9	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0									
1	28→A4	0	1	1	1	1	0	1	0	1	0	0	0	0									
1	NOP	0	0	0	0	0	0	0	0	0	0	0	0	1								2	000-7-6-5-4-3-2-1-0 1 bit serial MSB first
9	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0									
1	38→A4	0	1	1	1	1	0	1	1	1	0	0	0	0	0	1	0	1	1	0	0		
1	NOP	0	0	0	0	0	0	0	0	0	0	0	0	1								3	000-7-6-5-4-3-2-1-0 1 bit serial MSB first
9	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0									
1	48→A4	0	1	1	1	1	1	0	0	1	0	0	0	0									
1	NOP	0	0	0	0	0	0	0	0	0	0	0	0	1							7F	4	000-7-6-5-4-3-2-1-0 1 bit serial MSB first
9	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0									
1	58→A4	0	1	1	1	1	1	0	1	1	0	0	0	0									
1	NOP	0	0	0	0	0	0	0	0	0	0	0	0	1								5	000-7-6-5-4-3-2-1-0 1 bit serial MSB first
9	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0									
1	68→A4	0	1	1	1	1	1	1	0	1	0	0	0	0									
1	NOP	0	0	0	0	0	0	0	0	0	0	0	0	1								6	000-7-6-5-4-3-2-1-0 1 bit serial MSB first
9	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0									
1	78→A4	0	1	1	1	1	1	1	1	1	0	0	0	0									
1	NOP	0	0	0	0	0	0	0	0	0	0	0	0	1								7	000-7-6-5-4-3-2-1-0 1 bit serial MSB first
11	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0									

Concerning program ROM & pattern ROM dump, refer to the working ROM dump system approach below.

Under https://www.oguchi-rd.com/LSI%20products.php,

- System Design of ROM dump
- Dump Pattern Creation
  - Dump source pattern (PA.txt)
  - Source code of "rom_dump.cpp"
- Dump Analysis
  - Source code of "dump_analysis.cpp"

“PA.txt” (1,723KB) introduces the actual dump pattern for both program ROM and pattern ROM to be provided to μPD777s.

Example of Program ROM Dump Result (μPD777-010 for Epoch Video Cassette "Astro Command")

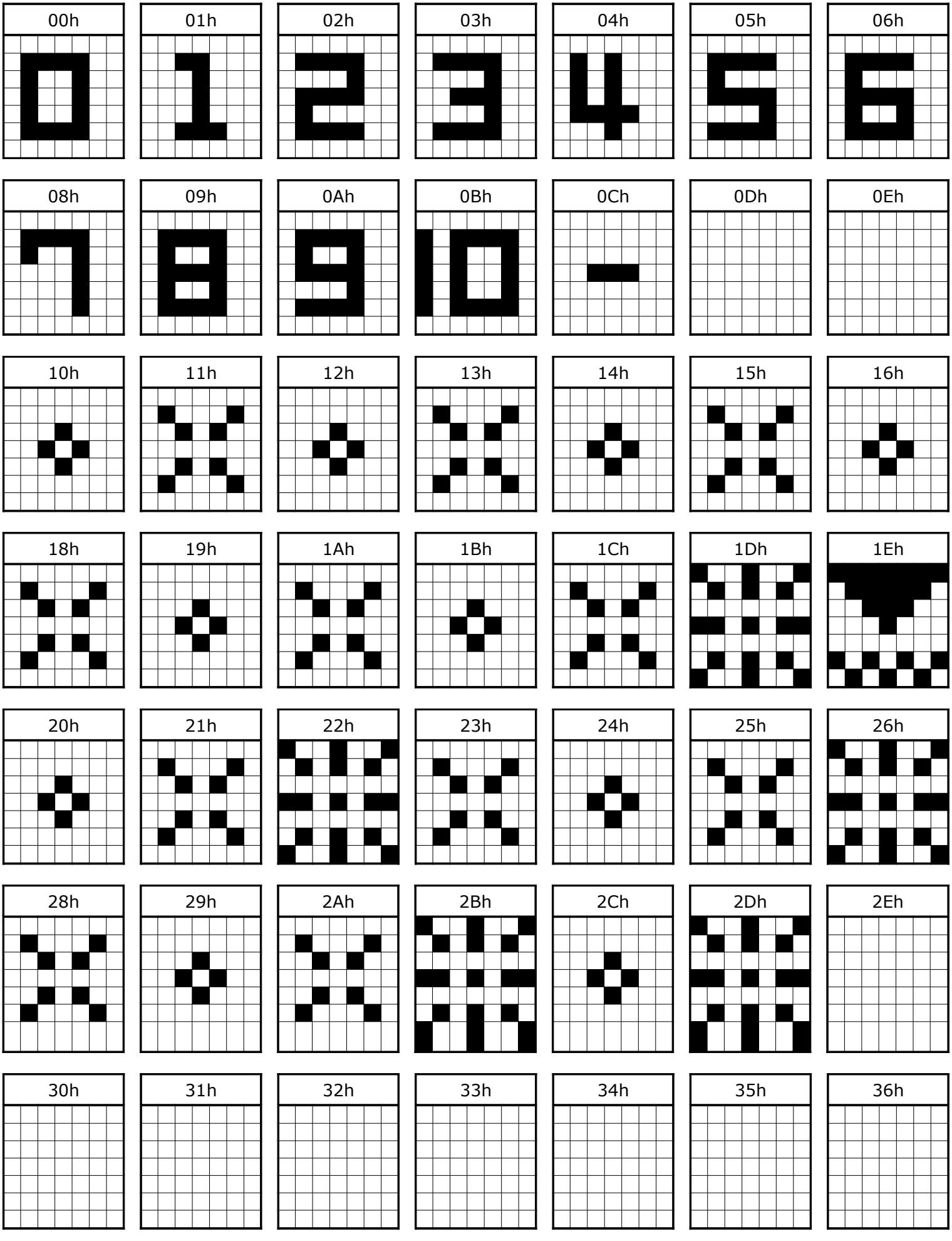
ADR	OUT	ADR	OUT	ADR	OUT	ADR	OUT	ADR	OUT	ADR	OUT	ADR	OUT	ADR	OUT	ADR	OUT	ADR	OUT	ADR	OUT	ADR	OUT	ADR	OUT	ADR	OUT	ADR	OUT
000	000	080	631	100	F1F	180	599	200	5BA	280	300	300	018	380	E21	400	D28	480	604	500	E61	580	EA6	600	B00	680	507	700	0E2	780	000
001	440	081	6D5	101	AA0	181	57E	201	6A0	281	101	301	301	381	020	401	510	481	D19	501	E3C	581	997	601	E15	681	3B3	701	880	781	000
003	59F	083	744	103	600	183	5BF	203	290	283	020	303	1A1	383	302	403	603	483	B1E	503	142	583	E76	603	086	683	612	703	E00	783	000
007	501	087	782	107	68D	187	57F	207	020	287	59A	307	08C	387	0C8	407	018	487	6B8	507	3AE	587	997	607	0FB	687	081	707	B7D	787	000
00F	388	08F	F7C	10F	E49	18F	04A	20F	060	28F	603	30F	B7A	38F	E58	40F	302	48F	34A	50F	08C	58F	018	60F	807	68F	614	70F	E87	78F	000
01F	389	09F	058	11F	020	19F	98F	21F	303	29F	A31	31F	67E	39F	E1F	41F	520	49F	8E5	51F	96E	59F	59E	61F	D28	69F	082	71F	681	79F	000
03F	600	0BF	599	13F	301	1BF	460	23F	08E	2BF	601	33F	680	3BF	F01	43F	5D8	4BF	E76	53F	E15	5BF	602	63F	933	6BF	618	73F	238	7BF	000
07E	680	0FE	390	17E	616	1FE	04A	27E	500	2FE	309	37E	70F	3FE	081	47E	63A	4FE	8B8	57E	084	5FE	280	67E	5D9	6FE	083	77E	B0E	7FE	000
07D	700	0FD	181	17D	28A	1FD	9F7	27D	142	2FD	5BD	37D	780	3FD	020	47D	E21	4FD	E61	57D	96E	5FD	9E9	67D	63C	6FD	620	77D	599	7FD	000
07B	780	0FB	39C	17B	020	1FB	9FE	27B	020	2FB	513	37B	B3D	3FB	5DC	47B	86F	4FB	5DC	57B	D28	5FB	5DC	67B	E5B	6FB	084	77B	640	7FB	000
077	018	0F7	743	177	3DE	1F7	599	277	5DC	2F7	63C	377	667	3F7	1C1	477	F1F	4F7	620	577	0CC	5F7	620	677	A51	6F7	630	777	0C0	7F7	000
06F	59F	0EF	78C	16F	5DE	1EF	102	26F	640	2EF	09D	36F	690	3EF	644	46F	C1C	4EF	282	56F	0D6	5EF	282	66F	E2E	6EF	085	76F	62A	7EF	000
05F	054	0DF	F7C	15F	38E	1DF	000	25F	E30	2DF	628	35F	77E	3DF	E6D	45F	917	4DF	8E7	55F	915	5DF	9E9	65F	666	6DF	620	75F	0A8	7DF	000
03E	481	0BE	599	13E	3D2	1BE	38C	23E	282	2BE	0AF	33E	784	3BE	020	43E	D28	4BE	5BF	53E	0CD	5BE	5BF	63E	E5B	6BE	086	73E	623	7BE	000
07C	85F	0FC	187	17C	382	1FC	598	27C	020	2FC	624	37C	5B8	3FC	89F	47C	821	4FC	281	57C	9A0	5FC	2A1	67C	A3D	6FC	618	77C	D28	7FC	000
079	029	0F9	39C	179	60F	1F9	2A8	279	060	2F9	0B2	379	481	3F9	E80	479	5DF	4F9	8B8	579	5DE	5F9	A95	679	E3C	6F9	087	779	301	7F9	000
073	029	0F3	711	173	3A2	1F3	9AE	273	018	2F3	628	373	081	3F3	38F	473	63C	4F3	8F5	573	60F	5F3	38D	673	144	6F3	614	773	2AE	7F3	000
067	029	0E7	742	167	5DE	1E7	481	267	59A	2E7	0B5	367	B1E	3E7	31B	467	E4E	4E7	D28	567	3A2	5E7	018	667	3AE	6E7	99D	767	142	7E7	000
04F	029	0CF	78E	14F	A2D	1CF	9F9	24F	61C	2CF	62D	34F	B79	3CF	686	44F	972	4CF	300	54F	141	5CF	301	64F	08C	6CF	603	74F	B1D	7CF	000
01E	638	09E	F7C	11E	5D8	19E	049	21E	E31	29E	0B8	31E	4C1	39E	333	41E	E2E	49E	102	51E	082	59E	0B5	61E	A75	69E	332	71E	59A	79E	000
03D	30A	0BD	5DD	13D	67E	1BD	99E	23D	018	2BD	628	33D	054	3BD	383	43D	000	4BD	0DC	53D	975	5BD	9EB	63D	018	6BD	326	73D	61C	7BD	000
07A	F45	0FA	602	17A	680	1FA	04A	27A	3A4	2FA	0BB	37A	301	3FA	162	47A	64A	4FA	80F	57A	D70	5FA	121	67A	A7D	6FA	326	77A	E31	7FA	000
075	F45	0F5	A02	175	70F	1F5	9F7	275	4C1	2F5	62D	375	481	3F5	020	475	6ED	4F5	D28	575	5DE	5F5	AAC	675	E15	6F5	326	775	318	7F5	000
06B	F45	0EB	5D0	16B	780	1EB	440	26B	020	2EB	0BE	36B	B5D	3EB	000	46B	E12	4EB	301	56B	607	5EB	682	66B	086	6EB	326	76B	682	7EB	000
057	028	0D7	644	157	054	1D7	020	257	018	2D7	630	357	5A0	3D7	5BB	457	972	4D7	096	557	2AA	5D7	230	657	A3D	6D7	326	757	334	7D7	000
02E	028	0AE	E21	12E	481	1AE	162	22E	3DC	2AE	0C1	32E	4C1	3AE	399	42E	0E4	4AE	1A1	52E	141	5AE	9F0	62E	D08	6AE	336	72E	68C	7AE	000
05C	5BA	0DC	8D9	15C	957	1DC	111	25C	018	2DC	62D	35C	058	3DC	020	45C	0EA	4DC	F7A	55C	08C	5DC	08E	65C	A51	6DC	000	75C	21C	7DC	000
038	516	0B8	E2E	138	020	1B8	9DD	238	3D2	2B8	0C4	338	481	3B8	E00	438	861	4B8	59E	538	1C1	5B8	182	638	59D	6B8	994	738	60A	7B8	000
070	59A	0F0	63C	170	E76	1F0	302	270	020	2F0	630	370	29D	3F0	FD7	470	87A	4F0	121	570	D03	5F0	684	670	081	6F0	5FD	770	D28	7F0	000
061	500	0E1	E4E	161	020	1E1	0E8	261	3DC	2E1	0C7	361	B37	3E1	6D5	461	0EA	4E1	301	561	5DC	5E1	230	661	A0D	6E1	53F	761	0E3	7E1	000
043	C0D	0C3	8D9	143	95D	1C3	0EF	243	5DA	2C3	636	343	05C	3C3	B7C	443	A85	4C3	1A1	543	65F	5C3	9B7	643	560	6C3	3B3	743	B53	7C3	000
006	441	086	E61	106	62D	186	99B	206	392	286	0CA	306	4C1	386	000	406	900	486	980	506	3A2	586	0DB	606	89A	686	0A0	706	300	786	000
00D	59F	08D	E3C	10D	695	18D	9F4	20D	920	28D	648	30D	05C	38D	5FE	40D	D80	48D	53B	50D	609	58D	9DD	60D	104	68D	A9C	70D	3AF	78D	000

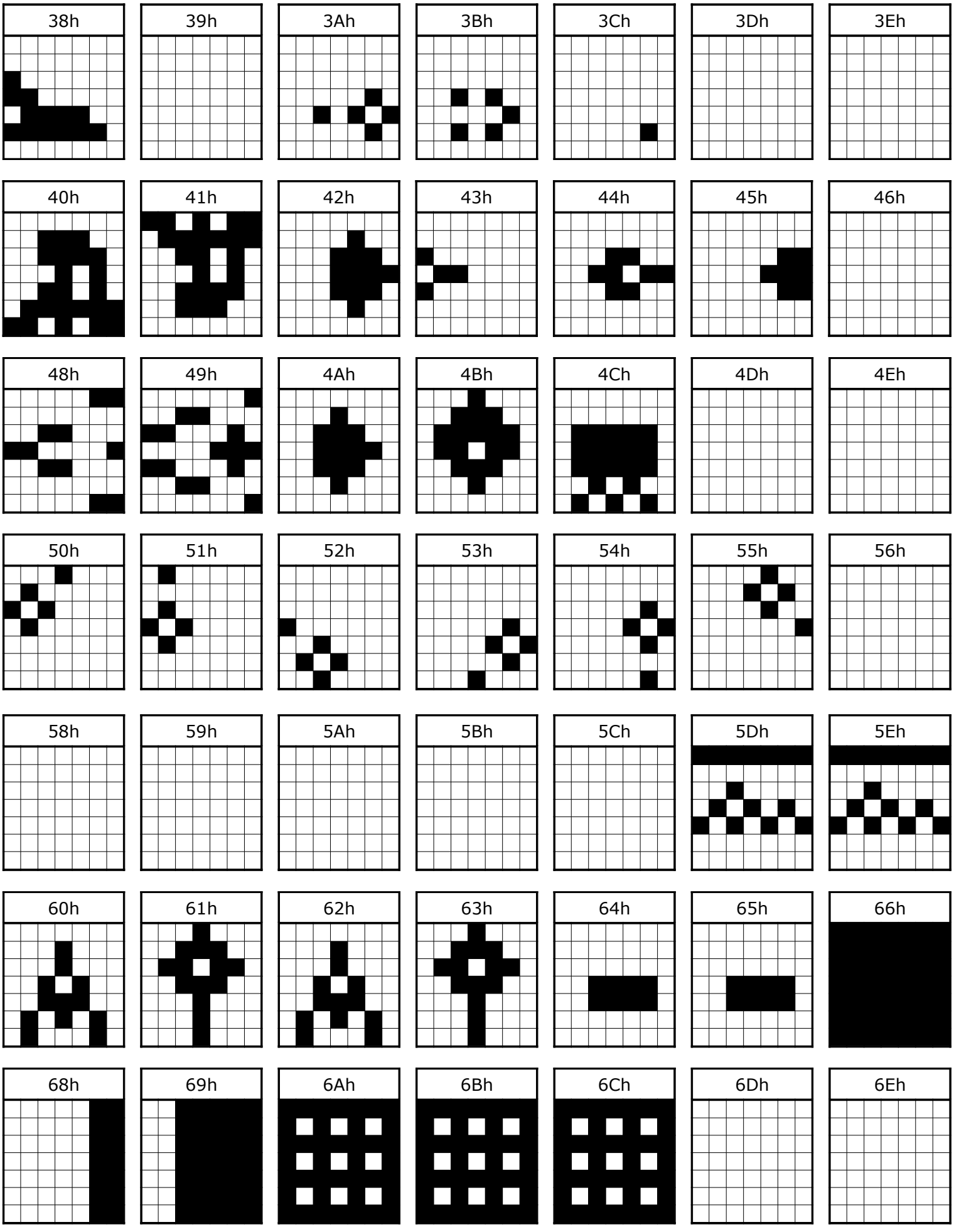
01B	57F	09B	E23	11B	738	19B	300	21B	59C	29B	0D3	31B	B2E	39B	0B6	41B	5FF	49B	5BA	51B	6CA	59B	102	61B	000	69B	38F	71B	B5D	79B	000
037	D1E	0B7	8D9	137	786	1B7	190	237	104	2B7	630	337	058	3B7	020	437	38F	4B7	640	537	930	5B7	018	637	5D9	6B7	53F	737	F1F	7B7	000
06E	C63	0EE	302	16E	F7C	1EE	9E7	26E	020	2EE	0F7	36E	020	3EE	5B2	46E	3A7	4EE	2A1	56E	018	5EE	9E9	66E	650	6EE	ACE	76E	A33	7EE	000
05D	F77	0DD	3AE	15D	058	1DD	008	25D	560	2DD	601	35D	058	3DD	081	45D	123	4DD	8AF	55D	302	5DD	018	65D	6D6	6DD	683	75D	164	7DD	000
03A	D35	0BA	08E	13A	301	1BA	9E7	23A	E73	2BA	308	33A	4C1	3BA	020	43A	301	4BA	ABC	53A	873	5BA	E00	63A	E12	6BA	238	73A	090	7BA	000
074	D35	0F4	0F0	174	395	1F4	303	274	5D9	2F4	D59	374	2BD	3F4	E87	474	449	4F4	150	574	87A	5F4	807	674	AD3	6F4	8A4	774	50A	7F4	000
069	600	0E9	8A6	169	1A6	1E9	150	269	64B	2E9	AC9	369	B2E	3E9	683	469	500	4E9	8AF	569	E3C	5E9	5BF	669	653	6E9	8CA	769	B1D	7E9	000
053	E00	0D3	8D9	153	395	1D3	008	253	E5B	2D3	020	353	2B9	3D3	238	453	441	4D3	601	553	1CC	5D3	640	653	28C	6D3	E87	753	300	7D3	000
026	620	0A6	5DC	126	73A	1A6	0EC	226	020	2A6	5BA	326	B2E	3A6	020	426	609	4A6	2AE	526	28F	5A6	2A0	626	188	6A6	683	726	3A4	7A6	000
04C	700	0CC	601	14C	B7C	1CC	9C1	24C	F01	2CC	640	34C	05C	3CC	39C	44C	2A0	4CC	8AF	54C	96E	5CC	98B	64C	144	6CC	238	74C	65A	7CC	000
018	780	098	2A2	118	5D8	198	0EB	218	AD4	298	281	318	481	398	359	418	84A	498	28A	518	3BF	598	2A0	618	141	698	A58	718	28E	798	000
031	AC5	0B1	8EC	131	63A	1B1	997	231	384	2B1	060	331	05C	3B1	620	431	608	4B1	8C1	531	085	5B1	997	631	8D5	6B1	A51	731	510	7B1	000
062	660	0E2	5D9	162	E21	1E2	0EA	262	3A0	2E2	020	362	B75	3E2	281	462	281	4E2	3AE	562	96E	5E2	3AC	662	59D	6E2	500	762	303	7E2	000
045	815	0C5	081	145	020	1C5	9AC	245	384	2C5	59B	345	5BF	3C5	365	445	288	4C5	5DC	545	D08	5C5	63F	645	081	6C5	997	745	1E8	7C5	000
00A	60A	08A	0FA	10A	E58	18A	300	20A	020	28A	E2F	30A	049	38A	599	40A	82F	48A	620	50A	972	58A	E6D	60A	A56	68A	000	70A	B1D	78A	000
015	688	095	8D6	115	E1F	195	0EF	215	303	295	862	315	B0A	395	38C	415	3AC	495	3AA	515	0E0	595	997	615	F00	695	38D	715	50E	795	000
02B	054	0AB	8EC	12B	018	1AB	500	22B	3BF	2AB	780	32B	008	3AB	5BB	42B	E00	4AB	5D8	52B	92C	5AB	018	62B	A6E	6AB	018	72B	B1D	7AB	000
056	E73	0D6	5FC	156	058	1D6	9E7	256	020	2D6	5FF	356	128	3D6	121	456	860	4D6	63A	556	945	5D6	058	656	E97	6D6	9EB	756	0E4	7D6	000
02C	658	0AC	170	12C	018	1AC	300	22C	060	2AC	BBB	32C	B2B	3AC	020	42C	624	4AC	E21	52C	A8E	5AC	018	62C	A6E	6AC	F7A	72C	0E6	7AC	000
058	054	0D8	650	158	300	1D8	0E0	258	601	2D8	5FE	358	020	3D8	3B5	458	688	4D8	8AF	558	6C5	5D8	054	658	5F9	6D8	018	758	B1D	7D8	000
030	E73	0B0	2AE	130	381	1B0	54A	230	302	2B0	522	330	600	3B0	68F	430	802	4B0	F1F	530	F6D	5B0	F7A	630	500	6B0	5BF	730	E00	7B0	000
060	600	0E0	8EC	160	391	1E0	9E7	260	282	2E0	EA6	360	68C	3E0	331	460	616	4E0	897	560	D28	5E0	302	660	664	6E0	38D	760	B02	7E0	000
041	6BB	0C1	604	141	1A6	1C1	300	241	A0B	2C1	020	341	E49	3C1	3A5	441	680	4C1	608	541	510	5C1	53C	641	E6D	6C1	018	741	B0E	7C1	000
002	74B	082	3AA	102	B7A	182	0E0	202	3AA	282	5FE	302	020	382	E87	402	391	482	2AA	502	018	582	EA1	602	A48	682	9EB	702	D28	782	000
005	E2F	085	E57	105	5D9	185	52E	205	020	285	501	305	301	385	5FF	405	381	485	1C1	505	D22	585	AE2	605	018	685	E61	705	601	785	000
00B	5DF	08B	520	10B	610	18B	9E7	20B	3AE	28B	020	30B	696	38B	0DC	40B	C37	48B	1C1	50B	972	58B	3A8	60B	5C0	68B	969	70B	301	78B	000
017	500	097	5DD	117	69E	197	300	217	020	297	604	317	29A	397	BD9	417	AA2	497	D03	517	5D9	597	D18	617	63C	697	308	717	282	797	000
02F	5B9	0AF	670	12F	E12	1AF	0E2	22F	054	2AF	280	32F	020	3AF	0C0	42F	300	4AF	5FE	52F	620	5AF	D27	62F	E4E	6AF	309	72F	B1D	7AF	000
05E	60F	0DE	3AA	15E	020	1DE	53C	25E	A75	2DE	E00	35E	601	3DE	BC2	45E	601	4DE	640	55E	6AD	5DE	D05	65E	A04	6DE	924	75E	280	7DE	000
03C	500	0BC	1C2	13C	E61	1BC	9E7	23C	058	2BC	AC7	33C	08B	3BC	098	43C	281	4BC	283	53C	E12	5BC	59E	63C	E61	6BC	5FE	73C	184	7BC	000
078	AD8	0F8	5FE	178	59E	1F8	D28	278	5D9	2F8	A57	378	B76	3F8	BA9	478	288	4F8	8B6	578	972	5F8	39C	678	E3C	6F8	0BC	778	102	7F8	000
071	000	0F1	53C	171	601	1F1	300	271	383	2F1	74C	371	39E	3F1	681	471	814	4F1	161	571	161	5F1	5DD	671	386	6F1	8AF	771	08C	7F1	000
063	607	0E3	5BE	163	280	1E3	09B	263	392	2E3	B90	363	088	3E3	21B	463	3AC	4E3	6BD	563	E1F	5E3	601	663	3AE	6E3	5DE	763	0D8	7E3	000
047	6C9	0C7	500	147	020	1C7	0CB	247	018	2C7	E57	347	36E	3C7	BE5	447	E00	4C7	29B	547	0AC	5C7	283	647	085	6C7	8F4	747	E58	7C7	000
00E	700	08E	5DC	10E	D28	18E	9BB	20E	058	28E	058	30E	312	38E	766	40E	86D	48E	980	50E	EE5	58E	A80	60E	A29	68E	601	70E	D28	78E	000
01D	78E	09D	620	11D	09D	19D	020	21D	018	29D	599	31D	6C0	39D	AAB	41D	656	49D	5BA	51D	92F	59D	309	61D	E23	69D	0E2	71D	F01	79D	000

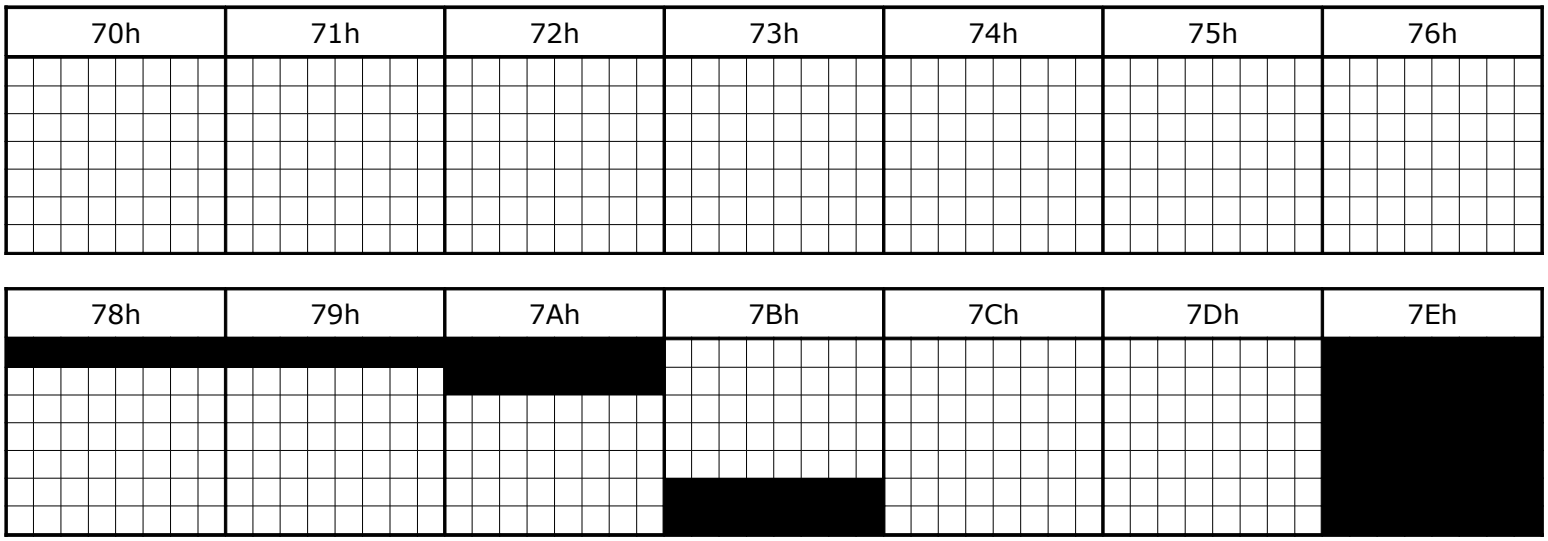
03B	F7C	0BB	2A2	13B	936	1BB	5DC	23B	020	2BB	39C	33B	B6D	3BB	F9A	43B	680	4BB	2AD	53B	5FC	5BB	5DD	63B	A29	6BB	603	73B	A33	7BB	000
076	6C4	0F6	8B6	176	141	1F6	602	276	638	2F6	3AC	376	6BA	3F6	600	476	836	4F6	80D	576	604	5F6	090	676	018	6F6	958	776	E15	7F6	000
06D	F7C	0ED	3AE	16D	E1F	1ED	A30	26D	5D9	2ED	08C	36D	5DF	3ED	769	46D	664	4ED	C11	56D	2A2	5ED	9D2	66D	510	6ED	EA6	76D	2AF	7ED	000
05B	6BF	0DB	D70	15B	90B	1DB	5DC	25B	481	2DB	0F4	35B	F4B	3DB	782	45B	688	4DB	980	55B	959	5DB	088	65B	D51	6DB	980	75B	807	7DB	000
036	F7C	0B6	F30	136	F1F	1B6	608	236	A59	2B6	AB2	336	29E	3B6	FD7	436	391	4B6	ED8	536	EA2	5B6	991	636	E77	6B6	C6A	736	A51	7B6	000
06C	6BA	0EC	D4D	16C	90B	1EC	2A2	26C	020	2EC	390	36C	B5B	3EC	BB8	46C	381	4EC	E87	56C	A97	5EC	084	66C	A13	6EC	980	76C	5DC	7EC	000
059	F7C	0D9	59E	159	000	1D9	020	259	28A	2D9	A57	359	088	3D9	74C	459	C37	4D9	683	559	2A2	5D9	ACF	659	668	6D9	601	759	608	7D9	000
032	707	0B2	601	132	D80	1B2	681	232	A5B	2B2	390	332	1C1	3B2	788	432	D06	4B2	238	532	952	5B2	950	632	E6D	6B2	280	732	2A2	7B2	000
064	78E	0E4	280	164	D80	1E4	291	264	060	2E4	B00	364	3A6	3E4	681	464	EBF	4E4	887	564	E76	5E4	680	664	A12	6E4	104	764	A51	7E4	000
049	5BA	0C9	9DB	149	5BD	1C9	9CB	249	5D9	2C9	E76	349	08A	3C9	21B	449	EBF	4C9	101	549	D06	5C9	5DD	649	F1F	6C9	182	749	CEB	7C9	000
012	612	092	5FA	112	121	192	142	212	302	292	AF7	312	B29	392	BE6	412	C6A	492	E87	512	5BE	592	3BD	612	5DC	692	086	712	5D1	792	000
025	38D	0A5	110	125	020	1A5	984	225	481	2A5	018	325	087	3A5	F9A	425	80D	4A5	ADD	525	640	5A5	500	625	608	6A5	0E0	725	640	7A5	000
04A	68F	0CA	020	14A	060	1CA	3B9	24A	A29	2CA	D18	34A	141	3CA	E80	44A	3A8	4CA	682	54A	3AB	5CA	601	64A	3AA	6CA	F1F	74A	E09	7CA	000
014	331	094	38F	114	5D8	194	508	214	020	294	5BD	314	A22	394	B8E	414	EA7	494	238	514	3AB	594	308	614	B49	694	302	714	A51	794	000
029	69C	0A9	69C	129	302	1A9	5FA	229	2AE	2A9	AF7	329	091	3A9	682	429	EA6	4A9	F6F	529	972	5A9	AE6	629	018	6A9	B25	729	E61	7A9	000
052	F7C	0D2	333	152	08D	1D2	500	252	A12	2D2	5BD	352	141	3D2	23B	452	917	4D2	AED	552	D06	5D2	5BD	652	0E5	6D2	058	752	5DC	7D2	000
024	302	0A4	327	124	942	1A4	642	224	29E	2A4	500	324	1AA	3A4	B9C	424	E76	4A4	5D9	524	D14	5A4	0A0	624	A04	6A4	018	724	604	7A4	000
048	384	0C8	327	148	481	1C8	E6D	248	A12	2C8	610	348	601	3C8	760	448	917	4C8	67E	548	C5D	5C8	0C0	648	665	6C8	599	748	283	7C8	000
010	380	090	383	110	929	190	020	210	060	290	919	310	0BC	390	78A	410	AD2	490	E5B	510	E87	590	9E8	610	E6D	690	380	710	B4E	790	000
021	696	0A1	5D9	121	020	1A1	54A	221	302	2A1	5DD	321	B5B	3A1	BE6	421	648	4A1	8A7	521	683	5A1	121	621	A04	6A1	59E	721	62F	7A1	000
042	70C	0C2	CD5	142	0B8	1C2	99C	242	2A8	2C2	603	342	5DF	3C2	681	442	6ED	4C2	300	542	218	5C2	640	642	A05	6C2	601	742	2AE	7C2	000
004	F7C	084	891	104	0BC	184	301	204	060	284	3A2	304	F4B	384	23B	404	E12	484	084	504	927	584	690	604	5F9	684	280	704	A51	784	000
009	701	089	5DC	109	F1F	189	640	209	481	289	128	309	303	389	AF1	409	917	489	8C8	509	F51	589	2A9	609	018	689	879	709	D28	789	000
013	E00	093	610	113	948	193	289	213	A21	293	020	313	164	393	74B	413	0E4	493	F1F	513	AED	593	9C9	613	668	693	5D9	713	E1F	793	000
027	702	0A7	282	127	5D8	1A7	020	227	020	2A7	5DD	327	090	3A7	680	427	0EA	4A7	F51	527	682	5A7	38F	627	E6D	6A7	821	727	B25	7A7	000
04E	A20	0CE	8E5	14E	63C	1CE	5D0	24E	4C1	2CE	601	34E	502	3CE	21B	44E	839	4CE	5DD	54E	238	5CE	503	64E	A12	6CE	3AF	74E	D28	7CE	000
01C	5BA	09C	D28	11C	E5B	19C	644	21C	2AA	29C	933	31C	301	39C	750	41C	821	49C	602	51C	F6F	59C	3A3	61C	058	69C	38F	71C	AD9	79C	000
039	1A1	0B9	F01	139	020	1B9	E21	239	060	2B9	F1F	339	0BC	3B9	F9A	439	E61	4B9	3AA	539	AED	5B9	620	639	018	6B9	681	739	402	7B9	000
072	68F	0F2	8C2	172	301	1F2	020	272	698	2F2	935	372	B04	3F2	BF9	472	0EA	4F2	AED	572	F8D	5F2	081	672	29D	6F2	2B3	772	403	7F2	000
065	291	0E5	602	165	121	1E5	A58	265	2FE	2E5	520	365	020	3E5	680	465	851	4E5	5BA	565	E87	5E5	634	665	D4D	6E5	602	765	030	7E5	000
04B	868	0CB	282	14B	0CA	1CB	E00	24B	A4E	2CB	018	34B	302	3CB	21B	44B	E3C	4CB	640	54B	683	5CB	082	64B	E76	6CB	31B	74B	028	7CB	000
016	630	096	8E8	116	AB9	196	9D1	216	020	296	5FB	316	4C1	396	B91	416	3AE	496	281	516	238	596	619	616	B6C	696	994	716	034	796	000
02D	3A1	0AD	D28	12D	F7A	1AD	59E	22D	3A2	2AD	168	32D	08B	3AD	76A	42D	08A	4AD	8D1	52D	A58	5AD	083	62D	018	6AD	5DD	72D	029	7AD	000
05A	5FC	0DA	300	15A	302	1DA	602	25A	640	2DA	020	35A	B16	3DA	085	45A	0FC	4DA	D14	55A	682	5DA	643	65A	E3C	6DA	67B	75A	038	7DA	000
034	164	0B4	8C4	134	94E	1B4	280	234	3AA	2B4	018	334	301	3B4	76C	434	811	4B4	C5D	534	238	5B4	994	634	602	6B4	3A2	734	028	7B4	000
068	5DD	0E8	D28	168	020	1E8	020	268	3D2	2E8	F1F	368	020	3E8	08C	468	83E	4E8	8D0	568	A38	5E8	520	668	B76	6E8	5D9	768	03C	7E8	000

051	670	0D1	300	151	D28	1D1	64A	251	020	2D1	060	351	600	3D1	76B	451	E3C	4D1	617	551	681	5D1	5DD	651	59B	6D1	A33	751	029	7D1	000
022	3AA	0A2	104	122	F1F	1A2	E6D	222	605	2A2	5BD	322	6D4	3A2	78C	422	1CC	4A2	E00	522	238	5A2	677	622	108	6A2	610	722	070	7A2	000
044	5FE	0C4	182	144	946	1C4	9A8	244	289	2C4	540	344	768	3C4	62E	444	28E	4C4	61F	544	923	5C4	3A1	644	8B8	6C4	308	744	028	7C4	000
008	500	088	8C2	108	018	188	303	208	020	288	6B8	308	78C	388	BD0	408	83E	488	288	508	E1B	588	9C2	608	548	688	309	708	074	788	000
011	600	091	5DC	111	302	191	08E	211	A4D	291	602	311	F7C	391	611	411	E15	491	101	511	A51	591	5BD	611	E00	691	D80	711	029	791	000
023	68C	0A3	610	123	510	1A3	502	223	303	2A3	2A1	323	610	3A3	B8C	423	085	4A3	67C	523	E76	5A3	082	623	A19	6A3	D80	723	078	7A3	000
046	E49	0C6	282	146	D35	1C6	126	246	397	2C6	6BC	346	764	3C6	601	446	0FB	4C6	3A0	546	A7E	5C6	0A0	646	59B	6C6	D80	746	028	7C6	000
00C	020	08C	9F8	10C	AD2	18C	F01	20C	3BF	28C	348	30C	788	38C	761	40C	833	48C	D14	50C	018	58C	501	60C	538	68C	601	70C	07C	78C	000
019	301	099	D28	119	5DD	199	6BD	219	086	299	301	319	FD0	399	78E	419	83E	499	C5D	519	5D9	599	121	619	AAD	699	308	719	029	799	000
033	616	0B3	301	133	3AA	1B3	299	233	020	2B3	601	333	656	3B3	BD0	433	D28	4B3	864	533	666	5B3	620	633	64C	6B3	80D	733	04A	7B3	000
066	289	0E6	0C8	166	020	1E6	9AA	266	060	2E6	281	366	765	3E6	659	466	6CC	4E6	302	566	E5B	5E6	688	666	6ED	6E6	5DC	766	451	7E6	000
04D	C6A	0CD	020	14D	D28	1CD	643	24D	0BC	2CD	308	34D	5FF	3CD	BD0	44D	29E	4CD	B38	54D	A7E	5CD	2A9	64D	E12	6CD	684	74D	004	7CD	000
01A	F1F	09A	5DC	11A	923	19A	E6D	21A	020	29A	34A	31A	FD0	39A	38C	41A	0EA	49A	E73	51A	E61	59A	9C9	61A	8BF	69A	2B2	71A	B51	79A	000
035	832	0B5	150	135	67E	1B5	020	235	656	2B5	D32	335	627	3B5	68F	435	807	4B5	59D	535	E3C	5B5	38F	635	0D0	6B5	AA8	735	445	7B5	000
06A	AD0	0EA	020	16A	690	1EA	548	26A	E00	2EA	A88	36A	6D3	3EA	333	46A	0CD	4EA	A0D	56A	144	5EA	39F	66A	0D6	6EA	59E	76A	500	7EA	000
055	68C	0D5	640	155	73F	1D5	B9F	255	664	2D5	601	355	740	3D5	687	455	6C5	4D5	0D6	555	3AE	5D5	501	655	A54	6D5	603	755	441	7D5	000
02A	E12	0AA	6C6	12A	780	1AA	642	22A	301	2AA	020	32A	788	3AA	337	42A	609	4AA	550	52A	088	5AA	283	62A	A33	6AA	6BB	72A	028	7AA	000
054	903	0D4	E12	154	B7C	1D4	2BD	254	289	2D4	302	354	F7C	3D4	327	454	0CD	4D4	018	554	A01	5D4	327	654	E61	6D4	282	754	449	7D4	000
028	301	0A8	020	128	5DA	1A8	648	228	121	2A8	A69	328	63B	3A8	327	428	603	4A8	A62	528	A1F	5A8	337	628	0E0	6A8	6B8	728	500	7A8	000
050	09D	0D0	E61	150	39E	1D0	5D8	250	020	2D0	5D9	350	741	3D0	6D6	450	0E0	4D0	59A	550	082	5D0	994	650	0E4	6D0	34A	750	441	7D0	000
020	903	0A0	060	120	3D2	1A0	B80	220	692	2A0	600	320	F7C	3A0	B7C	420	602	4A0	183	520	AF0	5A0	604	620	B56	6A0	80D	720	028	7A0	000
040	900	0C0	000	140	020	1C0	000	240	B7C	2C0	855	340	880	3C0	000	440	F6D	4C0	80D	540	9E4	5C0	958	640	B00	6C0	000	740	028	7C0	000

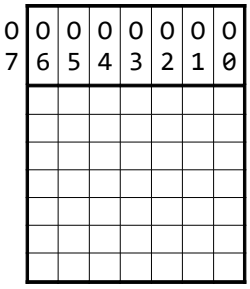
Example of Pattern ROM Dump Result (μPD777-010 for Epoch Video Cassette "Astro Command")



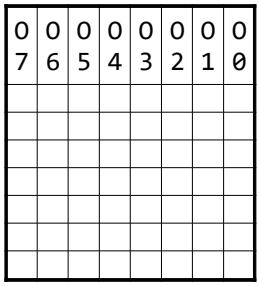

38h
39h
3Ah
3Bh
3Ch
3Dh
3Eh
40h
41h
42h
43h
44h
45h
46h
48h
49h
4Ah
4Bh
4Ch
4Dh
4Eh
50h
51h
52h
53h
54h
55h
56h
58h
59h
5Ah
5Bh
5Ch
5Dh
5Eh
60h
61h
62h
63h
64h
65h
66h
68h
69h
6Ah
6Bh
6Ch
6Dh
6Eh



7 x 7 Pattern Matrix (PTN[7:1] = 00h – 6Eh)



8 x 7 Pattern Matrix (PTN[7:1] = 70h – 7Eh)



NEC µPD777 Related PDF Files (Extracted)

- 777 Logic Schematics
- 777 Mask Layout
- 777 Die Micrograph
- µPD777 On-chip ROM Code Dump System Design
- 777 to YUV System Design
- Epoch Cassette Vision Cartridge
- Anatomy of Cassette Vision

Downloadable Files