

MỤC LỤC

CHƯƠNG 1: GIỚI THIỆU TẬP LỆNH TRONG NGÔN NGỮ VHDL	1
1.1 Những phần tử ngôn ngữ cơ bản :.....	1
1.1.1 Lời chú thích :	1
1.1.2 Những điều cần biết về ngôn ngữ VHDL :	1
1.1.3 Đối tượng dữ liệu :	1
1.1.4 Loại dữ liệu :	2
1.2 Toán tử dữ liệu :.....	4
1.3 Entity (thực thể):.....	6
1.4 Architecture (cấu trúc) :	6
1.4.1 Cú pháp cho dataflow model :	6
1.4.2 Cú pháp cho behavioral model :	7
1.4.3 Cú pháp của structural model :	7
1.5 Generic :	8
1.5.1 Cú pháp trong khai báo ENTITY :	8
1.5.2 Cú pháp trong khai báo component :	9
1.5.3 Cú pháp trong thuyết minh component :	9
1.6 Package (gói) :.....	10
1.6.1 Cú pháp khai báo PACKAGE:	10
1.6.2 Cú pháp khai báo thân chính Package:.....	11
1.7 Những câu lệnh đồng thời theo cấu trúc Dataflow :.....	12
1.7.1 Gán các tín hiệu đồng thời :	12
1.7.2 Gán tín hiệu có điều kiện :	12
1.7.3 Gán tín hiệu được chọn lựa :	13
1.7.4 Ví dụ cho kiểu dataflow :	13
1.8 Những câu lệnh tuần tự theo cấu trúc Behavioral :.....	14
1.8.1 Process :	14
1.8.2 Những phép gán tín hiệu tuần tự :	14
1.8.3 Phép gán biến :	15
1.8.4 Wait :	15
1.8.5 If then else :	15
1.8.6 Case:	16
1.8.7 Null :	16
1.8.8 For :	16
1.8.9 While :	17
1.8.10 Loop :	17
1.8.11 Exit :	17
1.8.12 Next :	17
1.8.13 Function (hàm) :	17
1.8.14 Procedure (thủ tục) :	19
1.8.15 Ví dụ về kiểu Behavioral :	20

1.9	Các câu lệnh kiểu Structural :	21
1.9.1	Khai báo Component :	21
1.9.2	Port map :	21
1.9.3	Open :	22
1.9.4	Generate :	22
1.9.5	Ví dụ về cách viết đoạn mã theo kiểu Structure:	23
1.10	Các thủ tục chuyển đổi :	25
1.10.1	Conv_integer () :	25
1.10.2	Conv_Std_Logic_Vector (,):	25
CHƯƠNG 2 : DÙNG NGÔN NGỮ VHDL MÔ TẢ CÁC MẠCH SỐ CƠ BẢN.....		27
2.1	Ngôn ngữ VHDL mô tả các cổng logic cơ bản:	27
2.1.1	Đoạn mã VHDL mô tả cổng NAND 2 ngõ vào:	27
2.1.2	Ngôn ngữ VHDL mô tả cổng NOR 3 ngõ vào:	28
2.1.3	Dùng ngôn ngữ VHDL mô tả một hệ thống báo động cho xe hơi:	29
2.2	Bộ giải mã LED 7 đoạn:	31
2.2.1	Xây dựng cấu trúc bộ giải mã LED 7 đoạn:	31
2.2.2	Ngôn ngữ VHDL mô tả mạch giải mã LED 7 đoạn:	34
2.2.3	Cấu trúc structural biểu diễn giải mã số thập phân ra Led 7 đoạn:	35
2.2.4	Cấu trúc dataflow biểu diễn giải mã số thập phân ra Led 7 đoạn:	37
2.2.5	Cấu trúc behavioral biểu diễn giải mã số thập phân ra Led 7 đoạn:	38
2.3	Bộ cộng:	38
2.3.1	Bộ cộng toàn phần (FA):	38
2.3.2	Bộ cộng toàn phần hai số nhị phân có nhiều hơn 1 bit:	40
2.3.3	Bộ cộng hai số nhị phân nhiều bit cho kết quả hiển thị nhanh:	41
2.4	Bộ trừ:	42
2.4.1	Bộ trừ một bit:	42
2.4.2	Sự tích hợp cả hai bộ cộng và bộ trừ trong cùng một mạch số:	43
2.5	Thành phần thực hiện các phép toán logic số học (ALU):	45
2.6	Bộ giải mã:	49
2.7	Bộ mã hóa:	52
2.8	2.8 Bộ ghép kênh:	53
2.9	Bộ đệm ba trạng thái:	57
2.10	Bộ so sánh:	58
2.11	Bộ dịch và bộ xoay (shifter / Rotator):	60
2.12	Bộ nhân:	62
2.13	Máy trạng thái hữu hạn FSM:	64
2.13.1	Mô hình máy trạng thái hữu hạn FSM (Finite-State-Machine):	65
2.13.2	Phương trình kích thích (Excitation Equation):	67
2.13.3	Phương trình trạng thái tiếp theo (Next-state Equation):	67
2.13.4	Bảng trạng thái tiếp theo (Next-state Table):	68

2.13.5	Ví dụ phân tích 1 Moore FSM:	70
2.13.6	Ví dụ phân tích Mealy FSM:	76
2.14	Các linh kiện tuần tự:	80
2.14.1	Các thanh ghi (Registers):	80
2.14.2	Thanh ghi tập tin (Register Files):	82
2.14.3	Bộ nhớ truy xuất ngẫu nhiên (Random Access Memory):	85
2.15	Bộ đếm (Counters):	88
2.15.1	Bộ đếm lên nhị phân (Binary Up Counter):	89
2.15.2	Mã VHDL cho bộ đếm lên 4 bit:	90
2.15.3	Bộ đếm lên xuống nhị phân (Binary Up-Down Counter):	91
2.15.4	Mã VHDL cho 1 bộ đếm lên xuống 4 bit như sau:	92
2.15.5	Bộ đếm lên xuống đọc song song :	93
2.15.6	Bộ đếm lên xuống BCD (BCD Up-Down Counter):	95
2.16	Thanh ghi dịch (Shift registers):	95
2.16.1	Thanh ghi dịch nối tiếp ra song song:	96
2.16.2	Thanh ghi dịch nối tiếp ra song song và song song ra nối tiếp:	97
CHƯƠNG 3 :	TÌM HIỂU KIT FPGA SPARTAN 3	100
3.1	Tổng quan kit FPGA Spartan 3 :	100
3.2	SRAM bất đồng bộ :	101
3.3	Led 7 đoạn:	105
3.4	Các công tắc trượt (SW), các nút ấn (PB) và các Led :	107
3.5	Cổng VGA :	107
3.6	Cổng PS/2 Mouse và Keyboard :	108
3.6.1	Bàn phím :	109
3.6.2	Mouse :	109
3.6.3	Nguồn cấp áp:	110
3.7	Cổng nối tiếp RS-232 :	110
3.8	Các nguồn xung clock :	111
3.9	Cách thiết lập các mode hoạt động cho FPGA :	111
3.10	Thiết lập cách lưu trữ cho Platform :	112
3.10.1	Default Option :	113
3.10.2	Flash Read option :	113
3.10.3	Disable Option :	114
3.11	Sự kết nối các board mở rộng vào kit Spartan 3 :	114
3.11.1	Port mở rộng A1:	115
3.11.2	Port mở rộng A2 :	116
3.11.3	Port mở rộng B1 :	117
CHƯƠNG 4 :	CÁC CỔNG GIAO TIẾP DÙNG TRÊN BOARD SPARTAN 3	119

4.1	Giao tiếp RS232 (cổng COM) :	119
4.2	Giao tiếp bàn phím PS/2 :	122
4.2.1	Sơ đồ chân kết nối:	122
4.2.2	Các tín hiệu của PS/2 :	122
4.2.3	Nguyên tắc truyền dữ liệu :	122
4.2.4	Mã quét bàn phím (Scancode) :	124
4.3	Giao tiếp VGA :	125
4.3.1	Sơ đồ chân kết nối :	125
4.3.2	Các tín hiệu của VGA :	125
4.3.3	Nguyên tắc tạo hình :	125
4.3.4	Nguyên tắc quét tín hiệu điện để tạo ảnh :	125
4.3.5	Một vài chuẩn Video điển hình cho TV và PC :	126
4.3.6	Giản đồ thời gian cho các tín hiệu của chuẩn VGA :	127
CHƯƠNG 5 : CÁC ỨNG DỤNG ĐÃ THỰC HIỆN		128
5.1	Đồng hồ và đếm sản phẩm :	129
5.2	Giao tiếp PS/2 :	129

MỤC LỤC HÌNH

Hình 2. 1 : Đoạn mã VHDL cho cổng NAND 2 ngõ vào.....	28
Hình 2. 2 : Cổng NOR 3 ngõ vào (a) đoạn mã VHDL; (b) sơ đồ mạch; (c) thời gian mô phỏng.	29
Hình 2. 3 : Giãn đồ xung của hệ thống báo động trong xe hơi: (a) Dạng xung trên lý thuyết; (b) Dạng xung trên thực tế.	30
Hình 2. 4 : Mạch báo động trong xe hơi (a) đoạn mã VHDL được viết dưới dạng dataflow; (b) mô phỏng giãn đồ xung.	31
Hình 2. 5 : Bảng chân trị của bộ giải mã 7 đoạn.	32
Hình 2. 6 : Mạch giải mã LED 7 đoạn.	34
Hình 2. 7 : Sơ đồ biểu diễn thời gian hiển thị một số trên Led 7 đoạn của một số thập phân tương ứng.	38
Hình 2. 8 : Bộ cộng toàn phần (a) bảng chân trị; (b) sơ đồ mạch; (c) ký hiệu logic.	39
Hình 2. 9 : Bộ cộng hai số nhị phân 8 bit.	40
Hình 2. 10 : (a) Mạch vận hành tín hiệu Carry-Lookahead từ c_1 đến c_4 ; (b) một mẫu bit của bộ cộng Carry-Lookahead.	42
Hình 2. 11 : Bộ trừ 1 bit (a) bảng chân trị; (b) sơ đồ mạch; (c) ký hiệu logic.	43
Hình 2. 12 : Mạch cộng và trừ chuỗi 8 bit nhị phân (a) bảng chân trị; (b) sơ đồ mạch; (c) ký hiệu logic.	44
Hình 2. 13 : Mạch ALU 4 bit.	46
Hình 2. 14 : Hoạt động của khối ALU (a) Bảng các trạng thái; (b) Bảng chân trị của LE; (c) Bảng chân trị của AE; (d) Bảng chân trị của CE.	47
Hình 2. 15 : Bìa karnaugh, biểu thức, sơ đồ mạch cho: (a) LE; (b) AE; (c) CE.	48
Hình 2. 16: Đoạn mã VHDL cho một khối ALU.	49
Hình 2. 17 : Dạng sóng mô phỏng cho 8 thuật toán cơ bản của khối ALU với hai giá trị ngõ vào là 5 và 3.	49
Hình 2. 18 : Một bộ giải mã 3 sang 8 (a) Bảng chân trị; (b) sơ đồ mạch; (c) ký hiệu logic.	50
Hình 2. 19 : Một bộ giải mã 3 sang 8 được xây dựng từ 7 bộ giải mã 1 sang 2.	51
Hình 2. 20 : Một bộ mã hóa 8 sang 3 (a) Bảng chân trị; (b) sơ đồ mạch; (c) ký hiệu logic.	52
Hình 2. 21 : Bảng chân trị cho một bộ mã hóa 8 sang 3 có sự ưu tiên.	53
Hình 2. 22 : Bộ ghép kênh từ 2 sang 1 (a) Bảng chân trị; (b) sơ đồ mạch; (c) ký hiệu logic.	54
Hình 2. 23 : Bộ ghép kênh 8 sang 1 (a) Bảng chân trị; (b) sơ đồ mạch; (c) ký hiệu logic.	54
Hình 2. 24 : Bộ ghép kênh 8 sang 1 có sử dụng (a) Bộ giải mã 3 sang 8; (b) 7 bộ ghép kênh 2 sang 1.	55
Hình 2. 25 : Dùng bộ ghép kênh 8 thành 1 biểu diễn hàm $F(x, y, z) = x'yz' + xy'z + xyz' + xyz$	57
Hình 2. 26 : Bộ đếm ba trạng thái (a) bảng chân trị; (b) ký hiệu logic; (c) bảng chân trị cho việc phân chia điều khiển cho mạch đếm ba trạng thái; (d) sơ đồ mạch.	58
Hình 2. 27 : Bộ so sánh 4 bit đơn giản cho (a) $X=3$; (b) $X \neq Y$; (c) $X < 5$	59
Hình 2. 28 : Bộ so sánh lặp (a) So sánh từng cặp bit x_i và y_i ; (b) 4-bit $X=Y$	60
Hình 2. 29 : Sự hoạt động của bộ dịch và bộ xoay.	60

Hình 2. 30 : Bộ dịch / bộ xoay 4 bit: (a) Bảng trạng thái hoạt động; (b) sơ đồ mạch; (c) ký hiệu logic.	61
Hình 2. 31 : Phép nhân (a) nhân bằng tay; (b) phương pháp thực hiện; (c) sơ đồ mạch....	63
Hình 2. 32 : Sơ đồ mạch của Moore FSM và Mealy FSM.	65
Hình 2. 33 : (a) Sơ đồ khối Moore FSM; (b) Sơ đồ khối Mealy FSM.....	67
Hình 2. 34 : Bảng trạng thái tiếp theo với 4 trạng thái và tín hiệu ngõ vào C.....	68
Hình 2. 35 : Bảng ngõ ra (a) Moore FSM; (b) Mealy FSM.	69
Hình 2. 36 : Sơ đồ các trạng thái trong một mạch tuần tự.....	70
Hình 2. 37 : Moore FSM đơn giản.....	71
Hình 2. 38 : Sơ đồ trạng thái đầy đủ của mạch Moore FSM.	73
Hình 2. 39 : Giãn đồ thời gian của Moore FSM mô phỏng bằng xilinx.	76
Hình 2. 40 : Mealy FSM đơn giản.	76
Hình 2. 41 : Bảng chân trị ngõ ra.	77
Hình 2. 42 : Trạng thái đầy đủ của Mealy FSM.	77
Hình 2. 43 : Tính toán thời gian mẫu cho Mealy FSM	78
Hình 2. 44 : Giãn đồ thời gian của Mealy FSM được mô phỏng bằng xilinx.	80
Hình 2. 45 : Thanh ghi 4 bit với mức xóa không đồng bộ.	81
Hình 2. 46 : Ký hiệu logic của thanh ghi.....	81
Hình 2. 47 : Giãn đồ mô phỏng cho thanh ghi 4 bit.	82
Hình 2. 48 : Mạch thanh ghi có thêm chân điều khiển.	83
Hình 2. 49 : Mạch hoàn chỉnh của thanh ghi 4x4.....	83
Hình 2. 50 : Tín hiệu mô phỏng cho ghi 4x4 với 1 Port ghi, 2 Port đọc.....	85
Hình 2. 51 : Ký hiệu logic của chip RAM.....	86
Hình 2. 52 : Mạch nhớ bit trong RAM.	86
Hình 2. 53 : Sơ đồ các ô nhớ dạng lưới trong chip RAM 4x4.	87
Hình 2. 54 : Bộ đếm lên nhị phân (a) Bảng chân trị; (b) Sơ đồ mạch; (c) Ký hiệu logic. ...	89
Hình 2. 55 : Bộ đếm lên 4 bit Sơ đồ mạch; bảng chân trị; ký hiệu logic.	90
Hình 2. 56 : Tín hiệu mô phỏng cho bộ đếm lên 4 bit.	91
Hình 2. 57 : Bộ cộng, trừ bán phần (a) Bảng chân trị; (b) Sơ đồ mạch; (c) Ký hiệu logic.	92
Hình 2. 58 : Bộ đếm lên xuống 4 bit: (a) Sơ đồ mạch; (b) Bảng chân trị; (c) Ký hiệu logic.	92
Hình 2. 59 : Tín hiệu mô phỏng cho bộ đếm lên xuống 4 bit.....	93
Hình 2. 60 : (a) Sơ đồ mạch đếm lên xuống 4 bit có sửa đổi ; (b) Bảng chân trị ; (c) ký hiệu logic của đếm lên xuống 4 bit có sửa đổi.	94
Hình 2. 61 : Bộ đếm BCD (a) bộ đếm lên; (b) bộ đếm xuống.	95
Hình 2. 62 : Bộ chuyển đổi 4 bit nối tiếp ra song song.	96
Hình 2. 63 : Tín hiệu mô phỏng của một bộ chuyển đổi 4 bit nối tiếp ra song song.	97
Hình 2. 64 : (a) Sơ đồ mạch thanh ghi dịch nối tiếp ra song song và song song ra nối tiếp; (b) Bảng chân trị ; (c) ký hiệu logic của thanh ghi dịch nối tiếp ra song song và song song ra nối tiếp.....	98
Hình 2. 65 : Tín hiệu mô phỏng thanh ghi dịch nối tiếp ra song song và song song ra nối tiếp.....	98
Hình 3. 1 : Sơ đồ khối kit Xilinx FPGA Spartan-3 Starter.	100
Hình 3. 2: Mạch in phía trước kit FPGA Xilinx Spartan-3 Starter.	101
Hình 3. 3 : Mạch in phía sau kit FPGA Xilinx Spartan-3 Starter.....	101

Hình 3. 4 : Sơ đồ kết nối giữa chân giữa FPGA và 2 SRAM 256Kx16.....	102
Hình 3. 5 : Bảng kết nối chân giữa FPGA với 18 đường địa chỉ của SRAM.....	103
Hình 3. 6 : Bảng kết nối chân giữa FPGA với chân OE và WE của.....	103
Hình 3. 7 : Bảng kết nối chân giữa IC10 với các chân của FPGA.....	104
Hình 3. 8 : Bảng kết nối chân giữa IC11 với các chân của FPGA.....	105
Hình 3. 9 : Sơ đồ bố trí các thanh của LED 7 đoạn.....	105
Hình 3. 10 : Bảng kết nối chân giữa LED 7 đoạn với chân của FPGA.....	106
Hình 3. 11 : Bảng kết nối tín hiệu điều khiển hiển thị 4 LED với chân của FPGA.....	106
Hình 3. 12 : Bảng hiển thị LED 7 đoạn tương ứng với 16 ký tự từ 0 đến F.....	106
Hình 3. 13 : Tín hiệu mô tả hiển thị các LED 7 đoạn bằng phương pháp quét led.....	107
Hình 3. 14 : Bảng kết nối chân giữa các công tắc trượt với các chân của FPGA.....	107
Hình 3. 15 : Bảng kết nối chân giữa các nút nhấn với các chân của FPGA.....	107
Hình 3. 16 : Bảng kết nối chân giữa 8 đèn LED với các chân của FPGA.....	107
Hình 3. 17 : Sơ đồ chân của cổng VGA.....	108
Hình 3. 18 : Bảng kết nối chân giữa các tín hiệu của cổng với các chân của FPGA.....	108
Hình 3. 19 : Bảng mã hóa hiển thị 3 bit cho 8 màu cơ bản.....	108
Hình 3. 20 : Sơ đồ chân của cổng PS/2.....	108
Hình 3. 21 : Mã quét bàn phím.....	109
Hình 3. 22 : Các mã điều khiển đặc biệt của bàn phím.....	109
Hình 3. 23 : Cấu trúc luồng bit quản lý cổng PS/2.....	110
Hình 3. 24 : Cách kết nối jumper trên board để chọn nguồn áp tùy người thiết kế.....	110
Hình 3. 25 : Sơ đồ chân của cổng RS-232.....	110
Hình 3. 26 : Sơ đồ kết nối chân giữa cổng RS-232 với các chân của FPGA.....	111
Hình 3. 27 : Kết nối chân giữa nguồn dao động xung clock với chân của FPGA.....	111
Hình 3. 28 : Bảng thiết lập các trạng thái hoạt động cho FPGA thông qua chân J8.....	112
Hình 3. 29 : Vị trí nút ấn để reset chương trình nạp cho kit và LED hiển thị.....	112
Hình 3. 30 : Sơ đồ kết nối jumper để lựa chọn các mode lưu trữ của FPGA.....	113
Hình 3. 31 : Sơ đồ kết nối chân giữa FPGA với Platform Flash ở chế độ Default.....	113
Hình 3. 32 : Sơ đồ kết nối chân giữa FPGA với Platform Flash ở chế độ Flash Read.....	114
Hình 3. 33 : Vị trí kết nối thêm các board mạch mở rộng trên board Spartan 3.....	114
Hình 3. 34 : Một số đặc tính của các port mở rộng A1, A2, B1.....	115
Hình 3. 35 : Cấu trúc chung của một port mở rộng.....	115
Hình 3. 36 : Bảng đồ chân kết nối giữa port mở rộng A1 với con FPGA spartan 3.....	116
Hình 3. 37 : Bảng đồ chân kết nối giữa port mở rộng A2 với con FPGA spartan 3.....	117
Hình 3. 38 : Bảng đồ chân kết nối giữa port mở rộng B1 với con FPGA spartan 3.....	118
Hình 4. 1 : Một áp dụng của RS-232.....	119
Hình 4. 2 : Các chân chức năng của DB25 và DB9 loại đầu đực.....	120
Hình 4. 3 : Các chân chức năng của DB25 và DB9 loại đầu cái.....	121
Hình 4. 4 : Nghi thức truyền và nhận dữ liệu giữa DTE và DCE.....	122
Hình 4. 5 : Chân kết nối của chuẩn PS/2 loại 5 chân và 6 chân.....	122
Hình 4. 6 : Thứ tự truyền data từ Keyboard đến Host.....	124
Hình 4. 7 : Thứ tự truyền data từ Host đến Keyboard.....	124
Hình 4. 8 : Mã Scancode của Keyboard.....	124
Hình 4. 9 : Chân kết nối của chuẩn VGA.....	125
Hình 4. 10 : Tín hiệu quét xen kẽ.....	126

<i>Hình 4. 11 : Tín hiệu quét liên tục</i>	126
<i>Hình 4. 12 : Thời gian thực hiện của tín hiệu Vertical Sync và Horizontal Sync.</i>	127
<i>Hình 4. 13 : Giảm đồ thời gian của tín hiệu Vertical Sync và Horizontal Sync.....</i>	127

CHƯƠNG 1 : GIỚI THIỆU TẬP LỆNH TRONG NGÔN NGỮ VHDL

VHDL là ngôn ngữ mô tả phần cứng cho các kiểu mạch số trong phạm vi các kết nối đơn giản của các cổng đến những hệ thống phức tạp. VHDL là viết tắt của VHSIC Hardware Description Language và VHSIC là viết tắt của Very High Speed Integrated Circuits. Trong chương này chỉ tóm tắt ngắn gọn nguyên lý cơ bản của VHDL và cú pháp của nó. Nhiều chức năng cao cấp của ngôn ngữ VHDL bị bỏ qua. Cho nên chúng ta cần phải tham khảo các tài liệu khác để có những cái nhìn chi tiết hơn.

1.1 Những phần tử ngôn ngữ cơ bản :

1.1.1 Lời chú thích :

Lời chú thích được chỉ ra sau hai dấu gạch nối liên tiếp (--) và được kết thúc ở cuối dòng.

Ví dụ : -- Đây là lời chỉ dẫn.

1.1.2 Những điều cần biết về ngôn ngữ VHDL :

Cú pháp nhận biết VHDL :

Một dãy của một hoặc nhiều ký tự viết hoa, ký tự thường, chữ số, đường gạch dưới .

Ký tự thường và ký tự hoa được xử lý như nhau.

Ký tự đầu tiên thường là một chữ cái.

Ký tự cuối cùng không thể là đường gạch dưới.

Không thể có 2 đường gạch dưới cùng một lúc.

1.1.3 Đối tượng dữ liệu :

Có 3 loại đối tượng dữ liệu : biến, hằng, tín hiệu.

Đối tượng dữ liệu tín hiệu đại diện cho tín hiệu logic trên đường dây trong mạch , một tín hiệu không có bộ nhớ do đó nếu nguồn tín hiệu bị mất thì tín hiệu không có giá trị.

Đối tượng dữ liệu biến nhớ nội dung của nó và dùng để tính toán trong mô hình hành vi.

Đối tượng dữ liệu hằng cần có 1 giá trị ban đầu khi khai báo và giá trị này không đổi.

Ví dụ : Signal x: bit;

 Variable y: integer;

 Constant one: STD_Logic_Vector {3 Downto 0} := "0001" ;

1.1.4 Loại dữ liệu :

1.1.4.1 Bit và Bit_vector :

Loại Bit và Bit_vector được xác định trước trong VHDL. Đối tượng của những loại này là giá trị '0' và '1'. Loại Bit_vector là một vector đơn giản của loại Bit. Một vector với tất cả các bit có cùng giá trị có thể được biểu diễn bằng từ khóa "others".

Ví dụ : Signal x: bit;
 Signal y: Bit_vector (7 downto 0);
 x <= '1';
 y <= "00000010";
 y <= (others => '0'); -- same as "00000000"

1.1.4.2 STD_Logic và STD_Logic_Vector :

Loại STD_Logic và STD_Logic_Vector cung cấp nhiều giá trị hơn loại Bit trong kiểu mạch thực chính xác hơn. Đối tượng của loại này có thể có những giá trị sau:

'0' -- mức 0
'1' -- mức 1
'Z' -- tổng trở cao
'-' -- không quan tâm
'L' -- mức 0 yếu
'H' -- mức 1 yếu
'U' -- không đặt giá trị ban đầu
'X' -- không xác định
'W' -- không xác định yếu

Loại STD_Logic và STD_Logic_Vector không được xác định trước vì thế phải khai báo 2 thư viện để sử dụng loại này:

```
LIBRARY IEEE;  
USE IEEE.STD_LOGIC_1164.ALL;
```

Nếu đối tượng loại STD_Logic_Vector được dùng như số nhị phân trong các thao tác số học, khi đó ta sử dụng lệnh "use" với hai cú pháp sau:

```
USE IEEE.STD_LOGIC_SIGNED.ALL; cho số có dấu.
```

USE IEEE.STD_LOGIC_UNSIGNED.ALL; cho số không dấu.

Một vector mà tất cả các bit có giá trị giống nhau có thể được biểu diễn ngắn gọn bằng cách sử dụng từ khóa “others” với cú pháp sau:

Ví dụ:

```

LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

SIGNAL x: STD_LOGIC;

SIGNAL y: STD_LOGIC_VECTOR(7 DOWNT0 0);

x <= 'Z';

y <= "00000001Z";

y <= (OTHERS => '0'); -- same as "00000000"
```

1.1.4.3 Integer :

Loại Integer được xác định trước để định nghĩa các đối tượng số nhị phân dùng với tính toán số học. Mặc định 1 tín hiệu khai báo Integer dùng tối đa 32 bit để chỉ một ký hiệu số. Integers cũng có thể dùng ít bit hơn với khai báo từ khóa RANGE.

Ví dụ :

```

SIGNAL x: INTEGER;

SIGNAL y: INTEGER RANGE -64 to 64;
```

1.1.4.4 Boolean :

Loại Boolean được xác định trước để định nghĩa các đối tượng chỉ có 2 giá trị TRUE hoặc FALSE

Ví dụ: SIGNAL x: BOOLEAN;

1.1.4.5 Bảng liệt kê Type :

Một bảng liệt kê cho phép người dùng chỉ rõ những giá trị mà đối tượng dữ liệu có thể có.

Cú pháp: **TYPE identifier IS (tr1 1, tr1 2, ...).**

Ví dụ: TYPE state_type IS (S1,S2,S3);

Signal state: state_type;

State <= S1;

1.1.4.6 Array :

Loại ARRAY nhóm các đối tượng dữ liệu riêng lẻ của cùng một loại thành một mảng một chiều hay nhiều chiều.

Cú pháp : **TYPE identifier IS ARRAY (range) OF type;**

Ví dụ : TYPE byte IS ARRAY(7 DOWNT0 0) OF BIT;
 TYPE memory_type IS ARRAY(1 TO 128) OF byte;
 SIGNAL memory: memory_type;
 memory(3) <= "00101101";

1.1.4.7 Subtype :

SUBTYPE là tập hợp con của một loại mà loại đó có sự ràng buộc về phạm vi.

Cú pháp : **SUBTYPE identifier IS type RANGE range;**

Ví dụ : SUBTYPE integer4 IS INTEGER RANGE -8 TO 7;
 SUBTYPE cell IS STD_LOGIC_VECTOR(3 DOWNT0 0);
 TYPE memArray IS ARRAY(0 TO 15) OF cell;

Một vài chuẩn Subtype

NATURAL – dãy số nguyên bắt đầu từ số 0.

POSITIVE – dãy số nguyên bắt đầu từ số 1.

1.2 Toán tử dữ liệu :

VHDL được xây dựng từ các toán tử được giới thiệu ở bảng dưới đây:

Toán tử Logic	Toán tử	Ví dụ
AND	And	a AND b
OR	Or	a OR b
NOT	Not	NOT a
NAND	Nand	a NAND b
NOR	Nor	a NOR b
XOR	Xor	a XOR b
XNOR	Xnor	a XNOR b
Toán tử số học (Arithmetic Operators)		

+	Phép cộng (addition)	$a + b$
-	Phép trừ (subtraction)	$a - b$
*	Phép nhân (multiplication (integer or floating point))	$a * b$
/	Phép chia (division (integer or floating point))	a / b
MOD	Lấy phần dư, dấu theo b (modulus (integer))	$a \text{ MOD } b$
REM	Lấy phần dư, dấu theo a (remainder (integer))	$a \text{ REM } b$
**	Lũy thừa (exponentiation)	$A ** 2$
&	Phép nối (concatenation)	'a' & 'b'
ABS	Trị tuyệt đối (absolute)	$a \text{ ABS } b$
Toán tử quan hệ (Relational Operators)		
=	Bằng	
/=	Không bằng	
<	Nhỏ hơn	
<=	Nhỏ hơn hoặc bằng	
>	Lớn hơn	
>=	Lớn hơn hoặc bằng	
Toán tử dịch (Shift Operators)		
sll	Dịch trái logic (shift left logical)	
srl	Dịch phải logic (shift right logical)	
sla	Dịch trái số học (shift left arithmetic)	
sra	Dịch phải số học (shift right arithmetic)	

rol	Xoay trái (rotate left)	
ror	Xoay phải (rotate right)	

1.3 Entity (thực thể):

Một khai báo ENTITY biểu thị một giao diện người dùng hoặc bên ngoài của modul giống với khai báo của một chức năng. Nó chỉ rõ tên của thực thể và giao diện của nó. Giao diện gồm có những tín hiệu vào và ra thực thể sử dụng từ khóa đại diện là IN và OUT .

Cú pháp : *ENTITY entity-name IS*
 PORT (list-of-port-names-and-types);
 END entity-name;

Ví dụ : LIBRARY IEEE;

```
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY Siren IS PORT(
    M: IN STD_LOGIC;
    D: IN STD_LOGIC;
    V: IN STD_LOGIC;
    S: OUT STD_LOGIC);

END Siren;
```

1.4 Architecture (cấu trúc) :

Thân ARCHITECTURE định nghĩa sự thực thi hiện thời của các chức năng của một ENTITY. Điều này giống với sự xác định hoặc sự thực thi của một chức năng. Cú pháp cho ARCHITECTURE khác nhau tùy thuộc vào mô hình (**dataflow, behavioral, or structural**) mà bạn sử dụng.

1.4.1 Cú pháp cho dataflow model :

```
ARCHITECTURE architecture-name OF entity-name IS  

    signal-declarations; -- khai báo tín hiệu  

BEGIN  

    concurrent-statements;
```

END architecture-name;

Những phát biểu concurrent được thực hiện một cách đồng thời.

Ví dụ ARCHITECTURE Siren_Dataflow OF Siren IS

 SIGNAL term_1: STD_LOGIC;

 BEGIN

 term_1 <= D OR V;

 S <= term_1 AND M;

 END Siren_Dataflow;

1.4.2 Cú pháp cho behavioral model :

ARCHITECTURE architecture-name **OF** entity-name **IS**

signal-declarations;

function-definitions;

procedure-definitions;

BEGIN

PROCESS-blocks;

 concurrent-statements;

END architecture-name;

Những câu lệnh bên trong process-block được thực hiện tuần tự, liên tục. Tuy nhiên chính process-block là concurrent-statements.

Ví dụ : ARCHITECTURE Siren_Behavioral OF Siren IS

 SIGNAL term_1: STD_LOGIC;

 BEGIN

 PROCESS (D, V, M)

 BEGIN

 term_1 <= D OR V;

 S <= term_1 AND M;

 END PROCESS;

 END Siren_Behavioral;

1.4.3 Cú pháp của structural model :

ARCHITECTURE architecture-name **OF** entity-name **IS**

component-declarations;

signal-declarations;

BEGIN

instance-name: **PORT MAP**-statements;

concurrent-statements;

END architecture-name;

Cho mỗi thành phần khai báo sử dụng cần có một kiến trúc hay một thực thể phù hợp cho các thành phần đó. Câu lệnh **PORT MAP** là câu lệnh đồng thời.

Ví dụ : **ARCHITECTURE** Siren_Structural **OF** Siren **IS**

COMPONENT myOR PORT (

in1, in2: IN STD_LOGIC;

out1: OUT STD_LOGIC);

END COMPONENT;

SIGNAL term1: STD_LOGIC;

BEGIN

U0: myOR PORT MAP (D, V, term1);

S <= term1 AND M;

END Siren_Structural;

1.5 Generic :

GENERIC cho phép thông tin đi qua ENTITY, ví dụ kích thước của Vector trong danh sách PORT sẽ không được biết cho đến khi thời gian chính xác. GENERIC của một ENTITY được thể hiện khi dùng từ khóa **GENERIC** trước danh sách PORT khai báo trong ENTITY. Một bộ nhận dạng được khai báo như GENERIC là một hằng và chỉ có thể được đọc. Bộ nhận dạng sau đó có thể được dùng trong khai báo ENTITY và những kiến trúc phù hợp của nó ở mọi nơi hằng số được đòi hỏi.

1.5.1 Cú pháp trong khai báo ENTITY :

ENTITY entity-name IS

GENERIC (identifier: type); -- with no default value

...

or

ENTITY entity-name IS

GENERIC (identifier: type := constant); -- with a default value given by the constant

...

Ví dụ ENTITY Adder IS

-- declares the generic identifier n having a default value 4

GENERIC (n: INTEGER := 4);

PORT (

-- the vector size is 3 downto 0 since n is 4

A, B: IN STD_LOGIC_VECTOR(n-1 DOWNT0 0);

Cout: OUT STD_LOGIC;

SUM: OUT STD_LOGIC_VECTOR(n-1 DOWNT0 0));

S: OUT STD_LOGIC);

END Siren;

Giá trị cho một GENERIC hằng cũng có thể được đề cập trong một câu lệnh khai báo Component hoặc một câu lệnh thuyết minh Component.

1.5.2 Cú pháp trong khai báo component :

COMPONENT component-name

GENERIC (identifier: type := constant);

-- with an optional value given by the constant

PORT (list-of-port-names-and-types);

END COMPONENT;

1.5.3 Cú pháp trong thuyết minh component :

label: component-name GENERIC MAP (constant) PORT MAP (association-list);

Ví dụ: ARCHITECTURE ...

COMPONENT mux2 IS

-- declares the generic identifier n having a default value 4

GENERIC (n: INTEGER := 4);

```

PORT (S: IN STD_LOGIC; -- select line
      D1, D0: IN STD_LOGIC_VECTOR(n-1 DOWNT0 0);-- data bus input
      Y: OUT STD_LOGIC_VECTOR(n-1 DOWNT0 0)); -- data bus output
END COMPONENT;

...

BEGIN

U0: mux2 GENERIC MAP (8) PORT MAP (mux_select, A, B, mux_out);

...

```

1.6 Package (gói) :

Một package cung cấp cơ chế để nhóm lại với nhau và chia sẻ khai báo mà được dùng cho một vài ENTITY. Chính một gói đó bao hàm cả một sự khai báo, tùy chọn, một thân chính. Khai báo gói và thân chính được lưu trữ cùng nhau trong một file riêng biệt từ phần còn lại của những đơn vị thiết kế. Tên file đưa cho file này cần giống tên package. Để hoàn thành thiết kế kết hợp chính xác nên dùng MAX+PLUS II. Trước tiên bạn cần kết hợp Package như một đơn vị riêng biệt. Sau đó bạn có thể kết hợp đơn vị mà dùng Package đó.

Khai báo Package và Body:

Khai báo PACKAGE chứa các khai báo có thể chia sẻ giữa các đơn vị ENTITY. Nó cung cấp giao diện mà các linh kiện có thể thấy trong đơn vị ENTITY khác. Tùy chọn PACKAGE BODY chứa đựng sự thực thi của các chức năng và các thủ tục được khai báo trong PACKAGE.

1.6.1 Cú pháp khai báo PACKAGE:

```

PACKAGE package-name IS

    type-declarations;

    subtype-declarations;

    signal-declarations;

    variable-declarations;

    constant-declarations;

    component-declarations;

    function-declarations;

    procedure-declarations;

END package-name;

```

1.6.2 **Cú pháp khai báo thân chính Package:**

PACKAGE BODY package-name IS

function-definitions; -- for functions declared in the package declaration

procedure-definitions; -- for procedures declared in the package declaration

END package-name;

Ví dụ LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

PACKAGE my_package IS

SUBTYPE bit4 IS STD_LOGIC_VECTOR(3 DOWNT0 0);

FUNCTION Shiftright (input: IN bit4) RETURN bit4; -- declare a function

SIGNAL mysignal: bit4; -- a global signal

END my_package;

PACKAGE BODY my_package IS

-- implementation of the Shiftright function

FUNCTION Shiftright (input: IN bit4) RETURN bit4 IS

BEGIN

RETURN '0' & input(3 DOWNT0 1);

END shiftright;

END my_package;

Để sử dụng PACKAGE, bạn chỉ đơn giản dùng một LIBRARY và câu lệnh USE cho Package đó. Trước khi kết hợp Modul dùng Package, trước tiên bạn cần kết hợp chính Package như một ENTITY cấp cao.

Cú pháp : **LIBRARY WORK;**

USE WORK.package-name.ALL;

Ví dụ : LIBRARY WORK;

USE WORK.my_package.ALL;

ENTITY test_package IS PORT (x: IN bit4; z: OUT bit4);

END test_package;

```

ARCHITECTURE Behavioral OF test_package IS
BEGIN
    mysignal <= x;
    z <= Shiftright(mysignal);
END Behavioral;

```

1.7 Những câu lệnh đồng thời theo cấu trúc Dataflow :

Phát biểu Concurrent sử dụng cho mô hình Dataflow được thi hành một cách đồng thời. Do đó thứ tự các phát biểu này không có ảnh hưởng ở kết quả ngõ ra.

1.7.1 Gán các tín hiệu đồng thời :

Gán một giá trị hoặc kết quả của ước lượng một biểu thức cho tín hiệu. Phát biểu này được thực thi khi nào tín hiệu trong biểu thức đó thay đổi giá trị. Tuy nhiên việc gán thực sự giá trị cho tín hiệu diễn ra sau thời gian trễ nào đó và không tức thời như những phép gán biến. Biểu thức có thể là các biểu thức logic hoặc số học.

Cú pháp : *signal <= expression;*

Ví dụ : *y <= '1';*

z <= y AND (NOT x);

Một vector mà tất cả bit có cùng giá trị có thể dùng từ khóa OTHERS như dưới đây:

SIGNAL x: STD_LOGIC_VECTOR(7 DOWNT0 0);

x <= (OTHERS => '0'); -- 8-bit vector of 0, same as "00000000"

1.7.2 Gán tín hiệu có điều kiện :

Chọn một hoặc vài giá trị khác nhau để gán cho tín hiệu dựa trên điều kiện khác nhau. Câu lệnh sẽ thực thi khi 1 số giá trị hay điều kiện thay đổi trong tín hiệu

Cú pháp : *signal <= value1 WHEN condition ELSE*

value2 WHEN condition ELSE

value3;

Ví dụ : *z <= in0 WHEN sel = "00" ELSE*

in1 WHEN sel = "01" ELSE

in2 WHEN sel = "10" ELSE

in3;

1.7.3 Gán tín hiệu được chọn lựa :

Chọn một hoặc vài giá trị khác nhau để gán cho tín hiệu dựa trên giá trị của biểu thức được chọn. Tất cả các trường hợp có thể có của biểu thức cần được đưa ra. Từ khóa OTHERS có thể dùng để chỉ rõ những trường hợp còn lại. Câu lệnh sẽ thực thi khi tín hiệu trong biểu thức hoặc một giá trị thay đổi.

Cú pháp : *WITH expression SELECT*

```

signal <= value1 WHEN choice1,
           value2 WHEN choice2 | choice3,
           ...
           value4 WHEN OTHERS;
```

Trong cú pháp ở trên nếu biểu thức bằng trường hợp 1 thì value1 được gán cho signal. Vì thế nếu biểu thức bằng trường hợp 2 hoặc trường hợp 3 thì value2 được gán cho tín hiệu. Nếu biểu thức không có trong các trường hợp trên thì value4 trong tùy chọn WHEN OTHERS được gán cho tín hiệu.

Ví dụ : WITH sel SELECT

```

z <=  in0 WHEN "00",
      in1 WHEN "01",
      in2 WHEN "10",
      in3 WHEN OTHERS;
```

1.7.4 Ví dụ cho kiểu dataflow :

-- outputs 1 if the 4-bit input is a prime number, 0 otherwise

LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

ENTITY Prime IS PORT (

number: IN STD_LOGIC_VECTOR(3 DOWNTO 0);

yes: OUT STD_LOGIC);

END Prime;

ARCHITECTURE Prime_Dataflow OF Prime IS

BEGIN

WITH number SELECT

```
yes <= '1' WHEN "0001" | "0010",
      '1' WHEN "0011" | "0101" | "0111" | "1011" | "1101",
      '0' WHEN OTHERS;
```

```
END Prime_Dataflow;
```

1.8 Những câu lệnh tuần tự theo cấu trúc Behavioral :

Mô hình behavioral cho phép những phát biểu thực thi liên tục giống như một chương trình máy tính thông thường. Phát biểu Sequential statements gồm nhiều chuẩn xây dựng như: gán biến, if – then – else, các vòng lặp.

1.8.1 Process :

Khối PROCESS chứa những phát biểu được thực thi tuần tự. Tuy nhiên chính phát biểu PROCESS là một concurrent statements (phát biểu đồng thời). Khối nhiều PROCESS trong 1 kiến trúc sẽ thực thi một cách đồng thời . Các khối xử lý này kết hợp với nhau thành concurrent statements khác.

Cú pháp : process-name: PROCESS (sensitivity-list)

 variable-declarations;

 BEGIN

 sequential-statements;

 END PROCESS process-name;

Danh sách tín hiệu nhạy được tách biệt bởi dấu phẩy (,) mà nó xử lý. Những từ khác, mỗi khi tín hiệu trong danh sách thay đổi giá trị , việc xử lý được thực thi tất cả phát biểu tuần tự theo danh sách. Sau khi phát biểu cuối cùng được thực thi , việc xử lý sẽ hoãn lại cho đến thời gian tiếp theo khi một tín hiệu trong danh sách thay đổi giá trị trước khi thực thi lần nữa.

Ví dụ: PROCESS (D, V, M)

 BEGIN

 term_1 <= D OR V;

 S <= term_1 AND M;

 END PROCESS;

1.8.2 Những phép gán tín hiệu tuần tự :

Gán giá trị cho một tín hiệu. Phát biểu này giống như bản sao của concurrent ngoại trừ nó được thực thi một cách tuần tự chỉ khi nào sự thực thi tiến đến nó.

Cú pháp: *signal* <= *expression*;

Ví dụ : *y* <= '1';

z <= *y* AND (NOT *x*);

1.8.3 Phép gán biến :

Gán 1 giá trị hoặc kết quả ước lượng của 1 biểu thức đến 1 biến. Giá trị này luôn gán cho biến ngay lập tức khi mà phát biểu này thực thi. Biến này chỉ biểu thị bên trong không xử lý (PROCESS).

Cú pháp: *signal* := *expression*;

Ví dụ : *y* := '1';

yn := NOT *y*;

1.8.4 Wait :

Khi 1 Process có danh sách nhảy, process luôn trì hoãn sau khi thực thi phát biểu trước đó. Một khả năng để sử dụng danh sách nhảy để trì hoãn Process là dùng phát biểu WAIT. Nó cần được phát biểu trước tiên trong PROCESS.

Cú pháp : WAIT UNTIL *condition*;

Ví dụ : -- suspend until a rising clock edge

 WAIT UNTIL *clock*'EVENT AND *clock* = '1';

1.8.5 If then else :

Cú pháp:

IF *condition* THEN

sequential-statements1;

ELSE

sequential-statements2;

END IF;

IF *condition1* THEN

sequential-statements1;

ELSIF *condition2* THEN

sequential-statements2;

ELSE

 sequential-statements3;

END IF;

Ví dụ: IF count /= 10 THEN -- not equal

 count := count + 1;

ELSE

 count := 0;

END IF;

1.8.6 Case:

Cú pháp:

CASE expression IS

WHEN choices => sequential-statements;

WHEN choices => sequential-statements;

 ...

WHEN OTHERS => sequential-statements;

END CASE;

Ví dụ: CASE sel IS

 WHEN "00" => z <= in0;

 WHEN "01" => z <= in1;

 WHEN "10" => z <= in2;

 WHEN OTHERS => z <= in3;

END CASE;

1.8.7 Null :

Phát biểu NULL không làm gì cả.

Cú pháp: NULL;

1.8.8 For :

Cú pháp : FOR identifier IN start [TO | DOWNTO] stop LOOP

 sequential-statements;

END LOOP;

Phát biểu LOOP cần giới hạn tĩnh cục bộ. Việc nhận biết được thực hiện ngầm vì thế không khai báo rõ biến là sự cần thiết.

Ví dụ : sum := 0;

```
FOR count IN 1 TO 10 LOOP
    sum := sum + count;
END LOOP;
```

1.8.9 While :

Cú pháp: *WHILE condition LOOP*
 sequential-statements;
 END LOOP;

1.8.10 Loop :

Cú pháp: *LOOP*
 sequential-statements;
 EXIT WHEN condition;
 END LOOP;

1.8.11 Exit :

Phát biểu EXIT chỉ dùng bên trong vòng lặp. Nó thực hiện hành động nhảy ra khỏi vòng lặp cuối và thường dùng kết hợp với phát biểu LOOP.

Cú pháp: *EXIT WHEN condition;*

1.8.12 Next :

Phát biểu NEXT chỉ có thể được dùng bên trong vòng lặp . Nó thực hiện bỏ qua phần cuối của các vòng lặp và bắt vòng lặp tiếp theo. Nó thường dùng kết hợp với phát biểu FOR.

Cú pháp : *NEXT WHEN condition;*

Ví dụ : sum := 0;

```
FOR count IN 1 TO 10 LOOP
    NEXT WHEN count = 3;
    sum := sum + count;
END LOOP;
```

1.8.13 Function (hàm) :

Cú pháp khai báo *FUNCTION*

FUNCTION function-name (parameter-list) RETURN return-type;

Cú pháp định nghĩa FUNCTION

```
FUNCTION function-name (parameter-list) RETURN return-type IS
BEGIN
    sequential-statements;
END function-name;
```

Cú pháp gọi FUNCTION

function-name (actuals);

Thông số trong danh sách thông số chỉ có thể hoặc là tín hiệu hoặc là biến của ngõ IN.

Ví dụ:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY test_function IS PORT (
    x: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    z: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END test_function;
ARCHITECTURE Behavioral OF test_function IS
    SUBTYPE bit4 IS STD_LOGIC_VECTOR(3 DOWNTO 0);
    FUNCTION Shiftright (input: IN bit4) RETURN bit4 IS
    BEGIN
        RETURN '0' & input(3 DOWNTO 1);
    END shiftright;
    SIGNAL mysignal: bit4;
BEGIN
    PROCESS
    BEGIN
        mysignal <= x;
        z <= Shiftright(mysignal);
    END PROCESS;
```

END Behavioral;

1.8.14 Procedure (thủ tục) :

Cú pháp khai báo PRODUCE

PROCEDURE procedure -name (parameter-list);

Cú pháp định nghĩa PRODUCE

PROCEDURE procedure-name (parameter-list) IS

BEGIN

sequential-statements;

END procedure-name;

Cú pháp gọi PRODUCE

procedure -name (actuals);

Thông số trong danh sách thông số là các biến của ngõ IN ,OUT hay INOUT.

Ví dụ:

LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

ENTITY test_procedure IS PORT (

x: IN STD_LOGIC_VECTOR(3 DOWNT0 0);

z: OUT STD_LOGIC_VECTOR(3 DOWNT0 0));

END test_procedure;

ARCHITECTURE Behavioral OF test_procedure IS

SUBTYPE bit4 IS STD_LOGIC_VECTOR(3 DOWNT0 0);

PROCEDURE Shiftright (input: IN bit4; output: OUT bit4) IS

BEGIN

output := '0' & input(3 DOWNT0 1);

END shiftright;

BEGIN

PROCESS

VARIABLE mysignal: bit4;

BEGIN

```
        Shiftright(x, mysignal);

        z <= mysignal;

    END PROCESS;

END Behavioral;

1.8.15 Ví dụ về kiểu Behavioral :

LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

ENTITY bcd IS PORT (

    I: IN STD_LOGIC_VECTOR(3 DOWNTO 0);

    Segs: OUT STD_LOGIC_VECTOR(1 TO 7));

END bcd;

ARCHITECTURE Behavioral OF bcd IS

BEGIN

    PROCESS(I)

    BEGIN

        CASE I IS

            WHEN "0000" => Segs <= "1111110";

            WHEN "0001" => Segs <= "0110000";

            WHEN "0010" => Segs <= "1101101";

            WHEN "0011" => Segs <= "1111001";

            WHEN "0100" => Segs <= "0110011";

            WHEN "0101" => Segs <= "1011011";

            WHEN "0110" => Segs <= "1011111";

            WHEN "0111" => Segs <= "1110000";

            WHEN "1000" => Segs <= "1111111";

            WHEN "1001" => Segs <= "1110011";

            WHEN OTHERS => Segs <= "0000000";

        END CASE;

    END PROCESS;

END Behavioral;
```

END Behavioral;

1.9 Các câu lệnh kiểu Structural :

Mô hình cấu trúc cho phép kết nối bằng tay một vài linh kiện với nhau sử dụng tín hiệu . Tất cả các linh kiện được sử dụng cần định nghĩa trước với phần ENTITY và ARCHITECTURE của chúng trong cùng 1 file hoặc các file riêng. Trong mô đun cao nhất, mỗi linh kiện sử dụng trong bảng kết nối được khai báo trước tiên dùng phát biểu COMPONENT. Sau đó những khai báo COMPONENT đó được áp dụng với linh kiện thật sự trong mạch bằng cách dùng phát biểu PORT MAP. Sau đó những tín hiệu được dùng kết nối các linh kiện với nhau theo bảng kết nối.

1.9.1 Khai báo Component :

Khai báo tên và giao diện của linh kiện được dùng trong mô tả mạch .Việc khai báo cho mỗi linh kiện phải được dùng phù hợp với ENTITY và ARCHITECTURE của linh kiện đó. Khai báo tên và giao diện phải phù hợp, chính xác tên và giao diện được chỉ rõ trong phần ENTITY của linh kiện đó.

Cú pháp

COMPONENT component-name IS

PORT (list-of-port-names-and-types);

END COMPONENT;

or

COMPONENT component-name IS

GENERIC (identifier: type := constant);

PORT (list-of-port-names-and-types);

END COMPONENT;

Ví dụ :

COMPONENT half_adder IS PORT (

 xi, yi, cin: IN STD_LOGIC;

 cout, si: OUT STD_LOGIC);

END COMPONENT;

1.9.2 Port map :

Phát biểu PORT MAP thuyết minh khai báo 1 linh kiện với linh kiện thật trong mạch bằng cách chỉ rõ kết nối như thế nào để các ứng dụng của linh kiện này được hình thành.

Cú pháp : *label: component-name PORT MAP (association-list); hoặc*

label: component-name GENERIC MAP (constant) PORT MAP (association-list);

Danh sách kết hợp có thể chỉ rõ dùng phương pháp hoặc positional hoặc named.

Ví dụ : kết hợp theo vị trí (positional association):

SIGNAL x0, x1, y0, y1, c0, c1, c2, s0, s1: STD_LOGIC;

U1: half_adder PORT MAP (x0, y0, c0, c1, s0);

U2: half_adder PORT MAP (x1, y1, c1, c2, s1);

Ví dụ : kết hợp theo tên (named association):

SIGNAL x0, x1, y0, y1, c0, c1, c2, s0, s1: STD_LOGIC;

U1: half_adder PORT MAP (cout=>c1, si=>s0, cin=>c0, xi=>x0, yi=>y0);

U2: half_adder PORT MAP (cin=>c1, xi=>x1, yi=>y1, cout=>c2, si=>s1);

1.9.3 Open :

Từ khóa OPEN dùng trong danh sách kết hợp của PORT MAP để chú ý rằng port ra riêng biệt không được kết nối hoặc sử dụng. Nó không dùng cho 1 port ngõ vào.

Ví dụ : U1: half_adder PORT MAP (x0, y0, c0, OPEN, s0);

1.9.4 Generate :

Phát biểu GENERATE làm việc như đoạn mã mở rộng. Nó cung cấp cách đơn giản để sao chép những linh kiện giống nhau.

Cú pháp :

label: FOR identifier IN start [TO | DOWNTO] stop GENERATE

port-map-statements;

END GENERATE label;

Ví dụ:

-- using a FOR-GENERATE statement to generate four instances of the full adder

-- component for a 4-bit adder

LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

```

ENTITY Adder4 IS PORT (
    Cin: IN STD_LOGIC;
    A, B: IN STD_LOGIC_VECTOR(3 DOWNT0 0);
    Cout: OUT STD_LOGIC;
    SUM: OUT STD_LOGIC_VECTOR(3 DOWNT0 0));
END Adder4;

ARCHITECTURE Structural OF Adder4 IS

    COMPONENT FA PORT (
        ci, xi, yi: IN STD_LOGIC;
        co, si: OUT STD_LOGIC);

    END COMPONENT;

    SIGNAL Carryv: STD_LOGIC_VECTOR(4 DOWNT0 0);

BEGIN

    Carryv(0) <= Cin;

    Adder: FOR k IN 3 DOWNT0 0 GENERATE

    FullAdder: FA PORT MAP (Carryv(k), A(k), B(k), Carryv(k+1), SUM(k));

    END GENERATE Adder;

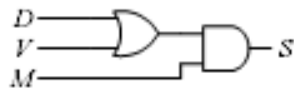
    Cout <= Carryv(4);

END Structural;

```

1.9.5 Ví dụ về cách viết đoạn mã theo kiểu Structure:

Ví dụ này dựa trên mạch sau :



-- declare and define the 2-input OR gate

```

LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

ENTITY myOR IS PORT (
    in1, in2: IN STD_LOGIC;
    out1: OUT STD_LOGIC);

```

```
END myOR;

ARCHITECTURE OR_Dataflow OF myOR IS
BEGIN
    out1 <= in1 OR in2; -- performs the OR operation
END OR_Dataflow;

-- declare and define the 2-input AND gate

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY myAND IS PORT (
    in1, in2: IN STD_LOGIC;
    out1: OUT STD_LOGIC);
END myOR;

ARCHITECTURE OR_Dataflow OF myAND IS
BEGIN
    out1 <= in1 AND in2; -- performs the AND operation
END OR_Dataflow;

-- topmost module for the siren

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY Siren IS PORT (
    M: IN STD_LOGIC;
    D: IN STD_LOGIC;
    V: IN STD_LOGIC;
    S: OUT STD_LOGIC);
END Siren;

ARCHITECTURE Siren_Structural OF Siren IS
-- declaration of the needed OR gate
    COMPONENT myOR PORT (
        in1, in2: IN STD_LOGIC;
```



```

        out1: OUT STD_LOGIC);

    END COMPONENT;

-- declaration of the needed AND gate

    COMPONENT myAND PORT (

        in1, in2: IN STD_LOGIC;

        out1: OUT STD_LOGIC);

    END COMPONENT;

-- signal for connecting the output of the OR gate
-- with the input to the AND gate

    SIGNAL term1: STD_LOGIC;

BEGIN

U0: myOR PORT MAP (D, V, term1);
U1: myAND PORT MAP (term1, M, S);
END Siren_Structural;

```

1.10 Các thủ tục chuyển đổi :

1.10.1 *Conv_integer* () :

Chuyển loại std_logic_vector thành Integer

Yêu cầu: *LIBRARY IEEE;*
 USE IEEE.STD_LOGIC_UNSIGNED.ALL;

Cú pháp : *CONV_INTEGER(std_logic_vector)*

Ví dụ : LIBRARY IEEE;
 USE IEEE.STD_LOGIC_UNSIGNED.ALL;
 SIGNAL four_bit: STD_LOGIC_VECTOR(3 DOWNT0 0);
 SIGNAL n: INTEGER;
 n := CONV_INTEGER(four_bit);

1.10.2 *Conv_Std_Logic_Vector* (,):

Chuyển loại Integer thành std_logic_vector

Yêu cầu : *LIBRARY IEEE;*

USE IEEE.STD_LOGIC_ARITH.ALL;

Cú pháp : *CONV_STD_LOGIC_VECTOR (integer, number_of_bits)*

Ví dụ : LIBRARY IEEE;

USE IEEE.STD_LOGIC_ARITH.ALL;

SIGNAL four_bit: STD_LOGIC_VECTOR(3 DOWNT0 0);

SIGNAL n: INTEGER;

four_bit := CONV_STD_LOGIC_VECTOR(n, 4);

CHƯƠNG 2 : DÙNG NGÔN NGỮ VHDL MÔ TẢ CÁC MẠCH SỐ CƠ BẢN

2.1 Ngôn ngữ VHDL mô tả các cổng logic cơ bản:

Một mạch số được mô tả bởi một phương trình Boolean đều có thể dễ dàng chuyển sang ngôn ngữ VHDL bằng cách sử dụng kiểu viết ‘dòng dữ liệu’ (dataflow). Ở cấp độ dòng dữ liệu, một mạch có thể được thiết lập từ các cổng AND, OR hay NOT khi mô tả các cổng này bằng ngôn ngữ VHDL ta sẽ dùng các câu lệnh đồng thời.

2.1.1 Đoạn mã VHDL mô tả cổng NAND 2 ngõ vào:

Dưới đây là đoạn mã VHDL viết cho cổng NAND 2 ngõ vào. Đây cũng là một đoạn mã mẫu minh họa cho cách viết ngôn ngữ VHDL. Dòng đầu tiên trong đoạn mã VHDL bắt đầu bằng hai dấu ‘ - ’ là dòng chú thích cho đoạn mã. Hai dòng lệnh LIBRARY và USE được dùng để chỉ rõ thư viện IEEE được sử dụng trong đoạn mã. Thư viện này chứa tất cả các phần tử logic cần thiết cho đoạn mã phía dưới. Hai dòng lệnh này tương đương với dòng lệnh ‘#include’ trong lập trình C++.

Mỗi một thành phần trong ngôn ngữ VHDL từ phần tử cổng NAND đơn giản cho đến bộ vi xử lý phức tạp thì đều bao gồm hai bộ phận: phần thực thể và phần thân cấu trúc. Phần thực thể tương tự như khai báo hàm trong C++. Nó khai báo tất cả các tín hiệu ngõ vào và ngõ ra trong mạch. Mỗi một thực thể phải được đặt tên, ví dụ trong đoạn mã bên dưới là NAND2gate. Thực thể chứa một danh sách PORT, nó sẽ quy định số lượng ngõ vào và ra của cổng NAND. Ví dụ trong đoạn mã x, y là những tín hiệu ngõ vào ở dạng STD_LOGIC và f là tín hiệu ngõ ra cũng ở dạng STD_LOGIC. Dạng STD_LOGIC cũng giống như dạng loại BIT, ngoại trừ nó chứa thêm những giá trị khác ngoài hai giá trị 0 và 1.

Phần thân cấu trúc nó chứa đoạn mã mô tả hoạt động của NAND 2 ngõ vào. Mỗi một thân cấu trúc cũng cần phải được đặt tên. Ví dụ trong đoạn mã phần thân cấu trúc có tên là Dataflow. Trong phần thân cấu trúc có thể có một hay nhiều câu lệnh đồng thời. Không giống như trong C++ nơi mà các dòng lệnh được thực thi một cách tuần tự, những dòng lệnh trong thân cấu trúc được thực thi một cách song song. Dấu ‘<=’ được dùng để gán cho một tín hiệu. Về phải của dấu ‘<=’ là biểu diễn phép toán logic giữa các biến số ngõ vào x và y, kết quả sẽ được gán cho ngõ ra f nằm ở bên trái của dấu ‘<=’. Đoạn mã mô tả cổng NAND bằng ngôn ngữ VHDL được viết như sau.

```
-- this is a dataflow model of a 2-input NAND gate
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;

ENTITY NAND2gate IS PORT (
    x: IN STD_LOGIC;
    y: IN STD_LOGIC;
    f: OUT STD_LOGIC);
END NAND2gate;

ARCHITECTURE Dataflow OF NAND2gate IS
BEGIN
    f <= x NAND y;          -- signal assignment
END Dataflow;
```

Hình 2. 1 : Đoạn mã VHDL cho cổng NAND 2 ngõ vào.

2.1.2 Ngôn ngữ VHDL mô tả cổng NOR 3 ngõ vào:

Dưới đây là đoạn mã VHDL cho cổng NOR 3 ngõ vào. Có 3 tín hiệu ngõ vào là x, y, z và một tín hiệu ngõ ra được khai báo trong phần thực thể. Trong ví dụ này có khai báo thêm hai tín hiệu nội: xory và xoryorz, cả hai tín hiệu này đều thuộc loại STD_LOGIC. Từ khóa SIGNAL trong phần thân cấu trúc được dùng để khai báo hai tín hiệu nội này. Các tín hiệu nội này được sử dụng như những nút tín hiệu trung gian trong mạch. Tất cả các câu lệnh gán tín hiệu được thực thi một cách đồng thời. Điều vừa nêu được minh họa rõ nét trong hình 2.2b.

Hình 2.2c là quá trình mô phỏng theo thời gian cho hoạt động của mạch hình 2.2b. Trong giản đồ xung này, chúng ta thấy ngõ ra f chỉ bằng 1 khi và chỉ khi tất cả các ngõ vào của nó đều phải có giá trị 0. Do đó f chỉ bằng 1 tại hai khoảng thời gian từ 0-100ns và từ 800-900ns, còn trong những khoảng thời gian khác f đều nhận giá trị 0.

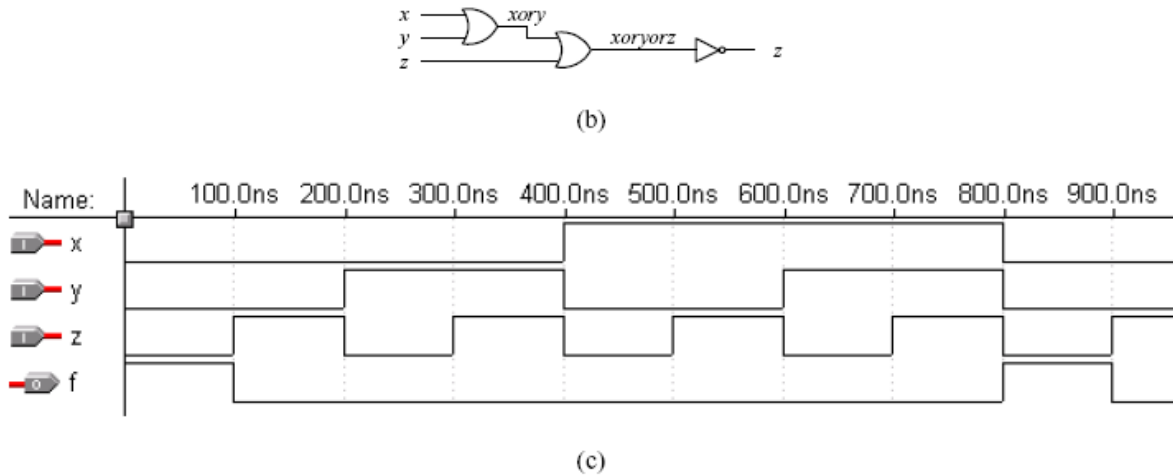
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;

ENTITY NOR3gate IS PORT (
    x: IN STD_LOGIC;
    y: IN STD_LOGIC;
    z: IN STD_LOGIC;
    f: OUT STD_LOGIC);
END NOR3gate;

ARCHITECTURE Dataflow OF NOR3gate IS
    SIGNAL xory, xoryorz : STD_LOGIC;
BEGIN
    xory <= x OR y;          -- three concurrent signal assignments

    xoryorz <= xory OR z;
    f <= NOT xoryorz;
END Dataflow;
```

(a)



Hình 2. 2 : Cổng NOR 3 ngõ vào (a) đoạn mã VHDL; (b) sơ đồ mạch; (c) thời gian mô phỏng.

2.1.3 Dùng ngôn ngữ VHDL mô tả một hệ thống báo động cho xe hơi:

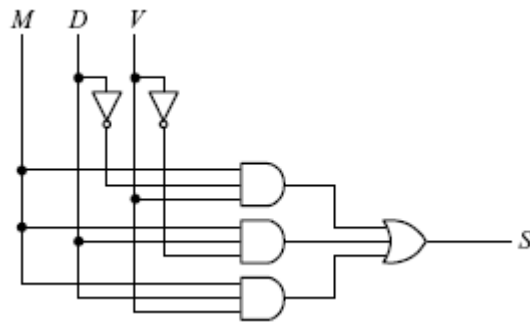
Trong một hệ thống báo động cho xe hơi, chúng ta thường kết nối hệ thống này với một cái còi báo động, để khi có một tác động từ bên ngoài nào vào hệ thống thì chuông sẽ vang lên. Theo ý tưởng này, chúng ta phải có: một công tắc điều khiển đóng ngắt chính cho hệ thống được đặt tên là M; một công tắc đại diện cho việc đóng mở cửa xe được đặt tên là D; một bộ phát hiện dao động được đặt tên là V. Chúng ta sẽ quy định mức logic cho từng ký hiệu này như sau: cửa xe mở khi D=1, trường hợp khác thì D=0; tương tự như thế, khi xe bị rung động thì V=1, trường hợp khác V=0; chúng ta muốn còi S reo lên thì S=1. Như vậy có ba trường hợp làm cho chuông báo hiệu reo lên là D=1 hoặc V=1 hoặc cả hai D=V=1, trong ba trường hợp trên công tắc điều khiển chính của hệ thống sẽ đóng lại làm chuông báo động vang lên tức là M=1. Tuy nhiên, vấn đề đặt ra ở đây là khi người chủ xe mở cửa xe ra vào bên trong lái xe thì họ không muốn còi báo động vang lên. Do đó lúc này công tắc điều khiển chính M=0 tương ứng với toàn bộ hệ thống báo động sẽ ngừng hoạt động bất kể D và V đang là 0 hay 1. Dựa trên những phân tích trên ta lập bảng chân trị cho hàm S gồm 3 biến M, D, V như sau:

M	D	V	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Từ bảng chân trị ta có thể viết được phương trình Boolean cho ngõ ra S như sau:

$$S = (MD'V) + (MDV') + (MDV)$$

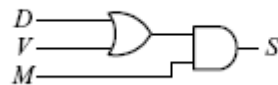
Từ phương trình trên ta sẽ thiết lập được sơ đồ mạch cho cả hệ thống báo động này như sau:



Ngoài ra ta có thể rút gọn biểu thức S bằng cách sử dụng các tiên đề và các định lý cơ bản trong đại số Boolean:

$$\begin{aligned}
 S &= (MD'V) + (MDV') + (MDV) \\
 &= M(D'V + DV' + DV) \\
 &= M(D(V' + V) + V(D' + D)) \\
 &= M(D(1) + V(1)) \\
 &= M(D + V)
 \end{aligned}$$

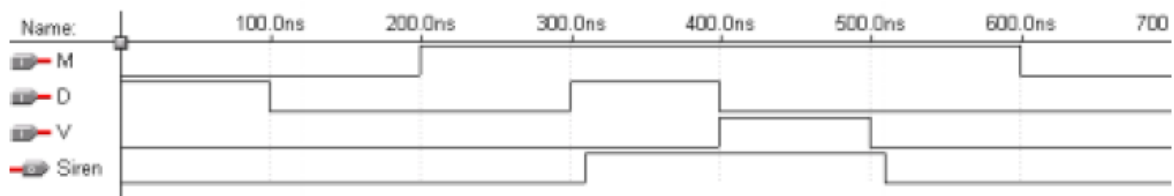
Từ biểu thức S rút gọn ta sẽ thiết lập được mạch báo động đơn giản hơn, sử dụng ít cổng logic hơn nhưng yêu cầu chống trộm vẫn đảm bảo.



Giãn đồ xung đóng ngắt chuông được mô tả rõ nét thông qua hình 2.3



(a)



(b)

Hình 2. 3 : Giãn đồ xung của hệ thống báo động trong xe hơi: (a) Dạng xung trên lý thuyết; (b) Dạng xung trên thực tế.

Ta nhận thấy có xuất hiện thời gian trễ khi ngắt và mở chuông. Điều này hoàn toàn phù hợp với thực tế khi thi công mạch.

Ta sẽ viết chương trình VHDL cho biểu thức logic của mạch báo động trong xe hơi:

$$S = (MD'V) + (MDV') + (MDV)$$

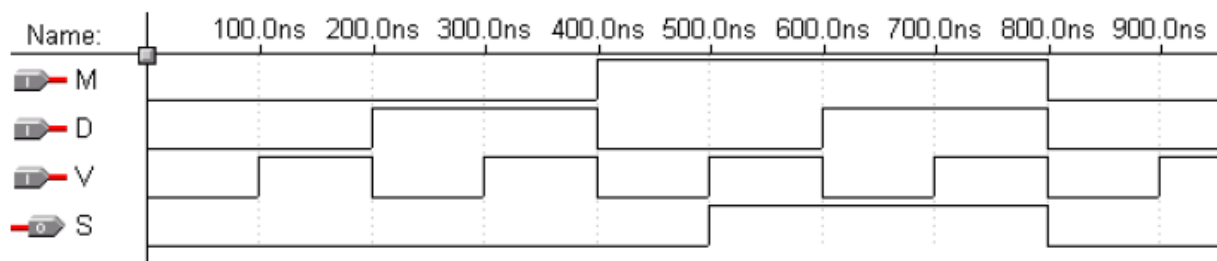
Đoạn mã này được viết ở cấp độ Dataflow không phải vì ta nhìn vào tên thân cấu trúc của nó là Dataflow để xác định. Mà vì mã hóa ở cấp độ Dataflow sẽ dùng các phương trình logic để mô tả mạch. Trong đoạn mã dưới đây ta sẽ dùng cách này để mô tả sự hoạt động của các cổng AND, OR, NOT bằng những câu lệnh gán tín hiệu đồng thời.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;

ENTITY Siren IS PORT (
    M: IN STD_LOGIC;
    D: IN STD_LOGIC;
    V: IN STD_LOGIC;
    S: OUT STD_LOGIC);
END Siren;

ARCHITECTURE Dataflow OF Siren IS
    SIGNAL term_1, term_2, term_3: STD_LOGIC;
BEGIN
    term_1 <= M AND (NOT D) AND V;
    term_2 <= M AND D AND (NOT V);
    term_3 <= M AND D AND V;
    S <= term_1 OR term_2 OR term_3;
END Dataflow;
```

(a)



(b)

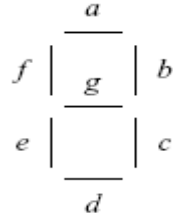
Hình 2. 4 : Mạch báo động trong xe hơi (a) đoạn mã VHDL được viết dưới dạng dataflow; (b) mô phỏng giãn đồ xung.

2.2 Bộ giải mã LED 7 đoạn:

2.2.1 Xây dựng cấu trúc bộ giải mã LED 7 đoạn:

Bây giờ chúng ta sẽ tổng hợp mạch cho một bộ giải mã 7 đoạn lái cho một bộ hiển thị LED 7 đoạn. Bộ giải mã 7 đoạn chuyển đổi một ngõ vào 4 bit thành 7 ngõ ra cho việc điều khiển

7 đèn trong bộ hiển thị LED 7 đoạn. 4 bit ngõ vào này mã hóa cho một trạng thái nhị phân tương ứng của một số thập phân. Cho một số thập phân ở ngõ vào, 7 đường tín hiệu ngõ ra được bật lên theo một trật tự đã định trước của bộ hiển thị LED để tượng trưng cho một số thập phân. Dưới đây là sơ đồ của bộ hiển thị LED 7 đoạn với các tên của từng đoạn được gán như sau:



Sự hoạt động của bộ giải mã 7 đoạn được giới thiệu trong bảng chân trị hình 2.5. 4 ngõ vào được giải mã là I_3, I_2, I_1, I_0 , và 7 ngõ ra mà mỗi ngõ ra thì được dán nhãn là seg a, seg b, ..., seg g. Mỗi một cách kết nối ở ngõ vào thì tượng trưng cho một số thập phân hiển thị trong LED 7 đoạn và chúng được biểu diễn trong cột Display. Mỗi một đoạn thì sáng lên khi nó ở trạng thái 1 và tắt đi khi nó nhận giá trị 0. Ví dụ trường hợp 4 bit ngõ vào là 0000 thì LED sẽ chỉ hiển thị lên 6 đoạn a, b, c, d, e và f tương đương với các đoạn này sẽ nhận giá trị 1 và chỉ duy nhất một đoạn g không hiển thị vì nó nhận giá trị 0.

Trong bảng dưới đây ta chú ý rằng các giá trị ngõ vào từ 1010 đến 1111 thì không được biểu thị, và ta cũng không cần quan tâm đến những giá trị của các đoạn LED hiển thị cho các giá trị đó.

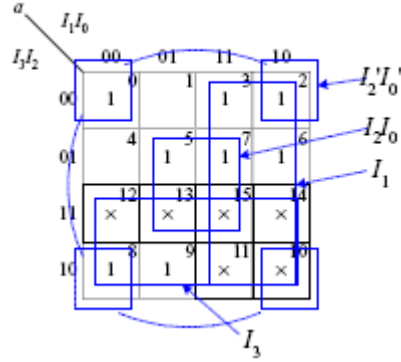
Inputs				Decimal digit	Display	seg a	seg b	seg c	seg d	seg e	seg f	seg g
I_3	I_2	I_1	I_0									
0	0	0	0	0		1	1	1	1	1	1	0
0	0	0	1	1		0	1	1	0	0	0	0
0	0	1	0	2		1	1	0	1	1	0	1
0	0	1	1	3		1	1	1	1	0	0	1
0	1	0	0	4		0	1	1	0	0	1	1
0	1	0	1	5		1	0	1	1	0	1	1
0	1	1	0	6		1	0	1	1	1	1	1
0	1	1	1	7		1	1	1	0	0	0	0
1	0	0	0	8		1	1	1	1	1	1	1
1	0	0	1	9		1	1	1	0	0	1	1
rest of the combinations						×	×	×	×	×	×	×

Hình 2. 5 : Bảng chân trị của bộ giải mã 7 đoạn.

Từ bảng chân trị ta có thể viết được biểu thức logic cho đoạn seg a một cách dễ dàng dựa vào những giá trị 1 trong cột seg a.

$$a = I_3'I_2'I_1'I_0' + I_3'I_2'I_1I_0' + I_3'I_2'I_1I_0 + I_3'I_2I_1'I_0 + I_3'I_2I_1I_0' + I_3'I_2I_1I_0 + I_3I_2'I_1'I_0' + I_3I_2'I_1'I_0$$

Muốn thu gọn biểu thức logic a để thuận tiện trong việc mô tả mạch sau này ta phải sử dụng phương pháp bìa Karnaugh đã được giới thiệu ở phần trên. Trạng thái nào của seg a không có trong bảng chân trị hình 2.5 thì ta có thể mặc định bằng 0 hoặc bằng 1 sao cho thuận lợi trong việc tối giản biểu thức.



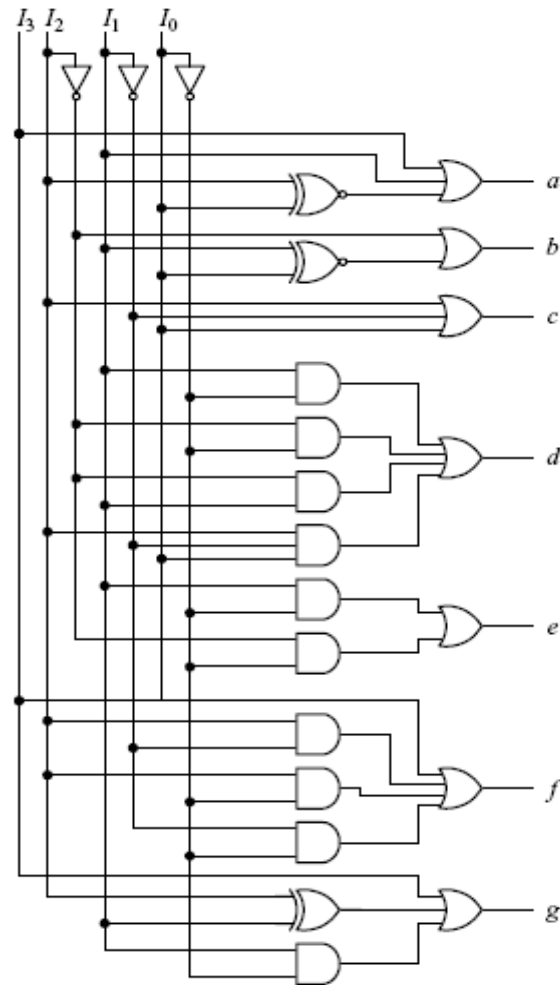
Từ bìa Karnaugh kết hợp với các phép biến đổi đơn giản trong đại số Boolean ta có thể viết gọn lại biểu thức a như sau:

$$a = I_3 + I_1 + I_2'I_0' + I_2I_0 = I_3 + I_1 + (I_2 \odot I_0)$$

Việc lập biểu thức Boolean cho các đoạn b, c, d, e, f, g được thực hiện tương tự như cho a. Cuối cùng ta sẽ có các kết quả sau:

$$\begin{aligned} b &= I_2' + (I_1 \odot I_0) \\ c &= I_2 + I_1' + I_0 \\ d &= I_1I_0' + I_2'I_0' + I_2'I_1 + I_2I_1'I_0 \\ e &= I_1I_0' + I_2'I_0' \\ f &= I_3 + I_2I_1' + I_2I_0' + I_1'I_0' \\ g &= I_3 + (I_2 \oplus I_1) + I_1I_0' \end{aligned}$$

Từ 7 biểu thức a, b, c, d, e, f, g ta sẽ vẽ được sơ đồ mạch giải mã LED 7 đoạn như sau:



Hình 2. 6 : Mạch giải mã LED 7 đoạn.

2.2.2 Ngôn ngữ VHDL mô tả mạch giải mã LED 7 đoạn:

Việc dùng ngôn ngữ VHDL để mô tả cho một mạch kết nối hay mạch tuần tự có thể được thực hiện bằng một trong ba cấp độ sau: *structural*, *dataflow* và *behavioral*.

Ở cấp độ *structural*, là cấp độ thấp nhất, trước tiên bạn phải tự thiết kế mạch. Phải vẽ mạch, bạn dùng VHDL để biểu diễn các thành phần và các cổng logic cần thiết cho mạch. Bạn phải dùng ngôn ngữ VHDL mô tả sơ đồ mạch biểu diễn một cách chính xác các cổng logic này được kết nối với nhau như thế nào.

Ở cấp độ *dataflow*, bạn dùng các hàm logic đã được xây dựng sẵn trong VHDL để gán cho các câu lệnh trong việc kết nối tín hiệu. Việc mô tả một mạch số, trước tiên bạn lại phải tự thiết kế mạch. Các phương trình Boolean dùng để mô tả mạch thì có thể được dễ dàng chuyển đổi thành các câu gán lệnh trong ngôn ngữ VHDL bằng các từ khóa logic đã có sẵn. Tất cả các câu lệnh được viết trong hai cây trúc *Structural* và *dataflow* đều được thực thi một cách đồng thời. Điều này thì trái ngược với những câu lệnh trong ngôn ngữ máy tính, nơi mà các câu lệnh này được thực thi một cách tuần tự. Mô tả mạch ở cấp độ *Behavioral*

thì là giống nhất với ngôn ngữ của máy tính. Bạn phải có tất cả các tiêu chuẩn của một chương trình ngôn ngữ cấp cao như FOR LOOP, WHILE LOOP, IF THEN ELSE, CASE và các sự gán biến. Những câu lệnh này được tập trung xử lý trong khối PROCESS và được thực thi một cách tuần tự.

2.2.3 Cấu trúc structural biểu diễn giải mã số thập phân ra Led 7 đoạn:

```
ENTITY myxnor2 IS PORT(
    i1, i2: IN BIT;
    o: OUT BIT);
END myxnor2;
ARCHITECTURE Dataflow OF myxnor2 IS
BEGIN
    o <= not (i1 XOR i2);
END Dataflow;

ENTITY myxor2 IS PORT(
    i1, i2: IN BIT;
    o: OUT BIT);
END myxor2;
ARCHITECTURE Dataflow OF myxor2 IS
BEGIN
    o <= i1 XOR i2;
END Dataflow;

ENTITY myand2 IS PORT(
    i1, i2: IN BIT;
    o: OUT BIT);
END myand2;
ARCHITECTURE Dataflow OF myand2 IS
BEGIN
    o <= i1 AND i2;
END Dataflow;

ENTITY myand3 IS PORT(
```

```
    i1, i2, i3: IN BIT;
    o: OUT BIT);
END myand3;
ARCHITECTURE Dataflow OF myand3 IS
BEGIN
    o <= (i1 AND i2 AND i3);
END Dataflow;

ENTITY myor2 IS PORT(
    i1, i2: IN BIT;
    o: OUT BIT);
END myor2;
ARCHITECTURE Dataflow OF myor2 IS
BEGIN
    o <= i1 OR i2;
END Dataflow;

ENTITY myor3 IS PORT(
    i1, i2, i3: IN BIT;
    o: OUT BIT);
END myor3;
ARCHITECTURE Dataflow OF myor3 IS
BEGIN
    o <= i1 OR i2 OR i3;
END Dataflow;
```

```

ENTITY myor4 IS PORT(
    i1, i2, i3, i4: IN BIT;
    o: OUT BIT);
END myor4;
ARCHITECTURE Dataflow OF myor4 IS
BEGIN
    o <= i1 OR i2 OR i3 OR i4;
END Dataflow;

ENTITY inv IS PORT(
    i: IN BIT;
    o: OUT BIT);
END inv;
ARCHITECTURE Dataflow OF inv IS
BEGIN
    o <= not i;
END Dataflow;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY bcd IS PORT(
    i0, i1, i2, i3: IN BIT;
    a, b, c, d, e, f, g: OUT BIT);
END bcd;

ARCHITECTURE Structural OF bcd IS
    COMPONENT inv PORT(
        i: IN BIT;
        o: OUT BIT);
    END COMPONENT;

```

```

    COMPONENT myand2 PORT(
        i1, i2: IN BIT;
        o: OUT BIT);
    END COMPONENT;
    COMPONENT myand3 PORT(
        i1, i2, i3: IN BIT;
        o: OUT BIT);
    END COMPONENT;
    COMPONENT myor2 PORT(
        i1, i2: IN BIT;
        o: OUT BIT);
    END COMPONENT;
    COMPONENT myor3 PORT(
        i1, i2, i3: IN BIT;
        o: OUT BIT);
    END COMPONENT;
    COMPONENT myor4 PORT(
        i1, i2, i3, i4: IN BIT;
        o: OUT BIT);
    END COMPONENT;
    COMPONENT myxnor2 PORT(
        i1, i2: IN BIT;
        o: OUT BIT);
    END COMPONENT;
    COMPONENT myxor2 PORT(
        i1, i2: IN BIT;
        o: OUT BIT);
    END COMPONENT;

```

```

    SIGNAL j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z: BIT;
BEGIN
    U1: INV port map(i2,j);
    U2: INV port map(i1,k);
    U3: INV port map(i0,l);
    U4: myXNOR2 port map(i2, i0, z);
    U5: myOR3 port map(i3, i1, z, a);
    U6: myXNOR2 port map(i1, i0, y);
    U7: myOR2 port map(j, y, b);
    U8: myOR3 port map(i2, k, i0, c);
    U9: myAND2 port map(i1, l, x);
    U10: myAND2 port map(j, l, w);
    U11: myAND2 port map(j, i1, v);
    U12: myAND3 port map(i2, k, i0, t);
    U13: myOR4 port map(x, w, v, t, d);
    U14: myAND2 port map(i1, l, s);
    U15: myAND2 port map(j, l, r);
    U16: myOR2 port map(s, r, e);
    U17: myAND2 port map(i2, k, q);
    U18: myAND2 port map(i2, l, p);
    U19: myAND2 port map(k, l, o);
    U20: myOR4 port map(i3, q, p, o, f);
    U21: myXOR2 port map(i2, i1, n);
    U22: myAND2 port map(i1, l, m);
    U23: myOR3 port map(i3, n, m, g);
END Structural;

```

2.2.4 Cấu trúc dataflow biểu diễn giải mã số thập phân ra Led 7 đoạn:

Dưới đây là đoạn mã VHDL cho bộ giải mã BCD ra LED 7 đoạn được viết ở cấp độ dataflow. Trong phần cấu trúc này thì việc gán 7 câu lệnh đồng thời cho 7 tín hiệu được sử dụng. Biểu thức logic a được chuyển đổi thành ngôn ngữ VHDL như sau:

$$a = I_3 + I_1 + (I_2 \odot I_0)$$

$$\text{Segs}(1) \leq I(3) \text{ OR } I(1) \text{ OR NOT } (I(2) \text{ XOR } I(0));$$

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY bcd IS PORT (
    I: IN STD_LOGIC_VECTOR (3 DOWNT0 0);
    Segs: OUT std_logic_vector (1 TO 7));
END bcd;

ARCHITECTURE Dataflow OF bcd IS
BEGIN
    Segs(1) <= I(3) OR I(1) OR NOT (I(2) XOR I(0));           -- seg a
    Segs(2) <= (NOT I(2)) OR NOT (I(1) XOR I(0));           -- seg b
    Segs(3) <= I(2) OR (NOT I(1)) OR I(0);                   -- seg c
    Segs(4) <= (I(1) AND NOT I(0)) OR (NOT I(2) AND NOT I(0)) -- seg d
               OR (NOT I(2) AND I(1)) OR (I(2) AND NOT I(1) AND I(0));
    Segs(5) <= (I(1) AND NOT I(0)) OR (NOT I(2) AND NOT I(0)); -- seg e
    Segs(6) <= I(3) OR (I(2) AND NOT I(1))                   -- seg f
               OR (I(2) AND NOT I(0)) OR (NOT I(1) AND NOT I(0));
    Segs(7) <= I(3) OR (I(2) XOR I(1)) OR (I(1) AND NOT I(0)); -- seg g
END Dataflow;

```

2.2.5 Cấu trúc behavioral biểu diễn giải mã số thập phân ra Led 7 đoạn:

Dưới đây là đoạn mã VHDL cho bộ giải mã BCD ra LED 7 đoạn được viết ở cấp độ behavioral. Trong phần cấu trúc này, một khối Process được sử dụng. Tất cả các câu lệnh trong khối Process được thực thi một cách tuần tự. Nếu trong cùng một cấu trúc mà có nhiều khối Process thì mỗi một khối Process sẽ được thực thi một cách đồng thời.

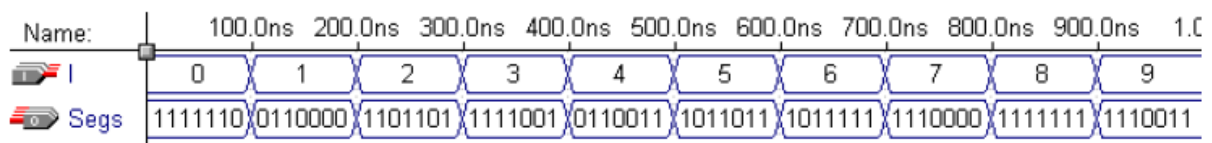
Các từ trong dấu “()” của khối Process để chỉ các biến đang xét trong khối process, nếu một trong các biến này thay đổi giá trị thì các dòng lệnh trong khối Process mới được thực thi từ đầu cho đến cuối khối Process đó.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY bcd IS PORT (
  I: IN STD_LOGIC_VECTOR (3 DOWNT0 0);
  Segs: OUT std_logic_vector (1 TO 7));
```

```
END bcd;

ARCHITECTURE Behavioral OF bcd IS
BEGIN
  PROCESS(I)
  BEGIN
    CASE I IS
      WHEN "0000" => Segs <= "1111110";
      WHEN "0001" => Segs <= "0110000";
      WHEN "0010" => Segs <= "1101101";
      WHEN "0011" => Segs <= "1111001";
      WHEN "0100" => Segs <= "0110011";
      WHEN "0101" => Segs <= "1011011";
      WHEN "0110" => Segs <= "1011111";
      WHEN "0111" => Segs <= "1110000";
      WHEN "1000" => Segs <= "1111111";
      WHEN "1001" => Segs <= "1110011";
      WHEN OTHERS => Segs <= "0000000";
    END CASE;
  END PROCESS;
END Behavioral;
```



Hình 2. 7 : Sơ đồ biểu diễn thời gian hiển thị một số trên Led 7 đoạn của một số thập phân tương ứng.

2.3 Bộ cộng:

2.3.1 Bộ cộng toàn phần (FA):

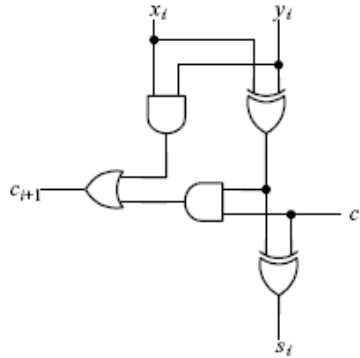
Để xây dựng cấu trúc cho một bộ cộng thực hiện phép toán cộng cho các số có hai giá trị nhị phân, $X = x_{n-1}...x_0$ và $Y = y_{n-1}...y_0$. Trước tiên ta cộng từng cặp bit lại với nhau x_i và y_i , sau đó ta cộng thêm bit nhớ c_i vào kết quả này ta sẽ được kết quả cuối cùng của phép cộng hai số nhị phân. Trong đó bit nhớ c_i là bit nhận giá trị 1 khi kết quả phép cộng trước đó của nó là có nhớ ($1+1=0$ viết 0 nhớ 1 sang cột tiếp theo). Do đó $s_i = x_i + y_i + c_i$ và $c_{i+1} = 1$ nếu kết quả phép cộng s_i là có nhớ sang cột kế tiếp. Mạch thực hiện phép cộng theo từng cặp bit như vậy ta gọi là bộ cộng toàn phần (FA) và bảng chân trị của nó được biểu diễn trong hình 2.8(a). Biểu thức logic của s_i và c_{i+1} được chứng minh như sau:

$$\begin{aligned} s_i &= x_i'y_i'c_i + x_i'y_ic_i' + x_iy_i'c_i' + x_iy_ic_i \\ &= (x_i'y_i + x_iy_i')c_i' + (x_i'y_i' + x_iy_i)c_i \\ &= (x_i \oplus y_i)c_i' + (x_i \oplus y_i)'c_i \\ &= x_i \oplus y_i \oplus c_i \\ c_{i+1} &= x_i'y_ic_i + x_iy_i'c_i + x_iy_ic_i' + x_iy_i'c_i \\ &= x_iy_i(c_i' + c_i) + c_i(x_i'y_i + x_iy_i') \\ &= x_iy_i + c_i(x_i \oplus y_i) \end{aligned}$$

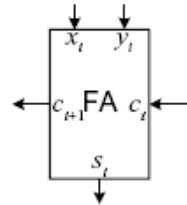
Từ hai biểu thức trên ta có thể vẽ sơ đồ mạch cho bộ cộng toàn phần được biểu diễn trong hình 2.8(b). Hình 2.8(c) biểu diễn ký hiệu logic của bộ cộng toàn phần.

x_i	y_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

(a)



(b)



(c)

Hình 2. 8 : Bộ cộng toàn phần (a) bảng chân trị; (b) sơ đồ mạch; (c) ký hiệu logic.

Đoạn mã VHDL cho bộ cộng toàn phần thực hiện phép cộng một cặp bit được viết theo cấu trúc Dataflow có dạng như sau:

```

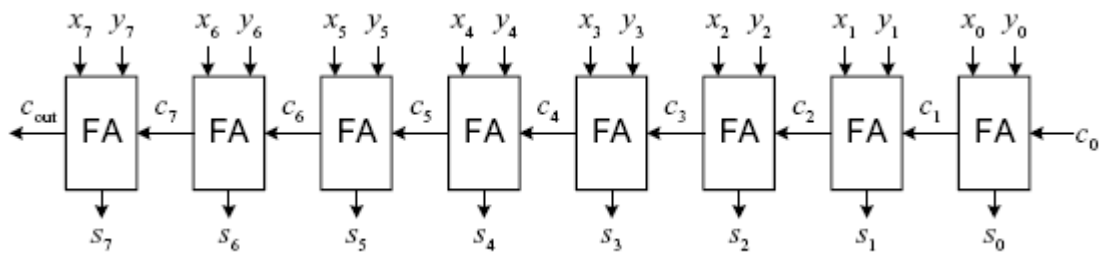
ENTITY FA IS PORT (
  ci, xi, yi: IN BIT;
  co, si: OUT BIT);
END FA;

ARCHITECTURE Dataflow OF FA IS
BEGIN
  co <= (xi AND yi) OR (ci AND (xi XOR yi));
  si <= xi XOR yi XOR ci;
END Dataflow;

```

2.3.2 Bộ cộng toàn phần hai số nhị phân có nhiều hơn 1 bit:

Phần trên ta đã khảo sát bộ cộng toàn phần cho một cặp bit, trong phần này ta sẽ trình bày phương pháp cộng hai số nhị phân 8 bit lại với nhau (Ripple Carry Adder). Sơ đồ mạch thực hiện việc cộng này được biểu diễn trong hình 2.9, nó bao gồm các bộ cộng toàn phần một cặp bit được mắc nối tiếp với nhau. Việc cộng được thực hiện từ cặp bit đầu tiên bên phải hay cặp bit có trọng số nhỏ nhất trong chuỗi bit nhị phân, lúc này ta phải set $c_0=0$. Mạch cộng tiếp tục thực hiện từ phải qua trái, $c_1=1$ nếu phép cộng cặp bit x_0 và y_0 với nhau có nhớ và $c_1=0$ cho trường hợp cộng không có nhớ. Cứ thực hiện công tuần tự như vậy cho đến bit c_{out} .



Hình 2. 9 : Bộ cộng hai số nhị phân 8 bit.

Dưới đây là đoạn mã VHDL được viết theo cấu trúc Structural cho bộ cộng hai số nhị phân 4 bit. Khi chúng ta cần lặp lại 4 lần bộ cộng toàn phần, chúng ta có thể dùng 4 lần câu lệnh PORT MAP hoặc là có thể sử dụng câu lệnh FOR-GENERATE để vận hành 4 thành phần này một cách tự động như trong đoạn mã biểu diễn dưới đây. Câu lệnh FOR k IN 3 DOWNT0 0 GENERATE quyết định lặp lại câu lệnh PORT MAP bao nhiêu lần và giá trị sử dụng cho phép đếm số lần lặp là k.


```

ENTITY Adder4 IS PORT (
    Cin: IN BIT;
    A, B: IN BIT_VECTOR(3 DOWNTO 0);
    Cout: OUT BIT;
    SUM: OUT BIT_VECTOR(3 DOWNTO 0));
END Adder4;

ARCHITECTURE Structural OF Adder4 IS
    COMPONENT FA PORT (
        ci, xi, yi: IN BIT;
        co, si: OUT BIT);
    END COMPONENT;

    SIGNAL Carryv: BIT_VECTOR(4 DOWNTO 0);

BEGIN
    Carryv(0) <= Cin;

    Adder: FOR k IN 3 DOWNTO 0 GENERATE
        FullAdder: FA PORT MAP (Carryv(k), A(k), B(k), Carryv(k+1), SUM(k));
    END GENERATE Adder;

    Cout <= Carryv(4);
END Structural;

```

2.3.3 Bộ cộng hai số nhị phân nhiều bit cho kết quả hiển thị nhanh:

Bộ cộng Ripple Carry Adder thì thực hiện việc cộng chậm bởi vì trạng thái nhớ ngõ vào của một cặp bit thì phụ thuộc vào trạng thái nhớ ngõ ra của cặp bit liền trước nó. Vì thế trước khi mẫu bit thứ i có thể có hiệu lực tại ngõ ra thì nó phải chờ đợi mẫu bit thứ $i-1$ đạt được giá trị ổn định trước. Trong bộ cộng hai số nhị phân nhiều bit cho kết quả hiển thị nhanh (Carry-Lookahead Adder) thì mỗi một mẫu bit loại trừ sự phụ thuộc vào trạng thái nhớ ngõ ra của tín hiệu trước đó, nó sử dụng giá trị của hai bit X và Y một cách trực tiếp để suy ra các tín hiệu cần thiết. Cho mỗi một mẫu bit thứ i thì tín hiệu nhớ ngõ ra c_{i+1} sẽ được bật lên 1 nếu một trong hai điều kiện sau đây là đúng:

$$x_i = 1 \text{ and } y_i = 1$$

Hay

$$(x_i = 1 \text{ or } y_i = 1) \text{ and } c_i = 1$$

Hay nói một cách khác:

$$c_{i+1} = (x_i y_i) + [(x_i + y_i) c_i].$$

Nếu chúng ta đặt:

$$g_i = x_i y_i$$

Và

$$p_i = x_i + y_i$$

Thì ta sẽ có cách viết tổng quát như sau:

$$c_{t+1} = g_t + p_t c_t.$$

Từ công thức tổng quát này ta có thể suy ra các công thức cụ thể sau:

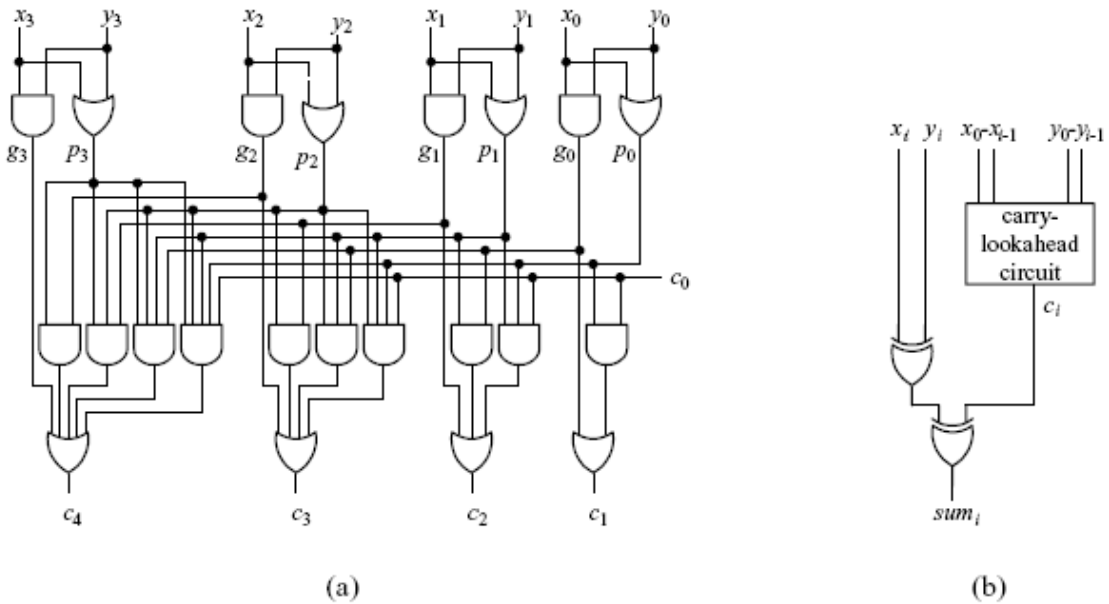
$$c_1 = g_0 + p_0 c_0$$

$$\begin{aligned} c_2 &= g_1 + p_1 c_1 \\ &= g_1 + p_1(g_0 + p_0 c_0) \\ &= g_1 + p_1 g_0 + p_1 p_0 c_0 \end{aligned}$$

$$\begin{aligned} c_3 &= g_2 + p_2 c_2 \\ &= g_2 + p_2(g_1 + p_1 g_0 + p_1 p_0 c_0) \\ &= g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0 \end{aligned}$$

$$\begin{aligned} c_4 &= g_3 + p_3 c_3 \\ &= g_3 + p_3(g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0) \\ &= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0 \end{aligned}$$

Từ biểu thức trên ta có thể vẽ được sơ đồ mạch cho bộ cộng Carry-Lookahead như sau:



Hình 2. 10 : (a) Mạch vận hành tín hiệu Carry-Lookahead từ c_1 đến c_4 ; (b) một mẫu bit của bộ cộng Carry-Lookahead.

2.4 Bộ trừ:

2.4.1 Bộ trừ một bit:

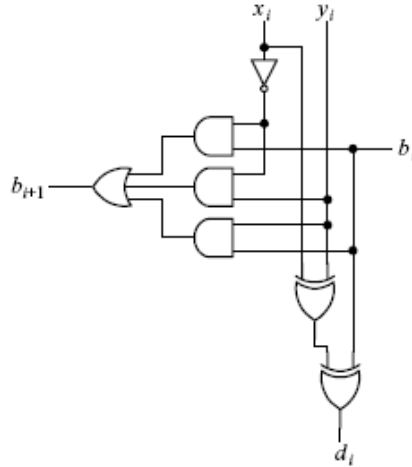
Chúng ta có thể xây dựng mạch cho bộ trừ thực hiện việc trừ một bit tương tự như phương pháp mà chúng ta đã dùng khi xây dựng bộ cộng toàn phần. Tuy nhiên bit tổng s_i trong phép cộng được thay thế bằng bit hiệu d_i trong phép trừ, và trạng thái nhớ ngõ vào, trạng

thái nhớ ngõ ra tín hiệu trong bộ cộng được thay thế bằng mượn ngõ vào (b_i) và mượn ngõ ra (b_{i+1}) tín hiệu. Do đó $d_i = x_i - y_i - b_i$ và $b_{i+1} = 1$ nếu chúng ta cần mượn trong phép trừ, những trường hợp khác $b_{i+1} = 0$. Bảng chân trị cho bộ trừ 1 bit được biểu diễn trong hình 2.11(a) thông qua hai biểu thức logic của d_i và b_{i+1} :

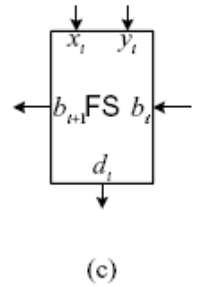
$$\begin{aligned} d_i &= x_i y' b_i + x_i' y_i b_i' + x_i y_i b_i' + x_i' y_i' b_i \\ &= (x_i' y_i + x_i y_i') b_i' + (x_i y_i' + x_i' y_i) b_i \\ &= (x_i \oplus y_i) b_i' + (x_i \oplus y_i) b_i \\ &= x_i \oplus y_i \oplus b_i \\ b_{i+1} &= x_i' y_i' b_i + x_i' y_i b_i' + x_i y_i' b_i + x_i y_i b_i \\ &= x_i' b_i (y_i' + y_i) + x_i y_i (b_i' + b_i) + y_i b_i (x_i' + x_i) \\ &= x_i' b_i + x_i y_i + y_i b_i \end{aligned}$$

x_i	y_i	b_i	b_{i+1}	d_i
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

(a)



(b)



(c)

Hình 2. 11 : Bộ trừ 1 bit (a) bảng chân trị; (b) sơ đồ mạch; (c) ký hiệu logic.

2.4.2 Sự tích hợp cả hai bộ cộng và bộ trừ trong cùng một mạch số:

Chúng ta có thể xây dựng một thành phần trong đó chứa đựng hai thành phần tách biệt là bộ cộng và bộ trừ bằng cách sửa đổi lại sơ đồ của mạch cộng Ripple Carry (hay có thể là mạch Carry-Lookahead Adder). Việc sửa đổi mạch để biểu diễn cho bộ trừ được thực hiện bằng cách cộng giá trị âm của hai toán hạng. Để phủ định một giá trị nhị phân như trong phần trước đã giới thiệu, chúng ta sẽ chuyển đổi tất cả các bit trong chuỗi bit từ 0 thành 1 và ngược lại một cách dễ dàng.

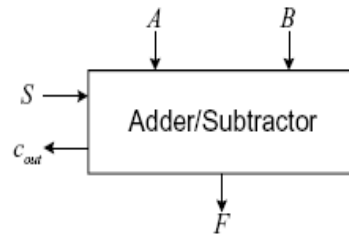
Ngõ vào S sẽ được dùng để lựa chọn chức năng của mạch là mạch cộng hay mạch trừ. Khi $S=1$ thì mạch hoạt động như một bộ trừ. Toán hạng B cần phải được đảo lại. Chúng ta nhớ rằng $x \oplus 1 = x'$, do đó để đảo B thì ta chỉ việc đem $B \oplus S = B'$ (khi $S=1$). Cuối cùng, việc cộng thêm 1 vào được thực hiện bằng cách bật bit nhớ tín hiệu c_0 lên 1. Mặt khác, để thực hiện chức năng của bộ cộng thì $S=0$. Khi $S=0$ thì toán hạng B sẽ không cần phải đảo trang

thái điều này được thực hiện nhờ cổng XOR. Trong trường hợp này, chúng ta cũng muốn $c_0 = S = 0$.

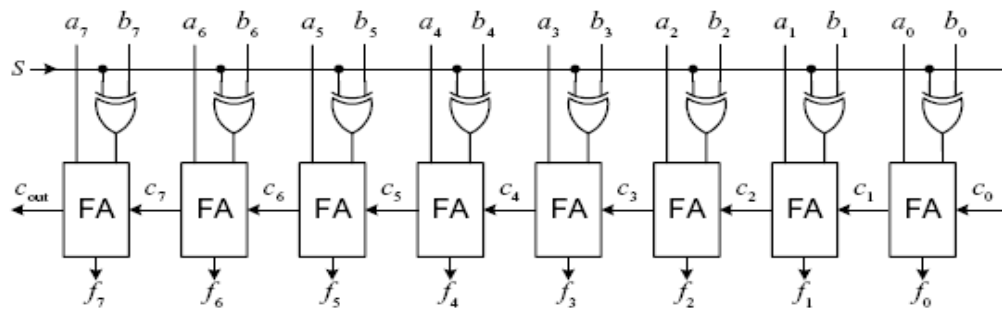
Một mạch thực hiện cả hai việc cộng và trừ được biểu diễn trong hình 2.12(b) và ký hiệu logic của nó được thể hiện trong hình 2.12(c).

S	Function	Operation
0	Add	$F = A + B$
1	Subtract	$F = A + B' + 1$

(a)



(c)



(b)

Hình 2. 12 : Mạch cộng và trừ chuỗi 8 bit nhị phân (a) bảng vận trị; (b) sơ đồ mạch; (c) ký hiệu logic.

Mã VHDL viết theo cấu trúc Behavioral cho mạch cộng và trừ chuỗi 8 bit nhị phân:

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY AddSub IS
  GENERIC(n: NATURAL :=8); -- default number of bits = 8
  PORT(A: IN std_logic_vector(n-1 downto 0);
        B: IN std_logic_vector(n-1 downto 0);
        subtract: IN std_logic;
        carry: OUT std_logic;
        sum: OUT std_logic_vector(n-1 downto 0));
END AddSub;

ARCHITECTURE Behavioral OF AddSub IS
  -- temporary result with one extra bit for carry
  SIGNAL result: std_logic_vector(n downto 0);
BEGIN
  PROCESS(subtract, A, B)
  BEGIN
    IF (subtract = '0') THEN -- addition
      --add the two operands with one extra bit for carry
      result <= ('0' & A)+('0' & B);
      sum <= result(n-1 downto 0); -- extract the n-bit result
      carry <= result(n); -- extract the carry bit from result
    ELSE -- subtraction
      result <= ('0' & A)-('0' & B);
      sum <= result(n-1 downto 0); -- extract the n-bit result
      carry <= result(n); -- extract the borrow bit from result
    END IF;
  END Behavioral;

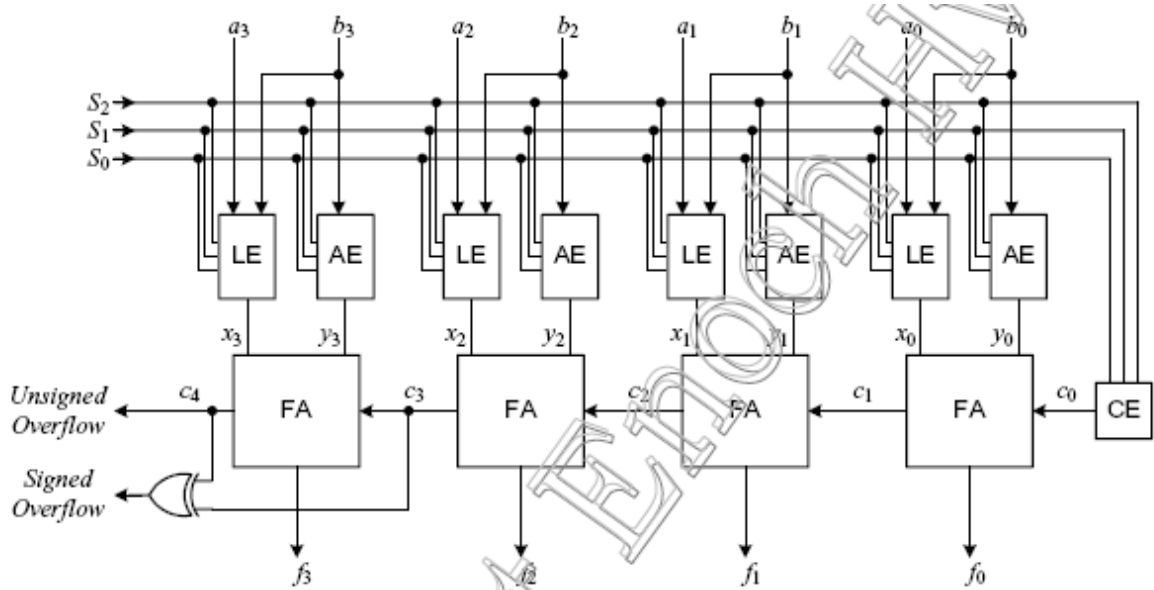
```

2.5 Thành phần thực hiện các phép toán logic số học (ALU):

Thành phần này gọi tắt là khối ALU là một trong những thành phần quan trọng trong một bộ vi xử lý, chúng đảm nhận trách nhiệm thực hiện các hoạt động liên quan đến số học hay các sự hoạt động của các mạch logic như bộ cộng, bộ trừ, cổng logic AND, OR. Để xây dựng mạch cho khối ALU, chúng ta có thể dùng cùng một ý tưởng như khi xây dựng cho một mạch thực hiện cả hai chức năng của bộ cộng và bộ trừ như đã được trình bày ở phần trên. Một lần nữa, chúng ta sẽ sử dụng bộ cộng Ripple Carry khi xây dựng sơ đồ khối và sau đó ta chèn thêm các mạch kết nối logic vào phía trước hai ngõ vào thuật toán của bộ cộng. Theo cách này, các ngõ vào sơ cấp sẽ được sửa đổi cho phù hợp với sự hoạt động mà nó cần biểu diễn trước khi nó đi qua bộ cộng toàn phần. Toàn bộ sơ đồ vận hành mạch của bộ ALU 4 bit được biểu diễn trong hình 2.13.

Chúng ta quan sát hình 2.13 thấy có hai mạch kết nối ở phía trước bộ cộng toàn phần (FA) đó là LE và AE. Bộ LE (logic extender) là bộ dùng để điều khiển tất cả mọi hoạt động logic, trong khi bộ AE (arithmetic extender) là bộ dùng để điều khiển tất cả mọi hoạt động về số học. Bộ LE thực thi các hoạt động logic chính xác từ hai ngõ vào sơ cấp là a_i và b_i trước khi kết quả của nó vượt ra ngõ x_i của bộ FA. Hay nói cách khác, bộ AE chỉ sửa đổi

ngõ vào b_i và cho ra kết quả tại ngõ ra y_i của bộ FA. Bộ FA nhận giá trị x_i và y_i sẽ thực thi các phép toán số học chính xác.



Hình 2. 13 : Mạch ALU 4 bit.

Chúng ta thấy từ sơ đồ mạch phối hợp bộ cộng và bộ trừ cho đến việc cộng và trừ, chúng ta chỉ cần sửa lại ngõ vào thứ hai của bộ FA y_i mà thôi, vì thế tất cả các hoạt động có thể được thực hiện với phép cộng. Do đó, bộ AE chỉ lấy giá trị của ngõ vào b_i và sửa đổi nó phù hợp với sự hoạt động được thực thi của mạch. Ngõ ra y_i của nó sẽ được kết nối đến ngõ vào thứ hai của bộ FA. Trong mạch phối hợp bộ cộng và bộ trừ, việc cộng được thực thi trong bộ FA. Khi các phép toán số học đang được thực thi, bộ LE phải cho tín hiệu qua mà không làm thay đổi từ ngõ vào sơ cấp a_i đến x_i trong bộ FA.

Không giống như bộ AE nơi chỉ sửa đổi các thuật toán, bộ LE thực thi các hoạt động logic chính xác. Do đó, nếu chúng ta muốn thực hiện phép toán A OR B, thì bộ LE sẽ lấy từng cặp bit thứ a_i và b_i (của A và B) thực hiện phép Or chúng lại với nhau. Do đó, bộ LE có hai ngõ vào a_i và b_i và ngõ ra của nó là x_i . Khi giá trị chuẩn bị hiển thị kết quả của các hoạt động logic, chúng ta không muốn bộ FA sửa đổi nó, mà đưa giá trị này hiển thị ra ngõ ra sơ cấp f_i . Điều này có thể thực hiện được bằng cách chúng ta bật cả hai giá trị y_i của bộ FA và c_0 xuống 0. Khi đó việc cộng bất kỳ số nào với 0 cũng không làm thay đổi giá trị ban đầu của nó.

Bộ CE đảm nhận việc nhớ trạng thái tín hiệu sơ cấp c_0 , giúp cho việc thực thi mạch được thực hiện một cách chính xác. Trong các hoạt động logic thì trạng thái bit nhớ tín hiệu c_0 luôn được đặt bằng 0.

S_2	S_1	S_0	Operation Name	Operation	X (LE)	Y (AE)	c_0 (CE)
0	0	0	Pass	Pass A to output	A	0	0
0	0	1	AND	$A \text{ AND } B$	$A \text{ AND } B$	0	0
0	1	0	OR	$A \text{ OR } B$	$A \text{ OR } B$	0	0
0	1	1	NOT	A'	A'	0	0
1	0	0	Addition	$A + B$	A	B	0
1	0	1	Subtraction	$A - B$	A	B'	1
1	1	0	Increment	$A + 1$	A	0	1
1	1	1	Decrement	$A - 1$	A	1	0

(a)

S_2	S_1	S_0	x_i
0	0	0	a_i
0	0	1	$a_i b_i$
0	1	0	$a_i + b_i$
0	1	1	a_i'
1	\times	\times	a_i

(b)

S_2	S_1	S_0	b_i	y_i
0	\times	\times	\times	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

(c)

S_2	S_1	S_0	c_0
0	\times	\times	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

(d)

Hình 2. 14 : Hoạt động của khối ALU (a) Bảng các trạng thái; (b) Bảng chân trị của LE; (c) Bảng chân trị của AE; (d) Bảng chân trị của CE.

Trong hình 2.14 các đường tín hiệu S_2, S_1 và S_0 được dùng để lựa chọn sự hoạt động của khối ALU. $S_2 = 1$, thì khối ALU sẽ được chọn lựa hoạt động theo các phép toán số học. $S_2 = 0$, thì khối ALU sẽ được chọn lựa hoạt động theo các phép toán logic. Hai đường S_0 và S_1 cho phép lựa chọn một trong bốn bộ LE hay AE hoạt động. Do đó mạch ALU có thể có 8 sự hoạt động khác nhau đó là:

- Cho một tín hiệu nào đó đi qua mà không làm đổi giá trị của nó.
- Thực hiện phép toán logic AND hai số A và B.
- Thực hiện phép toán logic OR hai số A và B.
- Thực hiện phép toán logic NOT cho một số nào đó.
- Thực hiện phép cộng hai số A và B.
- Thực hiện phép trừ hai số A và B.
- Thực hiện phép cộng 1 vào một số A.
- Thực hiện phép trừ A cho 1.

Trong 8 hoạt động trên, mỗi hoạt động sẽ có các giá trị X, Y và c_0 khác nhau.

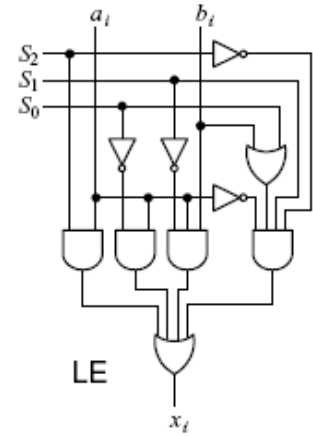
Trong các hình 2.14(b), (c) và (d) ta thấy x_i chỉ phụ thuộc vào 5 biến số S_2, S_1, S_0, a_i và b_i ; y_i chỉ phụ thuộc vào 4 biến số S_2, S_1, S_0 và b_i ; và c_0 chỉ phụ thuộc vào 3 ngõ vào lựa

chọn S_2, S_1, S_0 . Bìa Karnaugh và biểu thức tương ứng cho các biểu thức x_i , y_i và c_0 được biểu diễn trong hình 2.15.

x_i $a_i b_i$		$S_2 = 0$				$S_2 = 1$			
		00	01	11	10	00	01	11	10
$S_1 S_0$									
00	0	1	3	2	16	17	19	18	
01	4	5	7	6	20	21	23	22	
11	12	13	15	14	28	29	31	30	
10	8	9	11	10	24	25	27	26	

$$\begin{aligned} x_i &= S_2 a_i + S_0' a_i + S_1' a_i b_i + S_2' S_1 S_0 a_i' + S_2' S_1 a_i' b_i \\ &= S_2 a_i + S_0' a_i + S_1' a_i b_i + S_2' S_1 a_i' (S_0 + b_i) \end{aligned}$$

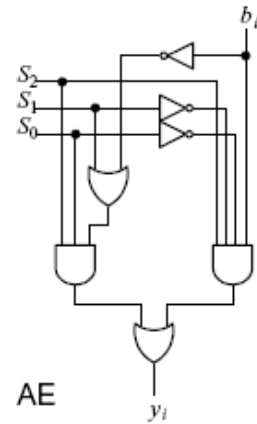
(a)



y_i $S_0 b_i$		00	01	11	10
$S_2 S_1$					
00	0	1	3	2	
01	4	5	7	6	
11	12	13	15	14	
10	8	9	11	10	

$$\begin{aligned} y_i &= S_2 S_1 S_0 + S_2 S_0 b_i' + S_2 S_1' S_0' b_i \\ &= S_2 S_0 (S_1 + b_i') + S_2 S_1' S_0' b_i \end{aligned}$$

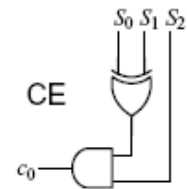
(b)



c_0 $S_1 S_0$		00	01	11	10
S_2					
0	0	1	3	2	
1	4	5	7	6	

$$\begin{aligned} c_0 &= S_2 S_1' S_0 + S_2 S_1 S_0' \\ &= S_2 (S_1 \oplus S_0) \end{aligned}$$

(c)



Hình 2. 15 : Bìa karnaugh, biểu thức, sơ đồ mạch cho: (a) LE; (b) AE; (c) CE.

Đoạn mã VHDL của bộ ALU được viết theo cấu trúc behavioral được biểu diễn trong hình 2.16. Dạng sóng mô phỏng các thuật toán của khối ALU khi hai số ngõ vào là 5 và 3 được biểu diễn trong hình 2.17.


```

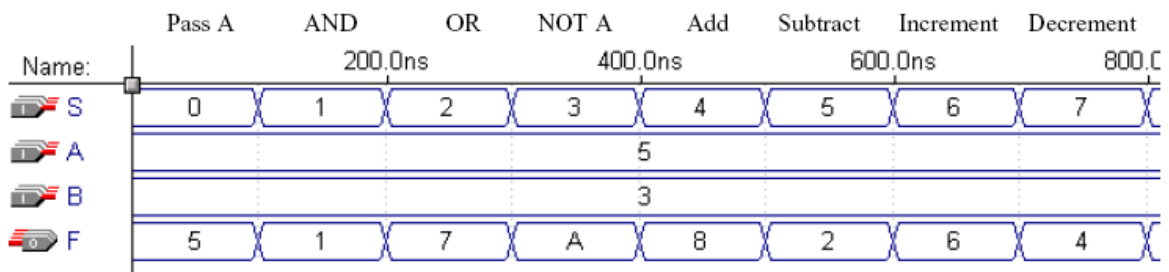
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-- The following package is needed so that the STD_LOGIC_VECTOR signals
-- A and B can be used in unsigned arithmetic operations.
USE ieee.std_logic_unsigned.all;

ENTITY alu IS PORT (
    S: IN std_logic_vector(2 downto 0);    -- select for operations
    A, B: IN std_logic_vector(3 downto 0); -- input operands
    F: OUT std_logic_vector(3 downto 0)); -- output
END alu;

ARCHITECTURE Behavior OF alu IS
BEGIN
    PROCESS(S, A, B)
    BEGIN
        CASE S IS
            WHEN "000" => -- pass A through
                F <= A;
            WHEN "001" => -- AND
                F <= A AND B;
            WHEN "010" => -- OR
                F <= A OR B;
            WHEN "011" => -- NOT A
                F <= NOT A;
            WHEN "100" => -- add
                F <= A + B;
            WHEN "101" => -- subtract
                F <= A - B;
            WHEN "110" => -- increment
                F <= A + 1;
            WHEN OTHERS => -- decrement
                F <= A - 1;
        END CASE;
    END PROCESS;
END Behavior;

```

Hình 2. 16: Đoạn mã VHDL cho một khối ALU.



Hình 2. 17 : Dạng sóng mô phỏng cho 8 thuật toán cơ bản của khối ALU với hai giá trị ngõ vào là 5 và 3.

2.6 Bộ giải mã:

Bộ giải mã còn được gọi là bộ phân kênh, nó có n ngõ ra, số ngõ ra này phụ thuộc vào số m bit lựa chọn ở ngõ vào. Mỗi quan hệ giữa n và m là $n = 2^m$. Trong bộ giải mã có thêm một

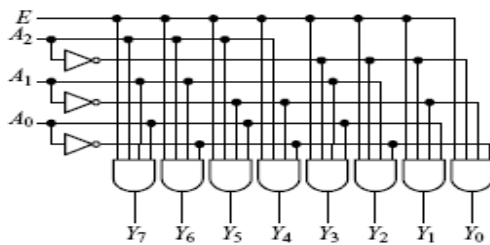
đường Enable để cho phép bộ giải mã hoạt động hay ngừng hoạt động. Khi $E=0$ thì tất cả các ngõ ra đều mang giá trị 0. Khi $E=1$ thì bộ giải mã sẽ hoạt động, nó sẽ lựa chọn ngõ ra nào đó để đưa dữ liệu đến tùy thuộc vào các ngõ vào lựa chọn m. Ví dụ một bộ giải mã 3 sang 8. Nếu ngõ vào địa chỉ là 101 thì ngõ ra Y_5 được lựa chọn để đưa dữ liệu ra (Y_5 lên mức cao), trong khi đó tất cả các ngõ ra còn lại đều không được lựa chọn (tích cực mức thấp).

Một bộ giải mã thường dùng rất nhiều thành phần và chúng ta muốn tại mỗi thời điểm chỉ có một thành phần được cho phép hoạt động mà thôi. Ví dụ trong một hệ thống nhớ lớn sử dụng nhiều con chip nhớ, tại mỗi thời điểm chỉ có một con chip nhớ được tích cực cho phép hoạt động mà thôi. Một ngõ ra của bộ giải mã sẽ được nối đến một ngõ vào tích cực trong mỗi con chip. Một địa chỉ được tạo ra từ bộ giải mã sẽ làm tích cực một con chip nhớ tương ứng. Bảng chân trị, sơ đồ mạch và ký hiệu logic của bộ giải mã 3 sang 8 được biểu diễn trong hình 2.18.

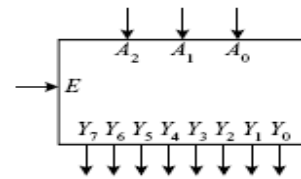
Một bộ giải mã kích cỡ lớn có thể sử dụng một vài các bộ giải mã nhỏ hơn. Ví dụ trong hình 2.19 sử dụng 7 bộ giải mã 1 sang 2 để xây dựng bộ giải mã 3 sang 8.

E	A_2	A_1	A_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	×	×	×	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

(a)

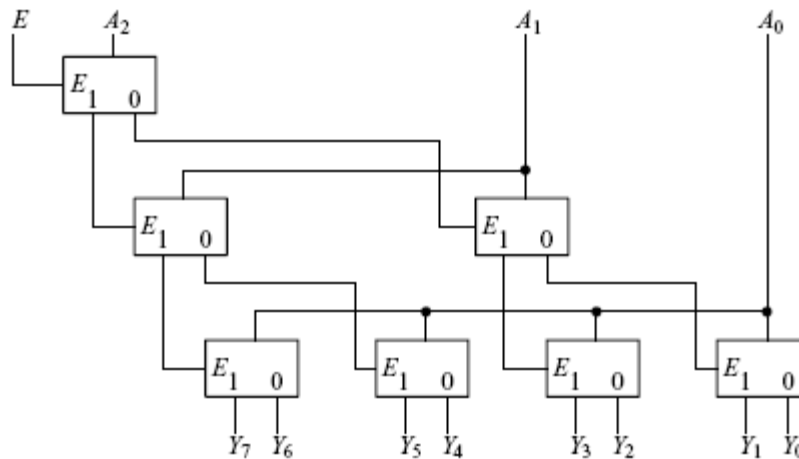


(b)



(c)

Hình 2. 18 : Một bộ giải mã 3 sang 8 (a) Bảng chân trị; (b) sơ đồ mạch; (c) ký hiệu logic.



Hình 2. 19 : Một bộ giải mã 3 sang 8 được xây dựng từ 7 bộ giải mã 1 sang 2.

Đoạn mã VHDL được viết theo cấu trúc Behavioral cho bộ giải mã 3 sang 8.

```
-- A 3-to-8 decoder
LIBRARY ieee;
USE IEEE.std_logic_1164.all;

ENTITY Decoder IS PORT(
    E: IN std_logic;           -- enable
    A: IN std_logic_vector(2 DOWNTO 0); -- 3 bit address
    Y: OUT std_logic_vector(7 DOWNTO 0)); -- data bus output
END Decoder;

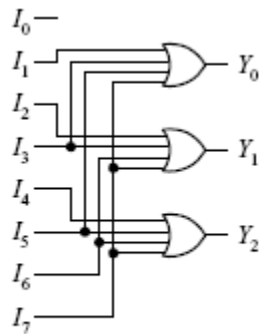
ARCHITECTURE Behavioral OF Decoder IS
BEGIN
    PROCESS (E, A)
    BEGIN
        IF (E = '0') THEN      -- disabled
            Y <= (OTHERS => '0'); -- 8-bit vector of 0
        ELSE
            CASE A IS          -- enabled
                WHEN "000" => Y <= "00000001";
                WHEN "001" => Y <= "00000010";
                WHEN "010" => Y <= "00000100";
                WHEN "011" => Y <= "00001000";
                WHEN "100" => Y <= "00010000";
                WHEN "101" => Y <= "00100000";
                WHEN "110" => Y <= "01000000";
                WHEN "111" => Y <= "10000000";
                WHEN OTHERS => NULL;
            END CASE;
        END IF;
    END PROCESS;
END Behavioral;
```

2.7 Bộ mã hóa:

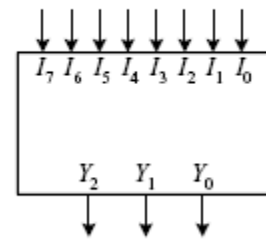
Một bộ mã hóa được xem như là sự đảo ngược của một bộ giải mã. Bộ mã hóa sẽ mã hóa 2^n bit dữ liệu ngõ vào thành mã n bit. Sự hoạt động của bộ mã hóa được hiểu như sau: chỉ một đường ngõ vào được bật lên 1, tất cả các ngõ vào còn lại đều bằng 0 ứng với một giá trị của Y_0, Y_1, Y_2 . Ví dụ khi $I_3=1$ thì 3 giá trị $Y_2Y_1Y_0 = 011$.

I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

(a)



(b)



(c)

Hình 2. 20 : Một bộ mã hóa 8 sang 3 (a) Bảng chân trị; (b) sơ đồ mạch; (c) ký hiệu logic.

Khuyết điểm của bộ mã hóa này là ở chỗ: Nếu hai hay nhiều ngõ vào I_i cùng được tích cực tại một thời điểm, thì ngõ ra sẽ mã hóa sai ngay lập tức. Ví dụ nếu ngõ vào 1 và 4 của bộ mã hóa 8 sang 3 cùng được tích cực một lúc. Khi đó Y_2 vừa nhận giá trị 1 của I_4 vừa mang giá trị 0 của I_1 . Để giải quyết vấn đề này, người ta mới đặt ra một quy luật ưu tiên cho thứ tự thực hiện của các ngõ vào I_i .

I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	Y_2	Y_1	Y_0	Z
0	0	0	0	0	0	0	0	×	×	×	0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	1	×	0	0	1	1
0	0	0	0	0	1	×	×	0	1	0	1
0	0	0	0	1	×	×	×	0	1	1	1
0	0	0	1	×	×	×	×	1	0	0	1
0	0	1	×	×	×	×	×	1	0	1	1
0	1	×	×	×	×	×	×	1	1	0	1
1	×	×	×	×	×	×	×	1	1	1	1

Hình 2. 21 : Bảng chân trị cho một bộ mã hóa 8 sang 3 có sự ưu tiên.

Từ hàng thứ 3 đến hàng thứ 10 trong hình 2.21 ta có thể viết được các biểu thức sau:

$$\begin{aligned}
 v_0 &= I_7' I_6' I_5' I_4' I_3' I_2' I_1' I_0 \\
 v_1 &= I_7' I_6' I_5' I_4' I_3' I_2' I_1 \\
 v_2 &= I_7' I_6' I_5' I_4' I_3' I_2 \\
 v_3 &= I_7' I_6' I_5' I_4' I_3 \\
 v_4 &= I_7' I_6' I_5' I_4 \\
 v_5 &= I_7' I_6' I_5 \\
 v_6 &= I_7' I_6 \\
 v_7 &= I_7
 \end{aligned}$$

Suy ra

$$\begin{aligned}
 Y_0 &= v_1 + v_3 + v_5 + v_7 \\
 Y_1 &= v_2 + v_3 + v_6 + v_7 \\
 Y_2 &= v_4 + v_5 + v_6 + v_7
 \end{aligned}$$

Từ bảng chân trị hình 2.21 ta có thể viết được biểu thức Z như sau:

$$Z = I_7 + I_6 + I_5 + I_4 + I_3 + I_2 + I_1 + I_0$$

Trong bảng chân trị hình 2.21 ta thấy I_7 là có độ ưu tiên cao nhất và I_0 là có độ ưu tiên thấp nhất. Ví dụ nếu ngõ vào I_3 có độ ưu tiên cao nhất thì ta không cần quan tâm đến những ngõ vào có độ ưu tiên thấp hơn là I_2, I_1, I_0 có trạng thái là 0 hay 1, ngõ ra của mạch sẽ là I_3 tức là giá trị nhị phân 011. Trong trường hợp không có ngõ vào nào được chọn thì ta cần thêm một ngõ ra Z để phân biệt sự khác nhau giữa trường hợp không có ngõ vào nào được chọn và trường hợp có một hay nhiều ngõ vào được chọn. Z=1 khi có một hay nhiều ngõ vào được chọn, trường hợp còn lại Z=0. Khi Z=0 tất cả các ngõ ra Y đều vô nghĩa.

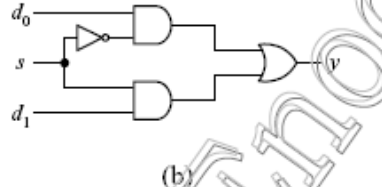
2.8 Bộ ghép kênh:

Bộ ghép kênh còn gọi là bộ MUX cho phép lựa chọn một tín hiệu ngõ vào từ n ngõ vào. Do đó n là ký hiệu cho các tín hiệu ngõ vào, s là ký hiệu cho ngõ vào lựa chọn. Mối quan hệ giữa n và s là $2^s = n$. Cấu trúc của một bộ ghép kênh 2 sang 1 đã được giới thiệu kỹ trong các chương trước, trong phần này ta sẽ nhắc lại cách xây dựng đó để minh họa mà thôi.

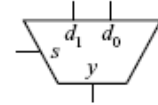
$$\begin{aligned} y &= s'd_1'd_0 + s'd_1d_0 + sd_1d_0' + sd_1d_0 \\ &= s'd_0(d_1' + d_1) + sd_1(d_0' + d_0) \\ &= s'd_0 + sd_1 \end{aligned}$$

s	d ₁	d ₀	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

(a)



(b)



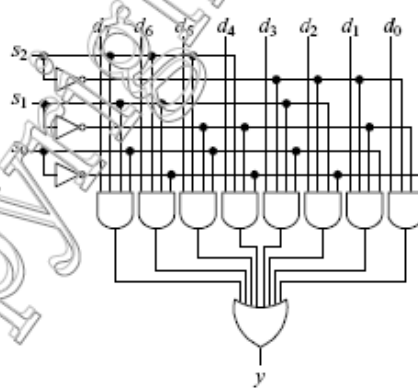
(c)

Hình 2. 22 : Bộ ghép kênh từ 2 sang 1 (a) Bảng chân trị; (b) sơ đồ mạch; (c) ký hiệu logic. Từ việc giới thiệu cách xây dựng bộ ghép kênh 2 sang 1 ta có thể xây dựng các bộ ghép kênh có kích thước lớn hơn như bộ ghép kênh 8 sang 1 tương tự như cách đã làm ở trên. Có 8 đường dữ liệu ngõ vào cho nên sẽ có 3 ngõ vào lựa chọn. Tùy thuộc vào giá trị nhị phân của 3 ngõ vào lựa chọn mà một trong 8 ngõ vào dữ liệu sẽ được chọn để đưa giá trị từ ngõ vào ấy đến ngõ ra. Ví dụ nếu giá trị lựa chọn là 101 thì ngõ vào d_5 được chọn và dữ liệu của d_5 sẽ được đưa đến ngõ ra.

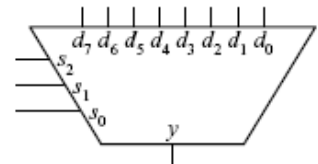
Để lập bảng chân trị cho bộ ghép kênh 8 sang 1 thì ta chú ý có 3 bit lựa chọn nên sẽ có $2^3=8$ trạng thái khác của ngõ vào dữ liệu.

s ₂	s ₁	s ₀	y
0	0	0	d ₀
0	0	1	d ₁
0	1	0	d ₂
0	1	1	d ₃
1	0	0	d ₄
1	0	1	d ₅
1	1	0	d ₆
1	1	1	d ₇

(a)

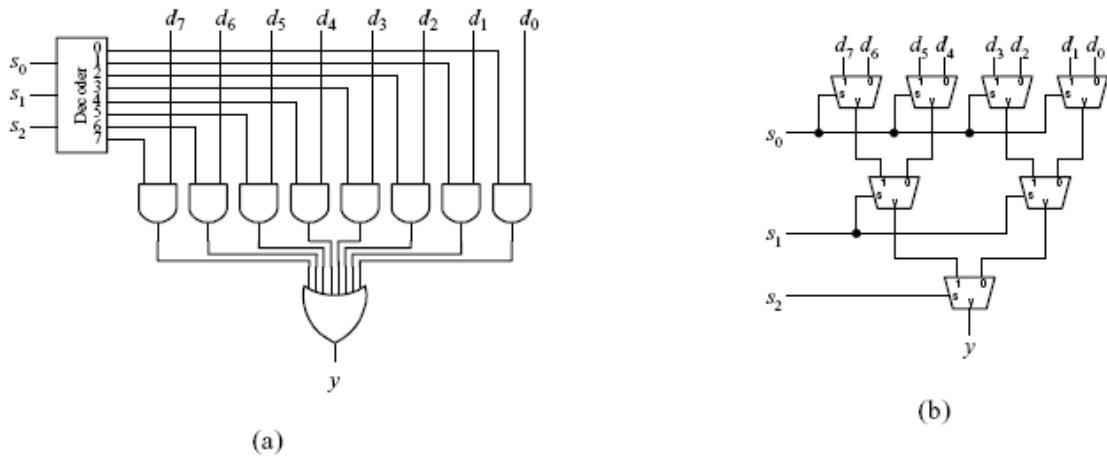


(b)



(c)

Hình 2. 23 : Bộ ghép kênh 8 sang 1 (a) Bảng chân trị; (b) sơ đồ mạch; (c) ký hiệu logic.



Hình 2. 24 : Bộ ghép kênh 8 sang 1 có sử dụng (a) Bộ giải mã 3 sang 8; (b) 7 bộ ghép kênh 2 sang 1.

Trong hình 2.23(b) ta phải hiểu rằng cổng AND đóng vai trò là một công tắc và nó được điều khiển bật bởi 3 ngõ vào lựa chọn dữ liệu. Khi cổng AND này được bật lên 1 dữ liệu d tương ứng qua cổng đó sẽ được đưa tới chân y tại ngõ ra, cùng thời điểm này tất cả các cổng AND còn lại đều nhận giá trị 0.

Trong mạch hình 2.23(b) ta còn sử dụng các cổng AND 4 ngõ vào trong đó có 3 ngõ vào lựa chọn dữ liệu để tích cực cổng. Chúng ta cũng có thể sử dụng cổng AND 2 ngõ vào như trong hình 2.24(a), một ngõ vào dữ liệu và một ngõ vào lựa chọn được điều khiển từ ngõ ra của bộ giải mã 3 sang 8. Tại mỗi thời điểm chỉ có một trong 8 ngõ ra bộ giải mã ở trạng thái tích cực các ngõ ra còn lại đều ở mức thấp.

Trong những bộ ghép kênh lớn hơn ta có thể được xây dựng từ các bộ ghép kênh nhỏ hơn. Ví dụ bộ ghép kênh 8 thành 1 có thể được xây dựng bằng 7 bộ ghép kênh 2 thành 1 như trong hình 2.24(b), 4 bộ ghép kênh 2 thành 1 ở trên cùng cung cấp 8 ngõ vào dữ liệu, và chúng được điều khiển bởi một đường lựa chọn duy nhất là s_0 . Ở cấp độ này nó sẽ lựa chọn một từ mỗi nhóm hai ngõ vào dữ liệu. Nhóm các bộ ghép kênh ở giữa cũng làm nhiệm vụ tương tự như 4 bộ ghép kênh ở trên và được điều khiển bởi đường s_1 . Cuối cùng là bộ ghép kênh ở dưới cùng sử dụng đường điều khiển là s_2 để lựa chọn một trong hai ngõ ra của bộ ghép kênh ở tầng giữa.

Ngôn ngữ VHDL cho bộ ghép kênh 4 thành 1 trong đó mỗi kênh có 8 bit được biểu diễn ở dưới đây. Hai cách viết khác nhau của cùng một bộ ghép kênh cũng được thể hiện: Cách 1: viết ở cấp độ behavioral, dùng một câu lệnh Process; Cách 2: viết theo cấp độ Dataflow, dùng câu lệnh gán tín hiệu đã lựa chọn đồng thời.

```
-- A 4-to-1 8-bit wide multiplexer
LIBRARY ieee;
USE IEEE.std_logic_1164.all;

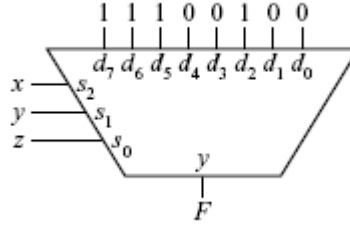
ENTITY Multiplexer IS
    PORT(S: IN std_logic_vector(1 DOWNTO 0);           -- select lines
          D0, D1, D2, D3: IN std_logic_vector(7 DOWNTO 0); -- data bus input
          Y: OUT std_logic_vector(7 DOWNTO 0));         -- data bus output
END Multiplexer;
```

```
-- Behavioral level code
ARCHITECTURE Behavioral OF Multiplexer IS
BEGIN
    PROCESS (S,D0,D1,D2,D3)
    BEGIN
        CASE S IS
            WHEN "00" => Y <= D0;
            WHEN "01" => Y <= D1;
            WHEN "10" => Y <= D2;
            WHEN "11" => Y <= D3;
            WHEN OTHERS => Y <= (OTHERS => 'U');           -- 8-bit vector of U
        END CASE;
    END PROCESS;
END Behavioral;

-- Dataflow level code
ARCHITECTURE Dataflow OF Multiplexer IS
BEGIN
    WITH S SELECT Y <=
        D0 WHEN "00",
        D1 WHEN "01",
        D2 WHEN "10",
        D3 WHEN "11",
        (OTHERS => 'U') WHEN OTHERS;                       -- 8-bit vector of U
END Dataflow;
```

Dùng bộ ghép kênh để biểu diễn một hàm:

Bộ ghép kênh có thể được dùng để biểu diễn một biểu thức Boolean một cách dễ dàng. Biểu thức có n biến thì ta sử dụng bộ ghép kênh có 2^n ngõ vào và n đường tín hiệu điều khiển lựa chọn, n biến ngõ vào sẽ được kết nối với n ngõ vào lựa chọn của bộ ghép kênh. Tùy theo giá trị của biến n mà một đường dữ liệu ngõ vào sẽ được lựa chọn và giá trị từ ngõ vào đó sẽ được đưa đến ngõ ra. Vì thế chúng ta cần phải kết nối 2^n đường dữ liệu d với các giá trị 0 hoặc 1. Khi d thứ i bằng 1 tức là ta sẽ viết được một tích số của các biến x, y, z tương ứng với giá trị tại d thứ i đó. Để hiểu rõ hơn về điều này ta sẽ xét một ví dụ dùng bộ ghép kênh để biểu diễn một hàm sau đây:



Hình 2. 25 : Dùng bộ ghép kênh 8 thành 1 biểu diễn hàm
 $F(x, y, z) = x'yz' + xy'z + xyz' + xyz.$

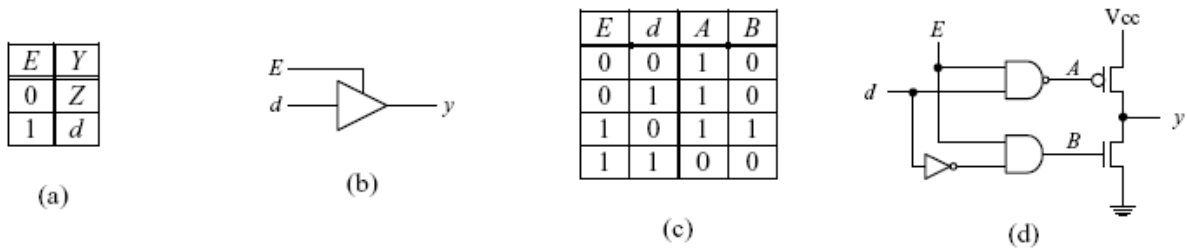
2.9 Bộ đệm ba trạng thái:

Bộ đệm ba trạng thái là bộ có ba trạng thái: 0, 1 và trạng thái thứ ba được biểu thị bằng Z. Giá trị của Z đặc trưng cho trạng thái trở kháng cao. Bộ đệm ba trạng thái được dùng để kết nối một vài dịch vụ trên cùng một bus. Một bus có một hay nhiều đường dây truyền tín hiệu. Nếu hai hay nhiều dịch vụ được kết nối một cách trực tiếp đến một bus mà không sử dụng bộ đệm ba trạng thái, thì những tín hiệu này sẽ bị sai lạc trong bus đó. Trong bộ đệm ba trạng thái có sử dụng chân E (enable) để điều khiển hoạt động của nó. Khi E=0, bộ đệm ba trạng thái không được tích cực và ngõ ra y ở trạng thái trở kháng cao. Khi E=1, bộ đệm được cho phép hoạt động và ngõ vào d sẽ đưa được dữ liệu của nó đến ngõ ra y.

Một mạch chỉ với một cổng logic thì không thể tạo ra trạng thái trở kháng cao được. Do đó để cung cấp trạng thái trở kháng cao, mạch đệm ba trạng thái sử dụng hai con transistor CMOS đặc biệt kết nối với các cổng logic như trong hình 2.26(d). Trong chương 5 ta sẽ giới thiệu chi tiết hơn về họ transistor CMOS này. Chương này ta sẽ chỉ giới thiệu họ CMOS một cách đơn giản mà thôi. Transistor pMOS ở trên dẫn khi ở mức 0, tức A=0. Khi nó dẫn thì tín hiệu mức cao từ nguồn V_{cc} sẽ được đưa ra y. Transistor nMOS ở phía dưới dẫn khi B=1, và tín hiệu mức thấp từ đất sẽ được đưa đến ngõ ra y. Khi cả hai con transistor này đều không dẫn ngõ ra y ở trạng thái trở kháng cao.

Việc có thêm hai con transistor họ CMOS này, chúng ta cần phải có một mạch để điều khiển chúng hoạt động liên kết với nhau để tạo thành một bộ đệm ba trạng thái. Bảng chân trị của mạch điều khiển được cho trong hình 2.26(c).

Khi E=0, cả hai con transistor CMOS đều tắt, y ở trạng thái trở kháng cao. Khi E=1 và d=0, nếu chúng ta muốn y=0 thì nMOS dẫn và pMOS tắt, lúc này ngõ ra y sẽ bị kéo xuống thấp; nếu chúng ta muốn y=1 thì nMOS tắt và pMOS dẫn, lúc này ngõ ra y sẽ bị đẩy lên mức cao do y được nối lên nguồn. Mạch có hai ngõ vào E và d nên bảng chân trị sẽ có 4 cặp giá trị là 00, 01, 10, 11. Dựa vào các giá trị này cộng thêm cách giải thích về hoạt động của các con transistor CMOS ta có thể hoàn thành bảng chân trị như hình 2.26(c).



Hình 2. 26 : Bộ đệm ba trạng thái (a) bảng chân trị; (b) ký hiệu logic; (c) bảng chân trị cho việc phân chia điều khiển cho mạch đệm ba trạng thái; (d) sơ đồ mạch.

Đoạn mã VHDL viết theo cấu trúc Behavioral cho bộ đệm ba trạng thái.

```

LIBRARY ieee;
USE IEEE.std_logic_1164.ALL;

ENTITY TriState_Buffer IS PORT (
    E: IN std_logic;
    d: IN std_logic_vector(7 DOWNTO 0);
    y: OUT std_logic_vector(7 DOWNTO 0));
END TriState_Buffer;

ARCHITECTURE Behavioral OF TriState_Buffer IS
BEGIN
    PROCESS (E, d)
    BEGIN
        IF (E = '1') THEN
            y <= d;
        ELSE
            y <= (OTHERS => 'Z');    -- to get 8 Z values
        END IF;
    END PROCESS;
END Behavioral;
    
```

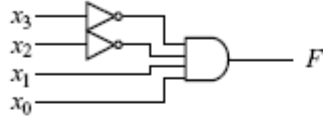
2.10 Bộ so sánh:

Thông thường để so sánh hai giá trị với nhau thì ta dùng các thuật ngữ bằng, lớn hơn và nhỏ hơn để nói lên kết quả của phép so sánh này. Một bộ so sánh là một mạch thực hiện nhiệm vụ so sánh hai số nhị phân gồm có nhiều bit. Để so sánh một giá trị là bằng hay không bằng một giá trị hằng số, thì một cổng AND cơ bản cần phải được sử dụng. Ví dụ để so sánh một biến x gồm 4 bit với hằng số cho trước là 3, mạch điện thực hiện phép so sánh này sẽ được biểu diễn trong hình 2.27(a). Ngõ ra của cổng AND là 1 khi ngõ vào của nó bằng với giá trị 3.

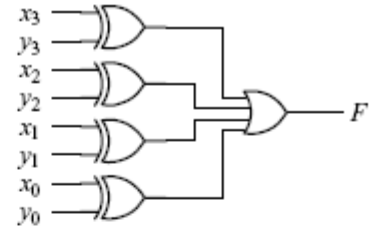
Các cổng XOR và XNOR có thể được sử dụng cho việc so sánh sự không bằng nhau hay sự bằng nhau giữa hai giá trị một cách thích hợp. Ngõ ra cổng XOR là 1 khi cả hai giá trị ngõ vào là khác nhau. Vì thế ta có thể sử dụng cổng XOR để so sánh từng cặp bit một của chuỗi bit nhị phân. Một bộ so sánh 4 bit không ngang bằng nhau được biểu diễn trong hình

2.27(b). trong hình 2.27(b), 4 cổng XOR được sử dụng, mỗi cổng XOR sẽ so sánh một cặp bit trong chuỗi bit nhị phân gồm 4 bit. Ngõ ra của các cổng XOR được nối đến cổng OR có 4 ngõ vào, vì thế nếu một trong các cặp bit tương ứng của hai số nhị phân mà khác nhau, thì kết quả ngõ ra F sẽ bằng 1. Tương tự như vậy, một bộ so sánh bằng nhau có thể được xây dựng bằng cách sử dụng cổng XNOR. Ngõ ra cổng XNOR bằng 1 khi cả hai ngõ vào của nó có cùng giá trị.

Để so sánh lớn hơn và nhỏ hơn, chúng ta có thể xây dựng một bảng chân trị và xây dựng sơ đồ mạch từ phương pháp thông thường. Ví dụ so sánh một số X gồm 4 bit nhỏ hơn 5, bảng chân trị, biểu thức và sơ đồ mạch được biểu diễn trong hình 2.27(c).



(a)

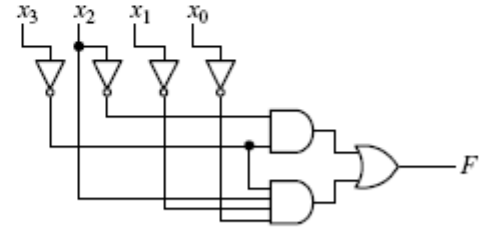


(b)

x_3	x_2	x_1	x_0	$X < 5$
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	×	×	×	0

$$(X < 5) = x_3'x_2' + x_3'x_2x_1'x_0'$$

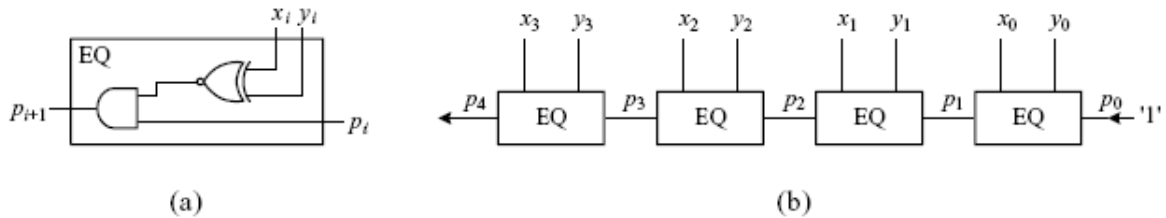
(c)



Hình 2. 27 : Bộ so sánh 4 bit đơn giản cho (a) $X=3$; (b) $X \neq Y$; (c) $X < 5$.

Để xây dựng một bộ so sánh hai số có nhiều bit, ta thường sử dụng cấu trúc của một mạch lặp, tức là sẽ xây dựng một mạch so sánh từng cặp bit tương ứng của hai số sau đó ta kết nối các mạch so sánh từng cặp bit này lại với nhau để tạo thành mạch so sánh nhiều bit. Một bộ so sánh một cặp bit gồm có hai ngõ vào bit cần so sánh x_i và y_i và một ngõ ra p_i để biểu diễn kết quả so sánh giữa hai cặp bit. $p_i=1$ nếu hai bit so sánh bằng nhau, ngược

lại $p_i=0$ nếu hai bit so sánh không bằng nhau. Ban đầu bit p_0 được bật lên 1, việc so sánh cứ thực hiện từ bộ so sánh từng cặp đầu tiên đến bộ so sánh từng cặp bit cuối cùng. Kết quả cuối cùng, nếu $p_4 = p_3 = p_2 = p_1 = p_0 = 1$ thì hai số được so sánh là bằng nhau, các trường hợp còn lại hai số đem so sánh là không bằng nhau.



Hình 2. 28 : Bộ so sánh lặp (a) So sánh từng cặp bit x_i và y_i ; (b) 4-bit $X=Y$.

2.11 Bộ dịch và bộ xoay (shifter / Rotator):

Bộ dịch và bộ xoay được sử dụng cho việc dịch chuyển các bit trong chuỗi bit nhị phân qua phải hoặc trái. Sự khác nhau giữa bộ dịch và bộ xoay được biểu diễn trong hình 2.29

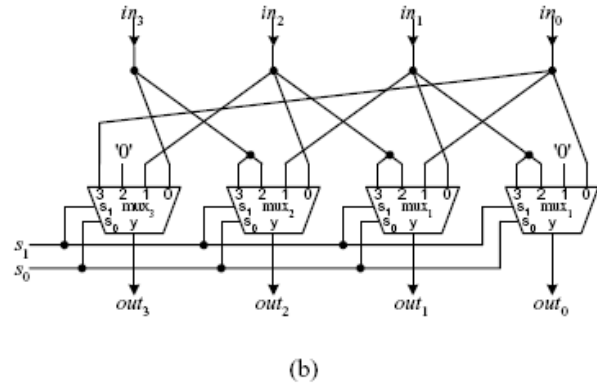
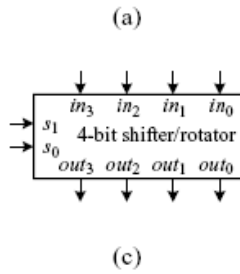
Operation	Comment	Example
Shift left with 0	Shift bits to the left one position. The leftmost bit is discarded and the rightmost bit is filled with a 0.	<pre> 1 0 1 1 0 1 0 0 ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ X 0 1 1 0 1 0 0 0 ← </pre>
Shift left with 1	Same as above except that the rightmost bit is filled with a 1.	<pre> 1 0 1 1 0 1 0 0 ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ X 0 1 1 0 1 0 0 1 ← </pre>
Shift right with 0	Shift bits to the right one position. The rightmost bit is discarded and the leftmost bit is filled with a 0.	<pre> 1 0 1 1 0 1 0 0 ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ → 0 1 0 1 1 0 1 0 X </pre>
Shift right with 1	Same as above except that the leftmost bit is filled with a 1.	<pre> 1 0 1 1 0 1 0 0 ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ → 1 0 1 1 0 1 0 0 X </pre>
Rotate left	Shift bits to the left one position. The leftmost bit is moved to the rightmost bit position.	<pre> 1 0 1 1 0 1 0 0 ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ 0 1 1 0 1 0 0 1 </pre>
Rotate right	Shift bits to the right one position. The rightmost bit is moved to the leftmost bit position.	<pre> 1 0 1 1 0 1 0 0 ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ 0 1 0 1 1 0 1 0 </pre>

Hình 2. 29 : Sự hoạt động của bộ dịch và bộ xoay.

Một bộ ghép kênh được dùng để dịch chuyển một bit qua bên trái hoặc qua phải của dòng bit ban đầu. Kích thước của bộ ghép sẽ quyết định số lượng các loại hoạt động có thể được thực thi. Ví dụ chúng ta có thể dùng một bộ ghép kênh 4 thành 1 để thực thi 4 hoạt động

đặc biệt được diễn tả trong hình 2.30(a). Hai ngõ vào lựa chọn s_0 và s_1 được sử dụng để lựa chọn một trong bốn hoạt động trong hình 2.30(a). Có 4 bit ngõ vào, chúng ta sẽ cần sử dụng 4 bộ ghép kênh 4 thành 1 như trong hình 2.30(b). Những ngõ vào của các bộ ghép kênh được kết nối như thế nào sẽ tùy thuộc vào 4 trạng thái hoạt động trong bảng 2.30(a).

S_1	s_0	Operation
0	0	Pass through
0	1	Shift left and fill with 0
1	0	Shift right and fill with 0
1	1	Rotate right



Hình 2. 30 : Bộ dịch / bộ xoay 4 bit: (a) Bảng trạng thái hoạt động; (b) sơ đồ mạch; (c) ký hiệu logic.

Trong ví dụ trên, khi $s_1 = s_0 = 0$, chúng ta muốn đưa thẳng bit đến ngõ ra mà không thực hiện việc dịch. Ví dụ chúng ta muốn đưa giá trị in_i đến ngõ ra out_i . Khi $s_1 = s_0 = 0$, d_0 của bộ ghép kênh được lựa chọn, do đó in_i được kết nối đến d_0 của bộ ghép kênh mux_i có ngõ ra là out_i . Khi $s_1 = 0$ và $s_0 = 1$, chúng ta muốn dịch sang trái, nghĩa là chúng ta muốn đưa giá trị in_i ra ngõ out_{i+1} . Với $s_1 = 0$ và $s_0 = 1$, d_1 của bộ ghép kênh được lựa chọn, do đó in_i được kết nối đến d_1 của bộ ghép kênh mux_{i+1} có ngõ ra là out_{i+1} . Trong sự lựa chọn này, chúng ta cũng muốn dịch bit 0, vì thế d_1 của bộ ghép kênh mux_0 được nối trực tiếp đến 0.

Ngôn ngữ VHDL cho bộ dịch / bộ xoay 8 bit được viết theo cấu trúc Behavioral có chức năng được giới thiệu ở trong hình 2.30(a).

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY shifter IS PORT (
    SHSel: IN std_logic_vector(1 downto 0);      -- select for operations
    input: IN std_logic_vector(7 downto 0);      -- input
    output: OUT std_logic_vector(7 downto 0));    -- output
END shifter;

ARCHITECTURE Behavior OF shifter IS
BEGIN
    process(SHSel, input)
    begin
        CASE SHSel IS
            WHEN "00" =>    -- pass through
                output <= input;
            WHEN "01" =>    -- shift left with 0
                output <= input(6 downto 0) & '0';
            WHEN "10" =>    -- shift right with 0
                output <= '0' & input(7 downto 1);
            WHEN OTHERS =>  -- rotate right
                output <= input(0) & input(7 downto 1);
        END CASE;
    END PROCESS;
END Behavior;

```

2.12 Bộ nhân

Trong trường học, chúng ta đã biết cách nhân hai số thập phân với nhau dựa trên phương pháp dịch và cộng. Bất kể số đó là số thập phân hay là số nhị phân cũng đều thực hiện phép nhân tương tự nhau. Trong thực tế việc nhân hai số nhị phân là dễ dàng hơn bởi vì bạn chỉ nhân hai giá trị 0 và 1, kết quả phép nhân cũng là 0 hoặc 1. Hình 2.31(a) biểu diễn cách nhân hai số 4 bit bằng tay, thừa số thứ nhất $M = m_3m_2m_1m_0$ nhân với thừa số thứ hai $Q = q_3q_2q_1q_0$ cho kết quả là tích $P = p_7p_6p_5p_4p_3p_2p_1p_0$.

Cách nhân hai số nhị phân được thực hiện hoàn toàn giống với cách nhân hai số thập phân. Đó là phương pháp nhân, dịch và cộng. Trong bộ nhân chúng ta phải sử dụng một thanh ghi để lưu trữ giá trị trung gian và giá trị cuối cùng.

Phép nhân cho kết quả nhanh hơn trong các mạch kết hợp có thể được xây dựng dựa trên cùng phương pháp trên. Trong các mạch kết hợp cổng, các cổng AND được sử dụng để nhân các bit để cho ra các kết quả trung gian và các bộ cộng được dùng để cộng tất cả các kết quả trung gian này lại với nhau để cho ra kết quả cuối cùng. Chúng ta thấy rằng việc AND hai bit thì cho cùng một kết quả với nhân hai bit.

Trong hình 2.31(c) biểu diễn sự kết nối của các bộ cộng toàn phần để cộng các kết quả trung gian và cộng kết quả cuối cùng. Bốn bộ cộng trong mỗi hàng được kết nối theo kiểu bộ cộng có nhớ trạng thái dư của bit trước (Ripple Carry Adder). Bit nhớ cuối cùng của bộ cộng được kết nối thẳng ra bit có trọng số lớn nhất của kết quả tích. Khi bắt đầu thực hiện phép nhân bit c_0 được set bằng 0.

Multiplicand (M)	1 1 0 1	
Multiplier (Q)	× 1 0 1 1	
	1 1 0 1	
Intermediate products {	1 1 0 1	
	0 0 0 0	
	+ 1 1 0 1	
Product (P)	1 0 0 0 1 1 1 1	

	m_3	m_2	m_1	m_0	
	× q_3	q_2	q_1	q_0	
	m_3q_0	m_2q_0	m_1q_0	m_0q_0	
	m_3q_1	m_2q_1	m_1q_1	m_0q_1	
	m_3q_2	m_2q_2	m_1q_2	m_0q_2	
	+ m_3q_3	m_2q_3	m_1q_3	m_0q_3	
	p_7	p_6	p_5	p_4	p_3
	p_2	p_1	p_0		

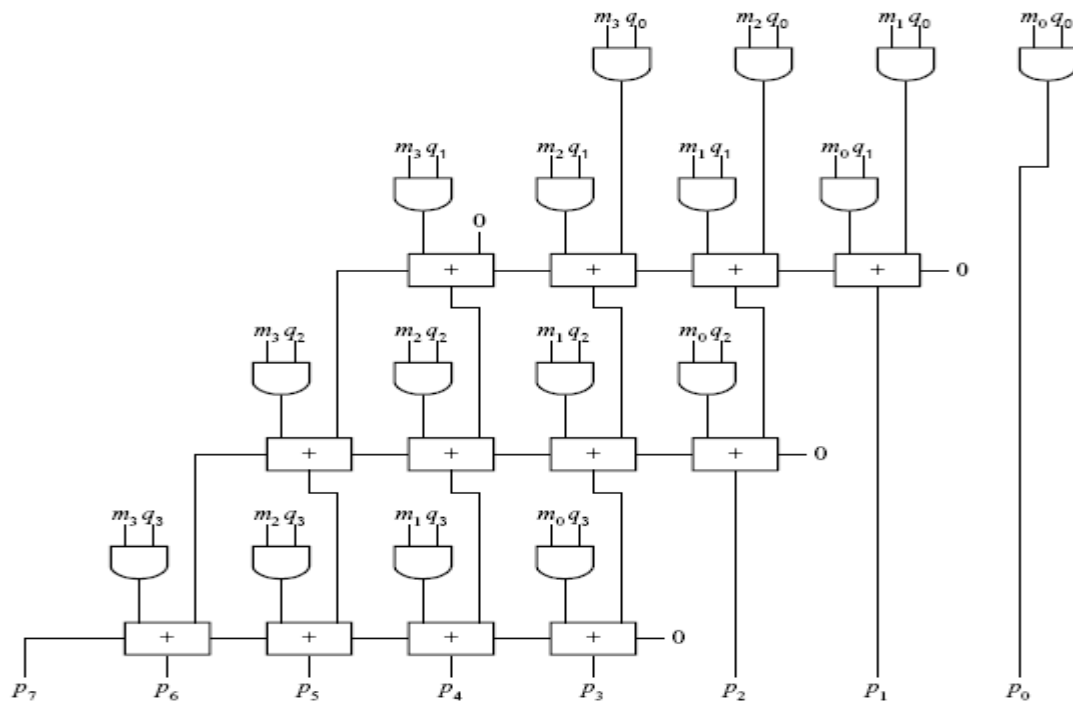
(a)

```

P = 0
FOR i = 0 TO 3
  IF  $q_i = 1$  THEN
    P = P + (M << i) // the operation M << i is to shift M to the left by
                      i bit position
  END IF
END FOR
// result is in P

```

(b)



(c)

Hình 2. 31 : Phép nhân (a) nhân bằng tay; (b) phương pháp thực hiện; (c) sơ đồ mạch.

2.13 Máy trạng thái hữu hạn FSM

Sự khác nhau chính giữa các mạch kết hợp và các mạch tuần tự đó là các mạch kết hợp chỉ phụ thuộc vào các ngõ vào hiện tại, trong khi ngoài các ngõ vào hiện tại, các mạch tuần tự còn tùy thuộc vào các ngõ vào trước đó. Các ngõ vào trước đó được nhớ trong bộ nhớ trạng thái, được tạo ra từ một hoặc nhiều mạch flip-flop. Nội dung của các mạch flip-flop bộ nhớ trạng thái tại bất kỳ thời điểm nào tương ứng với trạng thái hiện tại của mạch. Mạch thay đổi từ một trạng thái đến trạng thái tiếp theo khi nội dung của bộ nhớ trạng thái thay đổi.

Một mạch tuần tự hoạt động từng bước thông qua một chuỗi các trạng thái. Vì bộ nhớ trạng thái là hữu hạn, cho nên tổng số các trạng thái có khả năng khác nhau cũng hữu hạn. Vì lý do này, một mạch tuần tự cũng được xem như một máy trạng thái- hữu hạn (FSM). Dù chỉ có một số hữu hạn các trạng thái khác nhau, FSM có thể đi đến bất kỳ trạng thái nào khi cần thiết. Từ đó, chuỗi các trạng thái mà FSM đi qua có thể dài vô tận.

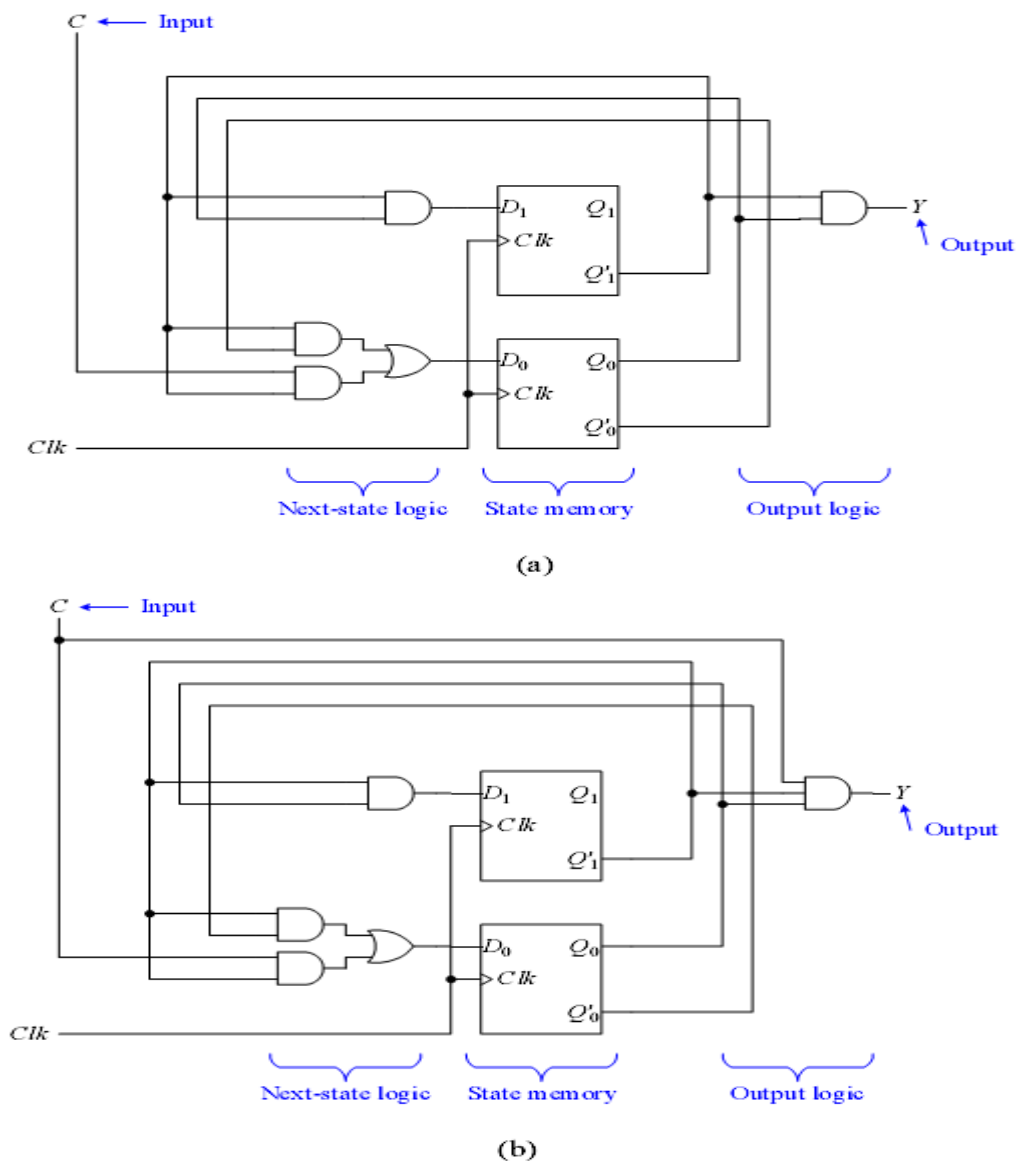
Để bổ sung bộ nhớ trạng thái, một máy trạng thái- hữu hạn chứa hai phần kết hợp: trạng thái logic tiếp theo và ngõ ra logic. Phụ thuộc vào trạng thái hiện tại của máy và các tín hiệu vào, trạng thái logic tiếp theo quyết định trạng thái tiếp theo là gì bằng việc thay đổi nội dung của bộ nhớ trạng thái. Để đưa ra trạng thái hiện tại và các ngõ vào, trạng thái logic tiếp theo tạo ra một giá trị mới tương đương trạng thái tiếp theo của máy. Bằng việc thay đổi các giá trị ngõ vào flip-flop, mạch trạng thái tiếp theo làm cho bộ nhớ trạng thái thay đổi đến giá trị mới. Giá trị mới của trạng thái tiếp theo được ghi vào bộ nhớ trạng thái tại cạnh tích cực của xung clock tiếp theo.

Tốc độ trong các chuỗi máy trạng thái- hữu hạn thông qua các trạng thái được quyết định bởi xung clock. Tại mỗi cạnh tích cực của tín hiệu xung clock, bộ nhớ thanh ghi trạng thái được cho phép và giá trị trạng thái tiếp theo được cất giữ vào trong các flip-flop. Yếu tố giới hạn tốc độ xung clock là trong thời gian mà nó thực hiện tất cả các thao tác dữ liệu để gán tới một trạng thái riêng biệt. Tất cả các thao tác dữ liệu được gán tới một trạng thái phải kết thúc trong một chu kỳ xung clock để các kết quả có thể được ghi vào trong các thanh ghi tại cạnh xung clock tích cực tiếp theo.

Phần kết hợp thứ hai ở một FSM là ngõ ra logic. Ngõ ra logic tạo các tín hiệu ngõ ra cần thiết cho FSM. Các tín hiệu ngõ ra tùy thuộc vào trạng thái hiện tại của máy và có thể hoặc không phụ thuộc vào các tín hiệu ngõ vào. Dù tín hiệu ngõ ra có phụ thuộc vào ngõ vào hay không vẫn có 2 kiểu của FSMs. Moore FSM là *FSM có ngõ ra của máy chỉ phụ thuộc vào trạng thái hiện tại và không phụ thuộc vào các tín hiệu vào*, trong khi Mealy FSM là *FSM có ngõ ra phụ thuộc vào cả trạng thái hiện tại và tín hiệu vào*.

2.13.1 Mô hình máy trạng thái hữu hạn FSM (Finite-State-Machine):

Hình 2.32a trình bày tổng quát sơ đồ cho Moore FSM mà các đầu ra của nó chỉ phụ thuộc vào trạng thái hiện tại của nó. Hình 2.32b cho thấy tổng quát sơ đồ cho Mealy FSM mà những đầu ra của nó phụ thuộc vào cả trạng thái hiện tại của máy lẫn các đầu vào nữa. Trong cả hai hình, chúng ta nhìn thấy các đầu vào mà trạng thái logic tiếp theo là những tín hiệu vào sơ cấp và trạng thái hiện tại của máy. Trạng thái logic tiếp theo tạo các giá trị kích thích để thay đổi bộ nhớ trạng thái. Một điểm khác nhau trong hai hình là đối với Moore FSM, ngõ ra logic chỉ có trạng thái hiện tại như ngõ vào của nó, trong khi đó đối với Mealy FSM, ngõ ra logic có cả trạng thái hiện tại và các tín hiệu vào như các ngõ vào của nó.



Hình 2. 32 : Sơ đồ mạch của Moore FSM và Mealy FSM.

Hình 2.32a và 2.32b là một mạch mẫu tương ứng của một Moore FSM và Mealy FSM. Hai mạch thì đồng nhất ngoại trừ ngõ ra của chúng. Đối với Moore FSM, mạch ngõ ra là một cổng AND 2 ngõ vào mà giá trị ngõ vào của nó lấy từ các ngõ ra của hai D flip-flop. Chú ý trạng thái FSM tương ứng với nội dung của bộ nhớ trạng thái, mà nội dung đó là các flip-flop. Nội dung (hay trạng thái) của một flip-flop tương ứng với giá trị ở ngõ ra Q (hay Q'). Từ đó, mạch này chỉ phụ thuộc vào trạng thái hiện tại của máy. Đối với Mealy FSM, mạch ngõ ra là một cổng AND 3 ngõ vào, mà hai ngõ vào của nó lấy từ các flip-flop, ngõ vào thứ ba cổng AND này được nối tới ngõ vào C sơ cấp. Với kết nối phụ này, ngõ ra mạch này tùy thuộc vào cả hai trạng thái hiện tại và ngõ vào.

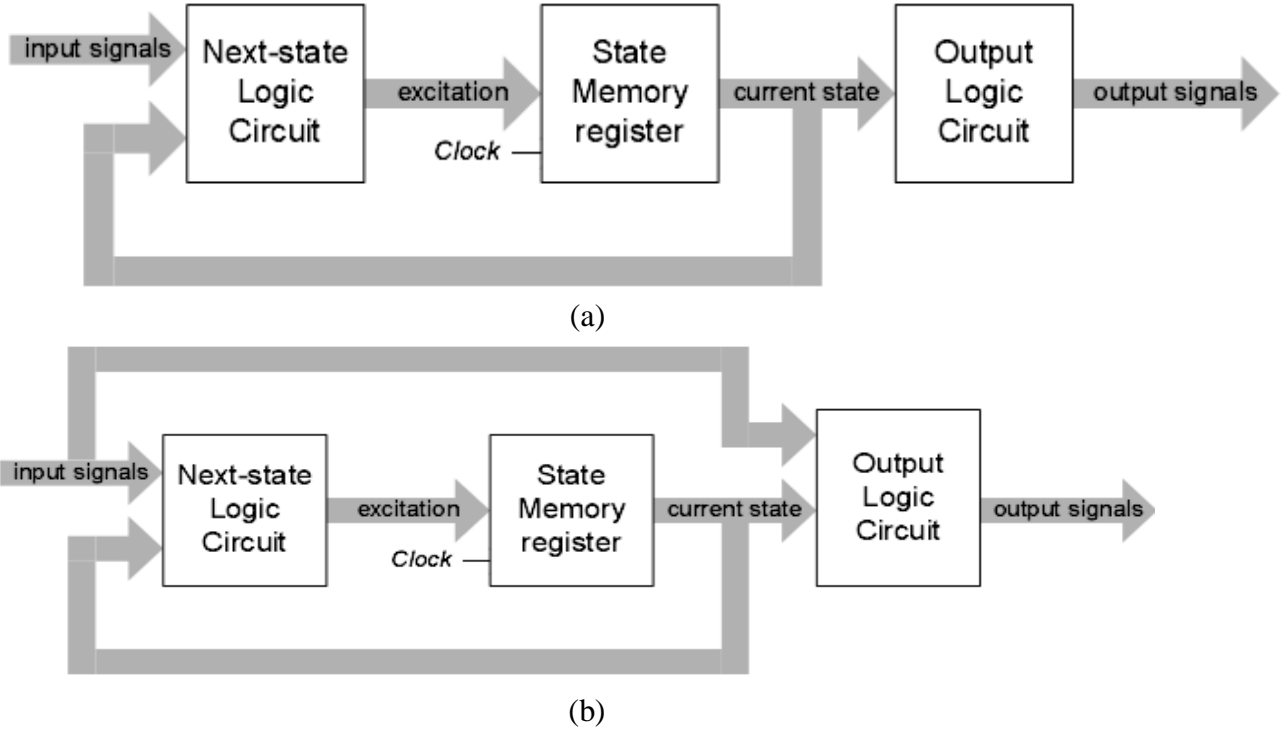
Với cả hai mạch, bộ nhớ trạng thái gồm có hai D flip-flop. Với hai flip-flop, tương ứng với bốn giá trị khác nhau. Từ đó, máy trạng thái- hữu hạn này có thể có bất kỳ 1 trong 4 trạng thái khác nhau. Trạng thái mà FSM này sẽ đi đến tiếp theo phụ thuộc vào giá trị tại các ngõ vào D flip-flop.

Mỗi flip-flop trong bộ nhớ trạng thái yêu cầu một mạch kết hợp để tạo một giá trị trạng thái tiếp theo cho các đầu vào của nó. Vì chúng ta có 2 D flip-flop, mỗi cái có 1 ngõ vào D, bởi vậy, trạng thái logic tiếp theo của mạch gồm có hai mạch kết hợp; một cho đầu vào D_0 và một cho D_1 . Các ngõ vào tới hai mạch kết hợp này là Q's, nó tương ứng cho trạng thái hiện tại tại các flip-flop và ngõ vào sơ cấp C. Chú ý là không cần thiết cho rằng ngõ vào C là một ngõ vào tới tất cả các mạch kết hợp. Trong mạch mẫu, chỉ duy nhất mạch kết hợp ở dưới phụ thuộc vào ngõ vào C.

Phân tích mạch tuần tự (Analysis of Sequential Circuits):

Thường khi chúng ta đưa ra một mạch tuần tự và cần biết sự hoạt động của nó. Phân tích mạch tuần tự là quá trình trong đó ta đưa ra cho một mạch tuần tự và ta muốn mô tả chính xác sự hoạt động của mạch đang có. Việc mô tả của một mạch tuần tự có thể là trong bảng trạng thái tiếp theo / bảng đầu ra, hay một sơ đồ trạng thái. Các bước phân tích của các mạch tuần tự như sau:

1. Dẫn xuất những phương trình kích thích từ mạch trạng thái logic tiếp theo.
2. Dẫn xuất ra những phương trình trạng thái tiếp theo bằng việc thế những phương trình kích thích vào các phương trình đặc tính của flip-flop.
3. Dẫn xuất bảng trạng thái tiếp theo- từ các phương trình trạng thái tiếp theo.
4. Dẫn xuất những phương trình ngõ ra (nếu có) từ mạch logic ngõ ra.
5. Dẫn xuất bảng ngõ ra (nếu có) từ những phương trình ngõ ra.
6. Vẽ sơ đồ trạng thái từ bảng trạng thái- kế tiếp và bảng ngõ ra.



Hình 2. 33 : (a) Sơ đồ khối Moore FSM; (b) Sơ đồ khối Mealy FSM

2.13.2 Phương trình kích thích (Excitation Equation):

Các phương trình kích thích là các phương trình của mạch logic trạng thái tiếp theo trong FSM. Vì trạng thái logic tiếp theo là một mạch kết hợp, bởi vậy, dẫn xuất ra các phương trình kích thích chỉ là để phân tích một mạch kết hợp như được thảo luận ở các phần trên. Mạch trạng thái tiếp theo được dẫn xuất ra bởi những phương trình "kích thích" flip-flop bằng việc làm cho chúng thay đổi trạng thái. Những phương trình này cung cấp những tín hiệu tới các ngõ vào của flip-flop, và được biểu thị như một chức năng của trạng thái hiện tại và các ngõ vào đến FSM. Trạng thái hiện thời được xác định bởi nội dung hiện tại của flip-flop, nghĩa là, tín hiệu ngõ ra của flip-flop Q (và Q'). Có một phương trình cho mỗi ngõ vào của flip-flop.

Sau đây là hai phương trình kích thích mẫu cho 2 D flip-flop. Phương trình đầu tiên cung cấp mạch trạng thái tiếp theo cho ngõ vào của D flip-flop 1, và phương trình thứ 2 cung cấp mạch cho ngõ vào của D flip-flop 0.

$$D_1 = Q_1'Q_0 \quad (1)$$

$$D_0 = Q_1'Q_0' + CQ_1' \quad (2)$$

2.13.3 Phương trình trạng thái tiếp theo (Next-state Equation):

Các phương trình trạng thái tiếp theo chỉ rõ trạng thái tiếp theo của các flip-flop là phụ thuộc vào ba thứ:

- 1) Trạng thái hiện tại của các flip-flop.
- 2) Hành vi chức năng của các flip-flop.
- 3) Các ngõ vào đến flip-flop.

Trạng thái hiện tại của các flip-flop chỉ là ngõ ra Q của các flip-flop.

Như vậy, để dẫn xuất ra các phương trình trạng thái tiếp theo, chúng ta thể các phương trình kích thích vào trong các phương trình đặc tính tương ứng của flip-flop.

Chẳng hạn, phương trình đặc tính cho D flip-flop là:

$$Q_{\text{next}} = D$$

Bởi vậy, thay hai phương trình kích thích (1) và (2) từ mục 2.13.2.1 vào trong phương trình đặc tính cho D flip-flop sẽ đưa cho chúng ta hai phương trình trạng thái kế tiếp như sau:

$$Q_{1\text{next}} = D_1 = Q_1'Q_0 \quad (3)$$

$$Q_{0\text{next}} = D_0 = Q_1'Q_0' + CQ_1' \quad (4)$$

2.13.4 Bảng trạng thái tiếp theo (Next-state Table):

Bảng trạng thái tiếp theo đơn giản là bảng chân trị được dẫn ra từ các phương trình trạng thái tiếp theo. Nó liệt kê mọi sự kết hợp giữa các giá trị của trạng thái hiện tại (Q) và các giá trị đầu vào, để các giá trị trạng thái tiếp theo (Q_{next}) nên là gì. Các giá trị trạng thái tiếp theo này có được bằng việc thế trạng thái hiện tại và các giá trị đầu vào vào trong những phương trình trạng thái tiếp theo thích hợp. Hình 2.34 cho thấy bảng trạng thái tiếp theo mẫu với những trạng thái hiện tại Q_1Q_0 bằng 00, 01, 10, và 11, và một tín hiệu ngõ vào C. Những mục trong bảng là các giá trị trạng thái tiếp theo $Q_{1\text{next}}$, $Q_{0\text{next}}$.

Current State Q_1Q_0	Next State	
	$Q_{1\text{next}}$	$Q_{0\text{next}}$
	$C = 0$	$C = 1$
00	01	01
01	10	11
10	00	00
11	00	00

Hình 2. 34 : Bảng trạng thái tiếp theo với 4 trạng thái và tín hiệu ngõ vào C.

Các giá trị trạng thái tiếp theo này có được từ việc thế các giá trị hiện tại Q_1Q_0 và giá trị ngõ vào C vào trong các phương trình trạng thái tiếp theo (3) và (4) từ mục 2.13.2.2 ở trên. Ví dụ, phần trên cùng bên trái nói cho chúng ta rằng nếu trạng thái hiện thời là 00 và điều kiện ngõ vào $C=0$ là đúng khi đó trạng thái tiếp theo mà FSM sẽ đi đến là 01. Vì 01 cũng là trạng thái tiếp theo từ trạng thái hiện tại 00 và điều kiện $C=1$ là đúng, điều này có nghĩa chuyển tiếp từ trạng thái 00 tới 01 không phụ thuộc vào ngõ vào điều kiện C, vì vậy đây là một sự chuyển tiếp vô điều kiện. Từ trạng thái 01, có hai chuyển tiếp có điều kiện: FSM sẽ chuyển đến trạng thái 10 nếu điều kiện $C=0$ là đúng, hoặc nếu $C=1$ nó sẽ chuyển đến trạng thái 11. Cả hai trạng thái 10 và 11 chuyển đến trạng thái 00 vô điều kiện.

Phương trình ngõ ra (Output Equation):

Các phương trình đầu ra là các phương trình được dẫn xuất ra từ ngõ ra mạch logic kết hợp trong FSM. Phụ thuộc vào kiểu FSM (Moore hay Mealy), các phương trình ngõ ra có thể chỉ phụ thuộc trên trạng thái hiện tại hoặc cả hai trạng thái hiện tại và các ngõ vào.

Với mạch Moore ở hình 2.32a, phương trình ngõ ra là :

$$Y = Q_1'Q_0 \quad (5)$$

Với mạch Mealy ở hình 2.32b, phương trình ngõ ra là

$$Y = CQ_1'Q_0 \quad (6)$$

Một kiểu FSM sẽ có nhiều tín hiệu ngõ ra, vì thế sẽ có một phương trình cho mỗi tín hiệu ngõ ra.

Bảng ngõ ra (Output Table):

Giống như bảng trạng thái kế tiếp, bảng ngõ ra là bảng chân trị được dẫn ra từ các phương trình ngõ ra. Các bảng ngõ ra cho Moore và Mealy có chút khác biệt. Với Moore FSM, các danh sách bảng ngõ ra cho mỗi sự kết hợp của giá trị hiện tại để các giá trị đầu ra nên là gì. Trong khi đó với Mealy FSM, các danh sách bảng ngõ ra cho mỗi sự kết hợp của trạng thái hiện tại và các giá trị ngõ vào để giá trị các ngõ ra là gì. Các giá trị ngõ ra này có được bằng việc thế trạng thái hiện tại và các giá trị ngõ vào vào trong những phương trình ngõ ra thích hợp hình 2.35a và 2.35b cho thấy rằng bảng ngõ ra mẫu cho Moore và Mealy được bắt nguồn từ phương trình ngõ ra (5) và (6) tương ứng ở mục 2.13.2.4 ở trên. Với Moore FSM, tín hiệu đầu ra Y chỉ phụ thuộc vào giá trị trạng thái hiện tại tại Q_1Q_0 . Trong khi đó, với Mealy FSM tín hiệu đầu ra Y thì phụ thuộc vào cả trạng thái hiện tại lẫn ngõ vào C.

Current State Q_1Q_0	Output Y
00	0
01	1
10	0
11	0

Current State Q_1Q_0	Output Y	
	$C = 0$	$C = 1$
00	0	0
01	0	1
10	0	0
11	0	0

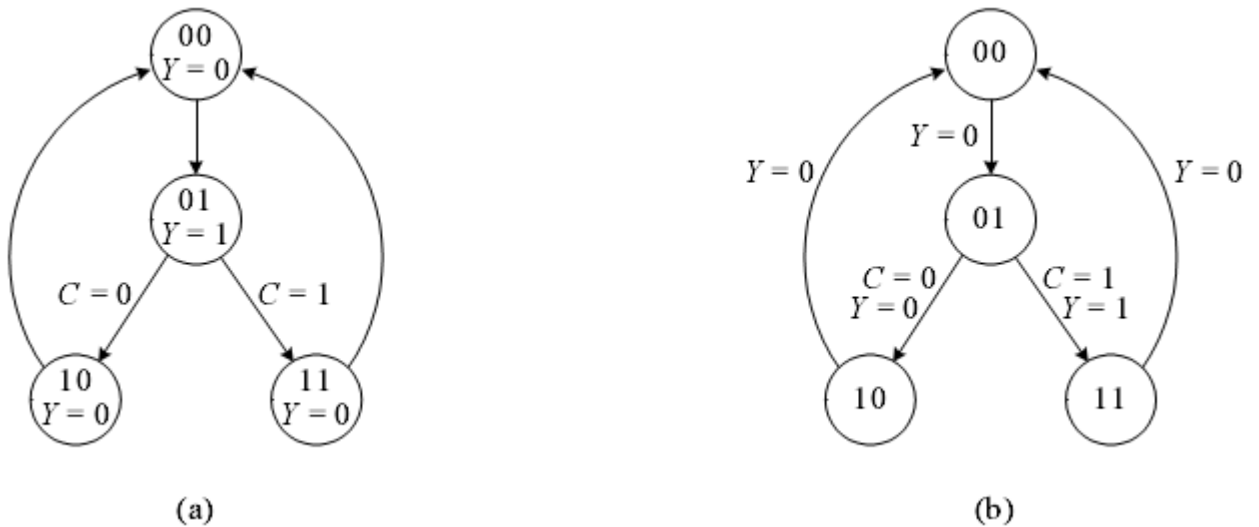
Hình 2. 35 : Bảng ngõ ra (a) Moore FSM; (b) Mealy FSM.

Sơ đồ trạng thái (State Diagram):

Một sơ đồ trạng thái là một đồ thị với các nút và các đường định hướng nối tới các nút. Sơ đồ trạng thái bằng đồ thị minh họa hoạt động của FSM. Có một nút cho mỗi trạng thái FSM và các nút này được gắn nhãn với trạng thái mà chúng đại diện. Với mỗi chuyển trạng thái của FSM có một đường định hướng kết nối cho hai nút. Đường định hướng bắt nguồn từ nút tương ứng cho trạng thái hiện tại mà FSM chuyển từ đó, và đi đến nút tương ứng cho trạng thái tiếp theo mà FSM chuyển tới. Các đường có thể có hoặc không có các nhãn trên

chúng. Các đường có các chuyển tiếp vô điều kiện từ trạng thái này sang trạng thái khác sẽ không có nhãn. Trong trường hợp này, chỉ có một đường có thể bắt nguồn từ nút đó. Chuyển tiếp có điều kiện từ một trạng thái sẽ có hai đường theo 2 hướng. Hai đường từ trạng thái này được gắn nhãn tương ứng với các điều kiện tín hiệu ngõ vào - một đường với nhãn khi mà điều kiện là đúng và đường khác với nhãn khi điều kiện là sai.

Hình 2.36a cho thấy một sơ đồ trạng thái nhỏ với bốn trạng thái, 00, 01, 10, và 11, và một tín hiệu ngõ vào C. Sơ đồ trạng thái này được bắt nguồn từ bảng trạng thái tiếp theo ở hình 2.34 và bảng đầu ra ở hình 2.35a. Có 3 chuyển tiếp vô điều kiện 00 tới 01, 10 tới 00, và 11 tới 00, và một chuyển tiếp có điều kiện từ 01 đến 10 hay 11. Với chuyển tiếp có điều kiện từ 01, nếu điều kiện $C=0$ là đúng, khi đó chuyển tiếp từ 01 đến 10 được thực hiện. Còn lại, nếu điều kiện $C=0$ là sai, nghĩa là $C=1$ là đúng, khi đó chuyển tiếp từ 01 đến 11 được thực hiện.



Hình 2. 36 : Sơ đồ các trạng thái trong một mạch tuần tự

Tín hiệu ngõ ra Y ở hình 2.36a được gắn nhãn bên trong mỗi nút biểu thị rằng ngõ ra chỉ phụ thuộc vào trạng thái hiện tại. Ví dụ, khi FSM ở trạng thái 01, ngõ ra Y là 1, trong khi, ở trạng thái 11, Y là 0. Từ đó, sơ đồ trạng thái này là cho Moore FSM.

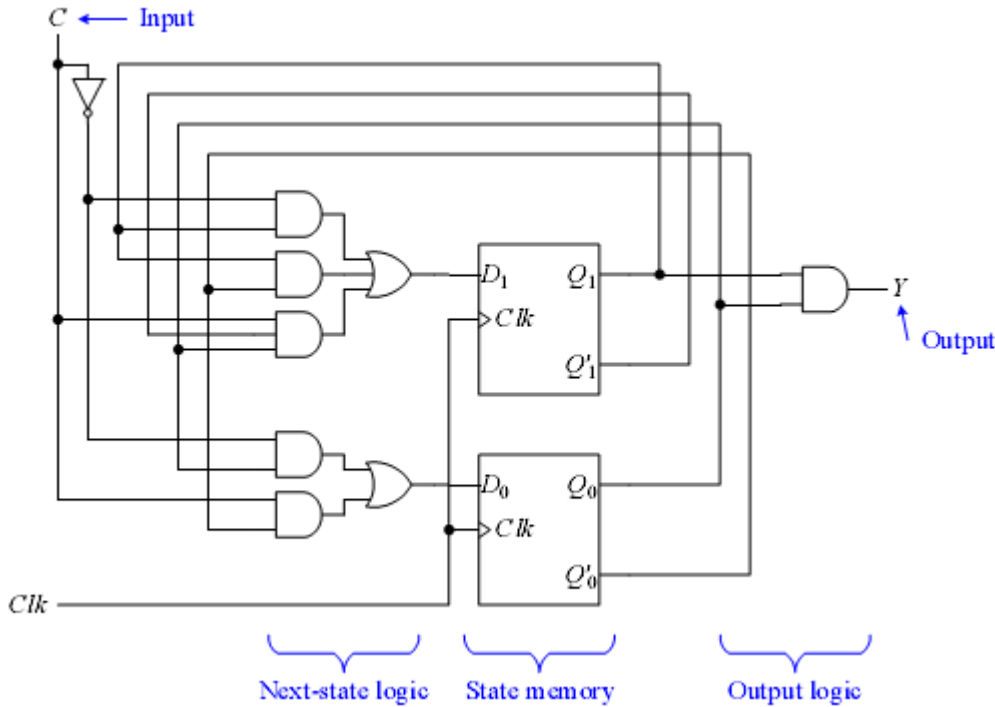
Trong hình 2.36b, ngõ ra Y được gắn nhãn trên đường đi biểu thị rằng ngõ ra là phụ thuộc vào cả hai trạng thái hiện tại và tín hiệu vào C. Ví dụ, khi FSM ở trạng thái 01, nếu FSM theo đường trái cho $C = 0$ để đến trạng thái 10, khi đó nó sẽ xuất ngõ ra là 0 cho Y. Tuy nhiên, nếu FSM theo đường bên phải cho $C = 1$ để đến trạng thái 11, khi đó nó sẽ xuất ngõ ra là 1 cho Y. Do đó đây là sơ đồ trạng thái cho Mealy FSM.

2.13.5 Ví dụ phân tích 1 Moore FSM:

Bây giờ chúng ta sẽ minh họa toàn bộ quá trình phân tích Moore FSM với 1 ví dụ.

Ví dụ 2.1:

Hình 2.37 cho thấy một mạch tuần tự đơn giản. Chúng ta kết luận rằng đây là một Moore kiểu FSM vì ngõ ra logic gồm 1 cổng AND 2 ngõ vào mà cổng chỉ phụ thuộc vào trạng thái hiện tại Q_1Q_0 . Chúng ta sẽ theo sáu bước ở trên để phân tích chi tiết của mạch này.



Hình 2. 37 : Moore FSM đơn giản

Bước 1 là chỉ ra phương trình kích thích là các phương trình cho mạch trạng thái logic tiếp theo. Những phương trình này thì tùy thuộc vào trạng thái hiện thời của các flip-flop Q_1 và Q_0 , và ngõ vào C . Một phương trình cần cho mọi ngõ vào dữ liệu của tất cả các flip-flop trong bộ nhớ trạng thái. Mạch mẫu của chúng ta có hai flip-flop với hai đầu vào D_1 , và D_0 , vì thế chúng ta có hai phương trình kích

$$D_1 = C'Q_1 + Q_1Q_0' + CQ_1'Q_0$$

$$D_0 = C'Q_0 + CQ_0'$$

Hai phương trình này có được từ việc phân tích hai mạch kết hợp mà cung cấp các ngõ vào D_1 và D_0 tới hai flip-flop. Cho ví dụ đặc biệt này, cả hai mạch kết hợp này chỉ đơn giản là mạch tạo tổng hai mức.

Bước 2 sẽ dẫn xuất ra phương trình trạng thái tiếp theo. Các phương trình này nói chúng ta biết trạng thái tiếp theo sẽ là trạng thái hiện tại nào của bộ nhớ trạng thái, hành vi chức năng của flip-flop và các ngõ vào đến flip-flop. Một phương trình cho mỗi flip-flop. Hành vi chức năng của flip-flop được mô tả bằng phương trình đặc tính của nó, mà của D flip-flop là $Q_{next} = D$. Các đầu vào tới flip-flop chỉ là các phương trình kích thích được dẫn xuất

ra từ bước 1. Từ đây, chúng ta thay phương trình kích thích vào trong phương trình đặc tính cho mỗi flip-flop để thu được phương trình trạng thái tiếp theo cho flip-flop đó. Với hai flip-flop trong ví dụ, chúng ta có hai phương trình trạng thái-tiếp theo, một cho Q_{1next} và một cho Q_{0next} .

$$\begin{aligned} Q_{1next} &= D_1 = C'Q_1 + Q_1Q_0' + CQ_1'Q_0 \\ Q_{0next} &= D_0 = C'Q_0 + CQ_0' \end{aligned}$$

Bước 3 sẽ dẫn xuất ra bảng trạng thái tiếp theo. Các giá trị trạng thái tiếp theo trong bảng có được bằng việc thay mọi kết hợp của trạng thái hiện tại và những giá trị đầu vào trong các phương trình trạng thái tiếp theo thu được trong bước 2. Trong ví dụ của chúng ta, có hai flip-flop, Q_1 và Q_0 , và ngõ vào C . Từ đó bảng sẽ có tám phần trạng thái tiếp theo. Có hai bit cho các phần – bit đầu tiên cho Q_{1next} , và bit thứ 2 cho Q_{0next} . Bảng trạng thái tiếp theo

Current State Q_1Q_0	Next State $Q_{1next}Q_{0next}$	
	$C = 0$	$C = 1$
00	00	01
01	01	10
10	10	11
11	11	00

Ví dụ để tìm Q_{1next} cho trạng thái hiện tại $Q_1Q_0 = 00$ và $C = 1$ (mục màu xanh), chúng ta thay giá trị $Q_1=0, Q_0=0$ và $C=1$ vào trong phương trình

$$Q_{1next} = C'Q_1 + Q_1Q_0' + CQ_1'Q_0 = (1' \cdot 0) + (0 \cdot 0') + (1 \cdot 0' \cdot 0) \text{ để được giá trị } 0.$$

Tương tự, chúng ta được Q_{0next} bằng cách thay cùng giá trị Q_0, Q_1 , và C vào trong phương trình $Q_{0next} = C'Q_0 + CQ_0' = (1' \cdot 0) + (1 \cdot 0') \text{ để được giá trị } 1.$

Bước 4 sẽ dẫn xuất ra phương trình ngõ ra từ mạch logic ngõ ra. Một phương trình ngõ ra cần cho mỗi tín hiệu ngõ ra. Ví dụ chúng ta, có chỉ có một tín hiệu đầu ra Y chỉ phụ thuộc vào trạng thái hiện tại của máy. Phương trình đầu ra cho Y là

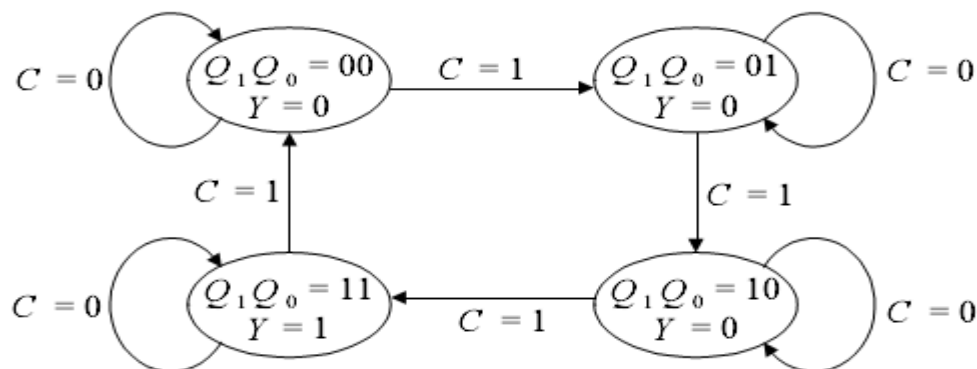
$$Y = Q_1Q_0$$

Bước 5 dẫn xuất bảng ngõ ra. Giống như bảng trạng thái tiếp theo, bảng đầu ra có được bằng việc thay tất cả các kết hợp có khả năng của các giá trị trạng thái hiện tại vào trong phương trình ngõ ra cho Moore FSM. Bảng ngõ ra cho ví dụ Moore FSM là

Current State Q_1Q_0	Output Y
00	0
01	0
10	0
11	1

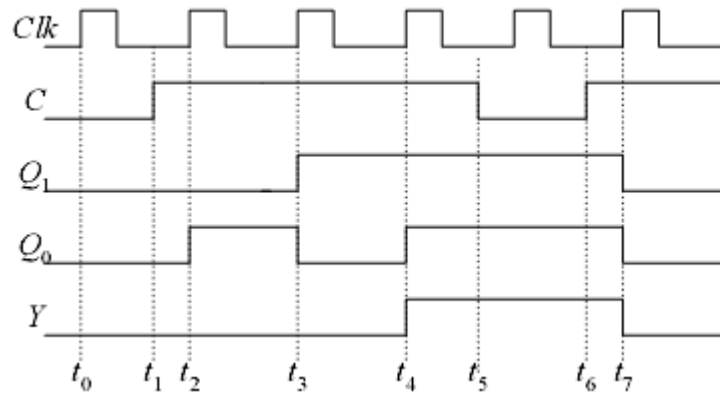
Bước 6 sẽ vẽ sơ đồ trạng thái, được dẫn xuất từ bảng trạng thái tiếp theo và bảng ngõ ra. Mọi trạng thái trong bảng trạng thái tiếp theo sẽ có một nút tương ứng có nhãn với trạng thái lập mã trong sơ đồ trạng thái. Mỗi phần trạng thái tiếp theo trong bảng trạng thái tiếp theo, sẽ tương ứng với một hướng đi. Hướng đi này bắt nguồn từ nút được gắn nhãn với trạng thái hiện tại và các kết thúc tại nút được gắn nhãn với phần trạng thái tiếp theo. Hướng có nhãn với những điều kiện đầu vào tương ứng.

Ví dụ, trong bảng trạng thái tiếp theo, khi trạng thái hiện tại Q_1Q_0 là 00 trạng thái tiếp theo $Q_{1next}Q_{0next}$ là 01 với ngõ vào $C=1$. Từ đó, trong sơ đồ trạng thái, có một hướng đi từ nút 00 tới nút 01 với nhãn $C=1$. Đối với Moore FSM, các ngõ ra chỉ phụ thuộc vào trạng thái hiện tại, như vậy các giá trị ngõ ra từ bảng ngõ ra được nằm bên trong mỗi nút của sơ đồ trạng thái. Sơ đồ trạng thái đầy đủ cho ví dụ của chúng ta được chỉ ở hình 2.38.



Hình 2. 38 : Sơ đồ trạng thái đầy đủ của mạch Moore FSM.

Sơ đồ tính toán thời gian mẫu cho sự thực hiện của mạch.



Hai D flip-flop được dùng trong mạch để kích cạnh dương flip-flop vì thế chúng thay đổi trạng thái của chúng tại mỗi cạnh lên xung clock. Đầu tiên, chúng ta giả thiết rằng hai flip-flop này ở trạng thái 0. Cạnh lên đầu tiên ở thời gian t_0 . Bình thường, flip-flop sẽ thay đổi trạng thái vào thời gian này, tuy nhiên, một khi $C=0$ giá trị các flip-flop vẫn không đổi. Vào thời gian t_1 , C thay đổi $C=1$, để tại cạnh lên xung clock tiếp theo vào thời gian t_2 , các giá trị flip-flop Q_1Q_0 thay đổi tới 01. Ở thời gian t_4 khi $Q_1Q_0=11$, đầu ra Y cũng thay đổi đến 1 vì $Y=Q_1*Q_0$. Vào thời gian t_5 , ngõ vào C rơi xuống 0 nhưng ngõ ra Y vẫn là 1. Q_1Q_0 vẫn là 11 cho dù có cạnh lên xung clock tiếp theo vì $C=0$. Vào thời gian t_6 , C thay đổi thành 1 và vì thế tại cạnh lên xung clock tiếp theo vào thời gian t_7 , Q_1Q_0 tăng dần lần nữa tới 00 và chu trình lặp lại.

Khi $C=1$, chu trình FSM thông qua bốn trạng thái để lặp lại. Khi $C=0$, FSM dừng lại tại trạng thái hiện tại cho đến khi C được tích cực lại. Nếu chúng ta giải thích 4 trạng thái mã hóa như 1 số thập phân, thì chúng ta có thể kết luận rằng mạch ở hình 2.37 là một bộ đếm lên modulo-4 mà chu trình thông qua 4 giá trị 0, 1, 2, và 3. Ngõ vào C cho phép hoặc không cho phép đếm.

Mã VHDL theo hành vi của Moore FSM trong ví dụ 2.1 như sau và giản đồ thời gian ở hình 2.39.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY MooreFSM IS PORT (
    clock: IN STD_LOGIC;
    reset: IN STD_LOGIC;
    C: IN STD_LOGIC;
    Y: OUT STD_LOGIC);
END MooreFSM;

ARCHITECTURE Behavioral OF MooreFSM IS
    TYPE state_type IS (s0, s1, s2, s3);
```

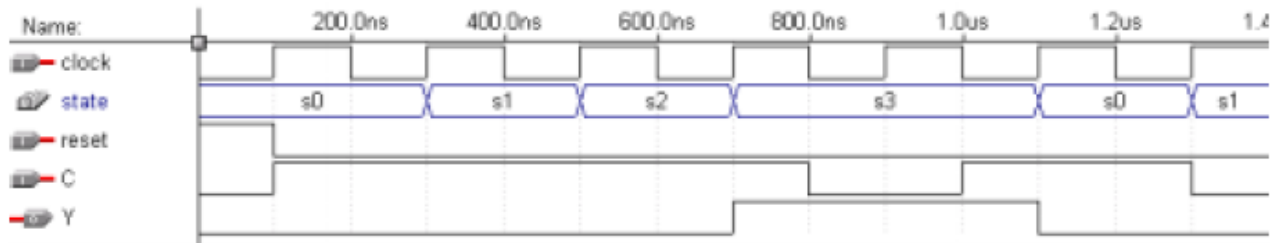
```
SIGNAL state: state_type;
BEGIN
  next_state_logic: PROCESS (clock)
  BEGIN
    IF (reset = '1') THEN
      state <= s0;
    ELSIF (clock'EVENT AND clock = '1') THEN
      CASE state is
        WHEN s0 =>
          IF C = '1' THEN
            state <= s1;
          ELSE
            state <= s0;
          END IF;
        WHEN s1 =>
          IF C = '1' THEN
            state <= s2;
          ELSE
            state <= s1;
          END IF;
        WHEN s2=>
          IF C = '1' THEN
            state <= s3;
          ELSE
            state <= s2;
          END IF;
        WHEN s3=>
          IF C = '1' THEN
            state <= s0;
          ELSE
            state <= s3;
          END IF;
      END CASE;
    END IF;
  END PROCESS;
  output_logic: PROCESS (state)
  BEGIN
    CASE state IS
      WHEN s0 =>
```

```

        y <= '0';
    WHEN s1 =>
        y <= '0';
    WHEN s2 =>
        y <= '0';
    WHEN s3 =>
        y <= '1';

    END CASE;
END PROCESS;
END Behavioral;

```

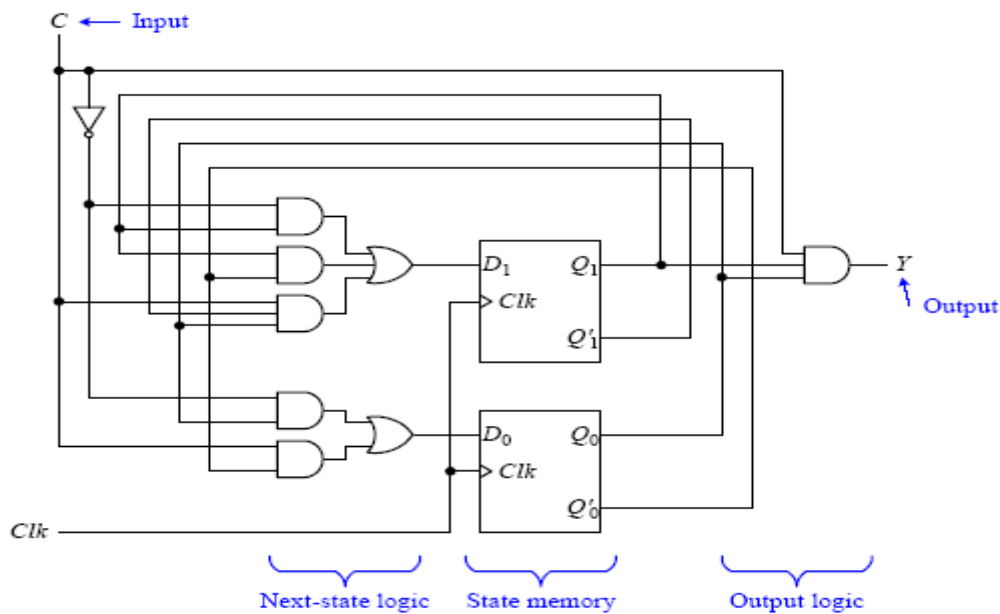


Hình 2. 39 : Giản đồ thời gian của Moore FSM mô phỏng bằng xilinx.

2.13.6 Ví dụ phân tích Mealy FSM:

Ví dụ 2.2: Minh họa quá trình để thực hiện một sự phân tích trên một Mealy FSM

Hình 2.40 cho thấy một Mealy FSM đơn giản. Mạch này cũng giống như mạch trong hình 2.37 ngoại trừ mạch ngõ ra, mà trong ví dụ này là một cổng AND 3-ngõ vào, nó không chỉ phụ thuộc vào ngõ vào hiện tại tại Q_1Q_0 mà còn phụ thuộc vào ngõ vào C.



Hình 2. 40 : Mealy FSM đơn giản.

Phân tích mạch này cũng giống như phân tích Moore FSM trong ví dụ 2.1 trên để tạo bảng trạng thái tiếp theo trong bước 3. Chỉ khác là dẫn xuất phương trình ngõ ra và bảng ngõ ra trong bước 4 và 5. Đối với Mealy FSM, phương trình ngõ ra thì tùy thuộc vào cả trạng thái hiện tại và trạng thái ngõ vào. Vì mạch chỉ có 1 tín hiệu ngõ ra, chúng ta được phương trình ngõ ra phụ thuộc vào C như sau.

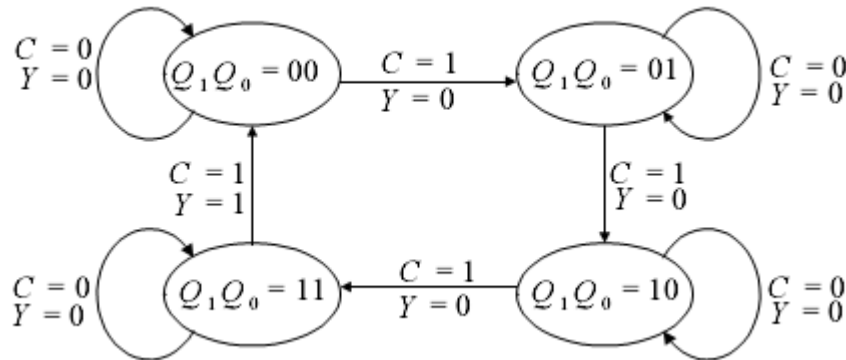
$$Y = CQ_1Q_0$$

Hình 2.41 chỉ bảng kết quả ngõ ra có được bằng cách thay tất cả các giá trị có thể có của Q_0, Q_1, C vào phương trình ngõ ra:

Current State Q_1Q_0	Output Y	
	$C = 0$	$C = 1$
00	0	0
01	0	0
10	0	0
11	0	1

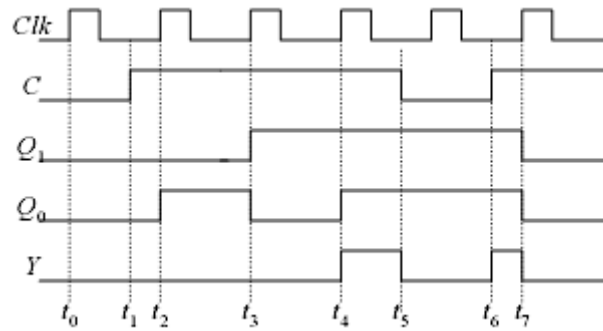
Hình 2. 41 : Bảng chân trị ngõ ra.

Với sơ đồ trạng thái, chúng ta không thể đặt giá trị ngõ ra vào bên trong một nút kế vì giá trị ngõ ra tùy thuộc vào trạng thái hiện tại và giá trị ngõ vào. Do đó, giá trị ngõ ra được đặt lên hướng đi tương ứng tới giá trị trạng thái hiện tại và giá trị ngõ vào như ở hình 2.42 . Tín hiệu ngõ ra Y là 0 cho tất cả các hướng ngoại trừ tín hiệu bắt nguồn từ trạng thái 11 có ngõ vào điều kiện C=1. Trên một hướng Y là 1.



Hình 2. 42 : Trạng thái đầy đủ của Mealy FSM.

Một sơ đồ tính toán thời gian mẫu được đưa vào hình 2.43 cho Moore FSM từ thời gian t_5 . Vào thời gian t_5 , ngõ vào C xuống 0, và vì vậy ngõ ra Y cũng xuống 0 vì $Y=C*Q_1*Q_0$. Vào thời gian t_6 , C tăng lên 1, và do đó Y cũng lên 1 ngay lập tức. Vì ngõ ra mạch là một mạch kết hợp, Y không thay đổi tại cạnh tích cực của xung clock, nhưng thay đổi ngay lập tức khi các ngõ vào thay đổi. Ở thời gian t_7 khi Q_1Q_0 thay đổi tới 00, Y lại thay đổi về 0.



Hình 2. 43 : Tính toán thời gian mẫu cho Mealy FSM

Ngoại trừ sự khác nhau trong mạch này là tạo tín hiệu ngõ ra Y như thế nào, Mealy FSM chạy giống như FSM Moore từ ví dụ 2.1 theo cách nó thay đổi từ một trạng thái tới trạng thái tiếp theo. Điều này là tất nhiên, vì thực tế là cả 2 bảng trạng thái tiếp theo đồng nhất. Như vậy, mạch Mealy FSM cũng là một bộ đếm lên modulo 4.

Mã VHDL theo hành vi của Mealy FSM trong ví dụ 2.2 như sau và giản đồ thời gian ở hình 2.44.

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY MealyFSM IS PORT (
    clock: IN STD_LOGIC;
    reset: IN STD_LOGIC;
    C: IN STD_LOGIC;
    Y: OUT STD_LOGIC);
END MealyFSM;

ARCHITECTURE Behavioral OF MealyFSM IS
    TYPE state_type IS (s0, s1, s2, s3);
    SIGNAL state: state_type;
BEGIN
    next_state_logic: PROCESS (clock)
    BEGIN
        IF (reset = '1') THEN
            state <= s0;
        ELSIF (clock'EVENT AND clock = '1') THEN
            CASE state is
                WHEN s0 =>
                    IF C = '1' THEN
                        state <= s1;
                    ELSE

```

```

                                state <= s0;
                                END IF;
                                WHEN s1 =>
                                    IF C = '1' THEN
                                        state <= s2;
                                    ELSE
                                        state <= s1;
                                    END IF;
                                WHEN s2=>
                                    IF C = '1' THEN
                                        state <= s3;
                                    ELSE
                                        state <= s2;
                                    END IF;
                                WHEN s3=>
                                    IF C = '1' THEN
                                        state <= s0;
                                    ELSE
                                        state <= s3;
                                    END IF;
                                END CASE;
                                END IF;
                                END PROCESS;
output_logic: PROCESS (state, C)
BEGIN
    CASE state IS
        WHEN s0 =>
            y <= '0';
        WHEN s1 =>
            y <= '0';
        WHEN s2 =>
            y <= '0';
        WHEN s3 =>
            IF (C = '1') THEN
                y <= '1';
            ELSE
                y <= '0';
            END IF;
        END CASE;
    END CASE;
```



Hình 2. 44 : Giãn đồ thời gian của Mealy FSM được mô phỏng bằng xilinx.

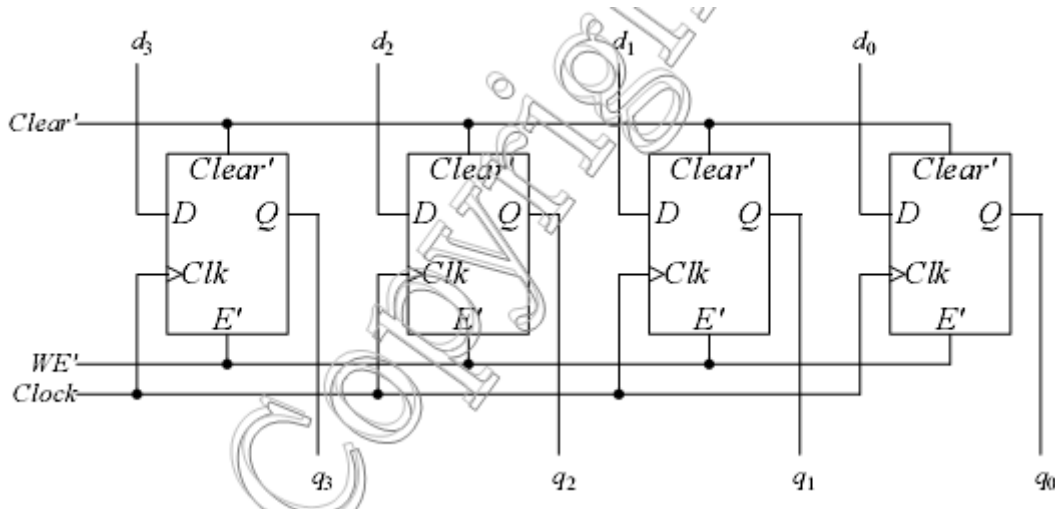
2.14 Các linh kiện tuần tự:

Trong hệ thống máy tính chúng ta thường muốn nhiều bit thông tin. Hơn nữa chúng ta muốn nhóm 1 vài bit lại với nhau và xem chúng như 1 thành phần, ví dụ như số nguyên được thành lập từ 8 bit. Trong chương này chúng ta sẽ xem xét các thanh ghi và mạch nhớ để lưu trữ nhiều bit thông tin. Các thanh ghi có nhiều chức năng hơn bằng cách thêm vào các chức năng đếm và dịch bit. Chúng ta sẽ xem xét 1 vài bộ đếm và thanh ghi dịch.

2.14.1 Các thanh ghi (Registers):

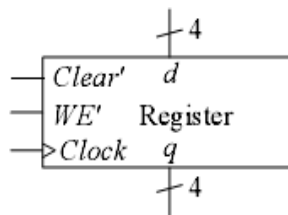
Khi chúng ta muốn lưu trữ 1 byte dữ liệu chúng ta phải kết hợp 8 flip-flop lại với nhau và chúng làm việc như 1 thành phần. Một thanh ghi chỉ là một mạch với 2 hay nhiều D flip-flop kết hợp lại với nhau bằng cách này tất cả chúng cùng làm việc chính xác với nhau và đồng bộ với 1 xung clock. Chỉ có 1 khác biệt là mỗi flip-flop trong nhóm được dùng để lưu trữ 1 bit khác nhau của dữ liệu.

Hình 2.45 chỉ ra 1 thanh ghi 4 bit với mức xóa không đồng bộ. Bốn flip-flop D tích cực mức thấp và dùng mức xóa không đồng bộ. Chú ý trong mạch các ngõ vào điều khiển Clk, WE', và Clear' được nối chung sao cho khi ngõ vào riêng biệt được tích cực, thì tất cả các flip-flop sẽ chạy chính xác với nhau. 4 Bit dữ liệu ngõ vào được kết nối từ d₀ đến d₃, trong khi đó 4 bit q₀ đến q₃ xem như 4 bit ngõ ra của thanh ghi. Khi chân cho phép ghi WE' (write enable) tích cực mức thấp được tích cực. Ví dụ WE'=0, dữ liệu tương đương trên chân d được lưu trữ vào thanh ghi (4 flip-flop) khi có cạnh xuống của xung clock tiếp theo. Khi WE' không tích cực, nội dung trong thanh ghi vẫn không đổi. Thanh ghi có thể được xóa không đồng bộ bằng cách tích cực chân Clear. Nội dung của thanh ghi luôn có trên các chân q, vì vậy không cần chân điều khiển đọc dữ liệu từ thanh ghi.



Hình 2. 45 : Thanh ghi 4 bit với mức xóa không đồng bộ.

Hình 2.46 chỉ ra ký hiệu logic của thanh ghi. Chân với số 4 trên tín hiệu d và q cho biết nó có độ rộng 4 bit vì vậy gọi là thanh ghi 4 bit.



Hình 2. 46 : Ký hiệu logic của thanh ghi.

Đoạn mã VHDL cho thanh ghi 4 bit. Chú ý mã này cũng giống cho D flip-flop đơn. Điểm khác biệt chính là dữ liệu ngõ vào và ra là 4 bit.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY reg4 IS
    PORT (
        Clock, Clear, WE' : IN STD_LOGIC;
        d : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        q : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END reg4;
ARCHITECTURE Behavior OF reg4 IS
BEGIN
    PROCESS(Clock, Clear)
    BEGIN
        IF Clear = '0' THEN
            q <= (others => '0'); -- same as "0000"
        ELSIF Clock'EVENT AND Clock = '1' THEN
            IF WE' = '0' THEN

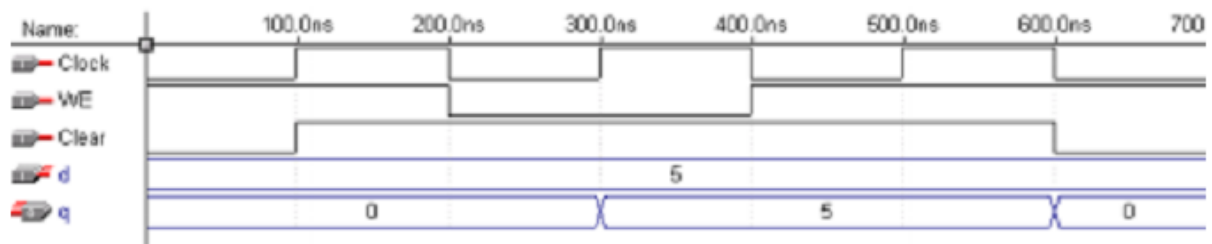
```

```

        q<=d;
    END IF;
END IF;
END PROCESS;
END Behavior;

```

Tín hiệu mô phỏng cho thanh ghi chỉ ở hình 2.47. Chú ý trong giản đồ sóng khi WE' tích cực 200ns, ngõ ra q không đổi tức khắc khi ngõ vào có giá trị 5. Sự thay đổi xuất hiện ở cạnh lên tiếp theo của xung clock tại 300ns. Mặt khác khi Clear tích cực tại 600ns, q được reset xuống 0 ngay.



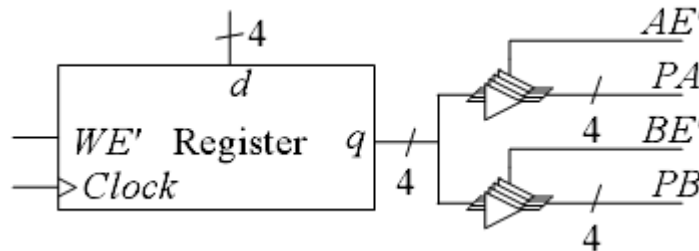
Hình 2. 47 : Giản đồ mô phỏng cho thanh ghi 4 bit.

2.14.2 Thanh ghi tập tin (Register Files):

Khi ta muốn lưu trữ vài số đồng thời, ta có thể dùng vài thanh ghi riêng trong mạch. Tuy nhiên có khi bạn muốn xử lý những thanh ghi này như một thành phần, tương tự như xác định các vị trí riêng lẻ của một mảng. Do đó thay vì có 1 vài thanh ghi ta muốn có 1 mảng thanh ghi. Mảng các thanh ghi này được xem như thanh ghi tập tin. Trong một thanh ghi tập tin, tất cả các tín hiệu điều khiển tương ứng đều được nối chung. Hơn nữa, *tất cả dữ liệu đầu vào và ra tương ứng cho tất cả các thanh ghi cũng được nối chung*. Nói cách khác, ví dụ *tất cả các chân d_3 của tất cả các thanh ghi được nối chung*. Vì thế thanh ghi tập tin chỉ có 1 cách set ngõ vào và ngõ ra cho tất cả các thanh ghi. Để chỉ rõ thanh ghi nào trong thanh ghi tập tin mà bạn muốn đọc/ghi, từ đâu/đến đâu, thì thanh ghi tập tin có các đường địa chỉ cho mục đích này.

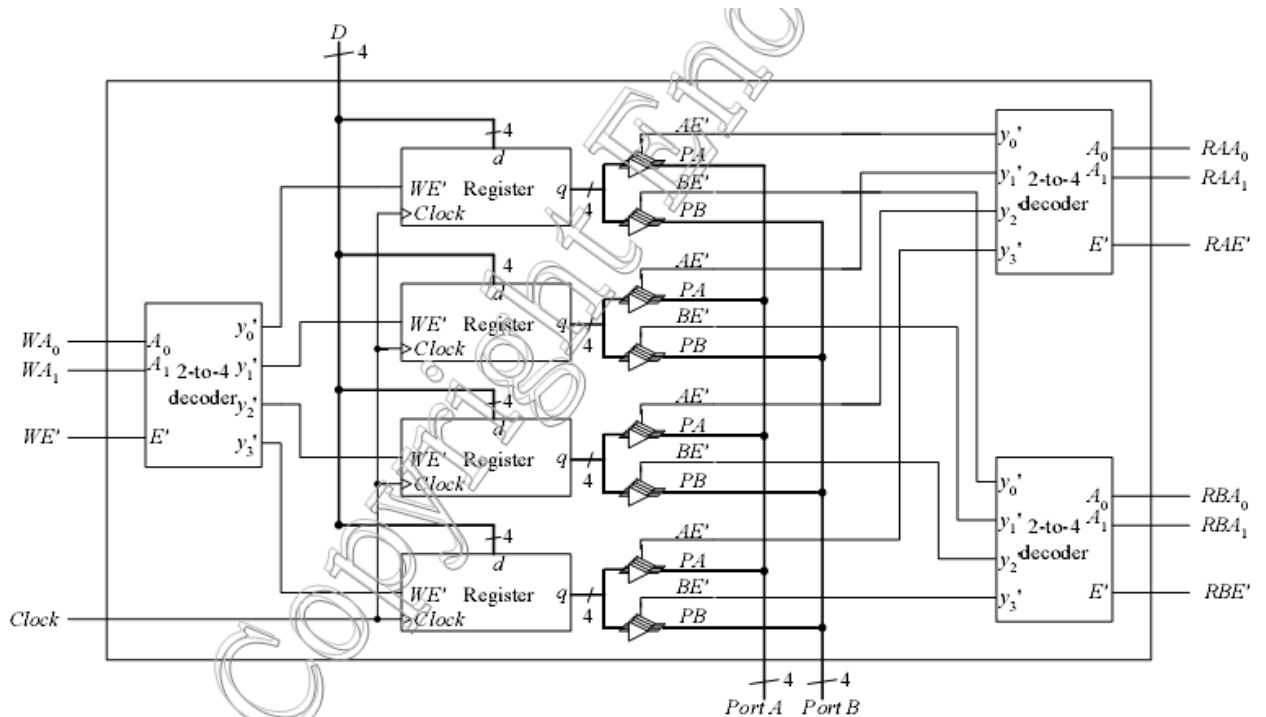
Trong một mạch vi xử lý cần có một ALU, thanh ghi tập tin thì thường được dùng cho những toán hạng nguồn của ALU. Khi ALU có hai toán hạng ngõ vào, chúng ta muốn thanh ghi tập tin có hai giá trị ngõ ra từ hai vị trí khác nhau của thanh ghi tập tin. Vì thế một thanh ghi tập tin thường có một Port ghi và hai Port đọc. Tất cả 3 Port sẽ có các chân địa chỉ và chân cho phép riêng. Port đọc có trạng thái trở kháng cao khi chân cho phép đọc không được tích cực. Trong chu kỳ hoạt động, dữ liệu trên Port đọc có giá trị lập tức sau khi chân cho phép đọc được tích cực, trong khi Port ghi xuất hiện mức tích cực ở cạnh lên tiếp theo của xung clock.

Mạch thanh ghi ở hình 2.45 không có chân điều khiển cho việc đọc dữ liệu ngõ ra. Để điều khiển khi ta muốn dữ liệu ngõ ra và đặt chân ngõ ra ở trạng thái trở kháng cao, chúng ta cần thêm vào bộ đệm 3 trạng thái ở mỗi ngõ ra. Tất cả các chân cho phép của bộ đệm 3 trạng thái được kết nối chung khi chúng ta muốn điều khiển tất cả ngõ ra cùng lúc. Hơn nữa chúng ta cần có 2 Port đọc. Ví dụ, 2 ngõ ra điều khiển của mỗi thanh ghi vì thế ta có thể kết nối 2 bộ đệm 3 trạng thái đến mỗi ngõ ra. Mạch thanh ghi được sửa đổi như hình 2.48.



Hình 2. 48 : Mạch thanh ghi có thêm chân điều khiển.

AE' và BE' là các chân cho phép đọc tín hiệu tương ứng của Port A và Port B. Mỗi 1 Port kết nối đến các chân cho phép của 4 bộ đệm 3 trạng thái khi set. Tất cả chúng đều tích cực mức thấp. PA và PB là 2 Port đọc 4 bit. Để chọn thanh ghi mà bạn muốn làm việc bạn dùng bộ giải mã để giải mã địa chỉ. Ngõ ra của bộ giải mã được dùng để tích cực chân cho phép đọc/ghi. Mạch hoàn chỉnh của 1 thanh ghi 4x4 (4 thanh ghi mỗi thanh ghi có độ rộng là 4 bit) được chỉ ra ở hình 2.49.



Hình 2. 49 : Mạch hoàn chỉnh của thanh ghi 4x4.

Đoạn mã VHDL cho thanh ghi 4x4 với 1 Port ghi 2 Port đọc:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all; -- needed for CONV_INTEGER()

ENTITY regfile IS PORT (
    clk: in STD_LOGIC; --clock
    WE: in STD_LOGIC; --write enable
    WA: in STD_LOGIC_VECTOR(1 DOWNTO 0); --write address
    D: in STD_LOGIC_VECTOR(7 DOWNTO 0); --input
    RAE, RBE: in STD_LOGIC; --read enable PORTsA&B
    RAA, RBA: in STD_LOGIC_VECTOR(1 DOWNTO 0); --read address PORTA&B
    A, B: out STD_LOGIC_VECTOR(7 DOWNTO 0)); --output PORTA&B
END regfile;

ARCHITECTURE Behavioral OF regfile IS
    SUBTYPE reg IS STD_LOGIC_VECTOR(7 DOWNTO 0);
    TYPE regArray IS ARRAY(0 to 3) OF reg;
    SIGNAL RF: regArray; --register file contents
    BEGIN
        WritePort: PROCESS (clk)
        BEGIN
            IF (clk'EVENT AND clk = '1') then
                IF (WE = '0') then
                    RF(CONV_INTEGER(WA)) <= D; --fn to convert from bit VECTOR to
                    integer
                END IF;
            END IF;
        END PROCESS;

        ReadPortA: PROCESS (RAA, RAE)
        BEGIN
            -- Read Port A
            IF (RAE = '0') then
                A <= RF(CONV_INTEGER(RAA)); --fn to convert from bit VECTOR to integer
            ELSE
                A <= (others => 'Z');
            END IF;
        END PROCESS;

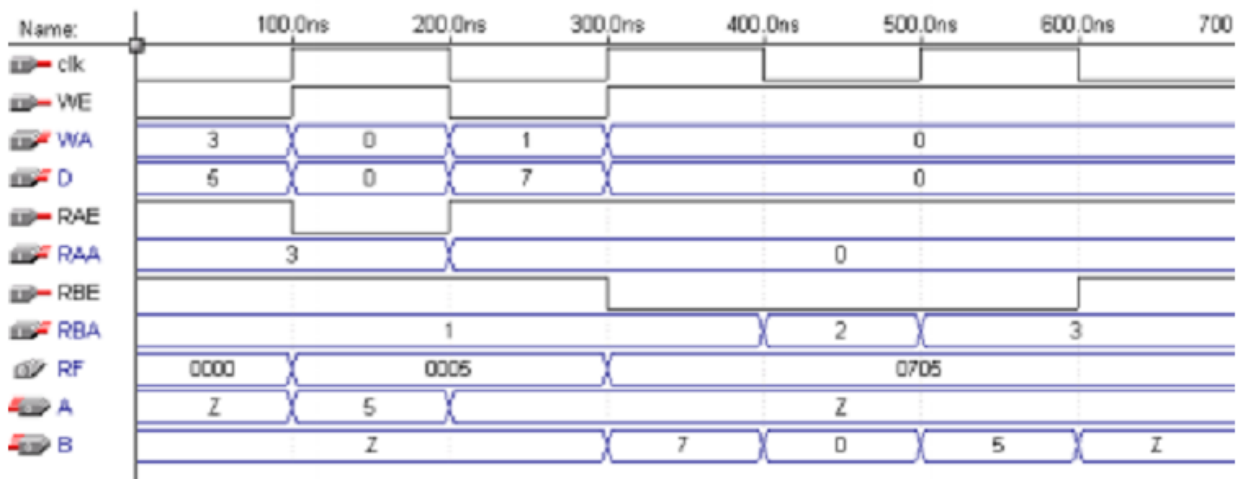
        ReadPortB: PROCESS (RBE, RBA)
```

```

BEGIN
-- Read Port B
  IF (RBE = '0') then
    B <= RF(CONV_INTEGER(RBA)); --fn to convert from bit VECTOR to integer
  ELSE
    B <= (others => 'Z');
  END IF;
END PROCESS;
END Behavioral;

```

Hình 2.50 cho thấy tín hiệu mô phỏng cho ghi 4x4 với 1 Port ghi , 2 Port đọc



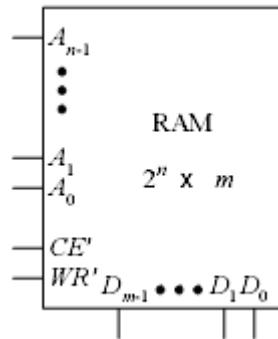
Hình 2. 50 : Tín hiệu mô phỏng cho ghi 4x4 với 1 Port ghi, 2 Port đọc.

2.14.3 Bộ nhớ truy xuất ngẫu nhiên (Random Access Memory):

Một thành phần quan trọng khác của hệ thống máy tính là bộ nhớ. Phần này có thể xem như là RAM hoặc ROM. Ta làm bộ nhớ cũng giống như làm thanh ghi tập tin nhưng với nhiều địa chỉ hơn. Tuy nhiên có nhiều lý do ta không thể làm như vậy. Một lý do đó là chúng ta muốn có bộ nhớ lớn và rẻ tiền. Vì thế chúng ta phải làm mỗi ô nhớ nhỏ đến khả năng có thể có. Một lý do khác là chúng ta muốn dùng chung đường Bus data cho cả việc đọc và ghi dữ liệu từ đâu/đến đâu của bộ nhớ. Điều này cho thấy rằng mạch nhớ chỉ cần có 1 Port dữ liệu mà không cần phải 2 hay 3 như thanh ghi tập tin. *Ký hiệu Logic chỉ tất cả các kết nối của chip RAM thông thường chỉ ở hình 2.51.* Có các đường dữ liệu D và các đường địa chỉ A.

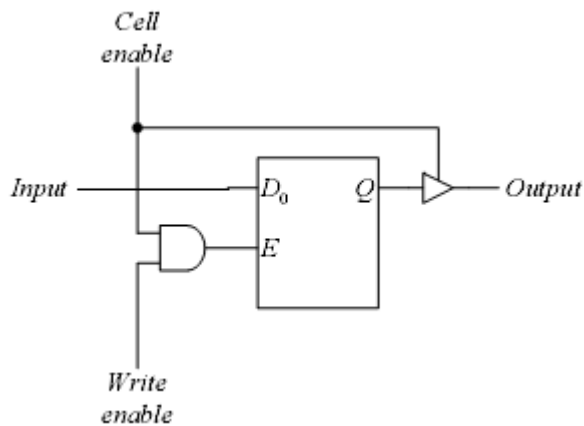
Đường dữ liệu cho cả ngõ vào và ngõ ra của dữ liệu ở vị trí từ đâu đến đâu được chỉ rõ bằng các đường địa chỉ. *Số lượng đường dữ liệu tùy thuộc vào có bao nhiêu bit được dùng để lưu trữ dữ liệu trong mỗi vị trí nhớ. Số lượng đường địa chỉ tùy thuộc vào có bao nhiêu vị trí trên chip.* Ví dụ 1 chip có 512 byte bộ nhớ sẽ có 8 đường dữ liệu (8 bit =1 byte) và 9

đường địa chỉ ($2^9 = 512$). Để có thể thêm các đường dữ liệu và địa chỉ, ta thường dùng 3 chân điều khiển là: chip enable (CE'), write enable (WR'), và xung clock. Cả CE' và WE' được tích cực mức thấp.



Hình 2. 51 : Ký hiệu logic của chip RAM.

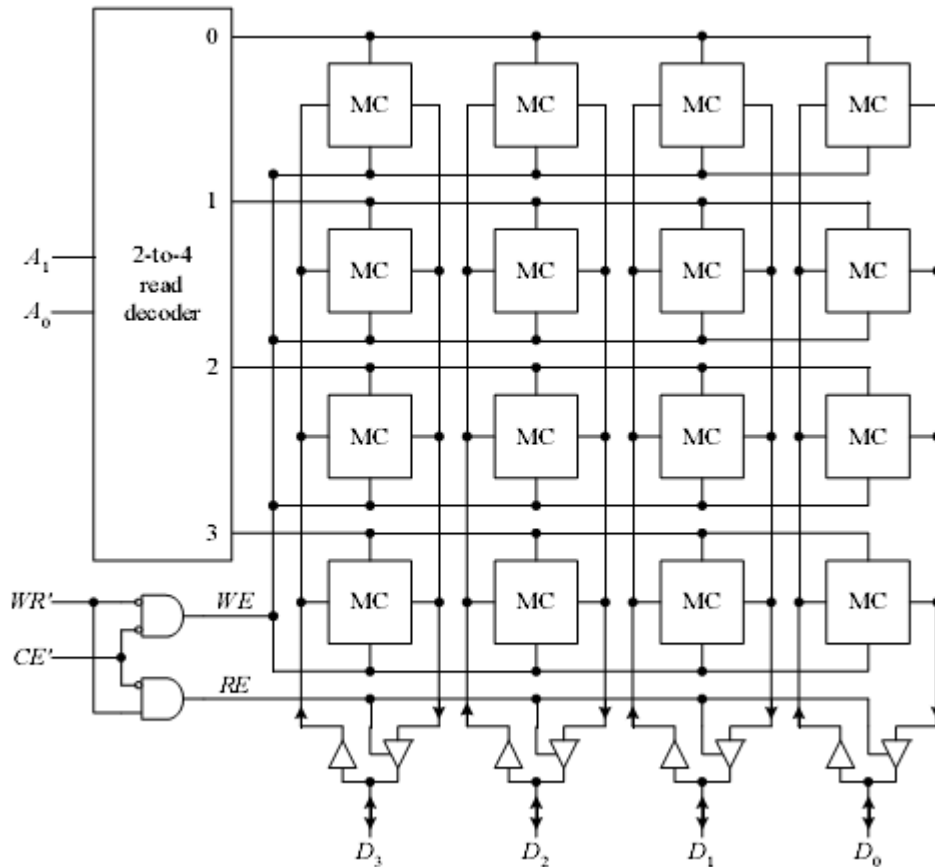
Mỗi bit trong RAM được lưu trữ trong 1 ô nhớ giống mạch ở hình 2.52. Phần tử nhớ chính trong ô nhớ là bộ chốt D có chân cho phép. Bộ đếm 3 trạng thái được nối ở ngõ ra của bộ chốt D để nó có thể lựa chọn đọc từ đâu. Tín hiệu cho phép ô được dùng để cho phép ô nhớ được đọc và ghi. Để đọc, tín hiệu cho phép ô được dùng để cho phép bộ đếm 3 trạng thái. Để ghi, tín hiệu cho phép ô cùng với tín hiệu cho phép ghi được dùng để cho phép bộ chốt để dữ liệu ở ngõ vào được chốt trong ô.



Hình 2. 52 : Mạch nhớ bit trong RAM.

Để tạo 1 chip RAM 4x4, chúng ta cần 16 ô nhớ dạng lưới 4x4 như hình 2.53. Mỗi hàng là 1 vị trí lưu trữ riêng và số ô nhớ trong mỗi hàng quyết định độ rộng bit ở mỗi vị trí. Vì vậy tất cả ô nhớ trong hàng có cùng địa chỉ. 1 Bộ giải mã được dùng để giải mã các địa chỉ. Trong ví dụ này là bộ giải mã 2-4 để giải mã 4 vị trí địa chỉ. Tín hiệu CE' cho phép chip, đặc biệt để cho phép chức năng đọc và ghi thông qua 2 cổng AND. Tín hiệu WE nội được tích cực (khi cả 2 tín hiệu CE' và WR' được tích cực) dùng để xác định cho phép ghi của tất cả các ô nhớ. Dữ liệu đi từ bus data bên ngoài qua ngõ vào bộ đếm và đến ngõ vào của mỗi ô nhớ.

Mục đích của việc dùng 1 bộ đệm ngõ vào cho mỗi đường dữ liệu là chỉ cần để tín hiệu bên ngoài vào để điều khiển đúng 1 linh kiện (bộ đệm) hơn là vài linh kiện (Ví dụ tất cả các ô nhớ trong cùng cột). Tùy thuộc vào địa chỉ được đưa ra mà hàng nào của các ô nhớ sẽ được ghi. Hoạt động đọc yêu cầu chân CE' được tích cực và chân WR' không được tích cực. Điều này sẽ tích cực tín hiệu RE nội, nó sẽ cho phép lần lượt 4 ngõ ra của bộ đệm 3 trạng thái ở cuối sơ đồ mạch. Nhắc lại, vị trí được đọc sẽ được chọn bằng địa chỉ.



Hình 2. 53 : Sơ đồ các ô nhớ dạng lưới trong chip RAM 4x4.

Mã VHDL cho chip RAM 16 x 4:

```

LIBRARY ieee;
USE ieee.STD_LOGIC_1164.ALL;
USE ieee.STD_LOGIC_arith.ALL;
USE ieee.STD_LOGIC_unsigned.ALL; -- needed for CONV_INTEGER()
ENTITY memory IS PORT (CE, WR: IN STD_LOGIC; --chip enable, write enable
    A: IN STD_LOGIC_VECTOR(3 DOWNTO 0); --address
    D: BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0) ); --data
END memory;
```

```

ARCHITECTURE Behavioral OF memory IS
BEGIN
    PROCESS (CE', WR')
        SUBTYPE cell IS STD_LOGIC_VECTOR(3 DOWNTO 0);
        TYPE memArray IS array(0 TO 15) OF cell;
        VARIABLE mem: memArray; --memory contents
        VARIABLE ctrl: STD_LOGIC_VECTOR(1 DOWNTO 0);
    BEGIN
        ctrl := CE & Wr; --group signals for CASE decoding
    CASE ctrl IS
        WHEN "10" =>                                     -- read
            D <= mem(CONV_INTEGER(A));                    -- fn TO convert from bit vecTOr TO integer
        WHEN "11" =>                                     -- write
            mem(CONV_INTEGER(A)) := D;                     -- fn TO convert from bit vecTOr TO integer
        WHEN OTHERS =>                                    -- invalid or not enable
            D <= (OTHERS => 'Z');
    END CASE;
    END PROCESS;
END Behavioral;

```

2.15 Bộ đếm (Counters):

Các bộ đếm, như tên gọi, dùng để đếm một chuỗi những giá trị. Tuy nhiên, có nhiều loại bộ đếm khác nhau phụ thuộc vào tổng số của các giá trị đếm, chuỗi những giá trị ngõ ra , hoặc là đếm lên hoặc đếm xuống.... Đơn giản nhất là bộ đếm modulo n là đếm dãy thập phân 0, 1, 2, lên tới n -1, và sau đó trở về 0. Một vài bộ đếm tiêu biểu được mô tả phía dưới.

Bộ đếm Modulo-n: đếm từ số thập phân 0 đến n-1 và trở về 0. Ví dụ, bộ đếm Modulo 5 đếm thập phân 0,1,2,3,4. Bộ đếm BCD: giống bộ đếm Modulo-n ngoại trừ n cố định là 10.

Bộ đếm n-bit nhị phân: giống như bộ đếm Modulo-n nhưng phạm vi từ 0 đến 2^n-1 và trở về 0 với n là số bit dùng trong bộ đếm. Ví dụ đếm 3-bit nhị phân tuần tự trong thập phân là 0,1,2,3,4,5,6,7.

Bộ đếm mã Gray: Chuỗi mã hóa để bất kỳ hai giá trị liên tiếp phải khác nhau chỉ 1 bit. Ví dụ 1 bộ đếm mã Gray 3 bit đếm lần lượt trong nhị phân là 000, 001, 011, 010, 110, 111, 101, 100.

Bộ đếm vòng: bắt đầu lần lượt với chuỗi bit 0 theo sau là bit 1. Bộ đếm chỉ đơn giản là dịch các bit qua trái của mỗi lần đếm. Ví dụ, bộ đếm vòng 4 bit tuần tự trong nhị phân là 0001, 0010, 0100, 1000.

2.15.1 Bộ đếm lên nhị phân (Binary Up Counter):

Một Bộ đếm nhị phân có thể được xây dựng bằng cách sửa đổi lại thanh ghi n bit tại dữ liệu ngõ vào của thanh ghi đến từ bộ cộng cho bộ đếm lên và mạch trừ cho bộ đếm xuống. Để bắt đầu với 1 giá trị được lưu trữ trong thanh ghi, tiếp tục đếm lên, đơn giản ta phải thêm 1 bit vào nó. Ta có thể dùng bộ cộng toàn phần được nói ở mục 2.3.1 như ngõ vào cho thanh ghi, nhưng chúng ta có thể làm tốt hơn. Bộ cộng toàn phần có 2 toán hạng cộng với số nhớ. Nhưng cái mà ta muốn chỉ là cộng thêm một, vì vậy toán hạng thứ hai của bộ cộng toàn phần luôn luôn là một. Vì cũng có thể cộng tín hiệu nhớ của bộ cộng, vì vậy, ta thật sự không cần ngõ vào cho toán hạng thứ hai. Bộ cộng được sửa đổi này chỉ cộng một toán hạng với số nhớ được gọi là bộ cộng bán phần (HA).

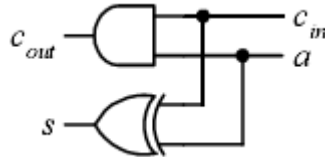
Bảng chân trị chỉ ở hình 2.54, ta chỉ có 1 toán hạng ngõ vào là a, c_{in} tương ứng với số nhớ ngõ vào và c_{out} tương ứng với số nhớ ngõ ra còn s là tổng của phép cộng. trong bảng chân trị ta chỉ đơn giản cộng a với số nhớ c_{in} để đưa ra tổng s và có thể có số nhớ ngõ ra c_{out} . Từ bảng chân trị, ta thu được 2 phương trình và mạch tương ứng như hình 2.54 và ký hiệu logic trong hình 2.54.

$$c_{out} = a c_{in}$$

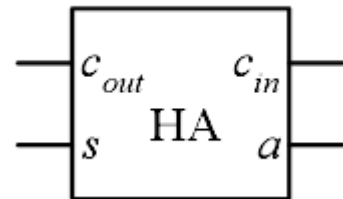
$$s = a \oplus c_{in}$$

a	c_{in}	c_{out}	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(a)



(b)

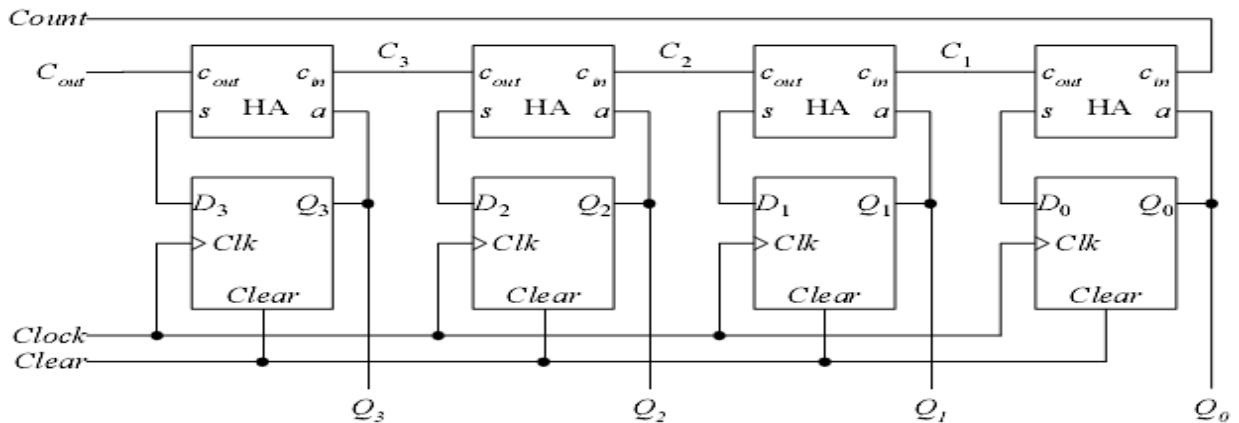


(c)

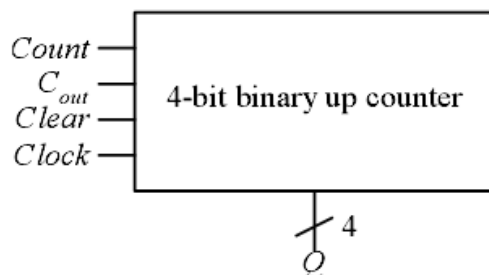
Hình 2. 54 : Bộ đếm lên nhị phân (a) Bảng chân trị; (b) Sơ đồ mạch; (c) Ký hiệu logic.

Các bộ cộng bán phần có thể kết hợp vòng thành bộ cộng toàn phần cộng n-bit. Mỗi ngõ vào toán hạng đến từ thanh ghi. Tín hiệu nhớ ngõ vào c_0 ban đầu dùng để cho phép tín hiệu đếm (count) vì bit '1' trên c_0 sẽ dẫn tới tăng một giá trị trong thanh ghi và '0' sẽ không có. Một mạch đếm lên 4-bit được chỉ như hình 2.55 với bảng chân trị và ký hiệu logic. Khi count được tích cực, bộ đếm sẽ tăng giá trị ở mỗi xung clock cho đến khi count không được tích cực. Khi count đạt đến 2^n-1 , tương đương với số nhị phân có tất cả '1', lần đếm tiếp

theo sẽ trở lại quay trở lại '0' vì việc thêm 1 bit vào một số nhị phân có tất cả các bit là '1' sẽ dẫn đến 1 bit tràn và tất cả các bit được reset về 0. Chân Clear cho phép reset bộ đếm không đồng bộ về 0.



Clear	Count	Operation
1	x	Reset counter to zero
0	0	No change
0	1	Count up



Hình 2. 55 : Bộ đếm lên 4 bit Sơ đồ mạch; bảng chân trị; ký hiệu logic.

Mã VHDL cho bộ đếm lên 4 bit:

```

ENTITY counter IS PORT (
    Clock: IN BIT;
    Clear: IN BIT;
    Count: IN BIT;
    Q : OUT INTEGER RANGE 0 TO 15);
END counter;
ARCHITECTURE Behavioral OF counter IS
BEGIN
    PROCESS (Clock, Clear)

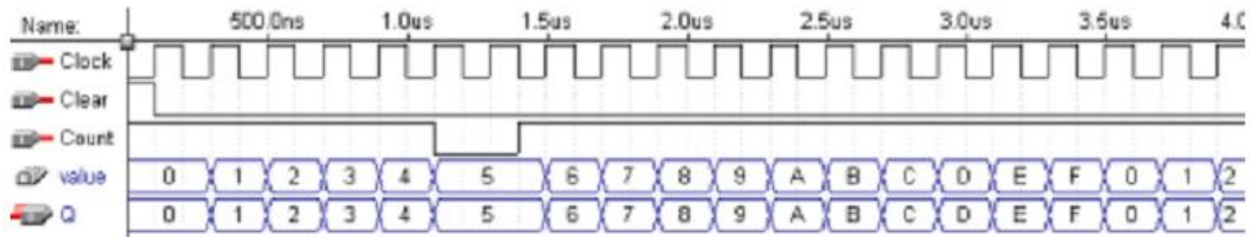
```

```

VARIABLE value: INTEGER RANGE 0 TO 15;
BEGIN
  IF Clear = '1' THEN
    value := 0;
  ELSIF (Clock'EVENT AND Clock='1') THEN
    IF Count = '1' THEN
      value := value + 1;
    END IF;
  END IF;
  Q <= value;
END PROCESS;
END Behavioral;

```

Tín hiệu mô phỏng ở hình sau



Hình 2. 56 : Tín hiệu mô phỏng cho bộ đếm lên 4 bit.

2.15.2 Bộ đếm lên xuống nhị phân (Binary Up-Down Counter):

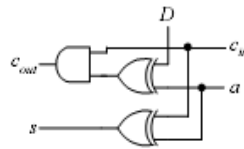
Chúng ta có thể thiết kế bộ đếm lên xuống n-bit giống như bộ đếm lên ngoại trừ việc cần cả bộ cộng và trừ cho dữ liệu ngõ vào thanh ghi. Bảng chân trị, mạch, ký hiệu logic của bộ cộng và trừ bán phần (HAS) ở hình 2.57a, hình 2.57b, hình 2.57c. Tín hiệu D để lựa chọn đếm lên hoặc xuống. Khi D=1 sẽ đếm xuống. Nửa trên của bảng chân trị là HA, nửa dưới là phép trừ của a-c_{in}, s là sự kết quả của phép trừ và c_{out}=1 nếu chúng ta cần mượn. Ví dụ 0-1, ta cần mượn nên c_{out}=1. Khi ta có a=2 và 2-1=1 nên s=a=1. Kết quả 2 phép tính là:

$$c_{out} = D' a c_{in} + D a' c_{in} = (D \oplus a) c_{in}$$

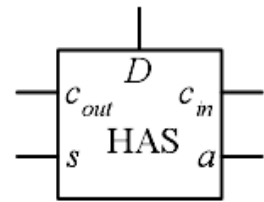
$$s = D' (a \oplus c_{in}) + D (a \oplus c_{in}) = a \oplus c_{in}$$

D	a	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	0	1
1	1	1	0	0

(a)



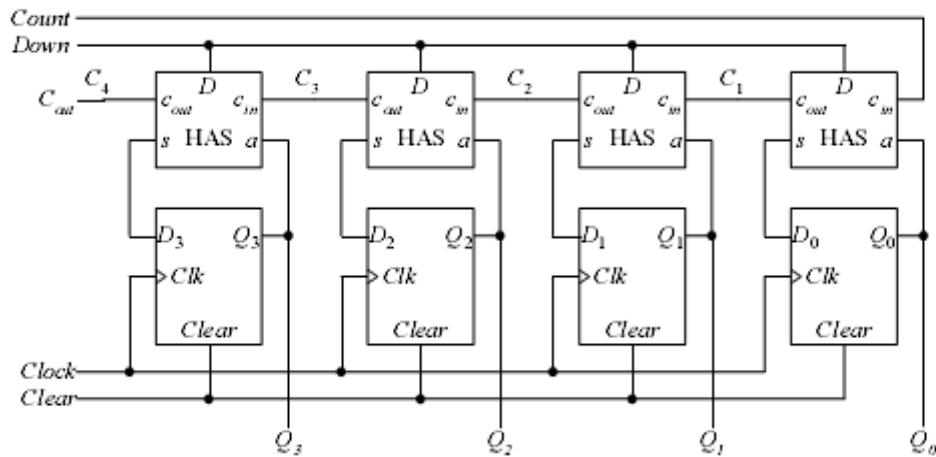
(b)



(c)

Hình 2. 57 : Bộ cộng ,trừ bán phần (a) Bảng chân trị; (b) Sơ đồ mạch; (c) Ký hiệu logic.

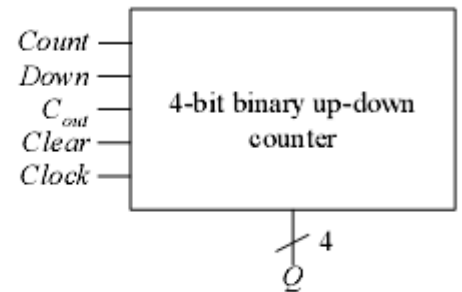
Một bộ đếm lên xuống 4 bit ở hình 2.58a. Bảng chân trị và ký hiệu logic ở hình 2.58b và c.



(a)

Clear	Count	Down	Operation
1	x	x	Reset counter to zero
0	0	x	No change
0	1	0	Count up
0	1	1	Count down

(b)



(c)

Hình 2. 58 : Bộ đếm lên xuống 4 bit: (a) Sơ đồ mạch; (b) Bảng chân trị; (c) Ký hiệu logic.

Mã VHDL cho 1 bộ đếm lên xuống 4 bit như sau:

```

ENTITY counter IS PORT (
    Clock: IN BIT;
    Clear: IN BIT;
    Count: IN BIT;

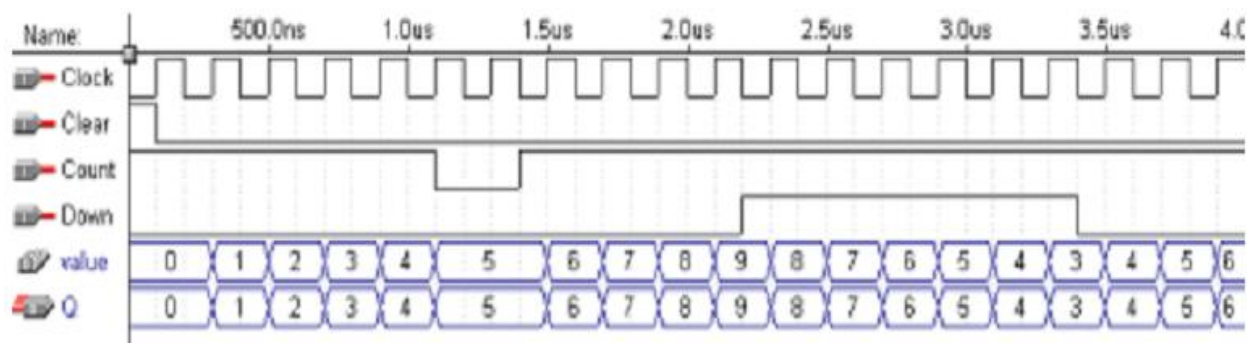
```

```

Down: IN BIT;
Q: OUT INTEGER RANGE 0 TO 15);
END counter;
ARCHITECTURE Behavioral OF counter IS
BEGIN
    PROCESS (Clock, Clear)
        VARIABLE value: INTEGER RANGE 0 TO 15;
    BEGIN
        IF Clear = '1' THEN
            value := 0;
        ELSIF (Clock'EVENT AND Clock='1') THEN
            IF Count = '1' THEN
                IF Down = '0' THEN
                    value := value + 1;
                ELSE
                    value := value - 1;
                END IF;
            END IF;
        END IF;
        Q <= value;
    END PROCESS;
END Behavioral;

```

Tín hiệu mô phỏng ở hình 2.59

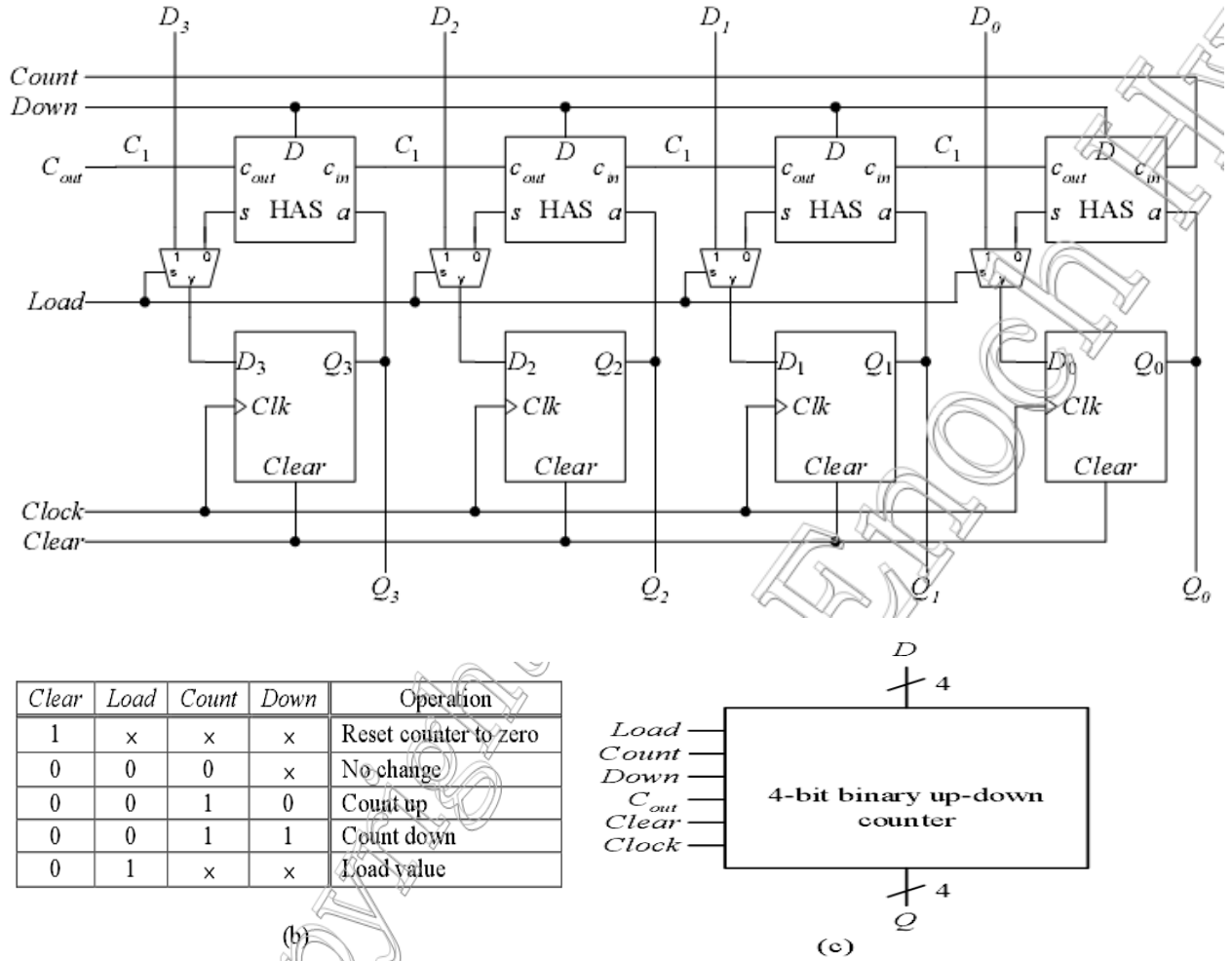


Hình 2. 59 : Tín hiệu mô phỏng cho bộ đếm lên xuống 4 bit.

2.15.3 Bộ đếm lên xuống đọc song song :

Để bộ đếm nhị phân linh hoạt hơn, ta cần bắt đầu chuỗi đếm với 1 số bất kỳ lớn hơn 0. Điều này dễ dàng được thực hiện bằng việc sửa đổi mạch đếm để cho phép nó mang một giá trị ban đầu. Với giá trị đã nạp vào trong thanh ghi, bây giờ chúng ta có thể đếm bắt đầu từ giá trị mới này. Sửa đổi mạch đếm như hình 2.60a.

Chỉ có 1 khác biệt giữa mạch này với mạch hình 2.58a là tổng của 2 ngõ vào bộ ghép kênh chính là ngõ ra s của HAS và ngõ vào D_i của flip-flop. Bằng cách làm này ngõ vào của flip-flop có thể được chọn từ giá trị ngõ vào bên ngoài nếu Load được tích cực hoặc từ giá trị đếm tiếp theo ở ngõ ra HAS nếu Load không được tích cực. Nếu ngõ ra HAS được chọn khi đó mạch làm việc chính xác như trước đây. Nếu ngõ vào bên ngoài được chọn, khi đó bất kỳ giá trị nào trên ngõ vào dữ liệu cũng được đọc vào thanh ghi. Bảng hoạt động và ký hiệu của mạch ở hình 2.60b và 2.60c.



Hình 2. 60 : (a) Sơ đồ mạch đếm lên xuống 4 bit có sửa đổi ; (b) Bảng chân trị ; (c) ký hiệu logic của đếm lên xuống 4 bit có sửa đổi.

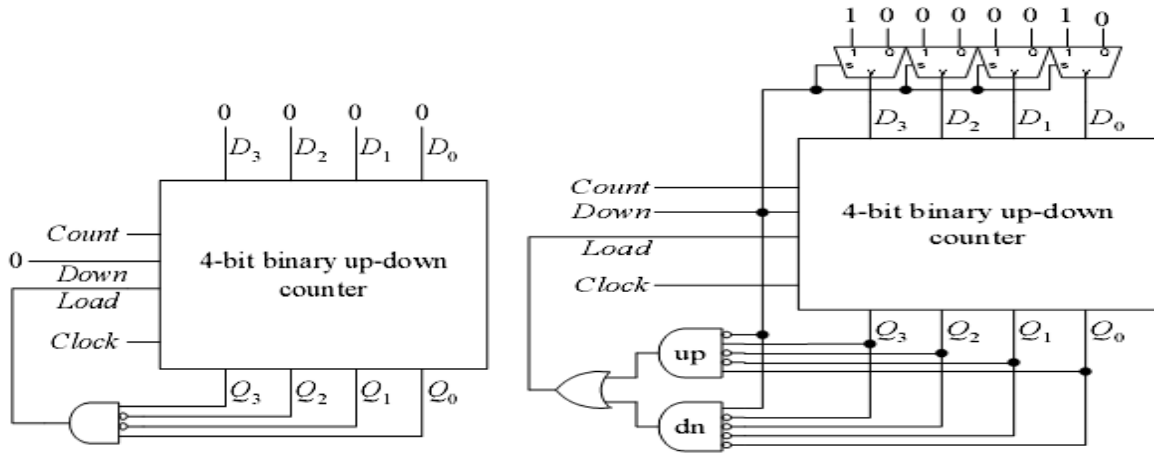
Chúng ta phải giữ chân Clear để bộ đếm có thể tạo giá trị ban đầu đến 0 tại bất kỳ thời điểm nào. *Chú ý rằng có sự khác biệt thời gian giữa việc tích cực chân CLea để reset bộ đếm về 0 khác với tích cực chân Load để đọc giá trị 0 và đặt dữ liệu ngõ vào là 0.* Trong trường hợp đầu tiên, bộ đếm được reset về 0 ngay lập tức sau khi chân Clear được tích cực trong khi trường hợp sau sẽ reset bộ đếm về 0 ở cạnh lên xung clock tiếp theo .

Với mạch này, việc đếm sẽ bắt đầu với 1 giá trị bất kỳ trong thanh ghi. Tuy nhiên khi bộ đếm tiến tới giá trị cuối của chuỗi đếm nó sẽ trở về 0 và không bắt đầu với giá trị mới này. Chúng ta có thể thêm một mạch so sánh đơn giản vào bộ đếm này để nó quay trở lại giá trị ngõ vào mới này hơn là 0 như ở mục tiếp theo.

2.15.4 Bộ đếm lên xuống BCD (BCD Up-Down Counter):

Vấn đề của bộ đếm lên xuống nhị phân với đọc song song là khi mà nó tiến về giá trị cuối của bộ đếm, thì nó luôn quay trở về 0. Nếu chúng ta muốn chuỗi đếm quay trở lại chu kỳ với giá trị được gán ban đầu sau mỗi lần như vậy, ta cần tích cực chân Load, tại bắt đầu của mỗi chu kỳ đếm và giá trị nạp ban đầu sẽ vẫn còn tồn tại trên các chân ngõ vào dữ liệu. Để tạo ra một tín hiệu tích cực chân Load, ta cần kiểm tra giá trị đếm là trị cuối của chuỗi đếm. Nếu ngõ ra là '1' để tích cực chân Load. Điều này nạp trở lại giá trị ban đầu và đếm tiếp tục với giá trị ban đầu này. Mạch này chỉ là mạch so sánh với các ngõ vào thiết lập từ thanh ghi và các ngõ khác là các hằng số mà ta dùng để kiểm tra. Ngõ ra của bộ so sánh sẽ tích cực chân Load.

Bộ đếm lên xuống BCD đếm từ 0 đến 9 cho chuỗi lên và ngược lại cho chuỗi xuống. Cho chuỗi lên khi đếm đến 9 chúng ta cần tích cực chân Load và đọc về '0'. Cho chuỗi xuống khi đếm đến 0 chúng ta cần tích cực chân Load và đọc về '9'. Đếm lên BCD ở hình 2.61 a, Đếm lên-xuống BCD ở hình 2.61b.



Hình 2. 61 : Bộ đếm BCD (a) bộ đếm lên; (b) bộ đếm xuống.

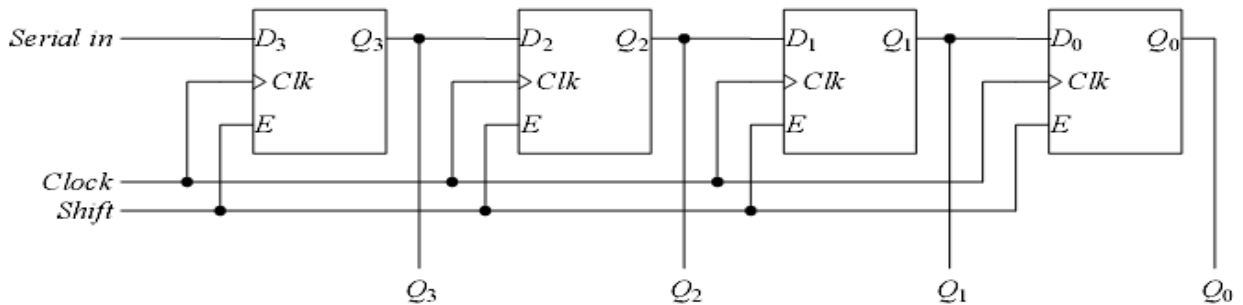
2.16 Thanh ghi dịch (Shift registers):

Giống như mạch kết hợp giữa quay và dịch chuyển. Đó là các thành phần dịch và quay tuần tự. Mạch để thực hiện chuyển dịch và quay được xây dựng cùng giống như vậy. Sự khác nhau duy nhất trong phần tuần tự là những thao tác được thực hiện trên giá trị được lưu trữ trong một thanh ghi. Cách dùng chính cho một thanh ghi dịch là để chuyển đổi từ một dòng

dữ liệu ngõ vào tuần tự thành một ngõ ra dữ liệu song song hay ngược lại. Để chuyển dữ liệu nối tiếp ra song song, các bit được chuyển vào trong thanh ghi tại mỗi chu kỳ xung clock và khi tất cả các bit (thường 8 bits) được chuyển vào trong thanh ghi, thanh ghi 8-bit có thể được đọc để xuất 8 bit ở ngõ ra song song. Để chuyển từ song song ra nối tiếp, trước tiên thanh ghi 8 bit đọc dữ liệu ngõ vào, các bit được dịch ra ngoài từng bit một, mỗi bit là 1 chu kỳ xung clock.

2.16.1 Thanh ghi dịch nối tiếp ra song song:

Hình 2.62 chỉ một bộ chuyển đổi 4 bit nối tiếp ra song song. Các bit dữ liệu ngõ vào được đưa vào từ đường Serial In. Khi chân Shift được tích cực, các bit dữ liệu được dịch vào trong. Tại xung Clock đầu, bit đầu tiên được đọc vào trong Q3. Tại xung Clock thứ 2, bit trong Q3 được đọc vào Q2 trong khi đó Q3 đọc bit tiếp theo của dòng dữ liệu ngõ vào nối tiếp. Cứ tiếp tục hết 4 xung clock thì 4 bits đã được dịch vào trong 4 flip-flop.



Hình 2. 62 : Bộ chuyển đổi 4 bit nối tiếp ra song song.

Mã VHDL theo cấu trúc cho thanh ghi dịch 4-bit vào nối tiếp ra song song như sau:

```
-- D flip-flop with enable
LIBRARY ieee;
USE IEEE.std_logic_1164.all;
ENTITY D_flipflop IS
    PORT(D, Clock,E:IN STD_LOGIC;
         Q : OUT STD_LOGIC);
END D_flipflop;
ARCHITECTURE Behavior OF D_flipflop IS
BEGIN
    PROCESS(Clock)
    BEGIN
        IF Clock'EVENT AND Clock = '1' THEN
            IF E = '1' THEN
                Q<=D;
            END IF;
        END IF;
    END IF;
END;
```



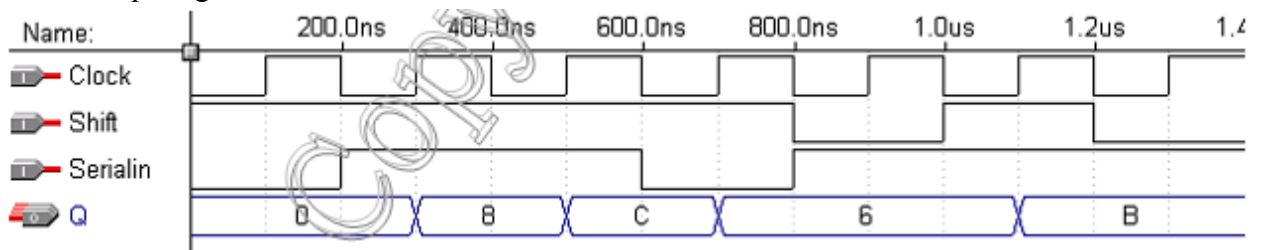
```

END PROCESS;
END Behavior;

-- 4-bit shift register
LIBRARY ieee;
USE IEEE.std_logic_1164.all;
ENTITY ShiftReg IS
    PORT(Serialin, Clock, Shift : IN STD_LOGIC;
          Q : OUT STD_LOGIC_VECTOR(3 downto 0));
END ShiftReg;
ARCHITECTURE Structural OF ShiftReg IS
    SIGNAL N0, N1, N2, N3 : STD_LOGIC;
    COMPONENT D_flipflop PORT (D, Clock,E:IN STD_LOGIC;
                                Q : OUT STD_LOGIC);
    END COMPONENT;
    BEGIN
        U1: D_flipflop PORT MAP (Serialin, Clock, Shift, N3);
        U2: D_flipflop PORT MAP (N3, Clock, Shift, N2);
        U3: D_flipflop PORT MAP (N2, Clock, Shift, N1);
        U4: D_flipflop PORT MAP (N1, Clock, Shift, N0);
        Q(3) <= N3;
        Q(2) <= N2;
        Q(1) <= N1;
        Q(0) <= N0;
    END Structural;

```

Tín hiệu mô phỏng ở hình 2.63



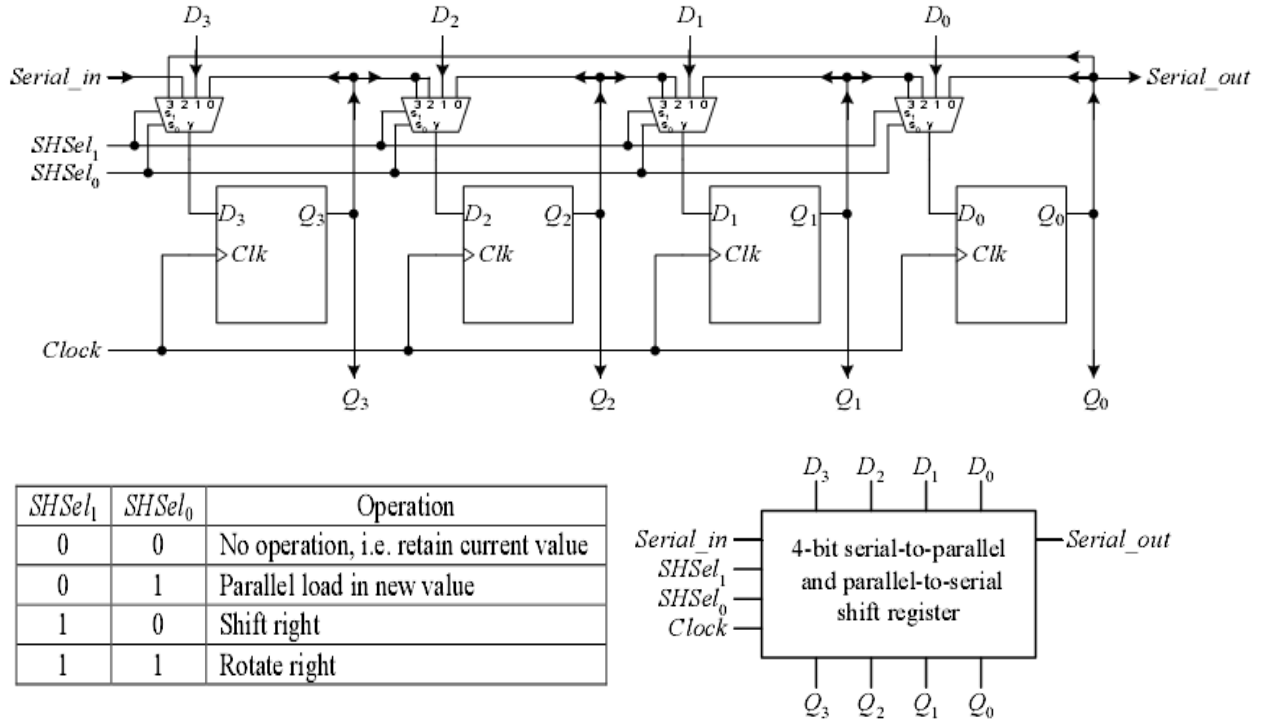
Hình 2. 63 : Tín hiệu mô phỏng của một bộ chuyển đổi 4 bit nối tiếp ra song song.

2.16.2 Thanh ghi dịch nối tiếp ra song song và song song ra nối tiếp:

Cả hoạt động chuyển nối tiếp ra song song và song song ra nối tiếp, chúng ta đều làm dịch bit từ trái qua phải thông qua thanh ghi. Điểm khác biệt giữa 2 giải thuật là khi bạn thực hiện đọc song song sau khi dịch hoặc ghi song song trước khi dịch. Để chuyển nối tiếp ra song song bạn phải đọc song song sau khi bit được dịch vào trong. Còn để chuyển song song ra nối tiếp, bạn phải ghi song song trước và sau đó dịch các bit ra ngoài như dòng nối

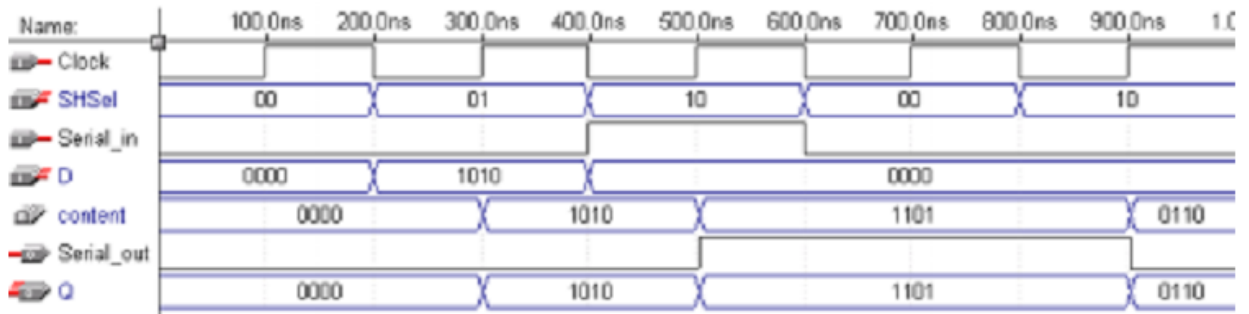
tiếp. Chúng ta có thể thực hiện cả hai chức năng vào trong mạch nối tiếp ra song song từ phần trước 1 cách đơn giản bằng việc thêm một chức năng Load song song vào mạch như trong hình 2.64a.

Bốn bộ ghép kênh làm việc với nhau để chọn những flip-flop mà bạn muốn giữ lại giá trị hiện tại, đọc vào một giá trị mới hay dịch những bit qua phải 1 vị trí bit. Hoạt động của mạch này tùy thuộc vào hai chân lựa chọn $SHSel_1$ Và $SHSel_0$, nó điều khiển ngõ vào của những bộ ghép kênh được chọn. Bảng hoạt động và ký hiệu logic ở hình 2.64b và 2.64c.



Hình 2. 64 : (a) Sơ đồ mạch thanh ghi dịch nối tiếp ra song song và song song ra nối tiếp; (b) Bảng chân trị ; (c) ký hiệu logic của thanh ghi dịch nối tiếp ra song song và song song ra nối tiếp.

Tín hiệu mô phỏng ở hình 2.65:



Hình 2. 65 : Tín hiệu mô phỏng thanh ghi dịch nối tiếp ra song song và song song ra nối tiếp.

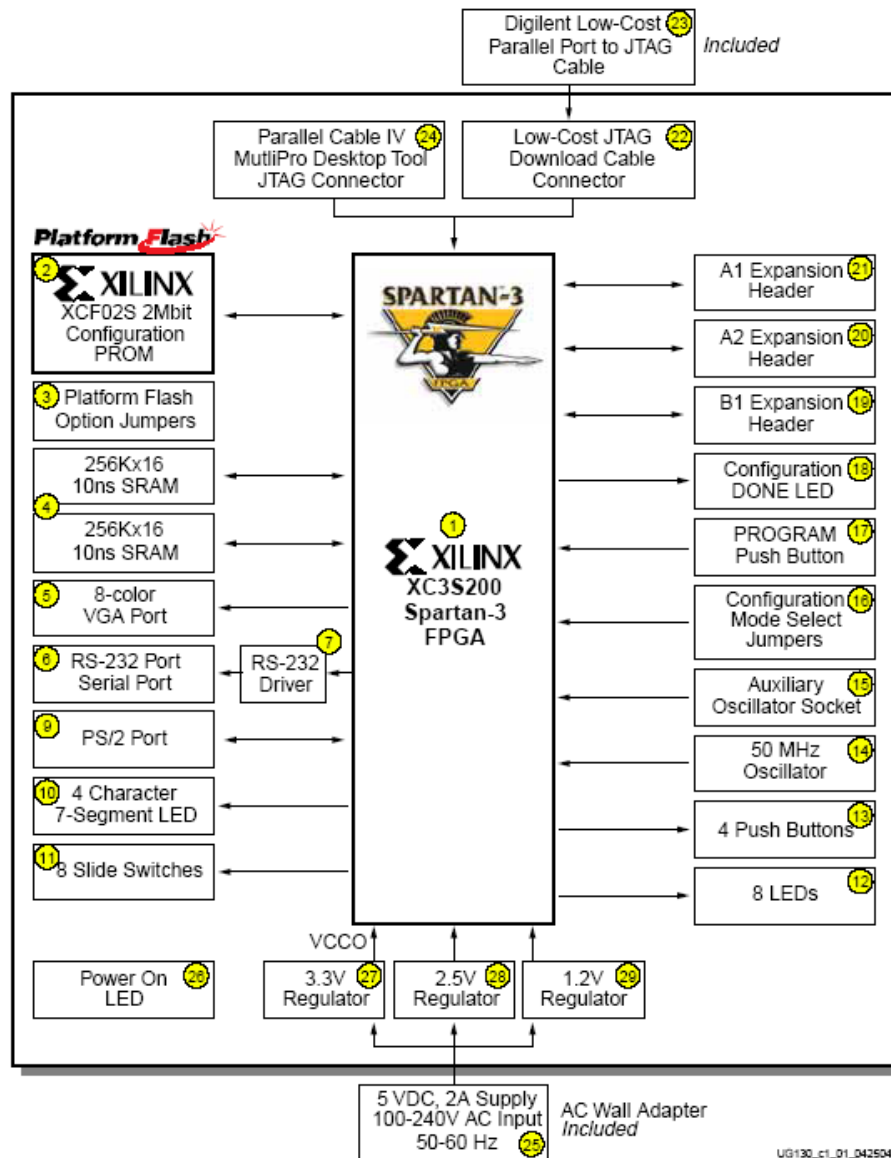
Mã VHDL mô tả cho thanh ghi dịch nối tiếp ra song song và song song ra nối tiếp.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY shiftreg IS PORT (
    Clock: IN STD_LOGIC;
    SHSel: IN STD_LOGIC_VECTOR(1 DOWNTO 0);
    Serial_in: IN STD_LOGIC;
    D: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    Serial_out: OUT STD_LOGIC;
    Q: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END shiftreg;
ARCHITECTURE Behavioral OF shiftreg IS
    SIGNAL content: STD_LOGIC_VECTOR(3 DOWNTO 0);
    BEGIN
        PROCESS(Clock)
            BEGIN
                IF (Clock'EVENT AND Clock='1') THEN
                    CASE SHSel IS
                        WHEN "01" => -- load
                            content <= D;
                        WHEN "10" => -- shift right, pad with bit from Serial_in
                            content <= Serial_in & content(3 DOWNTO 1);
                        WHEN OTHERS =>
                            NULL;
                    END CASE;
                END IF;
            END PROCESS;
            Q <= content;
            Serial_out <= content(0);
        END Behavioral;
```

CHƯƠNG 3 : TÌM HIỂU KIT FPGA SPARTAN 3

3.1 Tổng quan kit FPGA Spartan 3 :

Mạch nạp Xilinx Spartan-3 Starter Kit Board (của hãng Xilinx) được cung cấp với giá rẻ, dễ sử dụng để phát triển và đánh giá mạch số. Xilinx Spartan-3 Starter Kit Board là một mạch FPGA.



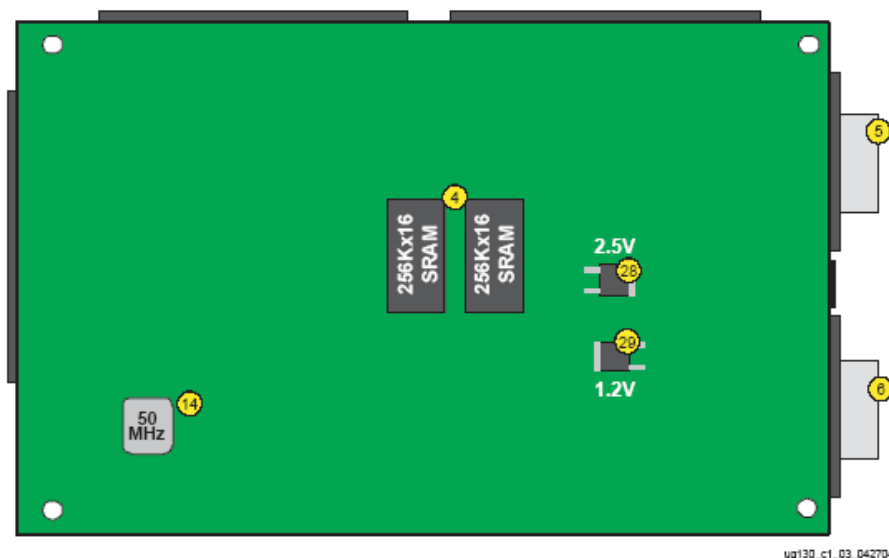
Hình 3. 1 : Sơ đồ khối kit Xilinx FPGA Spartan-3 Starter.

Các thành phần của Kit gồm có:

- 200000 cổng logic, tương đương 4320 tế bào logic (1).
- 2 Mbit PROM (2).

- 1 Mbyte SDRAM (4).
- Cổng màn hình VGA 3 bit, 8 màu (5).
- Cổng nối tiếp RS-232 (6).
- Cổng PS/2 dùng cho chuột hoặc bàn phím (9).
- Hệ thống 8 công tắc trượt, 4 nút nhấn và 4 LED 7 đoạn hiển thị (11, 13, 10).
- Xung clock 50 Mhz dùng dao động thạch anh (14).
- Khe cắm cho xung clock từ bên ngoài (15).
- Hệ thống các jumper để chọn chế độ làm việc cho các linh kiện trên board (3, 8, 16, 30, 31).
- 3 cổng cắm mở rộng, mỗi cổng 40 chân (19, 20, 21).
- Cổng JTAG kết nối với máy tính dùng để nạp chương trình vào kit Spartan 3 và gỡ rối (22).
- Hệ thống nguồn (25).

Hình 3. 2: Mạch in phía trước kit FPGA Xilinx Spartan-3 Starter.

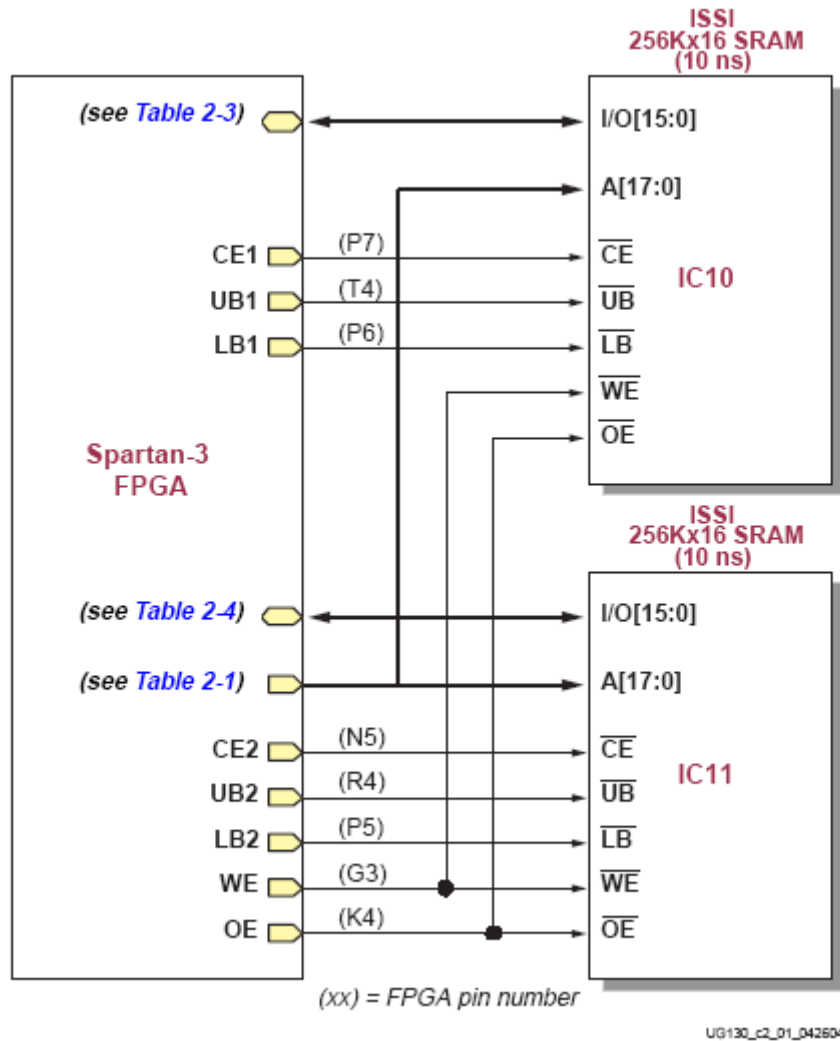


Hình 3. 3 : Mạch in phía sau kit FPGA Xilinx Spartan-3 Starter.

3.2 SRAM bất đồng bộ :

Kít gồm có 2 chip Ram 256K x 16 chia sẻ chung các ngõ điều khiển cho phép ghi (WE), cho phép ngõ ra (OE) và 18 đường địa chỉ. Chúng ta có thể sử dụng 2 SRAM một cách riêng biệt 256Kx16 hay cũng có thể sử dụng kết hợp 2 SRAM này lại thành 1 SRAM 256Kx32. Mỗi SRAM 256Kx16 được điều khiển bởi một con chip select riêng biệt thông qua chân (CE). Các chân điều khiển còn lại như CS, UB, LB được điều khiển riêng biệt.

Cả hai chip 256Kx16 SRAM cùng chia sẻ chung 18 đường điều khiển địa chỉ. Những đường địa chỉ này cũng được nối đến 18 chân của phần kết nối mở rộng A1 của board mạch. Sơ đồ kết nối các chân được biểu hiện rõ trong hình 2.5.



Hình 3. 4 : Sơ đồ kết nối giữa chân giữa FPGA và 2 SRAM 256Kx16.

Address Bit	FPGA Pin	A1 Expansion Connector Pin
A17	L3	35
A16	K5	33
A15	K3	34
A14	J3	31
A13	J4	32
A12	H4	29
A11	H3	30
A10	G5	27
A9	E4	28
A8	E3	25
A7	F4	26
A6	F3	23
A5	G4	24
A4	L4	14
A3	M3	12
A2	M4	10
A1	N3	8
A0	L5	6

Hình 3. 5 : Bảng kết nối chân giữa FPGA với 18 đường địa chỉ của SRAM

Tương tự như cách kết nối trên chân WE và OE cũng được nối đến phần kết nối mở rộng A1.

Signal	FPGA Pin	A1 Expansion Connector Pin
OE#	K4	16
WE#	G3	18

Hình 3. 6 : Bảng kết nối chân giữa FPGA với chân OE và WE của

Các chip select IC10 và IC11 cũng được kết nối đến các chân của FPGA theo sơ đồ chân như sau:

Signal	FPGA Pin	A1 Expansion Connector Pin
IO15	R1	
IO14	P1	
IO13	L2	
IO12	J2	
IO11	H1	
IO10	F2	
IO9	P8	
IO8	D3	
IO7	B1	19
IO6	C1	17
IO5	C2	15
IO4	R5	13
IO3	T5	11
IO2	R6	9
IO1	T8	7
IO0	N7	5
CE1 (chip enable IC10)	P7	
UB1 (upper byte enable IC10)	T4	
LB1 (lower byte enable IC10)	P6	

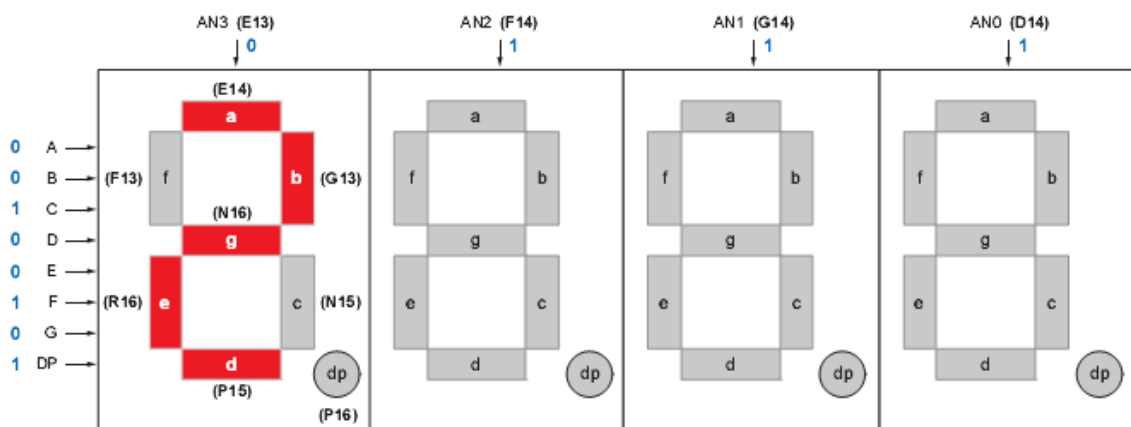
Hình 3. 7 : Bảng kết nối chân giữa IC10 với các chân của FPGA.

Signal	FPGA Pin
IO15	N1
IO14	M1
IO13	K2
IO12	C3
IO11	F5
IO10	G1
IO9	E2
IO8	D2
IO7	D1
IO6	E1
IO5	G2
IO4	J1
IO3	K1
IO2	M2
IO1	N2
IO0	P2
CE2 (chip enable IC11)	N5
UB2 (upper byte enable IC11)	R4
LB2 (lower byte enable IC11)	P5

Hình 3. 8 : Bảng kết nối chân giữa IC11 với các chân của FPGA.

3.3 Led 7 đoạn:

Các ký tự có thể được hiển thị bằng 4 LED 7 đoạn , được điều khiển bằng các chân I/O của người sử dụng như hình 2.9



Hình 3. 9 : Sơ đồ bố trí các thanh của LED 7 đoạn.

Mỗi số chia sẽ 8 đường tín hiệu chung để làm sáng các đoạn (segment) LED riêng biệt .
Mỗi 1 LED có 1 ngõ vào điều khiển anode riêng (tích cực mức thấp, từ AN3 đến AN0).

Để bật 1 đoạn trong LED 7 đoạn sáng lên, ta cho tín hiệu điều khiển riêng biệt cho đoạn tương ứng xuống mức 0 (dp = MSB , a = LSB) .

Segment	FPGA Pin
A	E14
B	G13
C	N15
D	P15
E	R16
F	F13
G	N16
DP	P16

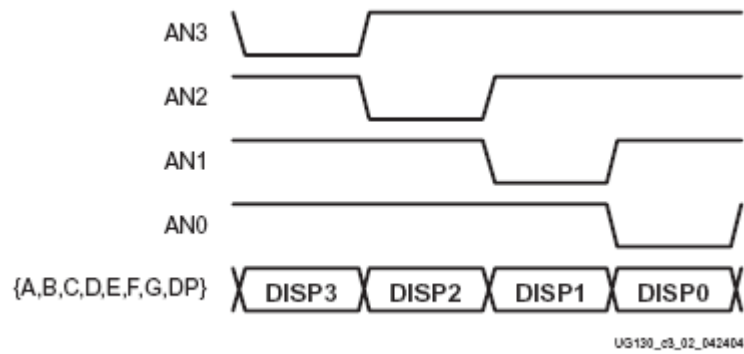
Hình 3. 10 : Bảng kết nối chân giữa LED 7 đoạn với chân của FPGA.

Anode Control	AN3	AN2	AN1	AN0
FPGA Pin	E13	F14	G14	D14

Hình 3. 11 : Bảng kết nối tín hiệu điều khiển hiển thị 4 LED với chân của FPGA.

Character	a	b	c	d	e	f	g
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0
A	0	0	0	1	0	0	0
b	1	1	0	0	0	0	0
C	0	1	1	0	0	0	1
d	1	0	0	0	0	1	0
E	0	1	1	0	0	0	0
F	0	1	1	1	0	0	0

Hình 3. 12 : Bảng hiển thị LED 7 đoạn tương ứng với 16 ký tự từ 0 đến F.



Hình 3. 13 : Tín hiệu mô tả hiển thị các LED 7 đoạn bằng phương pháp quét led.

3.4 Các công tắc trượt (SW), các nút ấn (PB) và các Led :

Có 8 công tắc trượt có nhãn SW7 (bên trái) đến SW0 (bên phải).

Switch	SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0
FPGA Pin	K13	K14	J13	J14	H13	H14	G12	F12

Hình 3. 14 : Bảng kết nối chân giữa các công tắc trượt với các chân của FPGA.

Khi UP hoặc ON công tắc kết nối chân FPGA lên V_{cc0} , logic cao. Khi DOWN hoặc OFF công tắc kết nối chân FPGA xuống GROUND , logic thấp. Công tắc hiển thị biểu thị khoảng 2 ms cho nảy giạt cơ khí . Một điện trở $4.7\text{ K}\Omega$ nối tiếp cung cấp bảo vệ đầu vào.

Các nút ấn: BTN3 ở ngoài cùng bên trái , BTN0 ở ngoài cùng bên phải. Khi ấn nút sẽ tạo mức cao ở chân FPGA.

Push Button	BTN3 (User Reset)	BTN2	BTN1	BTN0
FPGA Pin	L14	L13	M14	M13

Hình 3. 15 : Bảng kết nối chân giữa các nút nhấn với các chân của FPGA.

LED	LD7	LD6	LD5	LD4	LD3	LD2	LD1	LD0
FPGA Pin	P11	P12	N12	P13	N14	L12	P14	K12

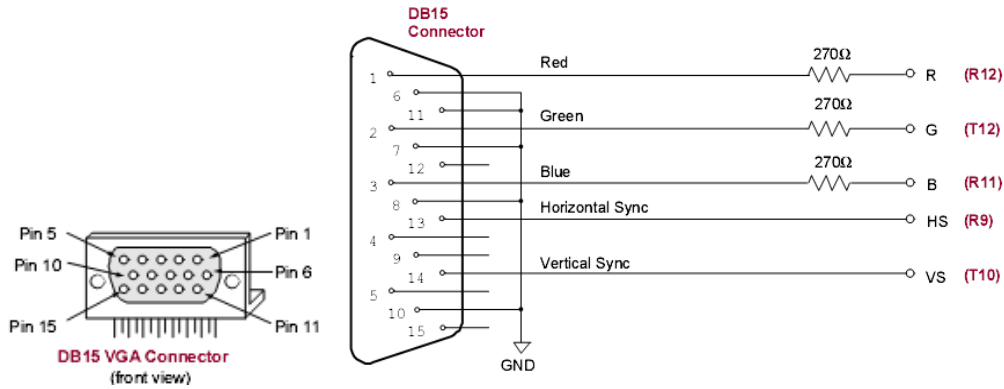
Hình 3. 16 : Bảng kết nối chân giữa 8 đèn LED với các chân của FPGA.

3.5 Cổng VGA :

Cổng hiển thị màn hình VGA và connector DB15.

Kết nối cổng này trực tiếp tới hầu hết những màn hình PC hay panel hiển thị màn hình LCD sử dụng một cáp monitor tiêu chuẩn

Điều khiển 5 tín hiệu VGA : Red , Green , Blue, Horizontal Sync, Vertical Sync , tất cả có sẵn trên connector VGA.



Hình 3. 17 : Sơ đồ chân của cổng VGA

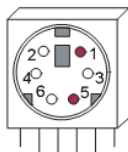
Signal	FPGA Pin
Red (R)	R12
Green (G)	T12
Blue (B)	R11
Horizontal Sync (HS)	R9
Vertical Sync (VS)	T10

Hình 3. 18 : Bảng kết nối chân giữa các tín hiệu của cổng với các chân của FPGA.

Red (R)	Green (G)	Blue (B)	Resulting Color
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White

Hình 3. 19 : Bảng mã hóa hiển thị 3 bit cho 8 màu cơ bản.

3.6 Cổng PS/2 Mouse và Keyboard :



PS/2 DIN Pin	Signal	FPGA Pin
1	DATA (PS2D)	M15
2	Reserved	—
3	GND	GND
4	Voltage Supply	—
5	CLK (PS2C)	M16
6	Reserved	—

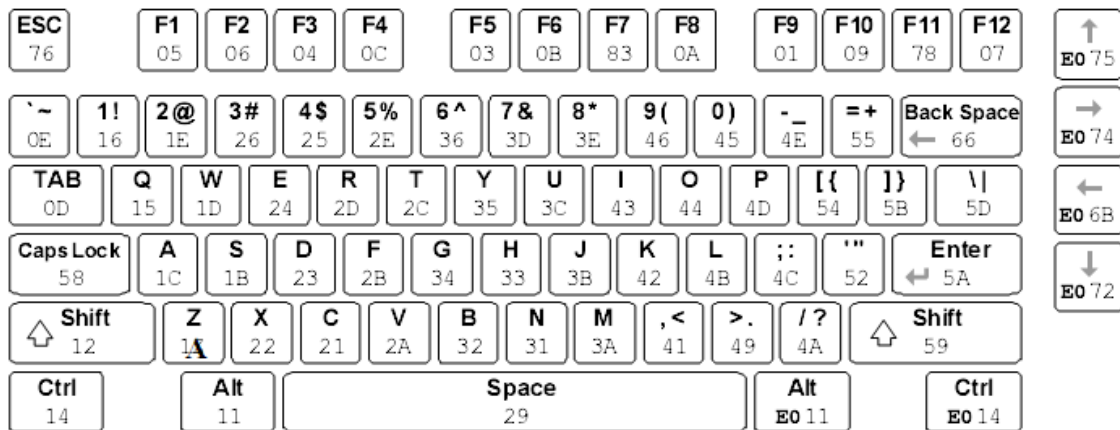
Hình 3. 20 : Sơ đồ chân của cổng PS/2.

Chuột và bàn phím đều sử dụng 2 dây của Bus nối tiếp PS/2 để trao đổi thông tin với thiết bị chủ (trong trường hợp này là FPGA Spartan III).

Bus PS/2 bao gồm xung clock và data. Cả 2 đều sử dụng từ 11 bit gồm : 1 bit start = 0; 8 bit data (LSB trước tiên); 1 bit odd parity; 1 bit stop = 1.

Tuy nhiên gói data của Mouse và keyboard là khác nhau.

3.6.1 Bàn phím :



Hình 3. 21 : Mã quét bàn phím.

Table 6-3: Common PS/2 Keyboard Commands

Command	Description																
ED	<p>Turn on/off Num Lock, Caps Lock, and Scroll Lock LEDs. The keyboard acknowledges receipt of an “ED” command by replying with an “FA”, after which the host sends another byte to set LED status. The bit positions for the keyboard LEDs appear in Table 6-4. Write a ‘1’ to the specific bit to illuminate the associated keyboard LED.</p> <p>Table 6-4: Keyboard LED Control</p> <table><tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr><tr><td colspan="5">Ignored</td><td>Caps Lock</td><td>Num Lock</td><td>Scroll Lock</td></tr></table>	7	6	5	4	3	2	1	0	Ignored					Caps Lock	Num Lock	Scroll Lock
7	6	5	4	3	2	1	0										
Ignored					Caps Lock	Num Lock	Scroll Lock										
EE	Echo. Upon receiving an echo command, the keyboard replies with the same scan code “EE”.																
F3	Set scan code repeat rate. The keyboard acknowledges receipt of an “F3” by returning an “FA”, after which the host sends a second byte to set the repeat rate.																
FE	Resend. Upon receiving a resend command, the keyboard resends the last scan code sent.																
FF	Reset. Resets the keyboard.																

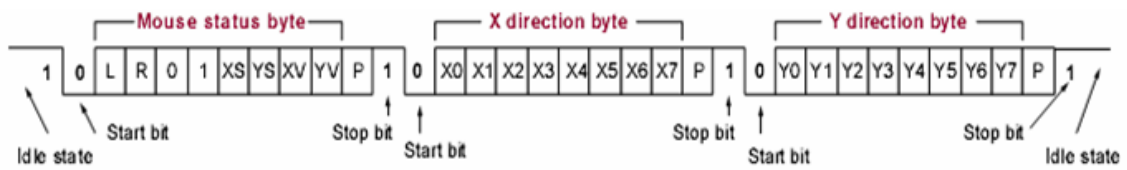
Hình 3. 22 : Các mã điều khiển đặc biệt của bàn phím.

3.6.2 Mouse :

Mouse tạo ra tín hiệu data và xung clock khi di chuyển, trường hợp còn lại các tín hiệu ở mức cao để cho biết trạng thái rảnh Idle.

Mỗi lần Mouse di chuyển nó gửi 3 từ 11 bit đến host. Mỗi từ 11 bit chứa bit start '0', tiếp theo là 8 bit data (đầu tiên là LSB), sau đó là bit parity lẻ, cuối cùng là stop bit '1'.

Data chỉ có giá trị ở cạnh xuống của xung clock , chu kỳ xung clock từ 20 Khz đến 30 Khz.



Hình 3. 23 : Cấu trúc luồng bit quản lý cổng PS/2.

3.6.3 Nguồn cấp áp:

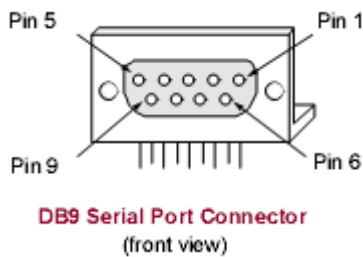
Hầu hết Mouse và Keyboard làm việc với áp 3.3 V hoặc 5V. Nguồn từ cổng PS/2 được chọn qua đường JP2.

PS/2 Port Supply Voltage	Jumper JP2 Setting
3.3V (DEFAULT)	3.3V JP2 VU
5V	3.3V JP2 VU

Hình 3. 24 : Cách kết nối jumper trên board để chọn nguồn áp tùy người thiết kế.

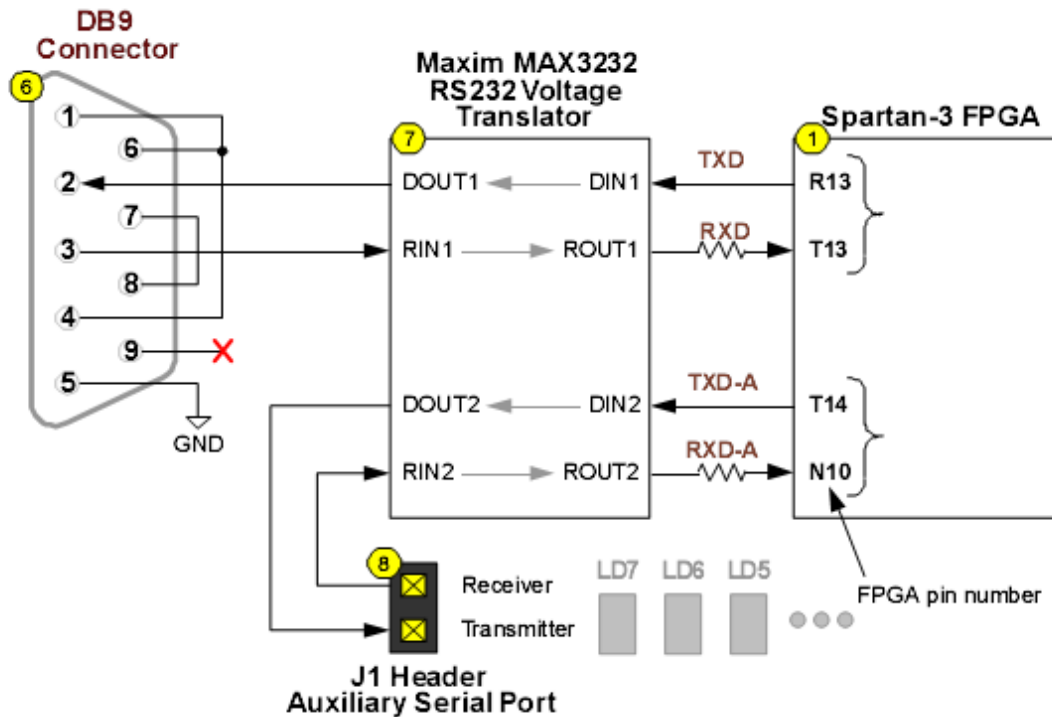
3.7 Cổng nối tiếp RS-232 :

RS 232 truyền và nhận tín hiệu xuất hiện trên connector DB9 female, nhãn J2. Sử dụng cáp nối tiếp truyền thẳng từ Kit FPGA đến cổng nối tiếp PC.



Signal	FPGA Pin
RXD	T13
TXD	R13
RXD-A	N10
TXD-A	T14

Hình 3. 25 : Sơ đồ chân của cổng RS-232.



Hình 3. 26 : Sơ đồ kết nối chân giữa cổng RS-232 với các chân của FPGA.

3.8 Các nguồn xung clock :











FPGA có thể sử dụng xung clock chính của nó là 50 Mhz hoặc sử dụng tần số khác lấy từ các bộ quản xung clock số (Digital Clock Managers DCMs) của FPGA.

Oscillator Source	FPGA Pin
50 MHz (IC4)	T9
Socket (IC8)	D9

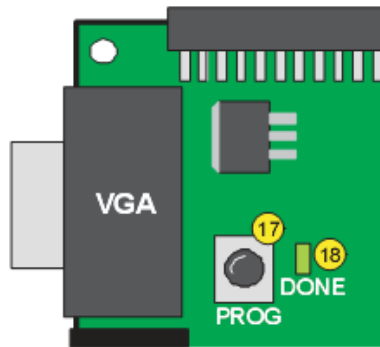
Hình 3. 27 : Kết nối chân giữa nguồn dao động xung clock với chân của FPGA.

3.9 Cách thiết lập các mode hoạt động cho FPGA :

Trong hầu hết các ứng dụng của Spartan 3, FPGA tự động boot khi được cấp nguồn hoặc ấn nút PROG. Khi sử dụng mode cấu hình Master Serial thì phải thiết lập JP1

Configuration Mode <M0:M1:M2>	Header J8 Settings	Jumper JP1 Setting	Description
Master Serial <0:0:0>		 JP1 or  JP1	DEFAULT. The FPGA automatically boots from the Platform Flash.
Slave Serial <1:1:1>		 JP1	The FPGA attempts to boot from a serial configuration source attached to either expansion connector A2 or B1.
Master Parallel <1:1:0>		 JP1	The FPGA attempts to boot from a parallel configuration source attached to the B1 expansion connector.
Slave Parallel <0:1:1>		 JP1	Another device connected to the B1 expansion connector provides parallel data and clock to load the FPGA.
JTAG <1:0:1>		 JP1	The FPGA waits for configuration via the four-wire JTAG interface.




Hình 3. 28 : Bảng thiết lập các trạng thái hoạt động cho FPGA thông qua chân J8.



Hình 3. 29 : Vị trí nút ấn để reset chương trình nạp cho kit và LED hiển thị.

Khi nút PROG được ấn thì FPGA sẽ cấu hình lại và đọc lại dữ liệu cấu hình của nó. LED DONE kết nối đến chân DONE của FPGA và sáng lên khi cấu hình FPGA thành công hay nói cách khác chương trình đã được nạp thành công vào kit Spartan 3.

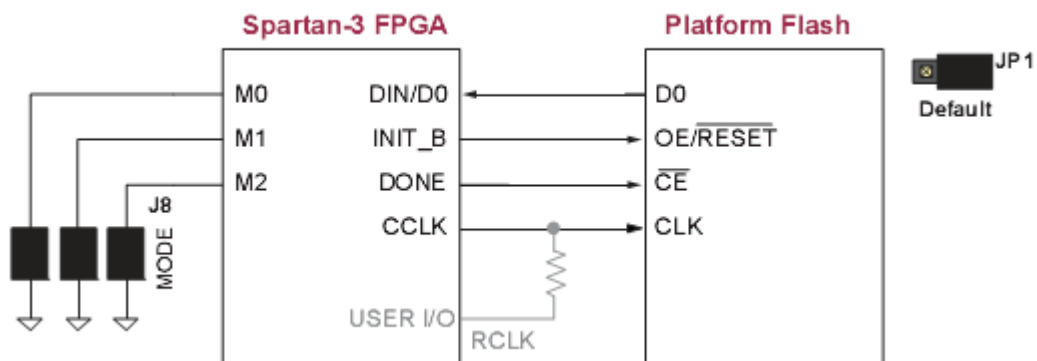
3.10 Thiết lập cách lưu trữ cho Platform :

Option	Jumper JP1 Setting	Description
Default	 JP1	The FPGA boots from Platform Flash. No additional data storage is available.
Flash Read	 JP1	The FPGA boots from Platform Flash, which is permanently enabled. The FPGA can read additional data from Platform Flash.
Disable	 JP1	Jumper removed. Platform Flash is disabled. Other configuration data source provides FPGA boot data.

Hình 3. 30 : Sơ đồ kết nối jumper để lựa chọn các mode lưu trữ của FPGA.

3.10.1 Default Option :

Hầu hết các ứng dụng đều sử dụng thiết lập này. Khi chân DONE của FPGA ở LOW thì Platform Flash được cho phép trong suốt thời gian cấu hình. Khi chân DONE của FPGA đi đến HIGH tại cuối quá trình cấu hình Platform Flash không được cho phép và ở chế độ nguồn ít.

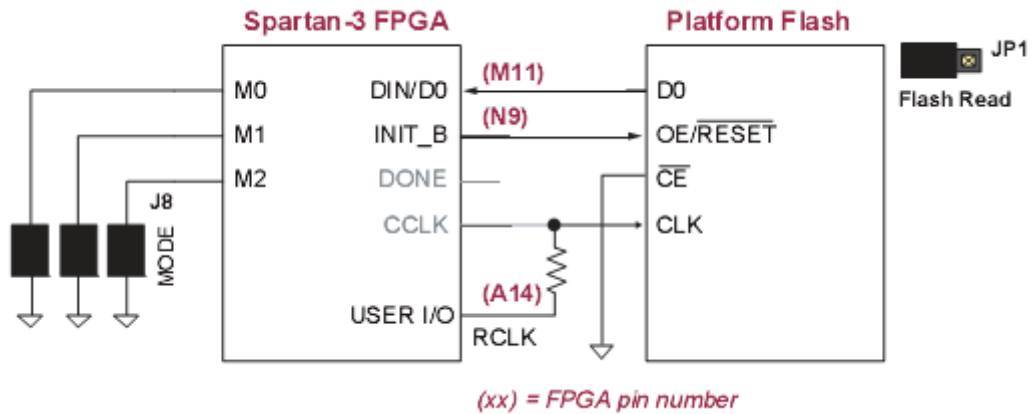


Hình 3. 31 : Sơ đồ kết nối chân giữa FPGA với Platform Flash ở chế độ Default.

3.10.2 Flash Read option :

Như ta đã biết ở phần tổng quan thì Platform có 2Mbit, trong đó chỉ sử dụng một phần nhỏ trong 1Mbit cho việc cấu hình dữ liệu. 1Mbit còn lại dùng trong việc lưu trữ những dữ liệu thay đổi như các số đếm nối tiếp, các hệ số toán học, một số MAC ID, hay đoạn mã cho một con vi xử lý được gắn vào trong một con FPGA.

Để cho phép FPGA đọc dữ liệu từ Platform thì jumper JP1 phải được kết nối như trong hình 1.32. Sau khi cấu hình hoàn tất cho FPGA thì FPGA sẽ gửi một tín hiệu lái cho chân INT_B lên mức cao, tức là chân N9 của FPGA lên mức cao. Tại thời điểm này Platform sẽ không được reset. Để đọc một dữ liệu tiếp theo thì Platform phải chờ đến xung clock kế tiếp của tín hiệu RCLK được đưa đến từ chân A14 của FPGA. Sau đó, ngõ ra của CCLK là 3 trạng thái và tạo ra một điện trở kéo lên làm cho áp V_{CCAUX} có giá trị là 2.5V. Dữ liệu nối tiếp được đọc từ Platform sẽ được chuyển đến FPGA qua chân M11.



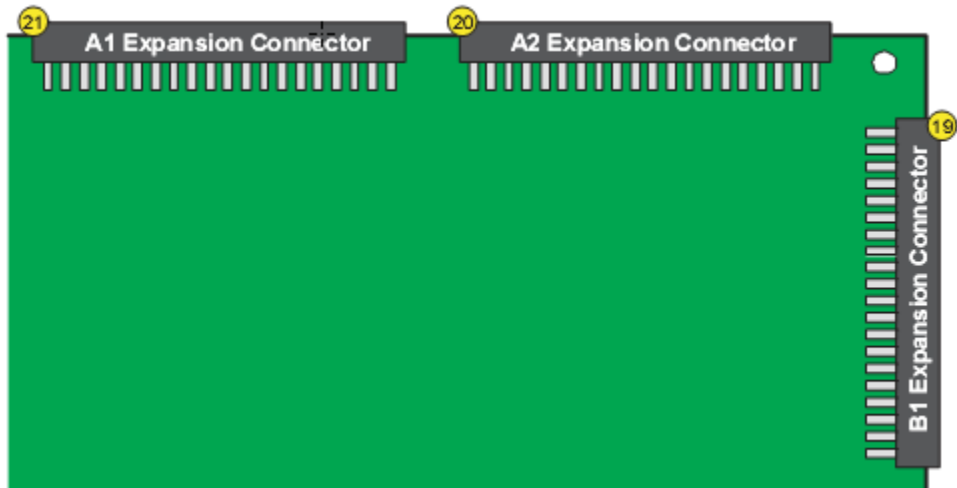
Hình 3. 32 : Sơ đồ kết nối chân giữa FPGA với Platform Flash ở chế độ Flash Read.

3.10.3 Disable Option :

Nếu chân JP1 được tháo ra thì Platform Flash không được cho phép ; điều này cho phép cấu hình mở rộng qua 1 board mở rộng kết nối đến 1 trong các connector mở rộng của Kit.

3.11 Sự kết nối các board mở rộng vào kit Spartan 3 :

Kit spartan 3 có thể kết nối với 3 board mạch mở rộng A1, A2 và B1, được gắn vào từ bên ngoài.



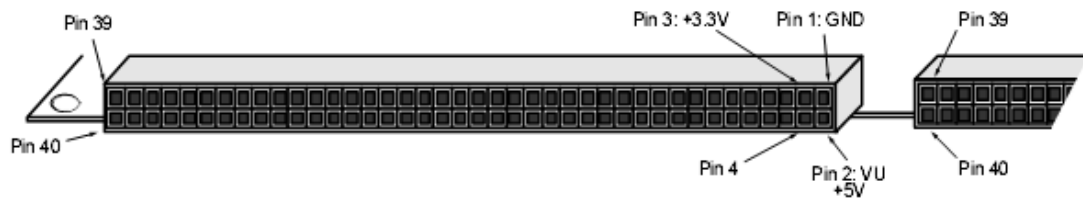
Hình 3. 33 : Vị trí kết nối thêm các board mạch mở rộng trên board Spartan 3.

Hình 2.34 là một bảng tổng hợp các đặc tính của mỗi port mở rộng. Port A1 có tối đa 32 chân xuất nhập dữ liệu, trong khi hai port còn lại thì được cung cấp đến 34 chân xuất nhập. Một vài chân được chia sẻ để sử dụng một số các chức năng khác trên board, những chân mà có thể được dùng cho việc giảm bớt hiệu quả của việc đếm xuất nhập cho những ứng dụng đặc biệt. Ví dụ những chân của port A1 được dùng chia sẻ với các đường địa chỉ của SRAM, với các chân OE và WE của SRAM.

Connector	User I/O	SRAM	JTAG	Serial Configuration	Parallel Configuration
A1	32	Address OE#, WE# Data[7:0] to IC10 only	√		
A2	34			√	
B1	34			√	√

Hình 3. 34 : Một số đặc tính của các port mở rộng A1, A2, B1.

Mỗi một port sẽ đảm nhận một nhiệm vụ khác nhau trong việc lập trình cho con FPGA trên kit Spartan 3. Ví dụ port A1 cung cấp thêm các phép toán logic để lái con FPGA và Platform Flash với dây cáp JTAG. Một cách tương tự như thế, port A2 và B1 cung cấp các sự kết nối theo cấu hình Master và Slave ở mode nối tiếp (cấu hình board chủ - tớ). Cuối cùng port B1 B1 cung cấp các sự kết nối theo cấu hình Master và Slave ở mode song song.



Hình 3. 35 : Cấu trúc chung của một port mở rộng.

Mỗi một port mở rộng có 40 chân, trong đó chân số 1 luôn luôn nối đất. Chân số 2 tạo ra điện áp 5V DC ở ngõ ra được lấy từ hệ thống chuyển đổi của đáp ứng nguồn. Chân số 3 luôn tạo ra một mức điện áp 3.3V DC điều chỉnh.

3.11.1 Port mở rộng A1:

Port mở rộng A1 chia sẻ sự kết nối với thiết bị 256Kx16 SRAM, đặc biệt là những đường địa chỉ của SRAM, những chân điều khiển WE và OE, IC10. Tương tự như vậy là cáp JTAG được tích cực từ chân 36 đến 40 của port. Chân 20 của port A1 được dùng cho cấu hình tín hiệu ở trạng thái DOUT / BUSY và chốt nó trong suốt quá trình FPGA xử lý cấu hình.

Schematic Name	FPGA Pin	Connector		FPGA Pin	Schematic Name
GND		1	2		VU (+5V)
VCCO (+3.3V)	VCCO (all banks)	3	4	(N8)	ADR0
DB0	(N7) SRAM IC10 IO0	5	6	(L5) SRAM A0	ADR1
DB1	(T8) SRAM IC10 IO1	7	8	(N3) SRAM A1	ADR2
DB2	(R6) SRAM IC10 IO2	9	10	(M4) SRAM A2	ADR3
DB3	(T5) SRAM IC10 IO3	11	12	(M3) SRAM A3	ADR4
DB4	(R5) SRAM IC10 IO4	13	14	(L4) SRAM A4	ADR5
DB5	(C2) SRAM IC10 IO5	15	16	(G3) SRAM WE#	WE
DB6	(C1) SRAM IC10 IO6	17	18	(K4) SRAM OE#	OE
DB7	(B1) SRAM IC10 IO7	19	20	(P9) FPGA DOUT/BUSY	CSA
LSBCLK	(M7)	21	22	(M10)	MA1-DB0
MA1-DB1	(F3) SRAM A6	23	24	(G4) SRAM A5	MA1-DB2
MA1-DB3	(E3) SRAM A8	25	26	(F4) SRAM A7	MA1-DB4
MA1-DB5	(G5) SRAM A10	27	28	(E4) SRAM A9	MA1-DB6
MA1-DB7	(H4) SRAM A12	29	30	(H3) SRAM A11	MA1-ASTB
MA1-DSTB	(J3) SRAM A14	31	32	(J4) SRAM A13	MA1-WRITE
MA1-WAIT	(K5) SRAM A16	33	34	(K3) SRAM A15	MA1-RESET
MA1-INT	(L3) SRAM A17	35	36	JTAG Isolation	JTAG Isolation
TMS	(C13) FPGA JTAG TMS	37	38	(C14) FPGA JTAG TCK	TCK
TDO-ROM	Platform Flash JTAG TDO	39	40	Header J7, pin 3	TDO-A

Hình 3. 36 : Bảng đồ chân kết nối giữa port mở rộng A1 với con FPGA spartan 3.

3.11.2 Port mở rộng A2 :

Port A2 chỉ được kết nối đến các chân của con FPGA và không chia sẻ chân với các thiết bị khác có sẵn trên kit. Chân 35 của port A2 nối đến một khe cắm nguồn xung clock phụ, với một dao động thạch anh được gắn thêm vào khe đó. Các chân từ 36 đến 40 được dùng để thiết lập cho FPGA hoạt động theo cấu hình Master – Slave ở mode nối tiếp.

Schematic Name	FPGA Pin	Connector		FPGA Pin	Schematic Name
GND		1	2		VU (+5V)
V _{CCO} (+3.3V)	V _{CCO} (all banks)	3	4	(E6)	PA-IO1
PA-IO2	(D5)	5	6	(C5)	PA-IO3
PA-IO4	(D6)	7	8	(C6)	PA-IO5
PA-IO6	(E7)	9	10	(C7)	PA-IO7
PA-IO8	(D7)	11	12	(C8)	PA-IO9
PA-IO10	(D8)	13	14	(C9)	PA-IO11
PA-IO12	(D10)	15	16	(A3)	PA-IO13
PA-IO14	(B4)	17	18	(A4)	PA-IO15
PA-IO16	(B5)	19	20	(A5)	PA-IO17
PA-IO18	(B6)	21	22	(B7)	MA2-DB0
MA2-DB1	(A7)	23	24	(B8)	MA2-DB2
MA2-DB3	(A8)	25	26	(A9)	MA2-DB4
MA2-DB5	(B10)	27	28	(A10)	MA2-DB6
MA2-DB7	(B11)	29	30	(B12)	MA2-ASTB
MA2-DSTB	(A12)	31	32	(B13)	MA2-WRITE
MA2-WAIT	(A13)	33	34	(B14)	MA2-RESET
MA2-INT/GCK4	(D9) Oscillator socket	35	36	(B3) FPGA PROG_B	PROG-B
DONE	(R14) FPGA DONE	37	38	(N9) FPGA INIT_B	INIT
CCLK	(T15) FPGA CCLK Connects to (A14) via 390Ω resistor	39	40	(M11)	DIN

Hình 3. 37 : Bảng đồ chân kết nối giữa port mở rộng A2 với con FPGA spartan 3.

3.11.3 Port mở rộng B1 :

Port B1 chỉ được kết nối đến các chân của con FPGA và không chia sẻ chân với các thiết bị khác có sẵn trên kit. Các chân từ 36 đến 40 được dùng để thiết lập cho FPGA hoạt động theo cấu hình Master – Slave ở mode nối tiếp. Các chân 5, 7, 9, 11, 13, 15, 17, 19 và 20 được dùng để thiết lập cho FPGA hoạt động theo cấu hình Master – Slave ở mode song song.

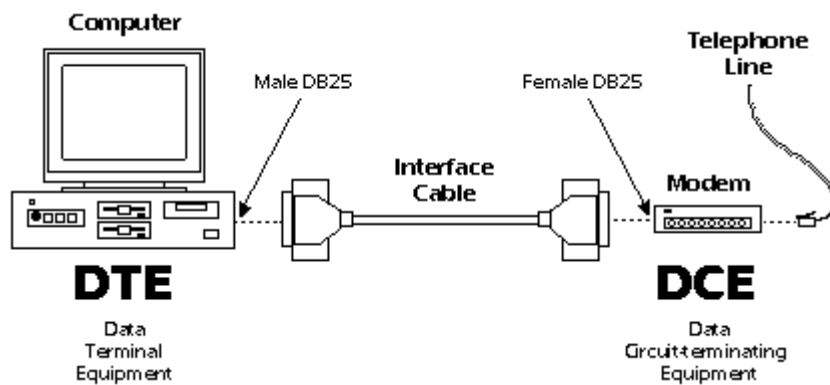
Schematic Name	FPGA Pin	Connector		FPGA Pin	Schematic Name
GND		1	2		VU (+5V)
V _{CCO} (+3.3V)	V _{CCO} (all banks)	3	4	(C10)	PB-ADR0
PB-DB0	(T3) FPGA RD_WR_B config	5	6	(E10)	PB-ADR1
PB-DB1	(N11) FPGA D1 config	7	8	(C11)	PB-ADR2
PB-DB2	(P10) FPGA D2 config	9	10	(D11)	PB-ADR3
PB-DB3	(R10) FPGA D3 config	11	12	(C12)	PB-ADR4
PB-DB4	(T7) FPGA D4 config	13	14	(D12)	PB-ADR5
PB-DB5	(R7) FPGA D5 config	15	16	(E11)	PB-WE
PB-DB6	(N6) FPGA D6 config	17	18	(B16)	PB-OE
PB-DB7	(M6) FPGA D7 config	19	20	(R3) FPGA CS_B config	PB-CS
PB-CLK	(C15)	21	22	(C16)	MB1-DB0
MB1-DB1	(D15)	23	24	(D16)	MB1-DB2
MB1-DB3	(E15)	25	26	(E16)	MB1-DB4
MB1-DB5	(F15)	27	28	(G15)	MB1-DB6
MB1-DB7	(G16)	29	30	(H15)	MB1-ASTB
MB1-DSTB	(H16)	31	32	(J16)	MB1-WRITE
MB1-WAIT	(K16)	33	34	(K15)	MB1-RESET
MB1-INT	(L15)	35	36	(B3) FPGA PROG_B	PROG-B
DONE	(R14) FPGA DONE	37	38	(N9) FPGA INIT_B	INIT
CCLK	(T15) FPGA CCLK Connects to (A14) via 390Ω resistor	39	40	(M11)	DIN

Hình 3. 38 : Bảng đồ chân kết nối giữa port mở rộng B1 với con FPGA spartan 3.

CHƯƠNG 4 : CÁC CỔNG GIAO TIẾP DÙNG TRÊN BOARD SPARTAN 3

4.1 Giao tiếp RS232 (cổng COM) :

Một chuẩn giao tiếp quan trọng được phát triển bởi tổ chức phi lợi nhuận chuyên trợ giúp các nhà sản xuất điện tử, gọi tắt là EIA (The Electronics Industries Association), là EIA-232, nó định nghĩa các đặc tính cơ, điện và chức năng của giao tiếp giữa một DTE và một DCE. Trong đó DTE là thiết bị đầu cuối dữ liệu (Data Terminal Equipment) và DCE là thiết bị đầu cuối mạch dữ liệu (Data Circuit-terminating Equipment). Chuẩn này được đề xuất năm 1962 gọi là RS-232. Một trong các cấu hình áp dụng được trình bày trong hình 3.1.



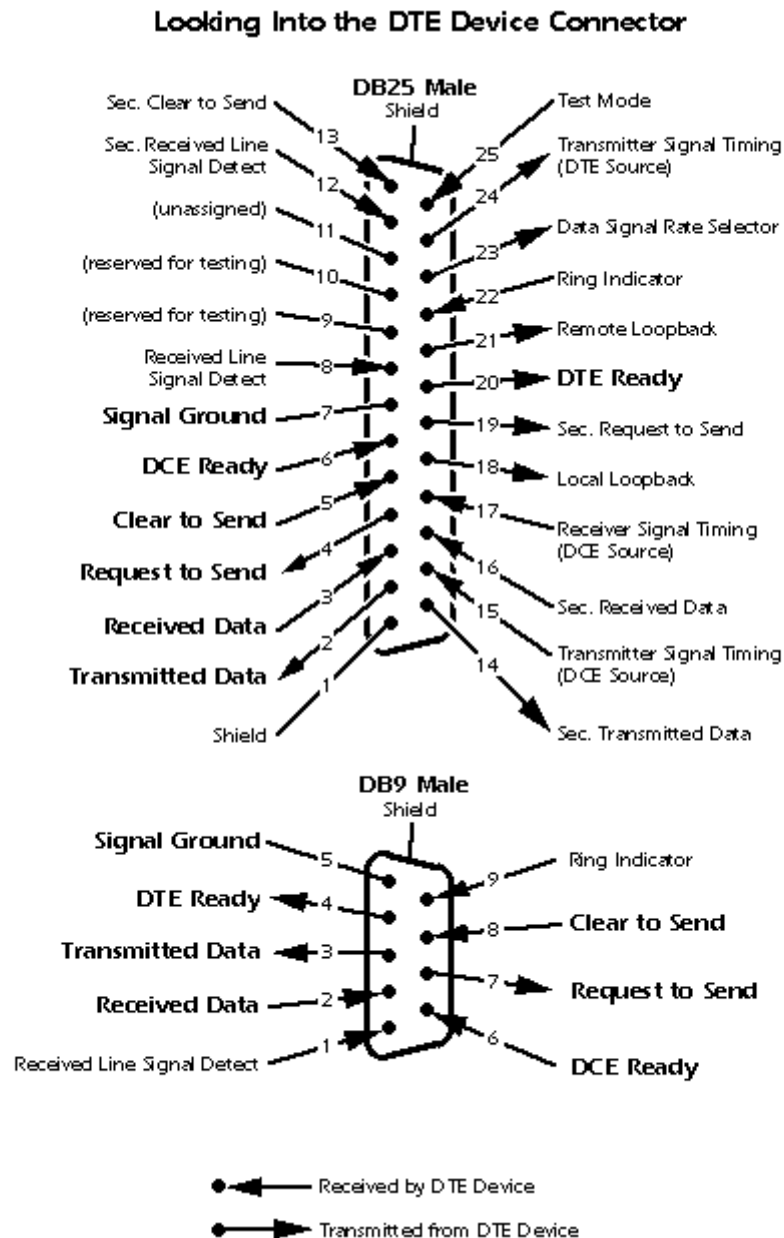
Hình 4. 1 : Một áp dụng của RS-232.

Trong hình 3.1 DTE thường là một máy tính (PC) còn DCE thường là một modem hay một thiết bị thu phát dữ liệu được kết nối với PC thông qua cổng COM (cổng RS-232).

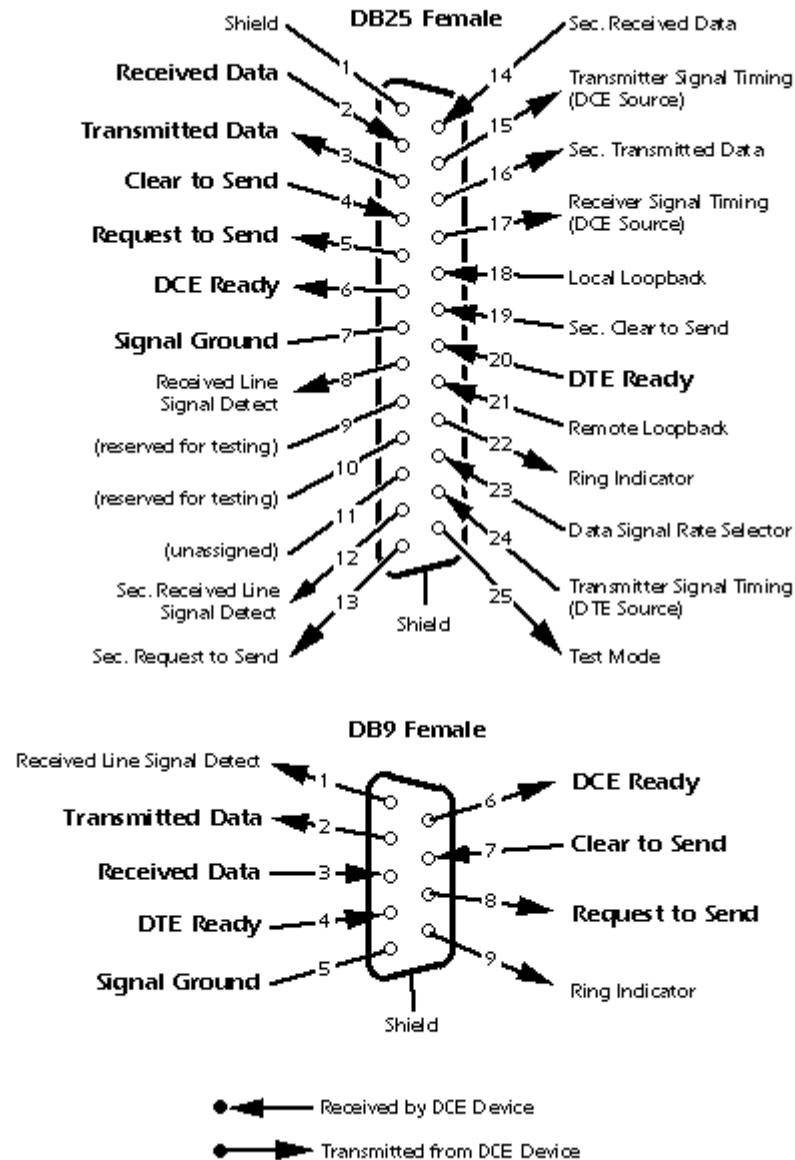
Trong quy định về cơ, chuẩn EIA-232 định nghĩa giao tiếp như là sợi cáp 25 dây với các đầu nối đực và cái DB-25. Chiều dài cáp không nên vượt quá 15m (50 feet). Một cách thực hiện kết nối khác của EIA-232 là dùng cáp 9 dây với các đầu nối đực và cái DB9.

Chỉ có 4 dây trong 25 dây giao tiếp được dùng cho các chức năng dữ liệu. 21 dây còn lại được dùng cho các chức năng khác như điều khiển, điều hòa thời gian, đất và kiểm tra. Trong chuẩn giao tiếp EIA-232, một tín hiệu về điện cũng tương tự như đường dữ liệu, một tín hiệu được gọi là ON nếu nó phát điện áp ít nhất +3V và OFF nếu nó phát điện áp với giá trị nhỏ hơn -3V.

Toàn bộ các chân chức năng được diễn tả cho các loại connectors DB25 và DB9 được mô tả chi tiết trong hình 3.2 và hình 3.3.



Hình 4. 2 : Các chân chức năng của DB25 và DB9 loại đầu đực.



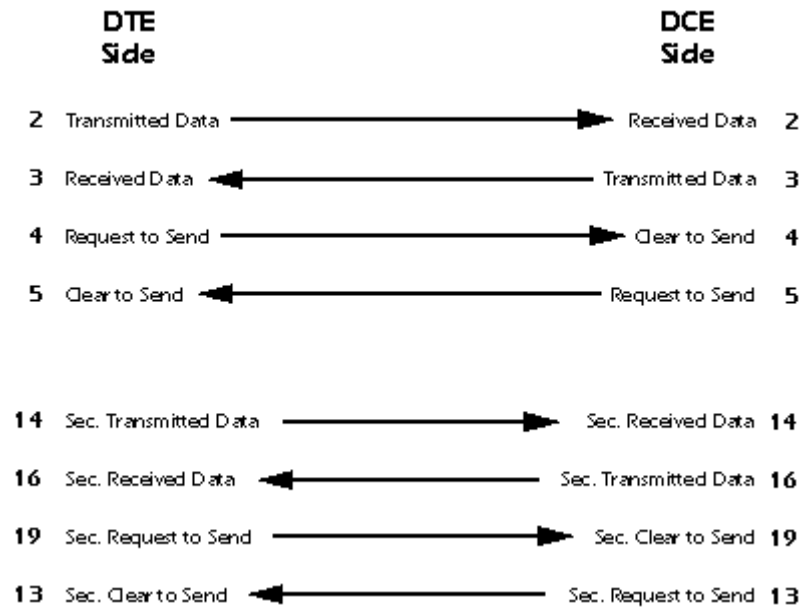
Hình 4. 3 : Các chân chức năng của DB25 và DB9 loại đầu cái.

DB25: thứ tự và chức năng được trình bày trong hình 3.3, mỗi đầu nối cái sẽ là ảnh gương của đầu đực. Như vậy mỗi chân chức năng đều có ảnh hoặc chiều trả lời theo hướng ngược lại để cho phép hoạt động song công. Tuy nhiên không phải chân nào cũng có chức năng ví dụ chân số 9 và 10 còn dùng để dự phòng và chân số 11 chưa được gán chức năng.

DB9: nhiều chân của DB25 không cần thiết cho kết nối đơn bất đồng bộ cho nên có thể giảm xuống còn 9 chân.

Chúng ta phải lưu ý về chức năng chân của đầu nối đực và đầu nối cái trong 2 loại cáp DB25 và DB9. Ta hãy chú ý các chân thứ 2, 3, 4, 5, 13, 14, 16 và 19 của đầu nối đực và đầu nối cái trong loại kết nối DB25 mặc dù có số thứ tự giống nhau nhưng đảm nhận hai quá trình trái ngược nhau trong truyền và nhận dữ liệu. Ví dụ chân số 2 của đầu đực là

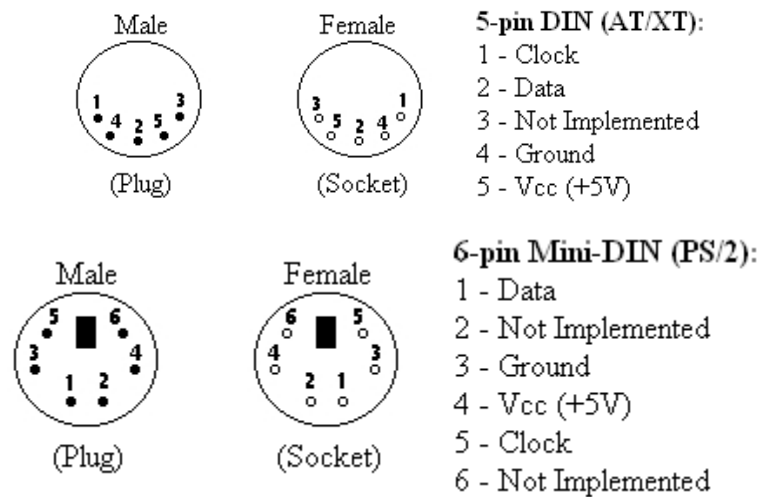
truyền dữ liệu thì chân số 2 của đầu cái đóng vai trò là chân nhận dữ liệu, tương tự như vậy cho các chân vừa nêu.



Hình 4. 4 : Nghi thức truyền và nhận dữ liệu giữa DTE và DCE.

4.2 Giao tiếp bàn phím PS/2 :

4.2.1 Sơ đồ chân kết nối:



Hình 4. 5 : Chân kết nối của chuẩn PS/2 loại 5 chân và 6 chân.

4.2.2 Các tín hiệu của PS/2 :

Bàn phím AT có chân kết nối tới 4 tín hiệu : Clock, Data, +5V, GND . Nguồn +5V được tạo bởi PC và mass GND cũng kết nối với mass của PC. Tín hiệu Clock và Data là kiểu “Open Collector”. Cả bàn phím và máy tính đều có điện trở kéo lên cho Clock và Data lên nguồn 5V.

4.2.3 Nguyên tắc truyền dữ liệu :

Khi nhấn 1 phím, bộ xử lý bàn phím gửi đến PC mã quét (scan-code) của phím được nhấn. Khi phím được nhấn, mã này gọi là **make-code**. Khi phím được nhả, mã này gọi là **break-code**.

Break-code gồm 2 byte: byte đầu là F0 (đối với bàn phím mở rộng), byte kế là mã make-code.

Scan code có 3 tiêu chuẩn: set 1, set 2, set 3. Bàn phím hiện nay thường sử dụng set 2.

Ví dụ:

Nhấn SHIFT: make-code = 12 .

Nhấn A: make-code = 1C

Nhả A: break-code = F0, 1C .

Nhả SHIFT: break-code = F0, 12 .

Bàn phím PS2 giao tiếp bằng giao thức nối tiếp bất đồng bộ 2 chiều .Xung clock được phát bởi bàn phím, tần số khoảng 10-16.7kHz .Các trạng thái hoạt động:

Data=high, clock=high: trạng thái rảnh .

Data=high, clock=low: trạng thái cấm giao tiếp .

Data=low, clock=high: trạng thái máy chủ được yêu cầu truyền dữ liệu .

4.2.3.1 Truyền dữ liệu từ bàn phím về máy chủ .

Các bước thực hiện:

Kiểm tra bus đang ở trạng thái rảnh .

Clock ở mức cao ít nhất 50us trước khi bàn phím gửi data .

Bàn phím gửi data từng khung dữ liệu 11 bit .

Dữ liệu được đọc tại cạnh xuống của clock .

Máy chủ có thể cấm giao tiếp bằng cách kéo clock xuống thấp .

Khi clock được giải phóng, bàn phím lại truyền tiếp dữ liệu chưa hoàn chỉnh .

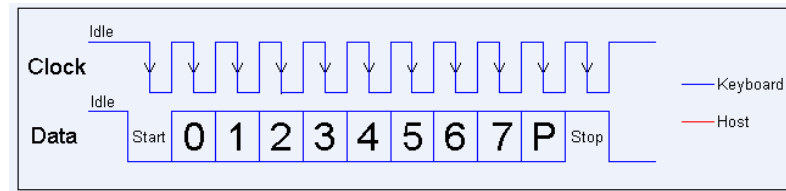
Mã được truyền nối tiếp từng byte, với khung truyền 11bit. Gói dữ liệu gửi từ Keyboard theo thứ tự sau:

1 start bit = 0;

8 data bits (LSB truyền trước);

1 parity bit (if number of ones is even, then parity bit = 1);

1 stop bit = 1.



Hình 4. 6 : Thứ tự truyền data từ Keyboard đến Host.

4.2.3.2 Truyền dữ liệu từ máy chủ đến bàn phím :

Các bước thực hiện:

Máy chủ cấm truyền từ bàn phím đến bằng cách kéo clock xuống thấp.

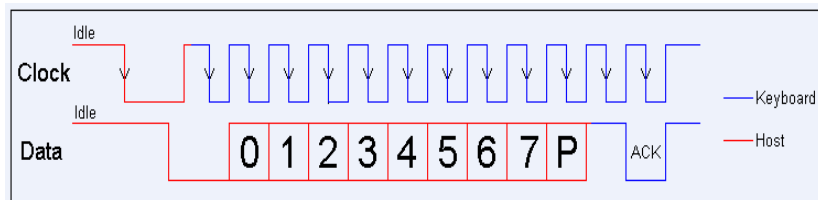
Máy chủ kéo data xuống thấp và giải phóng clock báo hiệu bàn phím phát xung clock bắt đầu truyền dữ liệu.

Dữ liệu được đọc tại cạnh lên của clock .

Sau khi bàn phím nhận stop bit Keyboard sẽ truyền tín hiệu ACK đến máy chủ kết thúc quá trình truyền dữ liệu.

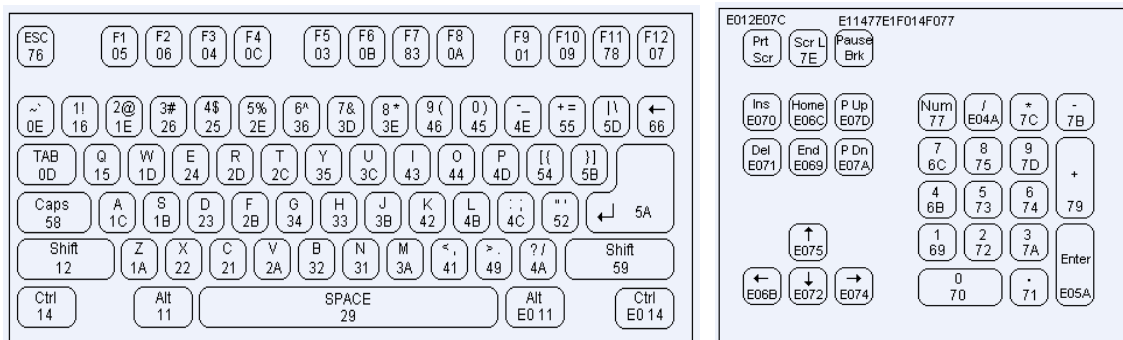
Data được truyền theo khung dữ liệu gồm 11-12bit như sau:

- 1 start bit = 0 .
- 8 data bit (LSB truyền trước) .
- 1 parity bit .
- 1 stop bit = 1 .
- 1 acknowledge bit (host only) .



Hình 4. 7 : Thứ tự truyền data từ Host đến Keyboard.

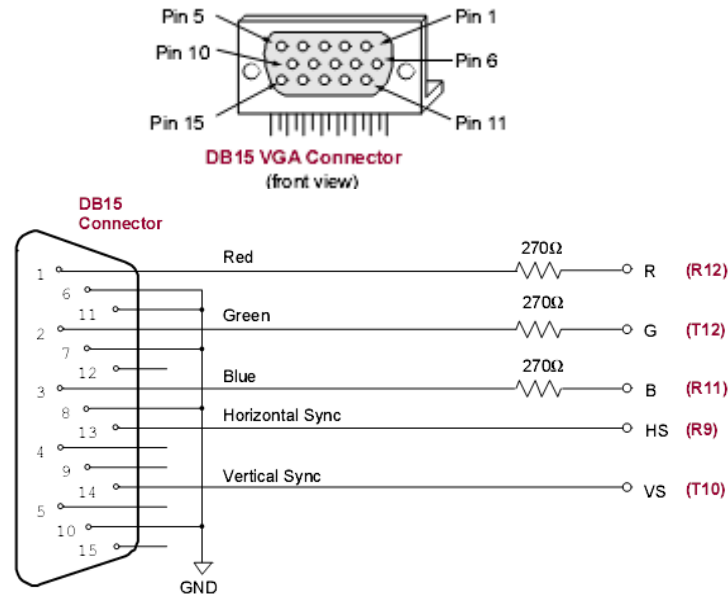
4.2.4 Mã quét bàn phím (Scancode) :



Hình 4. 8 : Mã Scancode của Keyboard.

4.3 Giao tiếp VGA :

4.3.1 Sơ đồ chân kết nối :



Hình 4. 9 : Chân kết nối của chuẩn VGA.

4.3.2 Các tín hiệu của VGA :

Red , Green , Blue : 3 tín hiệu màu cơ bản .

Horizontal Sync : xung đồng bộ quét ngang để tạo thành các dòng hình.

Vertical Sync : xung đồng bộ quét dọc để tạo thành các frame hình.

4.3.3 Nguyên tắc tạo hình :

Đèn điện tử quét các tia điện tử theo hướng từ trái sang phải , từ trên xuống dưới để tạo hình ảnh. Một bức ảnh được đưa lên trên màn hình TV hay máy vi tính bằng cách : quét 1 tín hiệu điện theo phương nằm ngang đi qua màn hình, mỗi dòng quét 1 lần . Ở cuối của mỗi dòng, có một tín hiệu được quét ngược về bên trái của màn hình (tín hiệu xóa ngang) và sau đó bắt đầu quét dòng tiếp theo. Tập hợp các dòng hoàn chỉnh tạo thành 1 tấm ảnh (còn gọi là 1 frame). Mỗi khi có 1 ảnh được quét xong thì có tín hiệu điện khác (tín hiệu xóa dọc) được quét quay ngược lên trên màn hình và bắt đầu ảnh (frame) tiếp theo . Chuỗi này được lặp lại ở một tốc độ đủ nhanh để các ảnh được hiển thị có sự chuyển động liên tục.

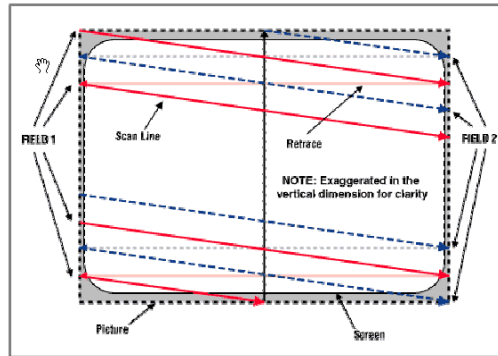
4.3.4 Nguyên tắc quét tín hiệu điện để tạo ảnh :

Có 2 nguyên tắc quét ảnh khác nhau : quét xen kẽ và quét liên tục .

Các tín hiệu TV sử dụng kiểu quét xen kẽ cổ điển, còn máy tính sử dụng kiểu quét liên tục (không xen kẽ). Hai 2 dạng quét này không phù hợp với nhau.

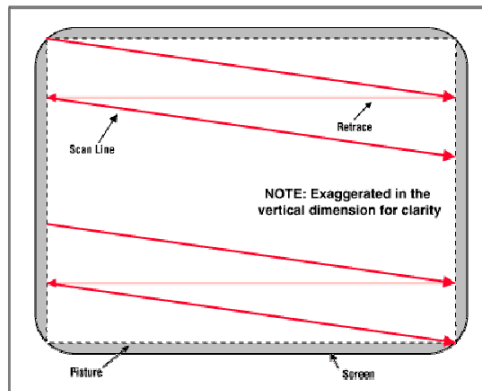
Quét xen kẽ thì mỗi ảnh (frame) được chia thành 2 ảnh con (còn gọi là mảnh). 2 mảnh làm thành 1 ảnh. Ảnh quét xen kẽ được vẽ trên màn hình trong 2 lần quét : đầu tiên quét

hàng ngang của màn hình 1 và sau đó quay ngược lên trên quét tiếp các dòng của màn hình 2 .
Màn hình 1 gồm các dòng từ 1 đến 262 ½ , màn hình 2 gồm các dòng từ 262 ½ đến 525.



Hình 4. 10 : Tín hiệu quét xen kẽ .

Quét liên tục , ảnh được tạo trên màn hình bằng cách quét tất cả các dòng trong 1 lần quét từ trên xuống dưới .



Hình 4. 11 : Tín hiệu quét liên tục .

4.3.5 Một vài chuẩn Video điển hình cho TV và PC :

Video Format	NTSC	PAL	HDTV/SDTV	VGA (PC)	XGA (PC)
Vertical Resolution Format (số dòng trên 1 frame)	Gần 480 dòng (tổng số 525 dòng)	Gần 575 dòng (tổng số 625 dòng)	1080 or 720 or 480 (18 định dạng khác nhau)	480	768
Horizontal Resolution Format (số pixel trên 1 dòng)	Xác định bởi băng thông từ 320 đến 650	Xác định bởi băng thông từ 320 đến 650	1920 or 704 or 640 (18 định dạng khác nhau)	640	1024
Horizontal Rate(KHz)	15.734	15.625	33.75-45	31.5	60
Vertical Rate (Hz)	30	25	30-60	60-80	60-80

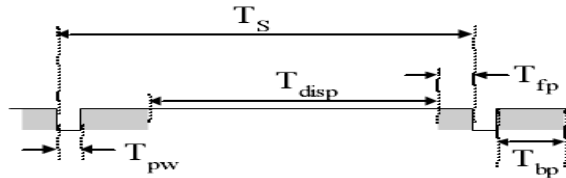
4.3.6 Giải đồ thời gian cho các tín hiệu của chuẩn VGA :

Tần số xung clock chính (tần số để hiển thị 1 pixel) là $f = 25 \text{ Mhz} = 0.04 \mu\text{s}$.

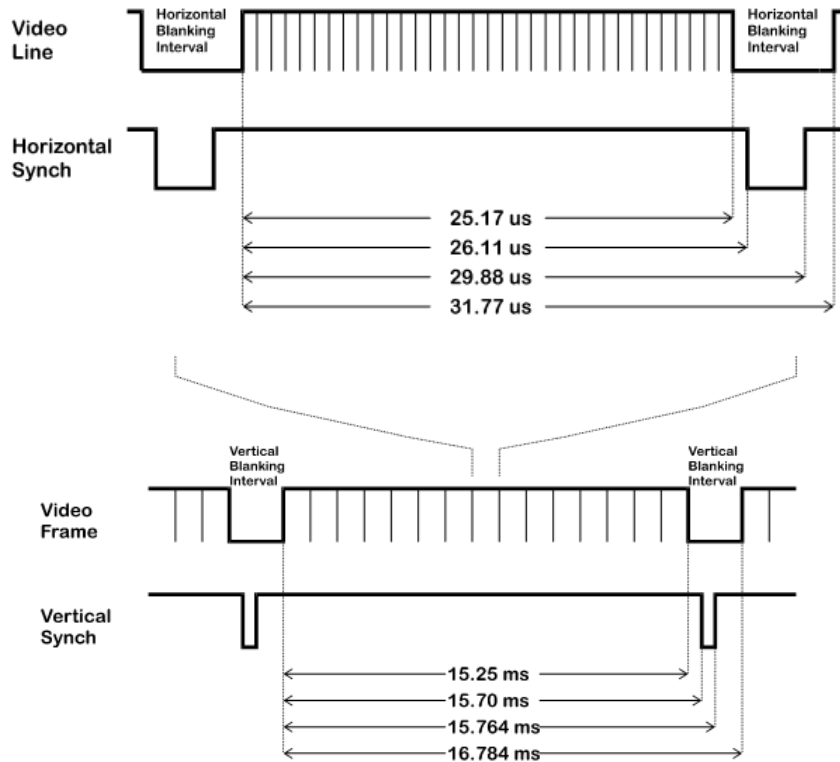
Do màn hình VGA loại CRT có kích thước theo tỉ lệ wide/high = 4/3 . Do đó độ phân giải màn hình VGA nên chọn theo tỉ lệ 4/3 tức là : 512/384 ; 640/480 ; 800/600 ; 1024/768....(để mỗi pixel được vuông hình ảnh không bị kéo dãn ra hay nén lại) .

Tần số làm tươi màn hình (tần số quét dọc) $f_V = 60 \text{ Hz}$, tần số quét ngang $f_H = 31250 \text{ Hz}$.

Symbol	Parameter	Vertical Sync			Horizontal Sync	
		Time	Clocks	Lines	Time	Clocks
T_S	Sync pulse time	16.7ms	416,800	521	32 μs	800
T_{disp}	Display time	15.36ms	384,000	480	25.6 μs	640
T_{pw}	VS pulse width	64 μs	1,600	2	3.84 μs	96
T_{fp}	VS front porch	320 μs	8,000	10	640 ns	16
T_{bp}	VS back porch	928 μs	23,200	29	1.92 μs	48



Hình 4. 12 : Thời gian thực hiện của tín hiệu Vertical Sync và Horizontal Sync.



Hình 4. 13 : Giải đồ thời gian của tín hiệu Vertical Sync và Horizontal Sync

CHƯƠNG 5 : SƠ ĐỒ KHỐI CỦA CÁC CORE VÀ LƯU ĐỒ GIẢI THUẬT THỰC HIỆN CÁC CORE

Sơ đồ khối

CHƯƠNG 6 : CÁC ỨNG DỤNG ĐÃ THỰC HIỆN

Các ứng dụng đã thực hiện sử dụng các tài nguyên trên Kit FpGA Spartan 3 gồm :

- Các công tắc trượt (SW0 đến SW6) .
- Nút ấn Pushbutton (PB0) .
- 4 Led 7 đoạn .
- Giao tiếp PS/2 .

6.1 Đồng hồ và đếm sản phẩm :

SW0 : làm công tắc cho phép đếm .

SW1 : công tắc để reset toàn bộ đồng hồ và đếm sản phẩm .

SW2 : chọn chế độ hiển thị đồng hồ hoặc đếm sản phẩm .

SW3 : chọn mode hiển thị giờ / phút hoặc phút /giây .

SW4 : chọn mode hiển thị đếm sản phẩm tự động hoặc bằng tay (dùng nút ấn PB0) .

SW5 : chọn mode để hiển thị đồng hồ dịch từ phải qua trái để hiển thị giờ / phút /giây .

6.2 Giao tiếp PS/2 :

SW6 : chọn mode để cho phép nhập từ bàn phím chuỗi ký tự để hiển thị Led và thực hiện dịch chuỗi ký tự này từ phải qua trái .