

NỘI DUNG

Chương 1. Kit DE2 – Development and Education Board	20
1.1 Tổng quan về Kit DE2.....	20
1.2 Đặc điểm của kit DE2	20
1.3 Tài liệu hỗ trợ.....	26
1.4 Ứng dụng trong giảng dạy và học tập:	27
1.5 Ứng dụng trong nghiên cứu và thiết kế.....	27
1.6 Một số ứng dụng minh họa.....	27
1.6.1 Ứng dụng trong truyền hình	27
1.6.2 Ứng dụng giao tiếp USB	28
1.6.3 Ứng dụng tạo bài hát karaoke và máy nghe nhạc.....	29
Chương 2. Hướng dẫn sử dụng Kit DE2	31
2.1 Kiểm tra Kit DE2	31
2.2 Hướng dẫn cài đặt USB-Blaster Driver	32
2.3 Thiết lập cấu hình ban đầu và thay đổi cấu hình mới cho Cyclon II FPGA	36
2.3.1 Thiết lập cấu hình ban đầu cho FPGA thông qua việc nạp cấu hình cho bộ nhớ EPPROM EPROM16 bằng AS mode:.....	39
2.4 Sơ đồ mạch và hoạt động của các linh kiện trên Kit DE2	40
2.4.1 Switches (công tắc) và Button (nút nhấn)	40
2.4.2 Leds.....	43
2.4.3 LED hiển thị bảy đoạn.....	45
2.4.4 LED hiển thị LCD.....	48

2.4.5 Ngõ vào xung Clock.....	61
2.4.6 Expansion Header (Jac cắm mở rộng)	62
2.4.7 VGA.....	66
2.4.8 Audio CODEC 24-bit.....	71
2.4.9 Cổng nối tiếp RS-232.....	72
2.4.10 Cổng nối tiếp PS/2	73
2.4.11 Mạch điều khiển mạng Fast Ethernet.....	74
2.4.12 TV Decoder.....	76
2.4.13 TV Encoder	78
2.4.14 USB Host and Device	79
2.4.15 Cổng hồng ngoại	81
2.4.16 Bộ nhớ SDRAM/SRAM/Flash	82
Chương 3. Hướng dẫn cài đặt và sử dụng phần mềm Control Panel để điều khiển kit DE2	89
3.1 Hướng dẫn cài đặt Control Panel điều khiển Kit DE2.....	89
3.2 Tổng quan về cấu trúc và hoạt động của Control Panel	92
3.3 Hướng dẫn sử dụng Control Panel.....	95
3.3.1 Điều khiển LEDs, LEDs bảy đoạn, LCD	95
3.3.2 Truy xuất bộ nhớ SDRAM/SRAM	96
3.3.3 Truy xuất bộ nhớ Flash (Flash memory).....	99
3.3.4 TOOLS – Multi-Port SRAM/SDRAM/Flash Controller... ..	102
3.3.5 VGA Display Control.....	103

Chương 4. Hướng dẫn thiết kế và thực hành môn học Hệ thống số trên Kit DE2.....	109
4.1 Hướng dẫn thực hành	109
4.1.1 Tạo một project trên Quartus II	109
4.1.2 Thiết kế một mạch điện đơn giản (cổng XOR) dùng Schematic trên Quartus II:	115
4.1.3 Gán pin.....	121
4.1.4 Mô phỏng mạch đã thiết kế :	126
4.1.5 Programming mạch đã thiết kế lên FPGA :	131
4.2 Nội dung thực hành môn Hệ thống số	135
4.2.1 Bài thực hành số 1 – Switchs, Lights, Multiplexers	135
4.2.2 Bài thực hành số 2 – Số và cách hiển thị	142
4.2.3 Bài thực hành số 3 – Latch, Flip-flop, Register	150
4.2.4 Bài thực hành số 4 – Bộ đếm (Counters)	157
4.2.5 Bài thực hành số 5 – Adder, Subtractor, Multiplier of two signed numbers in 2's- complement form.....	161
Chương 5. Hướng dẫn thiết kế và thực hành môn Thiết kế mạch dùng Verilog HDL trên Kit DE2.....	168
5.1 Hướng dẫn thực hành	168
5.1.1 Tạo một project trên Quartus II	168
5.1.2 Thiết kế một mạch điện đơn giản (cổng XOR) dùng Verilog trên Quartus II:	174
5.1.3 Gán pin.....	180
5.1.4 Mô phỏng mạch đã thiết kế :	185
5.1.5 Programming mạch đã thiết kế lên FPGA :	190
5.2 Nội dung thực hành môn Thiết kế mạch với Verilog HDL	194

5.2.1	Bài thực hành số 1 – Thiết kế mạch tổ hợp và mạch tuần tự đơn giản	194
5.2.2	Bài thực hành số 2 – Thực hành tìm hiểu thiết kế latches, flip-flops và counters.	198
5.2.3	Bài thực hành số 3 – Thiết kế hệ thống sử dụng xung Clock thời gian thực	208
5.2.4	Bài thực hành số 4 – Thực hành tìm hiểu thiết kế sử dụng State machine.....	210
5.2.5	Bài thực hành số 5 – Thực hành tìm hiểu phương pháp thiết kế onchip Memory trên FPGA và phương pháp sử dụng offchip Memory.....	214
Chương 6.	Hướng dẫn thiết kế và thực hành môn Kiến trúc máy tính nâng cao.....	227
6.1	Kiến thức tổng quát về vi xử lí Nios II	227
6.1.1	Giới thiệu tổng quan Nios II.....	227
6.1.2	Kiến trúc bộ xử lý Nios II.....	233
6.1.3	Mô hình lập trình.....	240
6.2	Hướng dẫn thực hành trên vi xử lí Nios II	273
6.2.1	Nios II System :.....	273
6.2.2	Mở một project mới.....	274
6.3	Nội dung thực hành môn Kiến trúc máy tính nâng cao	296
6.3.1	Bài thực hành số 1 – Thiết kế và sử dụng một hệ thống máy tính đơn giản	296
6.3.2	Bài thực hành số 2 – Điều khiển nhập xuất dữ liệu từ Vi xử lí.....	301
6.3.3	Bài thực hành số 3 – Tìm hiểu cách thức hoạt động và sử dụng Subroutine và Stack của Vi xử lí NiosII	313

6.3.4	Bài thực hành số 4 – Tìm hiểu cách thức hoạt động và sử dụng Polling và Interrupt của Vi xử lí NiosII	327
6.3.5	Bài thực hành số 5 – Tìm hiểu cách thức giao tiếp Bus	350
 Chương 7. Mô phỏng mô tả thiết kế bằng ModelSim.....		359
7.1	Giới thiệu	359
7.2	Mô phỏng pre-synthesis	360
7.3	Mô phỏng post-synthesis.....	372
7.3.1	Dùng Quartus tạo Verilog netlist cho việc mô phỏng post-synthesis.....	372
7.3.2	Dùng ModelSim để chạy mô phỏng post-synthesis	375
7.3.3	Mở lại project và waveform đã chạy mô phỏng.....	389

MỤC LỤC HÌNH

Hình 1.1 Board mạch DE2.....	20
Hình 1.2 Sơ đồ khối board mạch DE2.....	22
Hình 1.3 Ứng dụng trong xử lý ảnh và truyền hình	28
Hình 1.4 Ứng dụng giao tiếp USB	29
Hình 1.5 Ứng dụng trong xử lý âm thanh.....	29
Hình 2.1 Màn hình VGA mặc định	32
Hình 2.2 Chỉ vị trí driver cho hardware.....	33
Hình 2.3 Chỉ đường dẫn cho driver	34
Hình 2.4 Chỉ đường dẫn cho driver	34
Hình 2.5 Chỉ đường dẫn cho driver	35
Hình 2.6 Không cần kiểm tra driver	36
Hình 2.7 Driver đã được cài đặt thành công	36
Hình 2.8 Thiết lập cấu hình cho FPGA thông qua JTAG mode.....	38
Hình 2.9 Thiết lập cấu hình cho FPGA thông qua AS mode	39
Hình 2.10 Chức năng chống nảy cho Push button	40
Hình 2.11 Mạch thiết kế của switches và push button	41
Hình 2.12 Mapped pins giữa switches và FPGA	42
Hình 2.13 Mapped pins giữa Push button và FPGA	42
Hình 2.14 Mạch thiết kế của Leds.....	43
Hình 2.15 Mapped pins giữa LEDs và FPGA	44
Hình 2.16 Led 7 đoạn	45
Hình 2.17 Mạch thiết kế của LEDs 7 đoạn.....	46
Hình 2.18 Mapped pins giữa LEDs 7 đoạn và FPGA	48

Hình 2.19 Mạch thiết kế của LCD	49
Hình 2.20 Cấu tạo LCD	50
Hình 2.21 Mapped pins giữa LCD và FPGA.....	50
Hình 2.22 Thanh ghi điều khiển hoạt động của LCD	51
Hình 2.23 Bộ đếm địa chỉ	52
Hình 2.24 Bộ nhớ lưu giữ dữ liệu hiển thị.....	52
Hình 2.25 Bộ nhớ lưu giữ mẫu kí tự	54
Hình 2.26 Bộ nhớ lưu giữ tất cả các mẫu kí tự	55
Hình 2.27 Bộ tạo mẫu kí tự	56
Hình 2.28 Tập lệnh LCD.....	61
Hình 2.29 Mạch thiết kế của ngõ vào xung Clock.....	61
Hình 2.30 Mapped pins giữa ngõ vào xung Clock và FPGA	62
Hình 2.31 Mạch thiết kế giao tiếp giữa PIO và FPGA.....	63
Hình 2.32 Mapped pins giữa pin PIO và FPGA	66
Hình 2.33 Sơ đồ mạch VGA	67
Hình 2.34 Giản đồ định thời của tín hiệu HSYNC	68
Hình 2.35 Mô tả định thời cho việc đồng bộ hàng.....	69
Hình 2.36 Mô tả định thời cho việc đồng bộ cột	69
Hình 2.37 Mapped pins giữa ADV7123 và FPGA	71
Hình 2.38 Mạch thiết kế của Audio Codec.....	72
Hình 2.39 Mapped pins giữa Audio Codec và FPGA.....	72
Hình 2.40 Mạch thiết kế giao tiếp giữa RS-232 và FPGA.....	73
Hình 2.41 Mapped pins giữa RS-232 và FPGA.....	73
Hình 2.42 Mạch thiết kế giao tiếp giữa cổng PS/2 và FPGA	74
Hình 2.43 Mapped pins giữa cổng PS/2 và FPGA.....	74
Hình 2.44 Mạch thiết kế giao tiếp giữa DM9000A và FPGA.....	75

Hình 2.45 Mapped pins giữa DM9000A và FPGA.....	76
Hình 2.46 Mạch thiết kế giao tiếp giữa ADV7181 và FPGA.....	77
Hình 2.47 Mapped pins giữa ADV7181 và FPGA	78
Hình 2.48 TV encoder ADV7123 và FPGA.....	79
Hình 2.49 Mạch thiết kế giao tiếp USB giữa chip ISP1362 và FPGA	80
Hình 2.50 Mapped pins giữa ISP1362 và FPGA	81
Hình 2.51 Mạch giao tiếp giữa cổng hồng ngoại và FPGA	82
Hình 2.52 Mapped pins giữa cổng hồng ngoại và FPGA.....	82
Hình 2.53 Mạch giao tiếp thiết kế giữa DRAM và FPGA	83
Hình 2.54 Mạch giao tiếp thiết kế giữa SRAM và FPGA.....	83
Hình 2.55 Mạch giao tiếp thiết kế giữa FLASH và FPGA	84
Hình 2.56 Mapped pins giữa SDRAM và FPGA.....	85
Hình 2.57 Mapped pins giữa SRAM và FPGA	87
Hình 2.58 Mapped pins giữa FLASH và FPGA	88
 Hình 3.1 Giao diện cho việc cấu hình thiết kế lên FPGA	90
Hình 3.2 Giao diện Control Panel	91
Hình 3.3 Giao tiếp giữa Control Panel và các thiết bị ngoại vi trên FPGA	92
Hình 3.4 Sơ đồ khối giao tiếp giữa Control Panel và các thiết bị ngoại vi	94
Hình 3.5 Giao diện Control Panel điều khiển LEDs 7 đoạn	95
Hình 3.6 Giao diện Control Panel điều khiển LEDs đơn	96
Hình 3.7 Giao diện Control Panel điều khiển SDRAM	97
Hình 3.8 Giao diện Control Panel điều khiển FLASH.....	100
Hình 3.9 Giao diện Control Panel điều khiển Multi-Ports	102
Hình 3.10 Giao diện Control Panel điều khiển VGA.....	103
Hình 3.11 Giao diện Control Panel điều khiển Multi-ports	105

Hình 3.12 Cấu hình trên FPGA của Multi-ports.....	105
Hình 3.13 Màn hình VGA.....	106
Hình 3.14 Trình biến đổi ảnh	107
Hình 3.15 Giá trị ngưỡng của ảnh	108
Hình 4.1 Màn hình chính của Quartus.....	109
Hình 4.2 Tab File	110
Hình 4.3 Tạo project	110
Hình 4.4 Chỉ đường dẫn và tên project.....	111
Hình 4.5 Đường dẫn chưa tồn tại	111
Hình 4.6 Add các file sử dụng trong project.....	112
Hình 4.7 Chọn thiết bị FPGA.....	112
Hình 4.8 Thiết lập EDA tool	113
Hình 4.9 Hoàn thành việc tạo project	114
Hình 4.10 Màn hình chính sau khi tạo project hoàn thành.....	114
Hình 4.11 Thiết kế một mạch số đơn giản.....	115
Hình 4.12 Chọn công cụ thiết kế	115
Hình 4.13 Cửa sổ thiết kế mạch số.....	116
Hình 4.14 Lưu thiết kế	116
Hình 4.15 Chọn linh kiện	117
Hình 4.16 Các linh kiện đã được chọn	118
Hình 4.17 Đặt tên pin cho thiết kế.....	118
Hình 4.18 Thiết kế hoàn chỉnh	119
Hình 4.19 Cửa sổ trình biên dịch report	120
Hình 4.20 Cửa sổ mapped pin giữa thiết kế và FPGA	122
Hình 4.21 Cửa sổ gán pin.....	122

Hình 4.22 Cửa sổ liệt kê danh sách pin của FPGA	123
Hình 4.23 Cửa sổ sau gán pin	123
Hình 4.24 Dùng Microsoft Excel để tạo file gán pin	124
Hình 4.25 Import file gán pin.....	125
Hình 4.26 File gán pin tạo sẵn bởi Altera.....	126
Hình 4.27 Tạo waveform	127
Hình 4.28 Cửa sổ tạo waveform.....	127
Hình 4.29 Nhập tên signal của thiết kế.....	128
Hình 4.30 Dùng chức năng Node Finder.....	128
Hình 4.31 Tạo input waveform	129
Hình 4.32 Tạo mức logic "1"	129
Hình 4.33 Mức logic "1" đã được tạo.....	130
Hình 4.34 Thiết lập chế độ simulation.....	130
Hình 4.35 Waveform sau khi chạy mô phỏng	131
Hình 4.36 Nạp thiết kế lên FPGA	132
Hình 4.37 Thiết lập cổng giao tiếp giữa kit DE2 và Computer.....	132
Hình 4.38 Chọn cấu hình nạp thiết kế	133
Hình 4.39 Thiết kế chế độ nạp lên FPGA bằng AS mode.....	133
Hình 4.40 Chọn loại ROM tương ứng.....	134
Hình 4.41 Chọn file thiết kế .pof.....	135
Hình 4.42 Thiết kế đơn giản.....	136
Hình 4.43 Mạch số đơn giản	136
Hình 4.44 Mạch gồm 8 MUX 2-1	137
Hình 4.45 Mạch 8 MUX 2-1 với SW và LED	138
Hình 4.46 Mạch chọn kênh	139
Hình 4.47 Mạch chọn kênh 3 input	139

Hình 4.48 Mạch giải mã HEX.....	140
Hình 4.49 Bảng giải mã	140
Hình 4.50 Mạch chọn kênh và hiển thị.....	141
Hình 4.51 Mode hiển thị	142
Hình 4.52 Cửa sổ tạo Symbol	143
Hình 4.53 Mạch giải mã hiện thị HEXA	144
Hình 4.54 Mạch hiện thị từ 0 đến 15	145
Hình 4.55 Mạch FA	146
Hình 4.56 Mạch cộng FA 4 bit.....	147
Hình 4.57 Mạch latch.....	150
Hình 4.58 Giản đồ xung input của mạch latch RS.....	151
Hình 4.59 mạch D latch	152
Hình 4.60 Giản đồ xung input của D latch	152
Hình 4.61 Mạch D-Flipflop.....	153
Hình 4.62 Giản đồ xung input D Flipflop.....	154
Hình 4.63 Latch và Flipflop	155
Hình 4.64 Giản đồ xung input	156
Hình 4.65 Bảng nội dung hiển thị	160
Hình 4.66 Mạch cộng hai số có dấu bù 2	162
Hình 4.67 Mạch cộng hai số có dấu bù 2	164
Hình 4.68 Mạch nhân hai số có dấu bù 2	165
Hình 5.1 Màn hình chính Quartus	168
Hình 5.2 Tab File	169
Hình 5.3 Tạo project mới	169
Hình 5.4 Nhập đường dẫn và tên project.....	170

Hình 5.5 Đường dẫn chưa tồn tại	170
Hình 5.6 Add các file liên quan đến project	171
Hình 5.7 Chọn tên FPGA.....	171
Hình 5.8 Thiết lập thông số EDA.....	172
Hình 5.9 Project mới được tạo	173
Hình 5.10 Cửa sổ Quartus sau khi project mới được tạo	173
Hình 5.11 Mạch số đơn giản	174
Hình 5.12 Chọn môi trường thiết kế Verilog.....	174
Hình 5.13 Cửa sổ thiết kế Verilog.....	175
Hình 5.14 Lưu thiết kế	175
Hình 5.15 Mô tả thiết kế	176
Hình 5.16 Chọn template Verilog	176
Hình 5.17 Add file Verilog liên quan	177
Hình 5.18 Chỉ đường dẫn.....	178
Hình 5.19 Cửa sổ sau quá trình biên dịch.....	179
Hình 5.20 Cửa sổ mapped pin giữa thiết kế và FPGA	181
Hình 5.21 Cửa sổ gán pin.....	181
Hình 5.22 Cửa sổ liệt kê danh sách pin của FPGA	182
Hình 5.23 Cửa sổ sau gán pin	182
Hình 5.24 Dùng Microsoft Excel để tạo file gán pin	183
Hình 5.25 Import file gán pin.....	184
Hình 5.26 File gán pin tạo sẵn bởi Altera.....	185
Hình 5.27 Tạo waveform	186
Hình 5.28 Cửa sổ tạo waveform.....	186
Hình 5.29 Nhập tên signal của thiết kế.....	187
Hình 5.30 Dùng chức năng Node Finder	187

Hình 5.31 Tạo input waveform	188
Hình 5.32 Tạo mức logic "1"	188
Hình 5.33 Mức logic "1" đã được tạo.....	189
Hình 5.34 Thiết lập chế độ simulation.....	189
Hình 5.35 Waveform sau khi chạy mô phỏng	190
Hình 5.36 Nạp thiết kế lên FPGA	191
Hình 5.37 Thiết lập cổng giao tiếp giữa kit DE2 và Computer.....	191
Hình 5.38 Chọn cấu hình nạp thiết kế	192
Hình 5.39 Thiết kế chế độ nạp lên FPGA bằng AS mode.....	192
Hình 5.40 Chọn loại ROM tương ứng	193
Hình 5.41 Chọn file thiết kế .pof.....	194
Hình 5.42 Một mạch latch RS.....	199
Hình 5.43 Mạch thực hiện trên FPGA cho mạch RS latch	201
Hình 5.44 Mạch D latch.....	202
Hình 5.45 Mạch master-slave D flipflop	203
Hình 5.46 Mạch và dạng sóng ngõ vào cho phần 4	204
Hình 5.47 Một bộ đếm đồng bộ 4 bit	206
Hình 5.48 Dùng Tool tạo Mega Wizard	215
Hình 5.49 Tạo một Mega mới	216
Hình 5.50 Chọn các thông số như hình vẽ.....	217
Hình 5.51 Chọn thông số như trên hình	218
Hình 5.52 Gán registers cho inputs	219
Hình 5.53 Không tạo giá trị ban đầu cho SRAM.....	219
Hình 5.54 Simulation library.....	220
Hình 5.55 Chọn các loại dữ liệu cần tạo ra.....	221
Hình 5.56 Add dữ liệu tạo ra vào project hiện hành	221

Hình 5.57 Mô tả verilog của SRAM vừa tạo ra	222
Hình 6.1 Một ví dụ về hệ thống xử lí Nios II	228
Hình 6.2 Sơ đồ khối Nios II processor	234
Hình 6.3 Mạch mô tả phần cứng các thanh ghi phục vụ ngắn	252
Hình 6.4 Cấu trúc của Stack Pointer	266
Hình 6.5 Định dạng từ lệnh loại I.....	267
Hình 6.6 Định dạng từ lệnh loại R	268
Hình 6.7 Định dạng từ lệnh loại J.....	268
Hình 6.8 Nios system.....	273
Hình 6.9 Nios system đơn giản	274
Hình 6.10 Tạo project	275
Hình 6.11 Đặt tên cho Nios system	275
Hình 6.12 Cửa sổ SOPC	276
Hình 6.13 Chọn processor.....	276
Hình 6.14 Quay về SOPC Builder.....	277
Hình 6.15 Chọn On-Chip memory	278
Hình 6.16 Quay về SOPC Builder.....	279
Hình 6.17 Chọn PIO	279
Hình 6.18 Quay về SOPC Builder.....	280
Hình 6.19 Chọn JTAG UART.....	281
Hình 6.20 Quay về SOPC Builder.....	281
Hình 6.21 Thay đổi đặc tính cho Processor	282
Hình 6.22 Thay đổi Exception Vector.....	283
Hình 6.23 Tạo verilog files cho nios system	283
Hình 6.24 Top-level module	284

Hình 6.25 Programmer.....	285
Hình 6.26 Cửa sổ programmer	286
Hình 6.27 Giữ cửa sổ này trong suốt quá trình nạp FPGA	286
Hình 6.28 Chương trình assemble đơn giản	287
Hình 6.29 Cửa sổ Altera Monitor Program	288
Hình 6.30 Thiết lập cấu hình.....	288
Hình 6.31 Chọn file chương trình	289
Hình 6.32 Cửa sổ Debug.....	290
Hình 6.33 Chương trình dùng C.....	291
Hình 6.34 Tạo Breakpoints	292
Hình 6.35 Thiết lập điều kiện tạo Breakpoints	293
Hình 6.36 Thanh ghi Registers.....	294
Hình 6.37 Vùng nhớ của On-Chip memory.....	295
Hình 6.38 Thay đổi nội dung ô nhớ.....	295
Hình 6.39 SOPC Builder.....	296
Hình 6.40 Based address của các components.....	303
Hình 6.41 Thiết lập thông số cho PIO	305
Hình 6.42 Thiết lập thông số đồng bộ cho Input Port.....	306
Hình 6.43 Đổi thông số trên cửa sổ SOPC Builder.....	306
Hình 6.44 Mapping giữa Based address trên SOPC với address trong program ..	311
Hình 6.45 Based address	317
Hình 6.46 Load nội dung cho bộ nhớ	317
Hình 6.47 Nội dung đã được load vào bộ nhớ	318
Hình 6.48 Nios System	324
Hình 6.49 Sơ đồ khối của JTAG UART.....	328
Hình 6.50 Chọn và thiết lập thông số cho JTAG UART	330

Hình 6.51 Chọn và thiết lập thông số cho timer	331
Hình 6.52 Chọn interrupt request.....	331
Hình 6.53 JTAG UART Core Register Map	333
Hình 6.54 Data Register Bits	333
Hình 6.55 Control Register Bits.....	333
Hình 6.56 Số chu kì cho mỗi lệnh	335
Hình 6.57 Số chu kì cho mỗi lệnh	336
Hình 6.58 Số chu kì cho mỗi lệnh	337
Hình 6.59 Exception Vector.....	340
Hình 6.60 Thanh ghi của Timer	345
Hình 6.61 Mô hình Nios System dùng Avalon to External Bus Bridge	351
Hình 6.62 Giản đồ xung hoạt động của SRAM	352
Hình 6.63 Giao tiếp LED 7 đoạn.....	353
Hình 6.64 Connection giữa các components	354
Hình 6.65 Giao tiếp với RAM.....	355
Hình 6.66 Giao tiếp với RAM và LED 7 đoạn	357
 Hình 7.1 Cửa sổ ModelSim.....	360
Hình 7.2 Cửa sổ tạo project mới	361
Hình 7.3 Cửa sổ điền thông tin cho project	361
Hình 7.4 Tạo new file	362
Hình 7.5 Điền thông tin cho new file	362
Hình 7.6 Cửa sổ sau khi tạo project	363
Hình 7.7 Cửa sổ dùng cho mô tả thiết kế.....	363
Hình 7.8 Mô tả thiết kế D-Flipflop.....	364
Hình 7.9 Tạo new file	364

Hình 7.10 Mô tả Testbench cho thiết kế.....	365
Hình 7.11 Lưu mô tả testbench	365
Hình 7.12 Add file Testbench vào project.....	366
Hình 7.13 Chỉ đường dẫn đến file Testbench	366
Hình 7.14 Cửa sổ Workspace sau thiết kế	366
Hình 7.15 Compile thiết kế	367
Hình 7.16 Compile thành công	367
Hình 7.17 Thiết lập mô phỏng	368
Hình 7.18 Chọn thiết kế cần mô phỏng	368
Hình 7.19 Chọn tín hiệu dạng sóng cần quan sát.....	369
Hình 7.20 Cửa sổ dạng sóng	370
Hình 7.21 Thiết lập thời gian chạy mô phỏng	370
Hình 7.22 Chạy mô phỏng	370
Hình 7.23 Nhấn "No".....	371
Hình 7.24 Dạng sóng sau mô phỏng	371
Hình 7.25 Mô tả thiết kế D-Flipflop.....	372
Hình 7.26 Thiết lập thông số cho quá trình synthesis	373
Hình 7.27 Thiết lập thông số.....	374
Hình 7.28 Thư mục sau quá trình synthesis.....	375
Hình 7.29 Hai file quan trọng được tạo ra	375
Hình 7.30 Cửa sổ ModelSim.....	376
Hình 7.31 Tạo project mới	376
Hình 7.32 Đèn thông tin cho project mới	377
Hình 7.33 Add file thiết kế có sẵn	378
Hình 7.34 Chỉ đường dẫn đến file Verilog netlist.....	378
Hình 7.35 Tạo file mô tả thiết kế mới	378

Hình 7.36 Đèn thông tin cho new file	379
Hình 7.37 Cửa sổ mô tả thiết kế được mở ra	379
Hình 7.38 Mô tả Testbench cho thiết kế.....	380
Hình 7.39 Compile thiết kế	380
Hình 7.40 Mô tả thiết kế thành công	381
Hình 7.41 Thiết lập mô phỏng	381
Hình 7.42 Cửa sổ thiết lập mô phỏng.....	382
Hình 7.43 Chỉ Libraries cho thiết kế	383
Hình 7.44 File SDF được gọi trong Verilog netlist.....	384
Hình 7.45 Chọn thiết kế cần chạy mô phỏng.....	385
Hình 7.46 Cửa sổ waveform mở ra	385
Hình 7.47 Chọn tín hiệu cần xem waveform	386
Hình 7.48 Cửa sổ dạng sóng mô phỏng.....	387
Hình 7.49 Thiết lập thời gian chạy mô phỏng	387
Hình 7.50 Chạy mô phỏng	387
Hình 7.51 Nhấn "No"	388
Hình 7.52 Dạng sóng sau mô phỏng	388
Hình 7.53 Mở lại project.....	390
Hình 7.54 Chỉ đường dẫn project cần mở.....	390
Hình 7.55 Cửa sổ Transcript	391
Hình 7.56 Nhập dòng lệnh như trên	391
Hình 7.57 Nhập dòng lệnh như trên	391
Hình 7.58 Nhập dòng lệnh như trên	392

MỤC LỤC BẢNG

Bảng 6.1 Thao tác hỗ trợ bởi Nios II ALU	235
Bảng 6.2 Các thanh ghi đa dụng của NIOS II.....	242
Bảng 6.3 Các bit và tên của thanh ghi điều khiển.....	244
Bảng 6.4 Cho biết chi tiết các trường được định nghĩa trong thanh ghi status	245
Bảng 6.5 Các trường của thanh ghi điều khiển estatus	247
Bảng 6.6 Các trường của thanh ghi điều khiển bstatus	247
Bảng 6.7 Nhóm lệnh chuyển dữ liệu	256
Bảng 6.8 Lệnh logic và số học	257
Bảng 6.9 Các lệnh di chuyển.....	259
Bảng 6.10 Các lệnh so sánh	260
Bảng 6.11 Lệnh dịch và xoay.....	261
Bảng 6.12 Nhóm lệnh nhảy và gọi hàm không điều kiện	262
Bảng 6.13 Nhóm lệnh nhảy có điều kiện.....	263
Bảng 6.14 Nhóm lệnh điều khiển khác.....	264
Bảng 6.15 Các kiểu dữ liệu Nios II	264
Bảng 6.16 Mã hóa trường OP của các từ lệnh	269
Bảng 6.17 Mã hóa trường OPX của các từ lệnh loại R	270
Bảng 6.18 Danh sách các lệnh và lệnh tương đương	271
Bảng 6.19 Danh sách các macro hiện hành	272

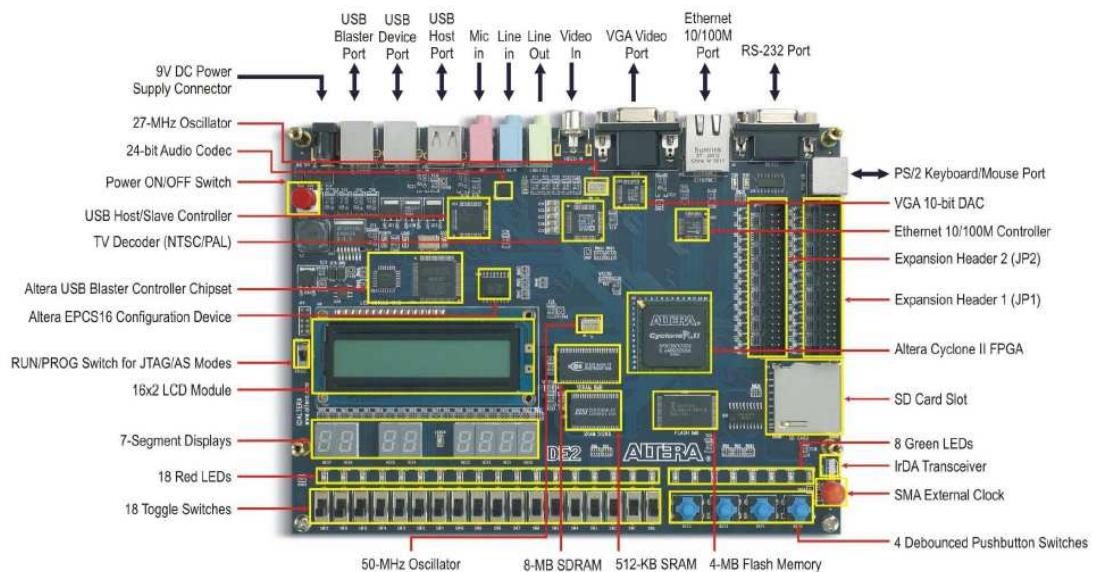
Chương 1. Kit DE2 – Development and Education Board

1.1 Tổng quan về Kit DE2

Mục đích của Kit DE2 là cung cấp cho sinh viên một phương tiện tối ưu để nghiên cứu về kỹ thuật số, cấu trúc máy tính và FPGA. Kit này sử dụng những công nghệ mới nhất cả về phần cứng lẫn công cụ CAD (Computer Aid Design) để giúp không chỉ sinh viên mà còn cả giáo viên có thể nghiên cứu được nhiều ứng dụng khác nhau. Kit cung cấp nhiều đặc điểm phù hợp cho công việc nghiên cứu cũng như phát triển những hệ thống số thông thường lẫn phức tạp trong phòng thí nghiệm của các trường đại học.

1.2 Đặc điểm của kit DE2

Dưới đây là hình ảnh của Kit DE2. Nó thể hiện bì mặt trên của Kit cũng như vị trí của những linh kiện trên Kit.



Hình 1.1 Board mạch DE2

Kit DE2 mang những đặc điểm cho phép người sử dụng có thể thiết kế từ những mạch điện đơn giản cho đến những thiết kế phức tạp như multimedia.

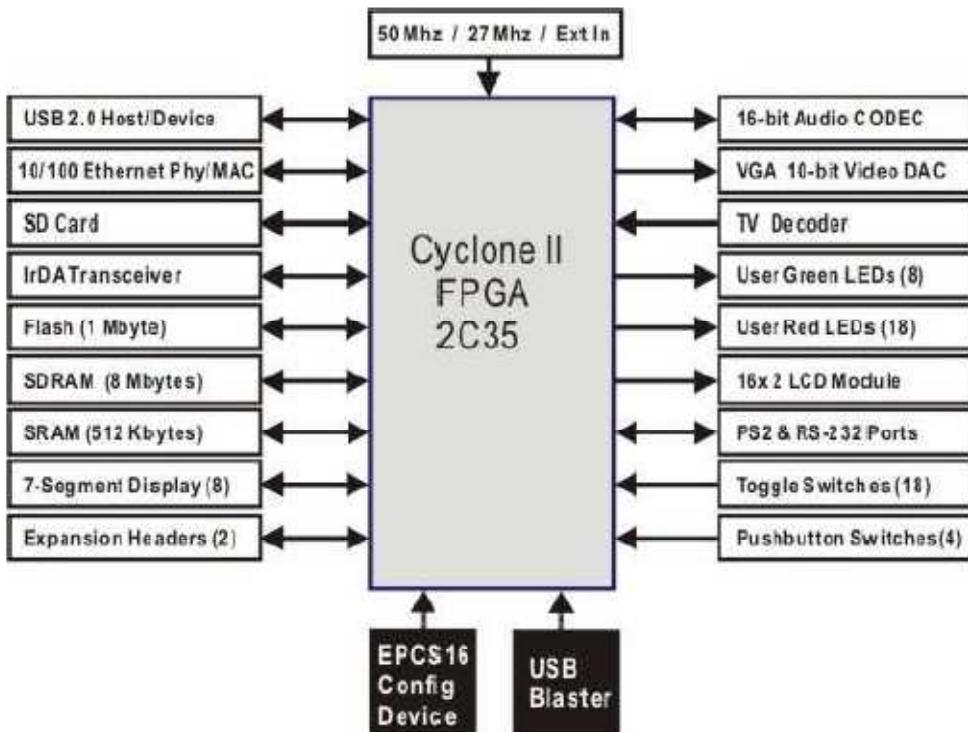
Kit DE2 gồm những linh kiện chính sau:

- Chip Cyclone II 2C35 FPGA 672 pins. Tất cả những linh kiện trên kit đều được kết nối sẵn với những pin của FPGA, điều này cho phép người sử dụng có thể điều khiển tất cả những linh kiện cũng như ứng dụng của chúng.
- Rom EPROM – Dùng để thiết lập cấu hình ban đầu cho thiết bị, hoạt động nối tiếp.
- USB Blaster – Dùng để cài đặt chương trình từ computer cho FPGA, hỗ trợ hai mode : JTAG và AS (Active Serial).
- 512 – Kbyte SRAM
- 8 – Mbyte SDRAM
- 4 – Mbyte Flash memory
- Khe cắm thẻ nhớ SD card.
- 18 toggle switches
- 4 push-button switches
- 18 red LEDs
- 9 green LEDs
- LED 7 đoạn (7-segments displays)
- LED hiện thị ký tự dùng LCD (16x2 character displays)
- Nguồn xung clock 50 MHz và 27 MHz.
- 24-bit CD-quality audio CODEC với những đầu cắm line-in, line-out, và microphone-in.
- VGA DAC (10-bit high-speed triple DACs) với đầu cắm VGA-out.
- TV Decoder (NTSC/PAL) với đầu cắm TV-in.
- Giao tiếp chuẩn RS-232 với đầu cắm 9 pin.
- Giao tiếp chuẩn PS/2 cho chuột và bàn phím.
- Giao tiếp USB 2.0 (cả host lẫn device)

- Giao tiếp Ethernet 10/100
- Giao tiếp hồng ngoại (IrDA)
- Hai cổng kết nối (header) dùng để giao tiếp với những thiết bị ngoại vi khác mà người sử dụng muốn kết nối vào Kit.

Đi kèm với những đặc tính phần cứng, Altera cũng cung cấp những giao tiếp I/O chuẩn và bảng điều khiển việc truy xuất những linh kiện trên Kit dựa trên phần mềm DE2 Control Panel.

Hình 1.2 mô tả sơ đồ khói của Kit DE2:



Hình 1.2 Sơ đồ khói board mạch DE2

Cyclone II 2C35 FPGA

- 33216 Les
- 105 M4K RAM blocks
- 483840 RAM bits

- 35 embedded multipliers
- 4 PLLs
- 475 I/O pins
- FineLine BGA 672-pin package.

 Serial Configuration device và USB Blaster circuit

- Rom EPICS16 Serial Configuration device
- USB Blaster for programing và user API control
- JTAG và AS programming modes

 SRAM:

- 512- Kbyte SRAM memory chip
- Được tổ chức như là 256K x 16 bits
- Có thể truy cập như là bộ nhớ cho vi xử lí Nios II hoặc truy cập thông qua bảng điều khiển Control Panel.

 SDRAM:

- 8-Mbyte Single Data Rate Synchronous Dynamic RAM.
- Được tổ chức như là 1M x 16 bits x 4 banks
- Có thể truy cập như là bộ nhớ cho vi xử lí Nios II hoặc truy cập thông qua bảng điều khiển Control Panel.

 Flash memory:

- 4-Mbyte NOR Flash memory
- 8-bit data bus
- Có thể truy cập như là bộ nhớ cho vi xử lí Nios II hoặc truy cập thông qua bảng điều khiển Control Panel.

 Khe cắm thẻ nhớ SD card:

- Truy xuất SD card bằng mode SPI

- Có thẻ truy cập như là bộ nhớ cho vi xử lí Nios II với DE2 SD card driver.

■ Pushbutton switches:

- 4 pushbutton switches
- Hồi phục lại tín hiệu bằng mạch Schmitt trigger.
- Ở trạng thái bình thường, tín hiệu ở mức cao; khi switch được nhấn, tín hiệu tạo ra một xung tích cực mức thấp và hồi phục lại trạng thái bình thường mức cao.

■ Toggle switches:

- 18 toggle switches
- Khi switch ở vị trí DOWN (gần cạnh của Kit DE2) thì tín hiệu ở mức thấp; ngược lại thì tín hiệu ở mức cao.

■ Clock inputs:

- Nguồn xung clock 50 MHz
- Nguồn xung clock 27 MHz
- Có thể sử dụng nguồn xung clock ngoài thông qua chân SMA.

■ Audio CODEC:

- Wolfson WM8731 24-bit sigma-delta audio CODEC
- Đầu cắm Line-in, Line-out, Microphone-in
- Tần số lấy mẫu : 8-96 KHz
- Ứng dụng cho MP3 players, recorders, PDAs, smart phones, voice recorders...

■ VGA output:

- Sử dụng ADV7123 240-MHz triple 10-bit high-speed video DAC
- Với đầu cắm 15-pin high-density D-sub

- Hỗ trợ độ phân giải 1600x1200 tại 100-Hz refresh rate.
- Có thể kết hợp với Cyclone II FPGA để thực thi một TV Encoder tốc độ cao.

 NTSC/PAL TV decoder circuit :

- Sử dụng ADV7181B Multi-format SDTV Video Decoder
- Hỗ trợ NTSC-(M,J,4.43), PAL-(B/D/G/H/I/M/N), SECAM
- Tích hợp 3 ADC 9-bit 54-MHz
- Hoạt động với nguồn xung clock 27-MHz
- Hỗ trợ Composite Video (CVBS) RCA jack input
- Hỗ trợ ngõ ra digital (8-bit/16-bit): ITU-R BT.656 YcrCb 4:2:2 output + HS, VS, FIELD
- Ứng dụng: DVD recorders, LCD TV, Digital TV, Portable Video devices, Set-top boxes.

 Bộ điều khiển 10/100 Ethernet

- Tích hợp MAC and PHY với giao tiếp vi xử lý thông thường.
- Hỗ trợ đường truyền 100Base-T và 10Base-T
- Hỗ trợ hoạt động kép tại 10Mb/s và 100Mb/s với auto-MDIX
- Hoàn toàn tương thích với cấu hình IEEE 802.3u
- Hỗ trợ IP/TCP/UDP checksum generation và checking
- Hỗ trợ back-pressure mode for half-duplex mode flow control.

 USB Host/Slave controller

- USB 2.0
- Hỗ trợ truyền data với high-speed và low-speed.
- Hỗ trợ USB chủ/tớ.
- Hai cổng USB (Cổng A cho chủ và Cổng B cho tớ).

- Cung cấp giao tiếp song song đến bộ vi xử lí; hỗ trợ Nios II bởi Terasic driver.
- Hỗ trợ Programmed I/O (PIO) và Direct Memory Access (DMA).

 **Cổng nối tiếp :**

- Một cổng giao tiếp RS-232
- Một cổng giao tiếp PS/2

 **Cổng giao tiếp hồng ngoại (IrDA)**

- Bộ truyền nhận tín hiệu 115.2 –kb/s
- Dòng điều khiển LED 32 mA
- Được bảo vệ bởi một lớp EMI
- IEC825-1 Class 1 eye safe
- Tín hiệu ngõ vào được xác nhận bởi tích cực cạnh.

 **Hai đầu nối mở rộng (40 pin)**

- 2x40 pin của 2 đầu nối được kết nối với 72 pin của Cyclone II I/O pins và 8 pin power và mass.
- Đầu nối 40 pin này có thể tương thích với cable chuẩn 40 pin được dùng cho ổ cứng IDE.
- Được bảo vệ bởi diode và điện trở.

1.3 Tài liệu hỗ trợ

Phần mềm đi kèm với Kit DE2 bao gồm Quartus II Web Edition CAD và Nios II Embedded Processor. Ngoài ra một số hướng dẫn và ứng dụng đơn giản để giúp sinh viên và giáo viên hiểu rõ về ứng dụng của Kit trong giảng dạy và nghiên cứu.

Thông thường những Kit FPGA được sản xuất cho mục đích giáo dục sẽ cung cấp nhiều đặc tính về phần cứng cũng như công cụ CAD để có thể tạo ra

nhiều ứng dụng trên Kit, nhưng có rất ít tài liệu hướng dẫn được kèm theo cho mục đích giảng dạy. Ưu điểm của Kit DE2 đó là, ngoài phần cứng và phần mềm, Altera còn cung cấp những bài thực hành với những khóa học cụ thể như Hệ thống số, Cấu trúc máy tính. Điều này giúp sinh viên trong các trường Đại học nhanh tiếp cận hơn với những công nghệ mới được trình bày trong những lớp lý thuyết.

1.4 Ứng dụng trong giảng dạy và học tập:

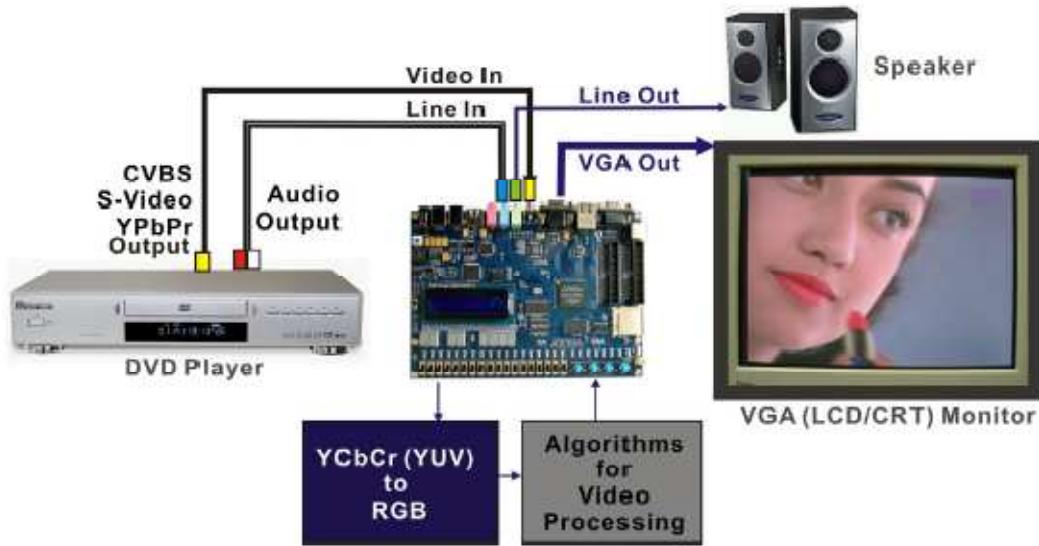
Những khóa học về thiết kế mạch logic và cấu trúc máy tính thường đề cập đến những thiết bị và linh kiện điện tử. Ngày nay khi mà công nghệ đang phát triển với tốc độ chóng mặt thì những giáo trình cũng như những thiết bị trong các phòng thí nghiệm cũng phải luôn được cập nhật những công nghệ và công cụ thiết kế hiện đại nhất, tuy nhiên nó vẫn phải đảm bảo giúp sinh viên nắm vững những kiến thức nền tảng cho đến những kiến thức cao hơn. Kit DE2 được thiết kế để đáp ứng được tất cả những yêu cầu trên.

1.5 Ứng dụng trong nghiên cứu và thiết kế

Với Chip Cyclone II FPGA tiên tiến, nhiều loại giao tiếp I/O và nhiều loại memory khác nhau, Kit DE2 sẽ giúp người sử dụng rất linh động trong việc thiết kế nhiều loại ứng dụng khác nhau. Cùng với những ứng dụng minh họa kèm theo Kit, người thiết kế có thể tạo ra những thí nghiệm thú vị về những ứng dụng như là audio, video, USB, network và memory. Kit DE2 cũng có thể thực thi được những ứng dụng nhúng sử dụng vi xử lí Nios II.

1.6 Một số ứng dụng minh họa

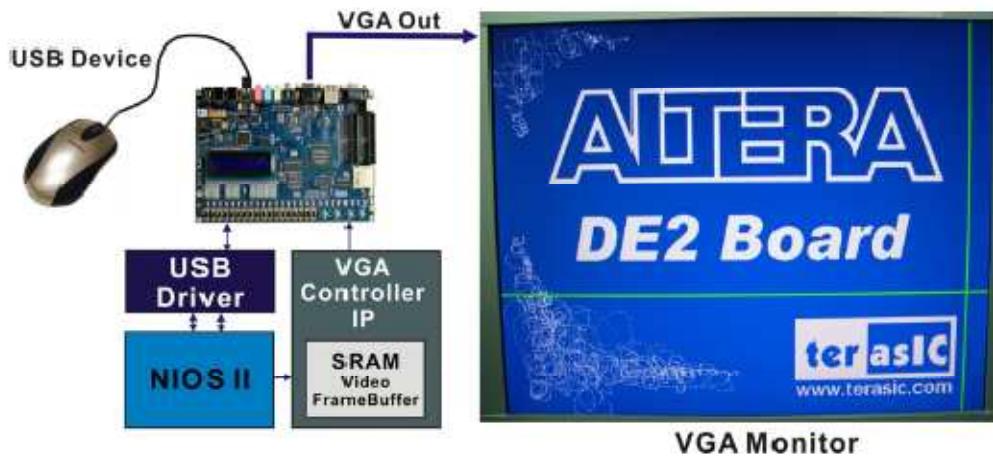
1.6.1 Ứng dụng trong truyền hình



Hình 1.3 Ứng dụng trong xử lý ảnh và truyền hình

- ✚ Bộ giải mã TV chất lượng cao
- ✚ Audio CD 24 bit chất lượng cao
- ✚ VGA monitor
- ✚ Tạo một nền thiết kế cho những ứng dụng trên video

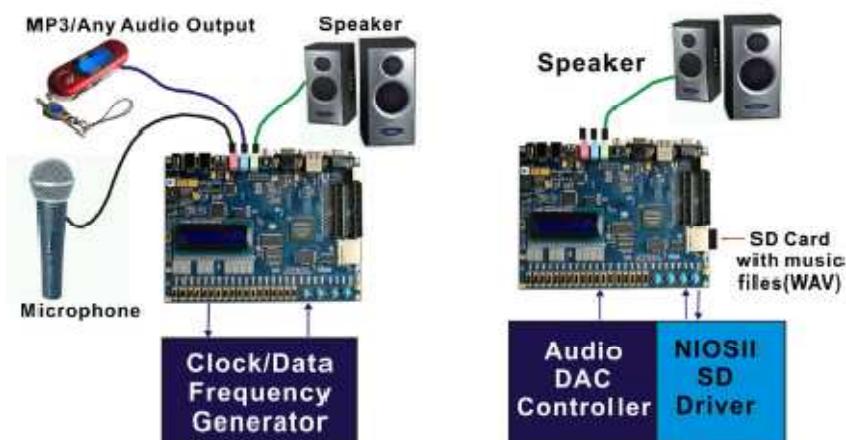
1.6.2 Ứng dụng giao tiếp USB



Hình 1.4 Ứng dụng giao tiếp USB

- ✚ Sử dụng giao tiếp USB trên Kit DE2.
- ✚ Sử dụng trình điều khiển thiết bị chủ và tớ cho Nios II
- ✚ Cung cấp minh họa về SRAM video buffer.

1.6.3 Ứng dụng tạo bài hát karaoke và máy nghe nhạc



Hình 1.5 Ứng dụng trong xử lý âm thanh

- ✚ CD audio 24 bit chất lượng cao

 Tạo một nền thiết kế cho những ứng dụng trên audio.

Chương 2. Hướng dẫn sử dụng Kit DE2

2.1 Kiểm tra Kit DE2

Khi mở nguồn trên Kit DE2, một chương trình đã được khởi tạo sẵn sẽ được nạp lên FPGA, chương trình này thể hiện một số đặc tính của Kit. Chương trình này cũng giúp người sử dụng kiểm tra Kit có hoạt động hay không. Để mở nguồn và kiểm tra Kit, ta thực hiện những bước sau:

- i. Kết nối dây cáp USB từ máy tính chứa phần mềm Quartus II đến cổng giao tiếp USB Blaster trên Kit DE2. Để có thể giao tiếp giữa máy tính và Kit DE2 thông qua cáp USB này, ta cần phải cài đặt trình điều khiển (driver) Altera USB Blaster.
- ii. Kết nối nguồn 9V đến Kit DE2.
- iii. Kết nối màn hình máy tính đến cổng giao tiếp VGA trên Kit DE2.
- iv. Kết nối headphone vào cổng giao tiếp Audio Line-out trên Kit DE2.
- v. Bật switch RUN/PROG trên Kit DE2 về vị trí RUN; vị trí PROG chỉ được dùng khi nạp chương trình cho mode AS (Active Serial).
- vi. Bật power của Kit DE2 bằng cách nhấn switch ON/OFF.

Sau khi bật power, nếu Kit DE2 không có vấn đề gì thì ta có thể quan sát những hiện tượng sau:

- Tất cả LED nhấp nháy.
- Tất cả LED bảy đoạn hiển thị số từ 0 – F theo chu kỳ lập lại.
- Màn hình LCD hiển thị dòng chữ : Welcome to the Altera DE2 Board
- Màn hình máy tính (VGA) hiển thị hình ảnh sau:



Hình 2.1 Màn hình VGA mặc định

- Bật switch SW17 xuống vị trí DOWN, ta sẽ nghe một âm thanh ở tần số 1-KHz
- Bật switch SW17 lên vị trí UP và kết nối đầu ra của máy nghe nhạc (radio, ipod, MP3...) vào cổng giao tiếp Line-in trên Kit DE2; Sau khi kết nối xong, ta sẽ nghe được âm thanh của máy nghe nhạc thông qua headphone đã kết nối vào Line-out trước đó.
- Ta có thể kết nối một microphone vào cổng giao tiếp Microphone-in trên Kit DE2; từ headphone ta có thể nghe được giọng nói của ta chộn lẫn trong tiếng nhạc.

2.2 Hướng dẫn cài đặt USB-Blaster Driver

Bo mạch DE2 được đóng gói bao gồm tất cả những phần cần thiết cho hoạt động của nó ngoài một bộ nguồn adapter 9 volt và cáp kết nối USB. Bo mạch được bảo vệ bởi một lớp kính nhằm hạn chế sự hư hỏng không mong muốn trong quá trình sử dụng.

Cắm nguồn adapter 9 volt vào bo mạch DE2. Sử dụng cáp USB để kết nối cổng USB (cổng nằm liền kề nút mở nguồn (power) trên bo mạch DE2 với cổng

USB trên máy tính để có thể chạy được những phần mềm Quartus II. Sau đó mở nguồn trên bo mạch DE2.

Máy tính sẽ nhận ra một thiết bị phần cứng được kết nối đến cổng USB, tuy nhiên nó sẽ không thể thực thi việc truyền nhận dữ liệu nếu trình điều khiển chưa được cài đặt. Bo mạch DE2 được lập trình bằng việc sử dụng cổng USB-Blaster với cơ chế hoạt động của Altera. Nếu trình điều khiển USB-Blaster chưa được cài đặt, màn hình “New Hardware Wizard” như Hình 2.2 sẽ xuất hiện.



Hình 2.2 Chỉ vị trí driver cho hardware

Bởi vì trình điều khiển cần thiết không có sẵn trên “Window Update Website”, ta chọn “No, not this time” để trả lời cho câu hỏi và nhấn “Next”. Một màn hình như trong Hình 2.3 sẽ xuất hiện.



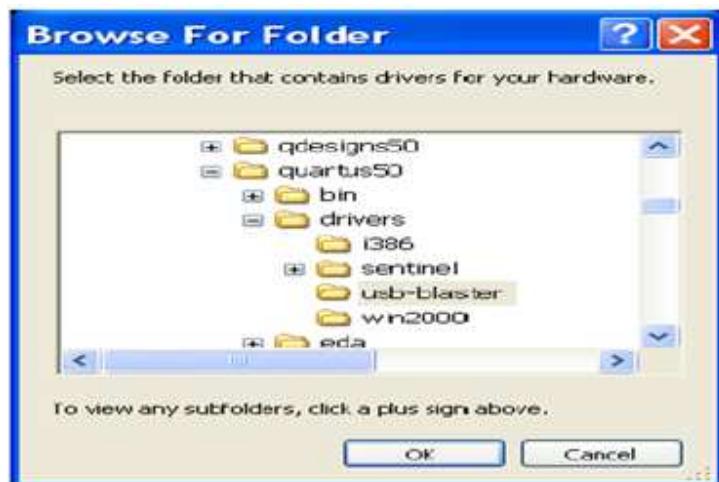
Hình 2.3 Chỉ đường dẫn cho driver

Trình điều khiển USB-Blaster có sẵn bên trong phần mềm cài đặt Quartus II, do đó ta chọn “Install from a specific location” và nhấn “Next” để ra màn hình như trong Hình 2.4.



Hình 2.4 Chỉ đường dẫn cho driver

Bây giờ, ta chọn “Search for the best driver in these locations” và nhấn “Browse” để xuất hiện hộp thoại như trong Hình 2.5. Tìm trình điều khiển mong muốn nằm trong thư mục “C:\altera\90\quartus\drivers\usb-blaster\x32”. Nhấn OK, sau đó màn hình sẽ chuyển về như trong Hình 2.4, nhấn Next. Lúc này thì việc cài đặt được bắt đầu, nhưng sẽ có một hộp thoại như trong Hình 2.6 sẽ xuất hiện để thông báo rằng trình điều khiển chưa hoàn thành kiểm tra “Window Logo”. Nhấn “Continue Anyway”.



Hình 2.5 Chỉ đường dẫn cho driver



Hình 2.6 Không cần kiểm tra driver

Bây giờ trình điều khiển sẽ được cài đặt như trong Hình 2.7. Nhấn Finish và ta có thể bắt đầu sử dụng bo mạch DE2.



Hình 2.7 Driver đã được cài đặt thành công

2.3 Thiết lập cấu hình ban đầu và thay đổi cấu hình mới cho Cyclon II FPGA

Qui trình tải một mạch thiết kế từ máy chủ lên Kit DE2 được mô tả chi tiết trong các phần hướng dẫn thực hành hệ thống số, verilog hoặc kiến trúc máy tính nâng cao trong Chương 4., Chương 5. hoặc Chương 6.. Ta nên đọc phần hướng

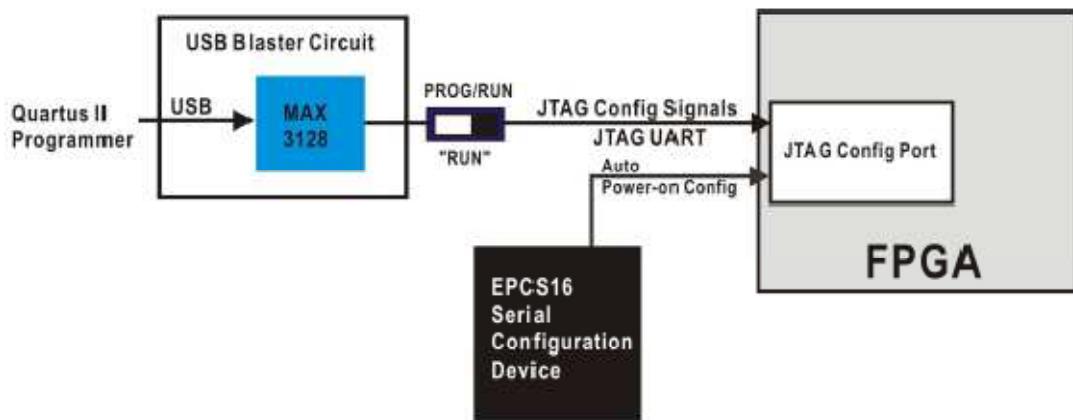
dẫn này trước và tham khảo nhanh những thông tin dưới đây để hiểu rõ qui trình sử dụng Kit.

Trên kit DE2 có một bộ nhớ EEPROM hoạt động nối tiếp, bộ nhớ này chứa dữ liệu để khởi tạo cấu hình ban đầu cho Cyclon II FPGA. Dữ liệu này sẽ tự động được tải lên FPGA từ EEPROM mỗi khi nguồn điện của Kit DE2 được cung cấp. Sau khi nguồn điện được cung cấp, FPGA sẽ điều khiển các thiết bị trên Kit DE2 (leds, switchs,...) hoạt động với dữ liệu khởi tạo. Sau đó, ta hoàn toàn có thể thay đổi cấu hình khởi tạo trên bằng việc tải trực tiếp lên FPGA một dữ liệu, một chương trình do ta thiết kế bằng phần mềm Quartus II. Bên cạnh đó ta cũng có thể thay đổi cấu hình khởi tạo bằng việc thay đổi dữ liệu trên bộ nhớ EEPROM. Hai phương pháp trên sẽ được liệt kê dưới đây:

- JTAG (Joint Test Action Group) programming: Trong phương pháp tải chương trình này, chuỗi bit cấu hình dữ liệu sẽ được tải trực tiếp lên Cyclone II FPGA. FPGA sẽ giữ cấu hình này cho đến khi nguồn điện không còn được cung cấp lên Kit DE2 nữa.
- AS (Active Serial) Programming: Trong phương pháp tải chương trình này, chuỗi bit cấu hình sẽ được tải lên bộ nhớ EPROM 16K hoạt động nối tiếp. Dữ liệu lưu trữ trên bộ nhớ này cố định không bị mất hoặc xóa khi không còn nguồn điện cung cấp. Mỗi khi nguồn điện được cung cấp lên Kit DE2, dữ liệu từ bộ nhớ EEPROM này sẽ tự động được tải lên Cyclone II FPGA.

Từng bước của quá trình thực thi việc tải chương trình bằng JTAG và AS sẽ được mô tả ở dưới. Ở cả hai phương pháp trên, Kit DE2 được kết nối với máy tính chủ thông qua cổng USB. Với kết nối này, Kit DE2 sẽ được nhận dạng bởi máy tính chủ như là một thiết bị USB Blaster. Trình điều khiển (driver) dùng để nhận dạng và giao tiếp giữa máy tính chủ với thiết bị USB Blaster cần được cài đặt trên máy

tính chủ (đã trình bày trong phần 2.2). Thiết lập cấu hình mới cho FPGA từ phần mềm QuartusII trên máy tính thông qua JTAG mode:

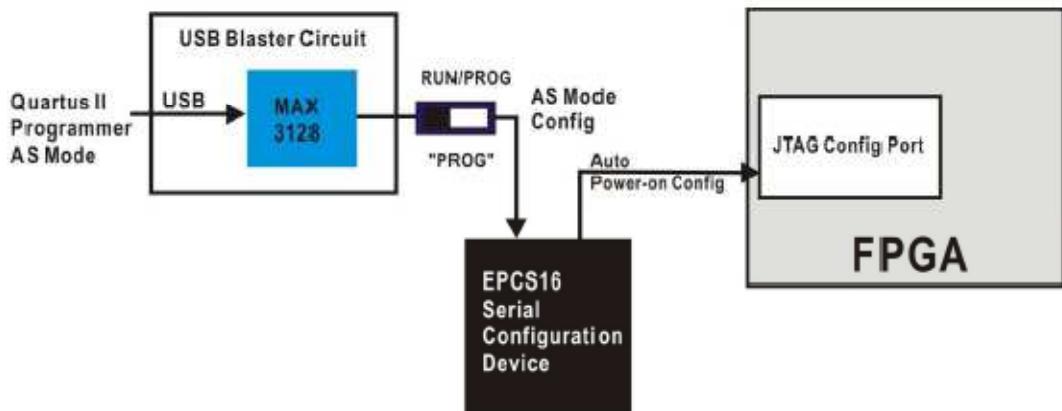


Hình 2.8 Thiết lập cấu hình cho FPGA thông qua JTAG mode

Để tải một chuỗi bit dữ liệu cấu hình lên FPGA, ta cần thực thi những bước sau:

- 1- Cung cấp nguồn cho Kit DE2.
- 2- Kết nối cáp USB đến cổng USB Blaster trên Kit DE2.
- 3- Bật switch RUN/PROG về vị trí RUN (phía trái của Kit).
- 4- Sử dụng công cụ Programmer trên phần mềm Quartus II để tải file chứa chuỗi bit dữ liệu cấu hình có định dạng .sof lên FPGA.

2.3.1 Thiết lập cấu hình ban đầu cho FPGA thông qua việc nạp cấu hình cho bộ nhớ EPPROM EPICS16 bằng AS mode:



Hình 2.9 Thiết lập cấu hình cho FPGA thông qua AS mode

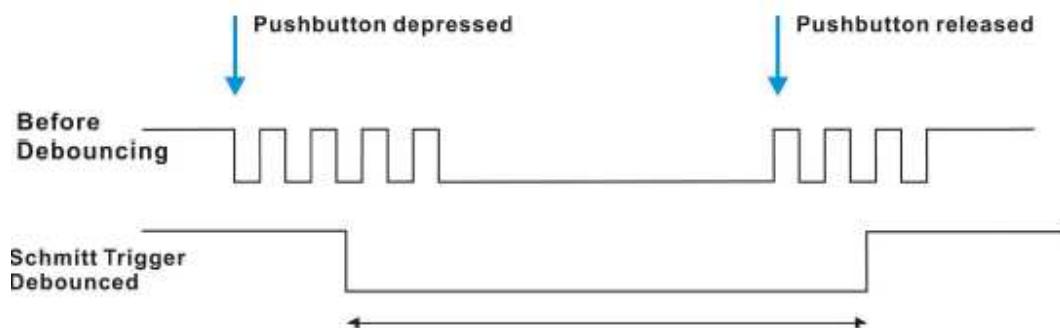
Để tải một chuỗi bit dữ liệu cấu hình lên bộ nhớ EPPROM EPICS16, ta cần thực thi những bước sau:

- 1- Cung cấp nguồn cho Kit DE2.
- 2- Kết nối cáp USB đến cổng USB Blaster trên Kit DE2.
- 3- Bật switch RUN/PROG về vị trí PROG (phía trái của Kit).
- 4- Sử dụng công cụ Programmer trên phần mềm Quartus II để tải file chứa chuỗi bit dữ liệu cấu hình có định dạng .pof lên bộ nhớ EPPROM EPICS16..
- 5- Khi quá trình tải chương trình hoàn thành, bật lại switch RUN/PROG về vị trí RUN và ngắt nguồn khỏi Kit, sau đó lại cung cấp nguồn lại. Thao tác này sẽ làm cho dữ liệu cấu hình mới từ bộ nhớ EPPROM EPICS16 được tự động nạp lên FPGA. Và dữ liệu cấu hình mới này sẽ được xem như dữ liệu cấu hình khởi tạo mới cho Kit DE2 mỗi lần ta cung cấp nguồn cho nó.

2.4 Sơ đồ mạch và hoạt động của các linh kiện trên Kit DE2

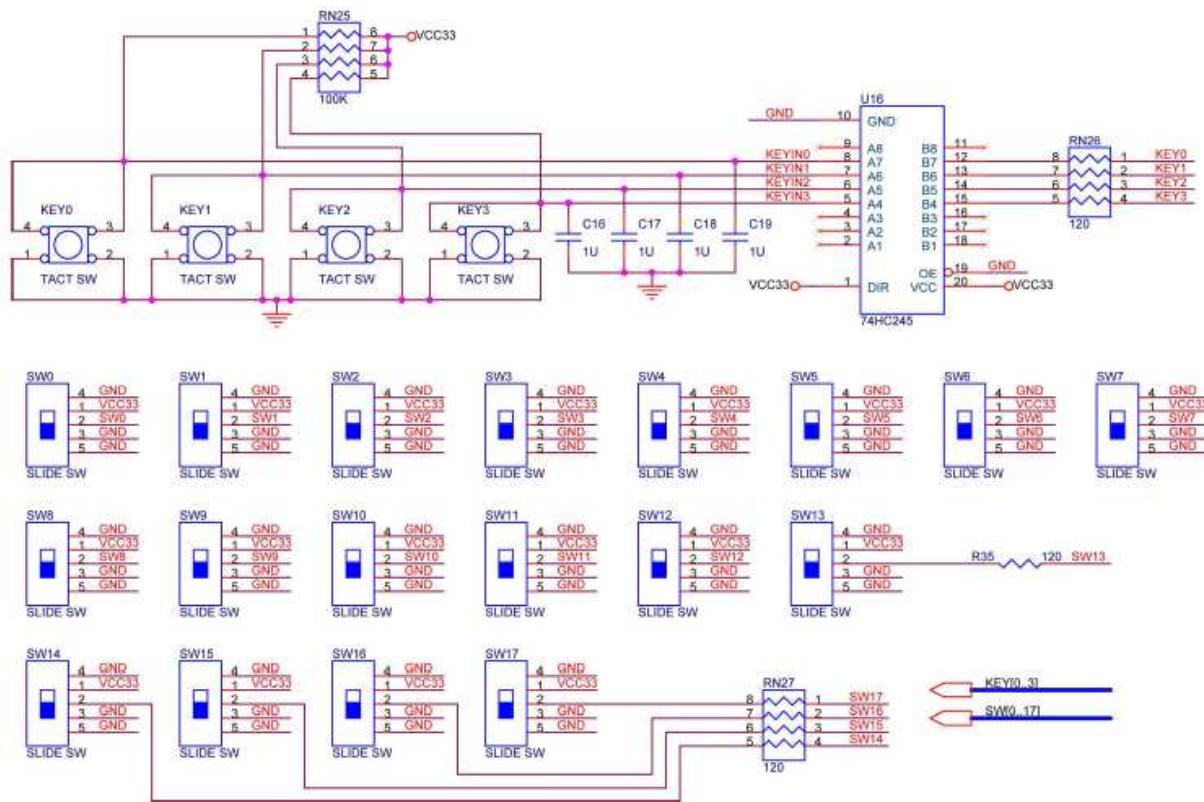
2.4.1 Switches (công tắc) và Button (nút nhấn)

Kit DE2 cung cấp bốn Switch nhấn. Khi không được nhấn tín hiệu ngõ ra của những Switch này ở mức cao, khi Switch được nhấn giá trị tín hiệu ngõ ra của mỗi Switch sẽ tích cực mức thấp và sẽ được phục hồi trở lại giá trị mức cao như cũ nhờ mạch Schmitt Trigger ngay khi Switch được thả ra. Bốn tín hiệu ngõ ra của mạch Schmitt Trigger được gọi là KEY0, ..., KEY3 được kết nối trực tiếp đến pin của Cyclone II FPGA. Những tín hiệu này thích hợp cho việc tạo tín hiệu xung clock hay tín hiệu Reset cho mạch.



Hình 2.10 Chức năng chống nảy cho Push button

Ngoài ra Kit DE2 cung cấp thêm 18 Switch bật tắt hoạt động như những công tắc điện. Khi những Switch này ở vị trí DOWN (gần cạnh của Kit), tín hiệu ngõ ra của nó sẽ ở điện áp mức thấp (0Volt) và ngược lại khi Switch ở vị trí UP, tín hiệu ngõ ra sẽ cung cấp điện áp mức cao (3.3V). Mỗi tín hiệu ngõ ra từ Switch này được nối trực tiếp đến pin của FPGA. Thông thường ta nên sử dụng điện áp mức thấp để điều khiển hoạt động của mạch thiết kế. Dưới đây là sơ đồ mạch kết nối của Switch trên Kit DE2 và danh sách liệt kê những tín hiệu ngõ ra từ Switch được kết nối đến những Pin tương ứng của Cyclone II FPGA.



Hình 2.11 Mạch thiết kế của switches và push button

Signal Name	FPGA Pin No.	Description
SW[0]	PIN_N25	Toggle Switch[0]
SW[1]	PIN_N26	Toggle Switch[1]
SW[2]	PIN_P25	Toggle Switch[2]
SW[3]	PIN_AE14	Toggle Switch[3]
SW[4]	PIN_AF14	Toggle Switch[4]
SW[5]	PIN_AD13	Toggle Switch[5]
SW[6]	PIN_AC13	Toggle Switch[6]
SW[7]	PIN_C13	Toggle Switch[7]
SW[8]	PIN_B13	Toggle Switch[8]
SW[9]	PIN_A13	Toggle Switch[9]
SW[10]	PIN_N1	Toggle Switch[10]
SW[11]	PIN_P1	Toggle Switch[11]
SW[12]	PIN_P2	Toggle Switch[12]
SW[13]	PIN_T7	Toggle Switch[13]
SW[14]	PIN_U3	Toggle Switch[14]
SW[15]	PIN_U4	Toggle Switch[15]
SW[16]	PIN_V1	Toggle Switch[16]
SW[17]	PIN_V2	Toggle Switch[17]

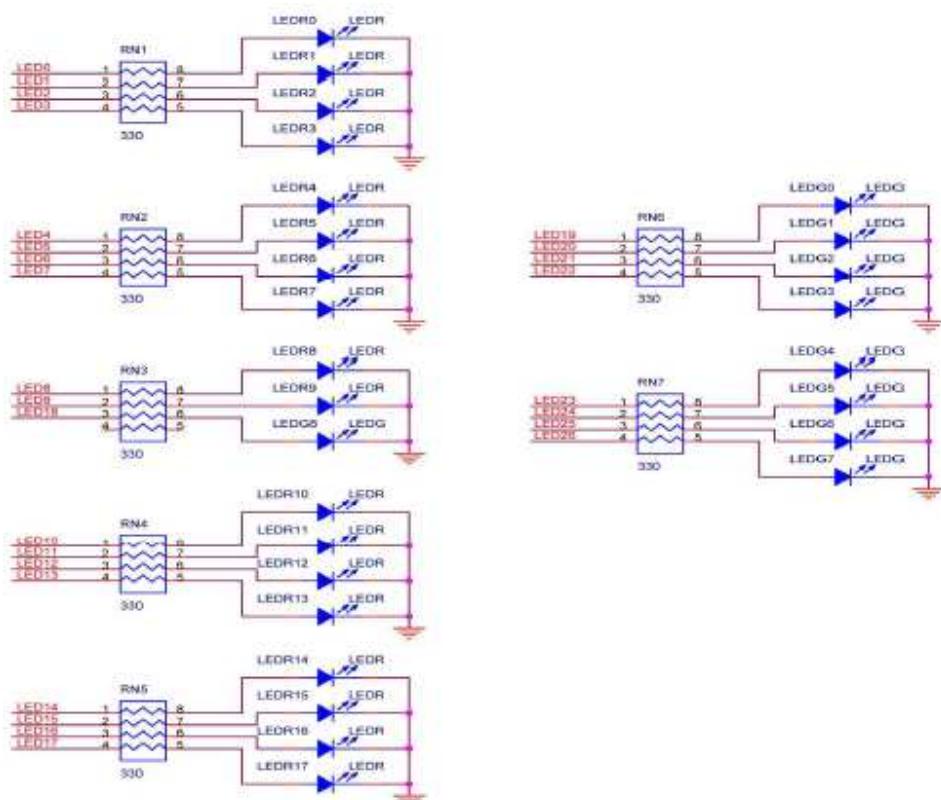
Hình 2.12 Mapped pins giữa switches và FPGA

Signal Name	FPGA Pin No.	Description
KEY[0]	PIN_G26	Pushbutton[0]
KEY[1]	PIN_N23	Pushbutton[1]
KEY[2]	PIN_P23	Pushbutton[2]
KEY[3]	PIN_W26	Pushbutton[3]

Hình 2.13 Mapped pins giữa Push button và FPGA

2.4.2 Leds

Kit DE2 cung cấp 27 LEDs, trong đó có 18 LEDs đỏ (được đặt ngay trên 18 Switch bật tắt), 8 LEDs xanh (được đặt ngay trên 4 Switch nhấn) và LED xanh thứ 9 nằm ngay giữa LED 7 đoạn. Mỗi LED được điều khiển trực tiếp bởi một tín hiệu từ pin của Cyclone II FPGA; LED sáng khi tín hiệu này có điện áp mức cao và ngược lại LED tắt khi tín hiệu có điện áp mức thấp. Dưới đây là sơ đồ mạch kết nối LED và danh sách liệt kê những tín hiệu ngõ vào của LED được kết nối đến những Pin tương ứng của Cyclone II FPGA.



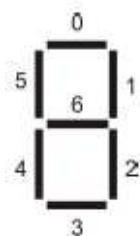
Hình 2.14 Mạch thiết kế của Leds

Signal Name	FPGA Pin No.	Description
LEDR[0]	PIN_AE23	LED Red[0]
LEDR[1]	PIN_AF23	LED Red[1]
LEDR[2]	PIN_AB21	LED Red[2]
LEDR[3]	PIN_AC22	LED Red[3]
LEDR[4]	PIN_AD22	LED Red[4]
LEDR[5]	PIN_AD23	LED Red[5]
LEDR[6]	PIN_AD21	LED Red[6]
LEDR[7]	PIN_AC21	LED Red[7]
LEDR[8]	PIN_AA14	LED Red[8]
LEDR[9]	PIN_Y13	LED Red[9]
LEDR[10]	PIN_AA13	LED Red[10]
LEDR[11]	PIN_AC14	LED Red[11]
LEDR[12]	PIN_AD15	LED Red[12]
LEDR[13]	PIN_AE15	LED Red[13]
LEDR[14]	PIN_AF13	LED Red[14]
LEDR[15]	PIN_AE13	LED Red[15]
LEDR[16]	PIN_AE12	LED Red[16]
LEDR[17]	PIN_AD12	LED Red[17]
LEDG[0]	PIN_AE22	LED Green[0]
LEDG[1]	PIN_AF22	LED Green[1]
LEDG[2]	PIN_W19	LED Green[2]
LEDG[3]	PIN_V18	LED Green[3]
LEDG[4]	PIN_U18	LED Green[4]
LEDG[5]	PIN_U17	LED Green[5]
LEDG[6]	PIN_AA20	LED Green[6]
LEDG[7]	PIN_Y18	LED Green[7]
LEDG[8]	PIN_Y12	LED Green[8]

Hình 2.15 Mapped pins giữa LEDs và FPGA

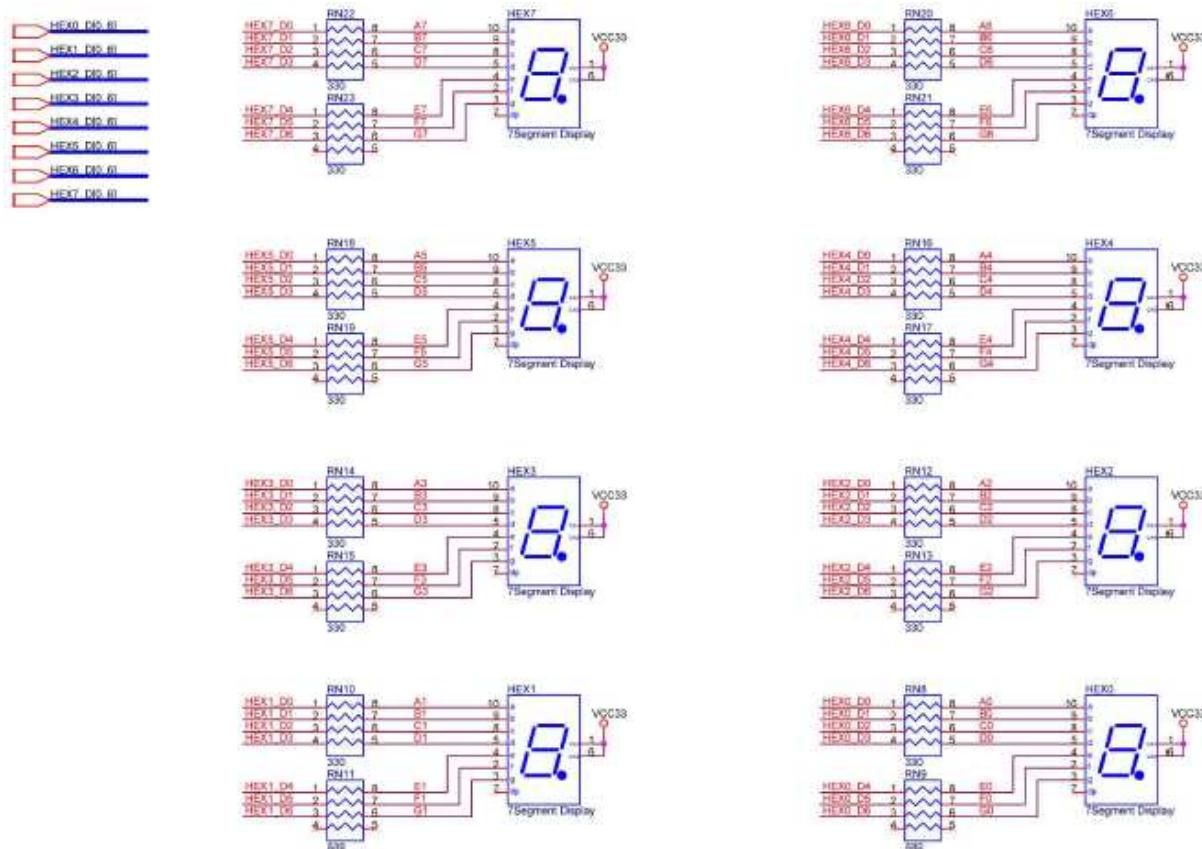
2.4.3 LED hiển thị bảy đoạn

Kit DE2 cung cấp 8 LED hiển thị bảy đoạn. LED sáng khi tín hiệu vào mức thấp và LED tắt khi tín hiệu vào mức cao. Bảy đoạn hiển thị trên LED được đánh số từ 0 đến 6, với vị trí được thể hiện như hình dưới, chú ý là dấu chấm(.) trên mỗi LED không được kết nối nên không thể sử dụng.



Hình 2.16 Led 7 đoạn

Dưới đây là sơ đồ mạch kết nối LED bảy đoạn và danh sách liệt kê những tín hiệu ngõ vào của LED bảy đoạn được kết nối đến những Pin tương ứng của Cyclone II FPGA.



Hình 2.17 Mạch thiết kế của LEDs 7 đoạn

Signal Name	FPGA Pin No.	Description
HEX0[0]	PIN_AF10	Seven Segment Digit 0[0]
HEX0[1]	PIN_AB12	Seven Segment Digit 0[1]
HEX0[2]	PIN_AC12	Seven Segment Digit 0[2]
HEX0[3]	PIN_AD11	Seven Segment Digit 0[3]
HEX0[4]	PIN_AE11	Seven Segment Digit 0[4]
HEX0[5]	PIN_V14	Seven Segment Digit 0[5]
HEX0[6]	PIN_V13	Seven Segment Digit 0[6]

HEX1[0]	PIN_V20	Seven Segment Digit 1[0]
HEX1[1]	PIN_V21	Seven Segment Digit 1[1]
HEX1[2]	PIN_W21	Seven Segment Digit 1[2]
HEX1[3]	PIN_Y22	Seven Segment Digit 1[3]
HEX1[4]	PIN_AA24	Seven Segment Digit 1[4]
HEX1[5]	PIN_AA23	Seven Segment Digit 1[5]
HEX1[6]	PIN_AB24	Seven Segment Digit 1[6]
HEX2[0]	PIN_AB23	Seven Segment Digit 2[0]
HEX2[1]	PIN_V22	Seven Segment Digit 2[1]
HEX2[2]	PIN_AC25	Seven Segment Digit 2[2]
HEX2[3]	PIN_AC26	Seven Segment Digit 2[3]
HEX2[4]	PIN_AB26	Seven Segment Digit 2[4]
HEX2[5]	PIN_AB25	Seven Segment Digit 2[5]
HEX2[6]	PIN_Y24	Seven Segment Digit 2[6]
HEX3[0]	PIN_Y23	Seven Segment Digit 3[0]
HEX3[1]	PIN_AA25	Seven Segment Digit 3[1]
HEX3[2]	PIN_AA26	Seven Segment Digit 3[2]
HEX3[3]	PIN_Y26	Seven Segment Digit 3[3]
HEX3[4]	PIN_Y25	Seven Segment Digit 3[4]
HEX3[5]	PIN_U22	Seven Segment Digit 3[5]
HEX3[6]	PIN_W24	Seven Segment Digit 3[6]
HEX4[0]	PIN_U9	Seven Segment Digit 4[0]
HEX4[1]	PIN_U1	Seven Segment Digit 4[1]
HEX4[2]	PIN_U2	Seven Segment Digit 4[2]
HEX4[3]	PIN_T4	Seven Segment Digit 4[3]
HEX4[4]	PIN_R7	Seven Segment Digit 4[4]

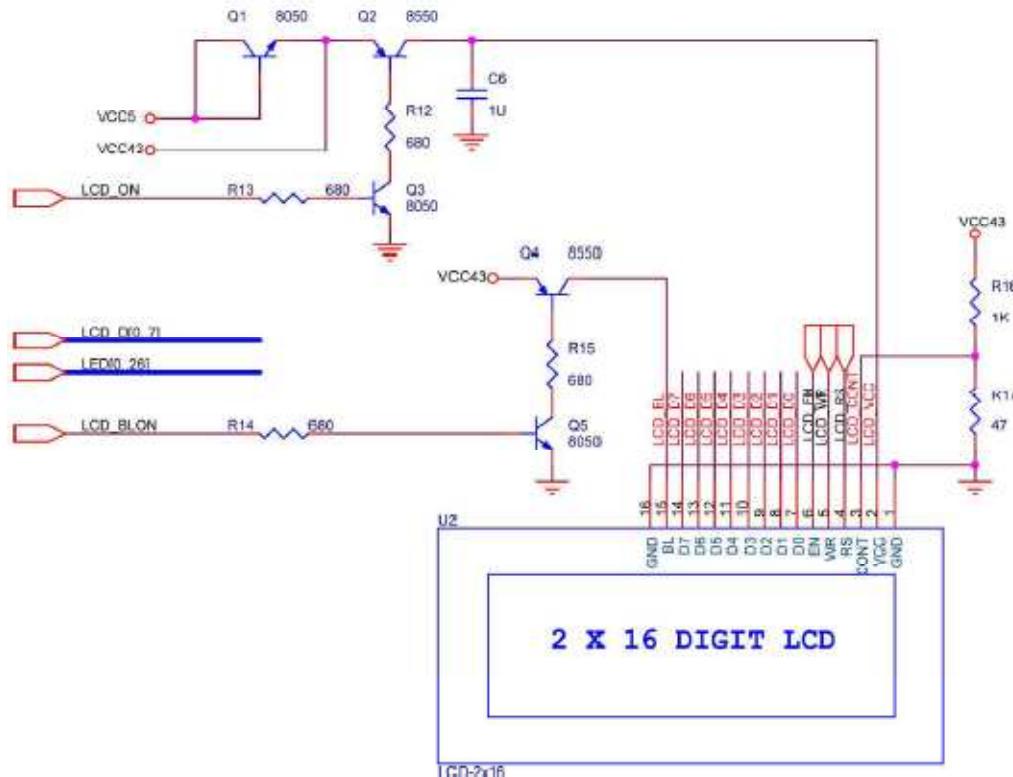
HEX4[5]	PIN_R6	Seven Segment Digit 4[5]
HEX4[6]	PIN_T3	Seven Segment Digit 4[6]
HEX5[0]	PIN_T2	Seven Segment Digit 5[0]
HEX5[1]	PIN_P6	Seven Segment Digit 5[1]
HEX5[2]	PIN_P7	Seven Segment Digit 5[2]
HEX5[3]	PIN_T9	Seven Segment Digit 5[3]
HEX5[4]	PIN_R5	Seven Segment Digit 5[4]
HEX5[5]	PIN_R4	Seven Segment Digit 5[5]
HEX5[6]	PIN_R3	Seven Segment Digit 5[6]
HEX6[0]	PIN_R2	Seven Segment Digit 6[0]
HEX6[1]	PIN_P4	Seven Segment Digit 6[1]
HEX6[2]	PIN_P3	Seven Segment Digit 6[2]
HEX6[3]	PIN_M2	Seven Segment Digit 6[3]
HEX6[4]	PIN_M3	Seven Segment Digit 6[4]
HEX6[5]	PIN_M5	Seven Segment Digit 6[5]
HEX6[6]	PIN_M4	Seven Segment Digit 6[6]
HEX7[0]	PIN_L3	Seven Segment Digit 7[0]
HEX7[1]	PIN_L2	Seven Segment Digit 7[1]
HEX7[2]	PIN_L9	Seven Segment Digit 7[2]
HEX7[3]	PIN_L6	Seven Segment Digit 7[3]
HEX7[4]	PIN_L7	Seven Segment Digit 7[4]
HEX7[5]	PIN_P9	Seven Segment Digit 7[5]
HEX7[6]	PIN_N9	Seven Segment Digit 7[6]

Hình 2.18 Mapped pins giữa LEDs 7 đoạn và FPGA

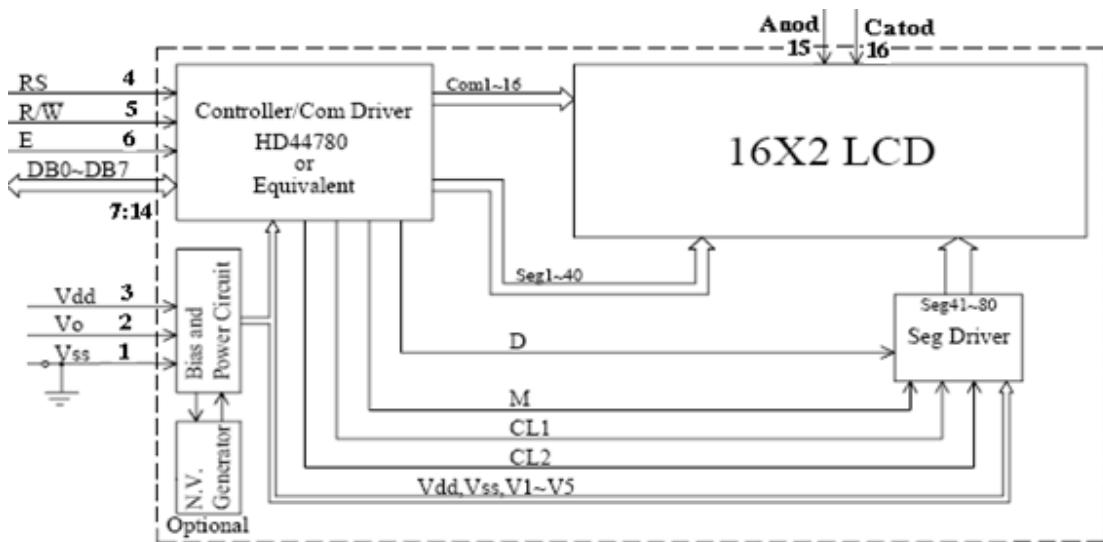
2.4.4 LED hiển thị LCD

Khối LED hiển thị LCD với kiểu chữ mặc định được tạo sẵn dùng để hiện thị những ký tự bằng việc gửi những câu lệnh thích hợp đến khói điều khiển hiển

thị LCD HD44780. Ta có thể tìm hiểu chi tiết hoạt động và chức năng của LCD trên Datasheet của nó. Sơ đồ mạch kết nối của LED LCD và những tín hiệu ngõ vào của LED LCD đến những Pin tương ứng của FPGA và danh sách liệt kê những tín hiệu ngõ vào của LED LCD đến những Pin tương ứng của Cyclone II FPGA.



Hình 2.19 Mạch thiết kế của LCD



Hình 2.20 Cấu tạo LCD

Signal Name	FPGA Pin No.	Description
LCD_DATA[0]	PIN_J1	LCD Data[0]
LCD_DATA[1]	PIN_J2	LCD Data[1]
LCD_DATA[2]	PIN_H1	LCD Data[2]
LCD_DATA[3]	PIN_H2	LCD Data[3]
LCD_DATA[4]	PIN_J4	LCD Data[4]
LCD_DATA[5]	PIN_J3	LCD Data[5]
LCD_DATA[6]	PIN_H4	LCD Data[6]
LCD_DATA[7]	PIN_H3	LCD Data[7]
LCD_RW	PIN_K4	LCD Read/Write Select, 0 = Write, 1 = Read
LCD_EN	PIN_K3	LCD Enable
LCD_RS	PIN_K1	LCD Command/Data Select, 0 = Command, 1 = Data
LCD_ON	PIN_L4	LCD Power ON/OFF
LCD_BLON	PIN_K2	LCD Back Light ON/OFF

Hình 2.21 Mapped pins giữa LCD và FPGA

2.4.4.1 Điều khiển hoạt động cho LCD:

Trong khối hiển thị LCD có một khói điều khiển LSI, khói điều khiển này có hai thanh ghi 8 bit, gồm một thanh ghi mã lệnh (IR) và một thanh ghi dữ liệu (DR).

Thanh ghi mã lệnh IR lưu giữ những mã lệnh như xóa hiển thị, dịch con trỏ và những địa chỉ của dữ liệu cần hiển thị nằm trong bộ nhớ DDRAM và bộ tạo ký tự CGRAM. Thanh ghi IR chỉ có thể được ghi vào từ một vi điều khiển (MPU).

Thanh ghi dữ liệu DR lưu giữ tạm thời những dữ liệu sẽ được ghi vào hoặc đọc ra từ DDRAM hay CGRAM. Khi địa chỉ được ghi vào IR, dữ liệu ứng với địa chỉ đó từ DDRAM hay CGRAM sẽ được lưu giữ trong DR. Để chọn thao tác với một trong hai thanh ghi trên, ta dùng tín hiệu chọn thanh ghi (RS).

RS	R/W	Operation
0	0	IR write as an internal operation (display clear, etc.)
0	1	Read busy flag (DB7) and address counter (DB0 to DB7)
1	0	Write data to DDRAM or CGRAM (DR to DDRAM or CGRAM)
1	1	Read data from DDRAM or CGRAM (DDRAM or CGRAM to DR)

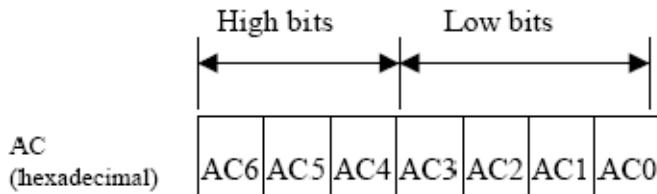
Hình 2.22 Thanh ghi điều khiển hoạt động của LCD

Busy Flag (BF)

Khi RS = 0 và R/W = 1, giá trị của cờ BF sẽ được đọc ra. Nếu BF = 1, báo hiệu khói điều khiển LSI đang bận thực hiện những thao tác bên trong nên nó sẽ không nhận thêm bất kỳ lệnh nào từ bên ngoài. Chỉ khi nào chắc chắn giá trị của cờ BF = 0 thì ta mới ghi lệnh kế tiếp.

Bộ đếm địa chỉ (AC):

Bộ đếm địa chỉ dùng để gán địa chỉ cho DDRAM hoặc CGRAM.



Hình 2.23 Bộ đếm địa chỉ

Bộ nhớ lưu giữ dữ liệu hiển thị (DDRAM):

DDRAM được dùng để lưu giữ kí tự (được mã hóa bởi một giá trị 8 bit) hiển thị trên LCD. Hình dưới đây thể hiện sự tương ứng giữa địa chỉ của DDRAM với vị trí hiển thị của nó trên LCD(2x16)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Hình 2.24 Bộ nhớ lưu giữ dữ liệu hiển thị

Giả sử để hiển thị kí tự nào đó lên vị trí ở hàng thứ 2, cột thứ 15 thì ta phải ghi 8 bit đã mã hóa cho kí tự đó vào ô nhớ có địa chỉ 4E (BCD) trong DDRAM, như vậy địa chỉ của AC sẽ là:

1	0	0	1	1	1	0
---	---	---	---	---	---	---

Muốn ghi dữ liệu vào DDRAM, trước hết ta phải định địa chỉ của DDRAM mà ta muốn dữ liệu đó sẽ được lưu vào (cũng chính là vị trí mà kí tự đó sẽ được hiển thị trên LCD). Việc định địa chỉ của DDRAM thông qua bộ đếm địa chỉ

AC[6:0], sau khi địa chỉ của DDRAM đã được xác định, ta sẽ ghi dữ liệu (ở đây chính là 8 bits mã hóa của mẫu kí tự mà ta muốn hiển thị) vào thông qua 8 bits DB[7:0].

Bộ nhớ ROM lưu giữ mẫu kí tự (Character Generator ROM Pattern)

Bộ nhớ ROM này có dung lượng 4096×8 bits nên có 12 đường địa chỉ A[11:0] để giải mã. Để tạo một mẫu kí tự 5×8 thì ta cần 8 ô nhớ 8 bits. Giả sử như trong hình dưới để lưu giữ một mẫu kí tự "b" trong ROM, nhà sản xuất phải dùng 8 ô nhớ có địa chỉ từ [011000100000] đến [011000100111]. Nội dung của mỗi ô nhớ gồm 8 bits O[7:0] được gán giá trị như hình dưới (Những bit O[7:5] được gán bằng 0). Khi muốn hiển thị kí tự "b" ra màn hình LCD, ta ghi 8 bits dữ liệu mã hóa của kí tự "b" vào bộ nhớ DDRAM (ghi giá trị vào DDRAM đã trình bày trong phần DDRAM), 8 bits dữ liệu mã hóa đó chính là 8 bits giá trị địa chỉ A[11:4] của ROM, vậy ta phải ghi dữ liệu 8 bits [01100010] vào trong bộ nhớ DDRAM để kí tự "b" hiển thị lên LED. Ta tự hỏi quá trình đọc dữ liệu từ 8 ô nhớ trong ROM để xuất ra LED như thế nào. Ta có thể hình dung đơn giản như sau, sau khi nhận 8 bits dữ liệu từ DDRAM, khởi động điều khiển sẽ hiểu 8 bits đó chính là 8 bits A[11:4] của ROM, nó sẽ tự động đọc liên tục (nối tiếp) dữ liệu từ 8 ô nhớ trong ROM mà bắt đầu từ địa chỉ A[11:4][0000] và chuyển nó thành một chuỗi dữ liệu song song 8x8 bits bằng bộ chuyển đổi nối tiếp sang song song và xuất ra LED.

EPROM Address											Data					
											LSB					
A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	D4	D3	D2	D1	00
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	0	1	1	0	0	0	0	0
0	0	1	1	0	0	0	0	1	1	0	0	0	0	1	0	0
0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0
0	1	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0
0	1	1	0	0	0	0	0	1	1	1	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0
0	1	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	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	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
Character code											Line position				← Cursor position	

Hình 2.25 Bộ nhớ lưu giữ mẫu kí tự

Dưới đây là bộ mẫu kí tự đã được nhà sản xuất tạo sẵn trong ROM của khối điều khiển và 8 bits dữ liệu mã hóa (ta dùng để ghi vào bộ nhớ DDRAM) tương ứng cho từng mẫu kí tự. (8 bits dữ liệu này cũng chính là A[11:4] của địa chỉ mà mẫu kí tự đó được lưu trong ROM).

Upper 4 bit Lower 4 bit	LLLL	LLLH	LLHL	LLHH	LHLL	LHLH	LHHL	LHHH	HILL	HLLH	HLHL	HLHH	HHLL	HHHL	HHHH
CG RAM (1)	Gap	P													
LLLL (2)		! 1AQa	q												
LLHL (3)		" 2BRbr													
LLHH (4)		# 3CSc	S												
LHLL (5)		\$ 4DTdt													
LHLH (6)		% 5E!eW													
LHHIIL (7)		& 6FVfV													
LHHHH (8)		' 7G!JgW													
IILLL (1)		(8HKhX													
HLLH (2)) 9IViY													
HLHL (3)		* : JZjz													
HLHH (4)		+ : KDKk	{												
HHLL (5)		; <L%1													
HHLH (6)		- - =M!M	>												
HHHL (7)		. >N^N	+												
HHHH (8)		/ ?O_o	+												

Hình 2.26 Bộ nhớ lưu giữ tất cả các mẫu kí tự

Ví dụ: bây giờ ta muốn hiện thị số "9" lên LCD ở hàng 2, cột 14 thì trước tiên ta phải ghi giá trị địa chỉ hexadecimal 4D [01001101] vào bộ đếm địa chỉ AC sau đó ghi 8 bits dữ liệu mã hóa của kí tự "9" là [00111001].

■ Bộ tạo kí tự RAM (CGRAM)

RAM có dung lượng 64x8 bits nên có 5 đường địa chỉ để giải mã A[5:0]. CGRAM cho phép người sử dụng tự tạo mẫu kí tự cho riêng mình. Vì chỉ có dung lượng 64x8 bits nên ta chỉ có thể tạo được tối đa 8 kí tự 5x8 hoặc 4 kí tự 5x10. Dưới đây là bảng mô tả sự tương quan giữa địa chỉ của CGRAM với DDRAM mà bộ điều khiển sẽ dựa vào đó để kiểm soát hoạt động.

Character Codes (DDRAM data)	CGRAM Address					Character Patterns (CGRAM data)					
7 6 5 4 3 2 1 0 High Low	5	4	3	2	1	0	7 6 5 4 3 2 1 0 High Low	*	*	*	0
0 0 0 0 * 0 0 0	0	0	0	1 0 0	0 0 0	0 0 0	*	*	0 0 0	0 0 0	Character pattern(1)
0 0 0 0 * 0 0 1	0	0	1	0 0 0	1 0 0	1 0 0	*	*	0 0 0	0 0 0	Cursor pattern
0 0 0 0 * 1 1 1	1	1	1	1 0 0	1 0 0	1 0 0	*	*	0 0 0	0 0 0	Character pattern(2)
				1 1 1	1 1 1	1 1 1	*	*	0 0 0	0 0 0	Cursor pattern

Hình 2.27 Bộ tạo mẫu kí tự

Để hiểu ý nghĩa của bảng trên, ta sẽ dùng một thí dụ để minh họa. Vì đây là CGRAM cho phép người sử dụng tự tạo mẫu kí tự riêng nên trước hết ta phải tạo mẫu kí tự mà mình muốn (mà không có sẵn trong CGROM nhà sản xuất đã cung cấp). Giả sử ta muốn tạo mẫu kí tự "R" 5x8 như trên hình vẽ. Vì đây là mẫu kí tự 5x8 nên sẽ cần 8 ô nhớ trong CGRAM để lưu giữ, ở đây lưu mẫu kí tự "R" vào 8 ô nhớ mà bắt đầu bằng ô nhớ có địa chỉ A[000000] và kết thúc là ô nhớ có địa chỉ A[000111] với nội dung của từng ô nhớ như bảng phía bên phải (CGRAM data) tượng trưng cho giá trị "1" được lưu trong CGRAM. Như vậy ta đã tạo xong một mẫu kí tự trong CGRAM.

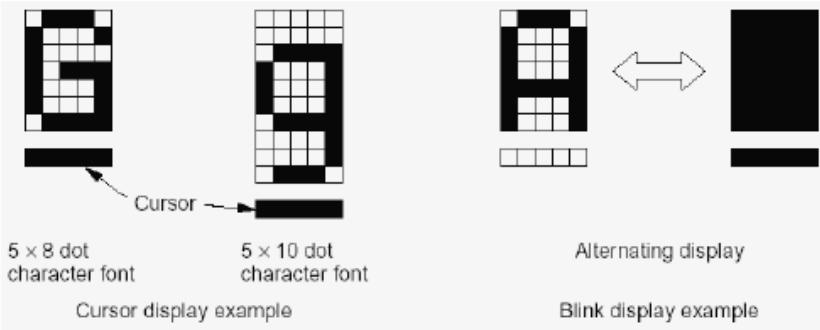
Giờ ta muốn hiện thị kí tự đó ra LCD, vậy làm sao ta biết được 8 bits dữ liệu đã được mã hóa cho mẫu kí tự trên để ghi vào DDRAM cho việc xuất ra. Việc mã hóa này khôi đi điều khiển sẽ tự động ngầm hiểu như sau: Nó lấy 3 bits địa chỉ của CGRAM A[5:3] làm 3 bits thấp D[2:0] của 8 bits mã hóa, 4 bits cao D[7:4] của 8 bits mã hóa sẽ gán bằng "0", còn bit thứ 4 D[3] nó sẽ không quan tâm. Như vậy 8 bits mã hóa cho mẫu kí tự "R" mà ta vừa tạo ở trên là D[0000x000]. Vậy để hiện thị kí tự "R" trên ta chỉ cần ghi 8 bits có giá trị [0000x000] (trong đó "x" có thể là "0" hay "1") vào bộ nhớ DDRAM. (ghi giá trị vào DDRAM đã trình bày trong phần DDRAM).

➡ Tập lệnh của LCD:

Tên lệnh	Hoạt động	t_{exe} (max)
Clear Display	Mã lệnh : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = 0 0 0 0 0 0 0 1 Lệnh Clear Display (xóa hiển thị) sẽ ghi một khoảng trống-blank (mã hiển kí tự 20H) vào tất cả ô nhớ trong DDRAM, sau đó trả bộ đếm địa AC=0, trả lại kiểu hiển thị gốc nếu nó bị thay đổi. Nghĩa là : Tất hiển thị, con trỏ dời về góc trái (hàng đầu tiên), chế độ tăng AC.	
Return home	Mã lệnh : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = 0 0 0 0 0 0 1 * Lệnh Return home trả bộ đếm địa chỉ AC về 0, trả lại kiểu hiển thị gốc nếu nó bị thay đổi. Nội dung của DDRAM không thay đổi.	1.52 ms
Entry mode set	Mã lệnh : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = 0 0 0 0 0 1 [I/D] [S] I/D : Tăng (I/D=1) hoặc giảm (I/D=0) bộ đếm địa chỉ hiển thị AC 1 đơn vị mỗi khi có hành động ghi hoặc đọc vùng DDRAM. Vị trí con trỏ cũng di chuyển theo sự tăng giảm này. S : Khi S=1 toàn bộ nội dung hiển thị bị dịch sang phải (I/D=0) hoặc sang trái (I/D=1) mỗi khi có hành động ghi vùng DDRAM. Khi S=0: không dịch nội dung hiển thị. Nội dung hiển thị không dịch khi đọc DDRAM hoặc đọc/ghi vùng CGRAM.	37 uS

<p>Display position 1 2 3 4 5 6 7 8</p> <p>DDRAM address 00 01 02 03 04 05 06 07</p> <p>For shift left 01 02 03 04 05 06 07 08</p> <p>For shift right 4F 00 01 02 03 04 05 06</p>	<p>Display position 1 2 3 4 5 6 7 8</p> <p>DDRAM address 00 01 02 03 04 05 06 07 40 41 42 43 44 45 46 47</p> <p>For shift left 01 02 03 04 05 06 07 08 41 42 43 44 45 46 47 48</p> <p>For shift right 27 00 01 02 03 04 05 06 67 40 41 42 43 44 45 46</p>	
---	---	--

Figure 3 1-Line by 8-Character Display Example Figure 5 2-Line by 8-Character Display Example

Display on/off control	<p>Mã lệnh : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 $DBx = 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad [D] \quad [C] \quad [B]$</p> <p>D: Hiển thị màn hình khi D=1 và ngược lại. Khi tắt hiển thị, nội dung DDRAM không thay đổi.</p> <p>C: Hiển thị con trỏ khi C=1 và ngược lại. Vị trí và hình dạng con trỏ, xem hình 8</p> <p>B: Nháy nháy kí tự tại vị trí con trỏ khi B=1 và ngược lại. Xem thêm hình 8 về kiểu nháy nháy. Chu kì nháy nháy khoảng 409,6ms khi mạch dao động nội LCD là 250kHz.</p>  <p>5 × 8 dot character font 5 × 10 dot character font Alternating display</p> <p>Cursor display example Blink display example</p>	37uS															
Cursor or display shift	<p>Mã lệnh : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 $DBx = 0 \quad 0 \quad 0 \quad 1 \quad [S/C] \quad [R/L] \quad * \quad *$</p> <p>Lệnh Cursor or display shift dịch chuyển con trỏ hay dữ liệu hiển thị sang trái mà không cần hành động ghi/đọc dữ liệu. Khi hiển thị kiểu 2 dòng, con trỏ sẽ nhảy xuống dòng dưới khi dịch qua vị trí thứ 40 của hàng đầu tiên. Dữ liệu hàng đầu và hàng 2 dịch cùng một lúc. Chi tiết sử dụng xem bảng bên dưới:</p> <table border="1" data-bbox="398 1208 1218 1423"> <thead> <tr> <th>S/C</th><th>R/L</th><th>Hoạt động</th></tr> </thead> <tbody> <tr> <td>0</td><td>0</td><td>Dịch vị trí con trỏ sang trái (Nghĩa là giảm AC một đơn vị).</td></tr> <tr> <td>0</td><td>1</td><td>Dịch vị trí con trỏ sang phải (Tăng AC lên 1 đơn vị).</td></tr> <tr> <td>1</td><td>0</td><td>Dịch toàn bộ nội dung hiển thị sang trái, con trỏ cũng dịch theo.</td></tr> <tr> <td>1</td><td>1</td><td>Dịch toàn bộ nội dung hiển thị sang phải, con trỏ cũng dịch theo.</td></tr> </tbody> </table>	S/C	R/L	Hoạt động	0	0	Dịch vị trí con trỏ sang trái (Nghĩa là giảm AC một đơn vị).	0	1	Dịch vị trí con trỏ sang phải (Tăng AC lên 1 đơn vị).	1	0	Dịch toàn bộ nội dung hiển thị sang trái, con trỏ cũng dịch theo.	1	1	Dịch toàn bộ nội dung hiển thị sang phải, con trỏ cũng dịch theo.	37uS
S/C	R/L	Hoạt động															
0	0	Dịch vị trí con trỏ sang trái (Nghĩa là giảm AC một đơn vị).															
0	1	Dịch vị trí con trỏ sang phải (Tăng AC lên 1 đơn vị).															
1	0	Dịch toàn bộ nội dung hiển thị sang trái, con trỏ cũng dịch theo.															
1	1	Dịch toàn bộ nội dung hiển thị sang phải, con trỏ cũng dịch theo.															

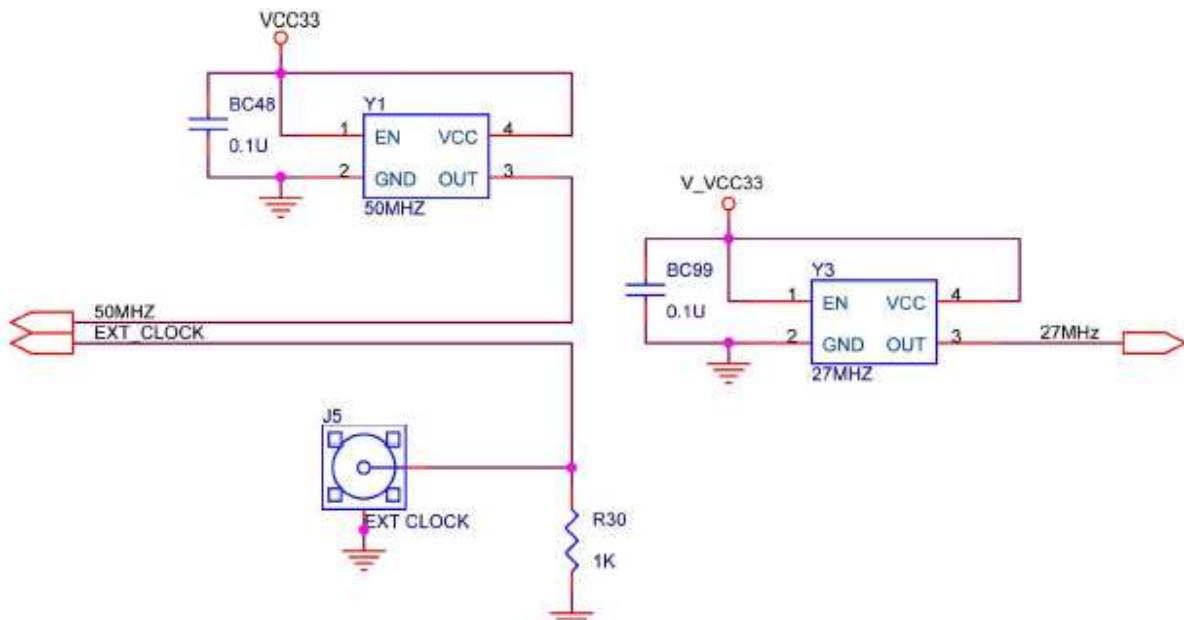
Function set	<p>Mã lệnh : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 $DBx = 0 \quad 0 \quad 1 \quad [DL] \quad [N] \quad [F] \quad * \quad *$</p> <p>DL: Khi DL=1, LCD giao tiếp với MPU bằng giao thức 8 bit (từ bit DB7 đến DB0). Ngược lại, giao thức giao tiếp là 4 bit (từ bit DB7 đến bit DB0). Khi chọn giao thức 4 bit, dữ liệu được truyền/nhận 2 lần liên tiếp. với 4 bit cao gửi/nhận trước, 4 bit thấp gửi/nhận sau.</p> <p>N : Thiết lập số hàng hiển thị. Khi N=0 : hiển thị 1 hàng, N=1: hiển thị 2 hàng.</p> <p>F : Thiết lập kiểu kí tự. Khi F=0: kiểu kí tự 5x8 điểm ảnh, F=1: kiểu kí tự 5x10 điểm ảnh.</p> <p>* <i>Chú ý:</i></p> <ul style="list-style-type: none"> • <i>Chỉ thực hiện thay đổi Function set ở đầu chương trình. Và sau khi được thực thi 1 lần, lệnh thay đổi Function set không được LCD chấp nhận nữa ngoại trừ thiết lập chuyển đổi giao thức giao tiếp.</i> • <i>Không thể hiển thị kiểu kí tự 5x10 điểm ảnh ở kiểu hiển thị 2 hàng</i> 	37uS
Set CGRAM address	<p>Mã lệnh : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 $DBx = 0 \quad 1 \quad [ACG][ACG][ACG][ACG][ACG][ACG]$</p> <p>Lệnh này ghi vào AC địa chỉ của CGRAM. Kí hiệu [ACG] chỉ 1 bit của chuỗi dữ liệu 6 bit. Ngay sau lệnh này là lệnh đọc/ghi dữ liệu từ CGRAM tại địa chỉ đã được chỉ định.</p>	37uS
Set DDRAM address	<p>Mã lệnh : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 $DBx = 1 \quad [AD] \quad [AD] \quad [AD] \quad [AD] \quad [AD] \quad [AD]$</p> <p>Lệnh này ghi vào AC địa chỉ của DDRAM, dùng khi cần thiết lập tọa độ hiển thị mong muốn. Ngay sau lệnh này là lệnh đọc/ghi dữ liệu từ DDRAM tại địa chỉ đã được chỉ định.</p> <p>Khi ở chế độ hiển thị 1 hàng: địa chỉ có thể từ 00H đến 4FH. Khi ở chế độ hiển thị 2 hàng, địa chỉ từ 00h đến 27H cho hàng thứ nhất, và từ 40h đến 67h cho hàng thứ 2. Xem chi tiết ở hình 4.</p>	37uS
Read BF and address	<p>Mã lệnh : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 $DBx = [BF] \quad [AC] \quad [AC] \quad [AC] \quad [AC] \quad [AC] \quad [AC] \quad (RS=0, R/W=1)$</p> <p>Như đã đề cập trước đây, khi cờ BF bật, LCD đang làm việc và lệnh tiếp theo (nếu có) sẽ bị bỏ qua nếu cờ BF chưa về mức thấp. Cho nên, khi lập trình điều khiển, bạn phải kiểm tra cờ BF trước khi ghi dữ liệu vào LCD.</p> <p>Khi đọc cờ BF, giá trị của AC cũng được xuất ra các bit [AC]. Nó là địa chỉ của CG hay DDRAM là tùy thuộc vào lệnh trước đó</p>	0uS
Write data to CG or DDRAM	<p>Mã lệnh : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 $DBx = \quad [Write data] \quad (RS=1, R/W=0)$</p> <p>Khi thiết lập RS=1, R/W=0, dữ liệu cần ghi được đưa vào các chân DBx từ mạch ngoài sẽ được LCD chuyển vào trong LCD tại địa chỉ được xác định từ lệnh ghi địa chỉ trước đó (lệnh ghi địa chỉ cũng xác định luôn vùng RAM cần ghi)</p> <p>Sau khi ghi, bộ đếm địa chỉ AC tự động tăng/giảm 1 tùy theo thiết lập Entry mode. Lưu ý là thời gian cập nhật AC không tính vào thời gian thực thi lệnh.</p> <p>Chi tiết về giao thức Ghi dữ liệu, xin xem hình 10.</p>	37uS t _{ADD} 4uS
Read data from CG	<p>Mã lệnh : DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 $DBx = \quad [Read data] \quad (RS=1, R/W=1)$</p>	37uS t _{ADD}

or DDRAM	<p>Khi thiết lập RS=1, R/W=1, dữ liệu từ CG/DDRAM được chuyển ra MPU thông qua các chân DBx (địa chỉ và vùng RAM đã được xác định bằng lệnh ghi địa chỉ trước đó).</p> <p>Sau khi đọc, AC tự động tăng/giảm 1 tùy theo thiết lập Entry mode, tuy nhiên nội dung hiển thị không bị dịch bất chấp chế độ Entry mode.</p> <p>Chi tiết hơn về giao thức đọc dữ liệu, xin xem hình 11.</p>	4uS
---------------------	---	-----

Hình 2.28 Tập lệnh LCD

2.4.5 Ngõ vào xung Clock

Kit DE2 cung cấp hai nguồn tín hiệu xung Clock 27 MHz và 50 MHz. Ngoài ra ta cũng có thể cung cấp nguồn xung Clock từ bên ngoài thông qua cổng ngõ vào SMA. Sơ đồ mạch kết nối nguồn xung Clock và danh sách liệt kê những tín hiệu nguồn xung Clock kết nối đến những Pin tương ứng của Cyclone II FPGA.



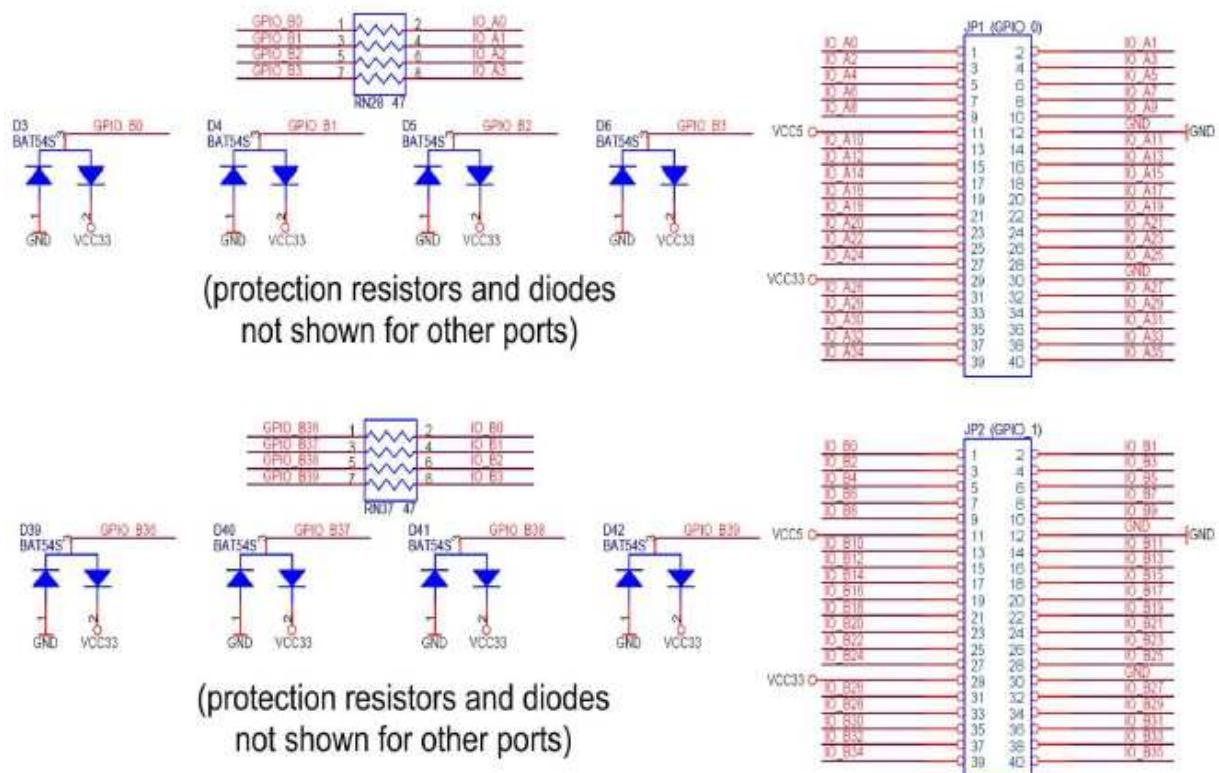
Hình 2.29 Mạch thiết kế của ngõ vào xung Clock

Signal Name	FPGA Pin No.	Description
CLOCK_27	PIN_D13	27 MHz clock input
CLOCK_50	PIN_N2	50 MHz clock input
EXT_CLOCK	PIN_P26	External (SMA) clock input

Hình 2.30 Mapped pins giữa ngõ vào xung Clock và FPGA

2.4.6 Expansion Header (Jac cắm mở rộng)

Kit DE2 cung cấp thêm hai expansion headers 40-pins. Mỗi header được kết nối trực tiếp đến 36 pins của Cyclone II FPGA, 1 pin nguồn DC +5V (VCC5), 1 pin nguồn DC +3.3V (VCC33) và 2 pins GND. Hình dưới mô tả một phần của mạch cho 4 pins của mỗi expansion headers, mạch đầy đủ sẽ gồm 40 pins cho mỗi expansion headers. Mỗi pin từ expansion header được kết nối đến hai diode và một điện trở dùng để bảo vệ khỏi hiện tượng quá áp hay hạ áp.



Hình 2.31 Mạch thiết kế giao tiếp giữa PIO và FPGA

Dưới đây là bảng liệt kê 80 pins của hai expansion header và 80 pins của FPGA được kết nối tương ứng .

Signal Name	FPGA Pin No.	Description
GPIO_0[0]	PIN_D25	GPIO Connection 0[0]
GPIO_0[1]	PIN_J22	GPIO Connection 0[1]
GPIO_0[2]	PIN_E26	GPIO Connection 0[2]
GPIO_0[3]	PIN_E25	GPIO Connection 0[3]
GPIO_0[4]	PIN_F24	GPIO Connection 0[4]
GPIO_0[5]	PIN_F23	GPIO Connection 0[5]
GPIO_0[6]	PIN_J21	GPIO Connection 0[6]
GPIO_0[7]	PIN_J20	GPIO Connection 0[7]
GPIO_0[8]	PIN_F25	GPIO Connection 0[8]
GPIO_0[9]	PIN_F26	GPIO Connection 0[9]
GPIO_0[10]	PIN_N18	GPIO Connection 0[10]
GPIO_0[11]	PIN_P18	GPIO Connection 0[11]
GPIO_0[12]	PIN_G23	GPIO Connection 0[12]
GPIO_0[13]	PIN_G24	GPIO Connection 0[13]
GPIO_0[14]	PIN_K22	GPIO Connection 0[14]
GPIO_0[15]	PIN_G25	GPIO Connection 0[15]
GPIO_0[16]	PIN_H23	GPIO Connection 0[16]
GPIO_0[17]	PIN_H24	GPIO Connection 0[17]
GPIO_0[18]	PIN_J23	GPIO Connection 0[18]
GPIO_0[19]	PIN_J24	GPIO Connection 0[19]
GPIO_0[20]	PIN_H25	GPIO Connection 0[20]
GPIO_0[21]	PIN_H26	GPIO Connection 0[21]
GPIO_0[22]	PIN_H19	GPIO Connection 0[22]
GPIO_0[23]	PIN_K18	GPIO Connection 0[23]
GPIO_0[24]	PIN_K19	GPIO Connection 0[24]

GPIO_0[25]	PIN_K21	GPIO Connection 0[25]
GPIO_0[26]	PIN_K23	GPIO Connection 0[26]
GPIO_0[27]	PIN_K24	GPIO Connection 0[27]
GPIO_0[28]	PIN_L21	GPIO Connection 0[28]
GPIO_0[29]	PIN_L20	GPIO Connection 0[29]
GPIO_0[30]	PIN_J25	GPIO Connection 0[30]
GPIO_0[31]	PIN_J26	GPIO Connection 0[31]
GPIO_0[32]	PIN_L23	GPIO Connection 0[32]
GPIO_0[33]	PIN_L24	GPIO Connection 0[33]
GPIO_0[34]	PIN_L25	GPIO Connection 0[34]
GPIO_0[35]	PIN_L19	GPIO Connection 0[35]
GPIO_1[0]	PIN_K25	GPIO Connection 1[0]
GPIO_1[1]	PIN_K26	GPIO Connection 1[1]
GPIO_1[2]	PIN_M22	GPIO Connection 1[2]
GPIO_1[3]	PIN_M23	GPIO Connection 1[3]
GPIO_1[4]	PIN_M19	GPIO Connection 1[4]
GPIO_1[5]	PIN_M20	GPIO Connection 1[5]
GPIO_1[6]	PIN_N20	GPIO Connection 1[6]
GPIO_1[7]	PIN_M21	GPIO Connection 1[7]
GPIO_1[8]	PIN_M24	GPIO Connection 1[8]
GPIO_1[9]	PIN_M25	GPIO Connection 1[9]
GPIO_1[10]	PIN_N24	GPIO Connection 1[10]
GPIO_1[11]	PIN_P24	GPIO Connection 1[11]
GPIO_1[12]	PIN_R25	GPIO Connection 1[12]
GPIO_1[13]	PIN_R24	GPIO Connection 1[13]
GPIO_1[14]	PIN_R20	GPIO Connection 1[14]

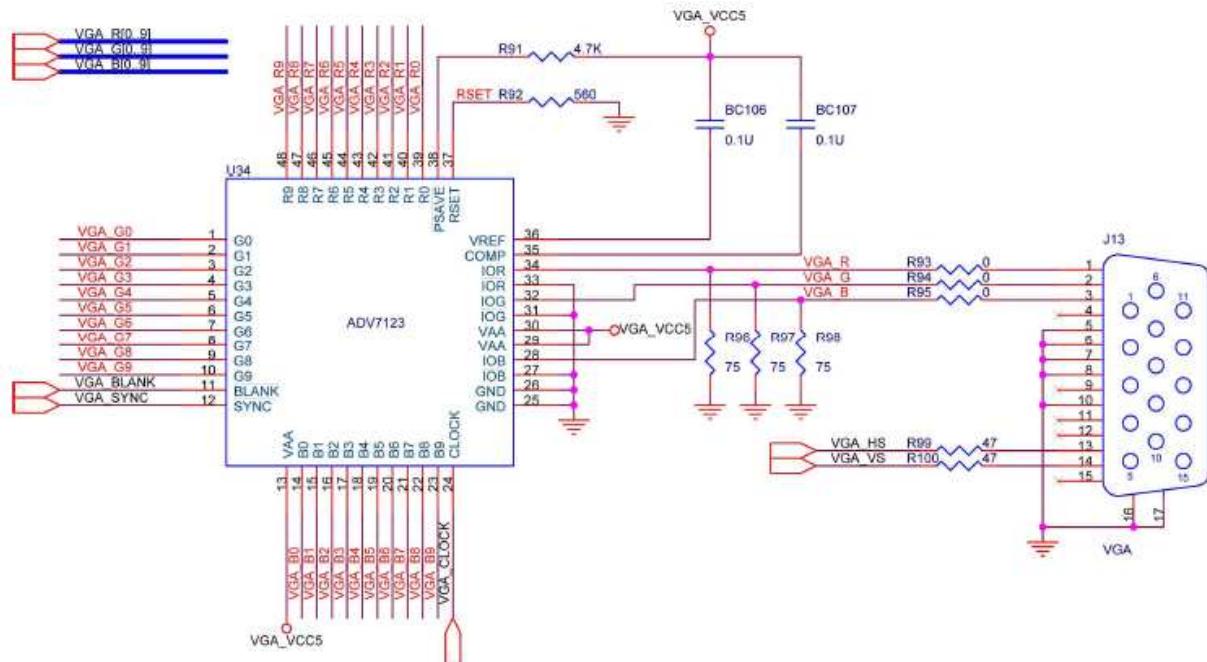
GPIO_1[15]	PIN_T22	GPIO Connection 1[15]
GPIO_1[16]	PIN_T23	GPIO Connection 1[16]
GPIO_1[17]	PIN_T24	GPIO Connection 1[17]
GPIO_1[18]	PIN_T25	GPIO Connection 1[18]
GPIO_1[19]	PIN_T18	GPIO Connection 1[19]
GPIO_1[20]	PIN_T21	GPIO Connection 1[20]
GPIO_1[21]	PIN_T20	GPIO Connection 1[21]
GPIO_1[22]	PIN_U26	GPIO Connection 1[22]
GPIO_1[23]	PIN_U25	GPIO Connection 1[23]
GPIO_1[24]	PIN_U23	GPIO Connection 1[24]
GPIO_1[25]	PIN_U24	GPIO Connection 1[25]
GPIO_1[26]	PIN_R19	GPIO Connection 1[26]
GPIO_1[27]	PIN_T19	GPIO Connection 1[27]
GPIO_1[28]	PIN_U20	GPIO Connection 1[28]
GPIO_1[29]	PIN_U21	GPIO Connection 1[29]
GPIO_1[30]	PIN_V26	GPIO Connection 1[30]
GPIO_1[31]	PIN_V25	GPIO Connection 1[31]
GPIO_1[32]	PIN_V24	GPIO Connection 1[32]
GPIO_1[33]	PIN_V23	GPIO Connection 1[33]
GPIO_1[34]	PIN_W25	GPIO Connection 1[34]
GPIO_1[35]	PIN_W23	GPIO Connection 1[35]

Hình 2.32 Mapped pins giữa pin PIO và FPGA

2.4.7 VGA

Bo mạch DE2 có một ngõ raVGA D-SUB 16 pin. Những tín hiệu đồng bộ VGA được cung cấp trực tiếp từ FPGA Cyclone II, và một con chip xử lí tín hiệu số sang tương tự video DAC ADV7123 với tốc độ cao 10 bit được dùng để tạo ra những tín hiệu tương tự (đỏ, xanh lam, xanh lục). Sơ đồ mạch VGA được cho ở

Hình 2.33 và có thể hỗ trợ độ phân giải lên đến 1600x1200 pixels ở tốc độ 100MHz.

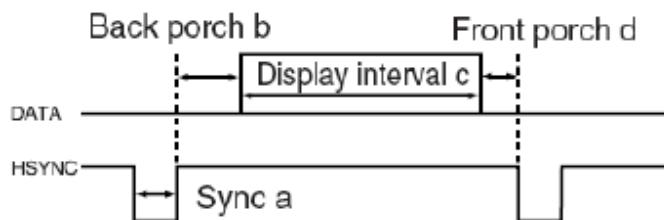


Hình 2.33 Sơ đồ mạch VGA

Đặc tả thời gian cho những dữ liệu về đồng bộ VGA cũng như dữ liệu RGB (đỏ, xanh lục, xanh lam) có thể được tìm thấy trên nhiều trang web về đào tạo (ví dụ, chỉ cần tìm kiếm trên google với từ khóa “VGA signal timing”). Hình 2.34 minh họa những ràng buộc cơ bản về thời gian cho mỗi hàng (horizontal) để hiện thị trên màn hình VGA. Một tín hiệu xung tích cực mức thấp trong một khoảng thời gian nhất định (khoang thời gian a như trong hình) được cấp đến tín hiệu ngõ vào đồng bộ hàng (hsync) của màn hình để cho biết dấu hiệu kết thúc một hàng dữ liệu và bắt đầu một hàng dữ liệu kế tiếp. Những ngõ vào RGB trên màn hình phải tắt (điều khiển về 0V) trong một khoảng thời gian gọi là “back porch” (b) sau khi xung tín hiệu ngõ vào đồng bộ hàng (hsync) xuất hiện, và tiếp tục theo sau khoảng

thời gian (b) này là khoảng thời gian hiển thị (c). Trong suốt khoảng thời gian hiện thị, tín hiệu RGB điều khiển mỗi pixel hiện thị đọc theo hàng. Cuối cùng, trong một khoảng thời gian gọi là “front porch” (d) những tín hiệu RGB phải tắt đi lần nữa trước khi tín hiệu xung ngõ vào đồng bộ hàng (hsync) kế tiếp xuất hiện để bắt đầu một hàng mới. Đặc tả về thời gian cho tín hiệu đồng bộ cột (vsync) được trình bày tương tự trong Hình 2.34, nó chỉ khác đó là xung tín hiệu vsync cho biết dấu hiệu kết thúc một khung (frame) màn hình và bắt đầu một khung màn hình kế tiếp. Hình 2.35 và Hình 2.36 mô tả những khoảng thời gian đồng bộ hàng và cột a, b, c và d ứng với những độ phân giải khác nhau.

Thông tin chi tiết về việc sử dụng chip Video DAC ADV7123 được mô tả rõ trong datasheet của nó mà ta có thể tìm thấy trên website của nhà sản xuất. Việc gán pin giữa FPGA Cyclone II và chip ADV7123 được liệt kê trong Hình 2.37.



Hình 2.34 Giản đồ định thời của tín hiệu HSYNC

VGA mode		Horizontal Timing Spec				
Configuration	Resolution(HxV)	a(us)	b(us)	c(us)	d(us)	Pixel clock(Mhz)
VGA(60Hz)	640x480	3.8	1.9	25.4	0.6	25 (640/c)
VGA(85Hz)	640x480	1.6	2.2	17.8	1.6	36 (640/c)
SVGA(60Hz)	800x600	3.2	2.2	20	1	40 (800/c)
SVGA(75Hz)	800x600	1.6	3.2	16.2	0.3	49 (800/c)
SVGA(85Hz)	800x600	1.1	2.7	14.2	0.6	56 (800/c)
XGA(60Hz)	1024x768	2.1	2.5	15.8	0.4	65 (1024/c)
XGA(70Hz)	1024x768	1.8	1.9	13.7	0.3	75 (1024/c)
XGA(85Hz)	1024x768	1.0	2.2	10.8	0.5	95 (1024/c)
1280x1024(60Hz)	1280x1024	1.0	2.3	11.9	0.4	108 (1280/c)

Hình 2.35 Mô tả định thời cho việc đồng bộ hàng

VGA mode		Vertical Timing Spec			
Configuration	Resolution (HxV)	a(lines)	b(lines)	c(lines)	d(lines)
VGA(60Hz)	640x480	2	33	480	10
VGA(85Hz)	640x480	3	25	480	1
SVGA(60Hz)	800x600	4	23	600	1
SVGA(75Hz)	800x600	3	21	600	1
SVGA(85Hz)	800x600	3	27	600	1
XGA(60Hz)	1024x768	6	29	768	3
XGA(70Hz)	1024x768	6	29	768	3
XGA(85Hz)	1024x768	3	36	768	1
1280x1024(60Hz)	1280x1024	3	38	1024	1

Hình 2.36 Mô tả định thời cho việc đồng bộ cột

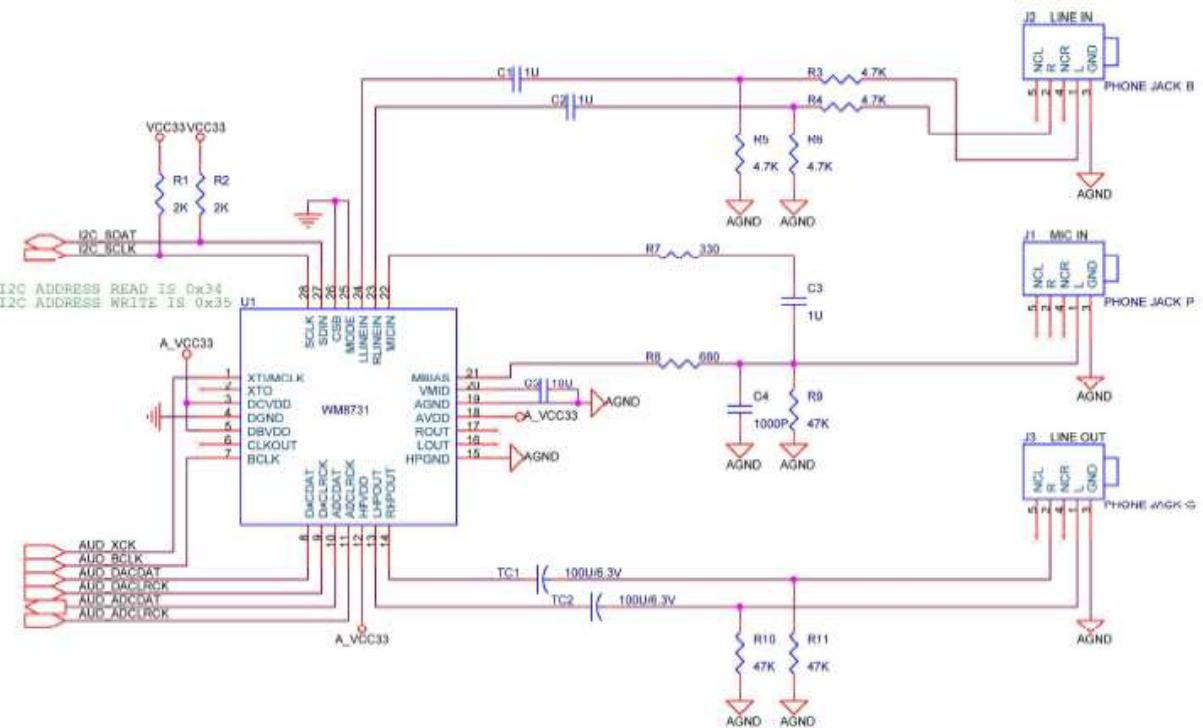
Signal Name	FPGA Pin No.	Description
VGA_R[0]	PIN_C8	VGA Red[0]
VGA_R[1]	PIN_F10	VGA Red[1]
VGA_R[2]	PIN_G10	VGA Red[2]
VGA_R[3]	PIN_D9	VGA Red[3]
VGA_R[4]	PIN_C9	VGA Red[4]
VGA_R[5]	PIN_A8	VGA Red[5]
VGA_R[6]	PIN_H11	VGA Red[6]
VGA_R[7]	PIN_H12	VGA Red[7]
VGA_R[8]	PIN_F11	VGA Red[8]
VGA_R[9]	PIN_E10	VGA Red[9]
VGA_G[0]	PIN_B9	VGA Green[0]
VGA_G[1]	PIN_A9	VGA Green[1]
VGA_G[2]	PIN_C10	VGA Green[2]
VGA_G[3]	PIN_D10	VGA Green[3]
VGA_G[4]	PIN_B10	VGA Green[4]
VGA_G[5]	PIN_A10	VGA Green[5]
VGA_G[6]	PIN_G11	VGA Green[6]
VGA_G[7]	PIN_D11	VGA Green[7]
VGA_G[8]	PIN_E12	VGA Green[8]
VGA_G[9]	PIN_D12	VGA Green[9]
VGA_B[0]	PIN_J13	VGA Blue[0]
VGA_B[1]	PIN_J14	VGA Blue[1]
VGA_B[2]	PIN_F12	VGA Blue[2]
VGA_B[3]	PIN_G12	VGA Blue[3]
VGA_B[4]	PIN_J10	VGA Blue[4]

VGA_B[5]	PIN_J11	VGA Blue[5]
VGA_B[6]	PIN_C11	VGA Blue[6]
VGA_B[7]	PIN_B11	VGA Blue[7]
VGA_B[8]	PIN_C12	VGA Blue[8]
VGA_B[9]	PIN_B12	VGA Blue[9]
VGA_CLK	PIN_B8	VGA Clock
VGA_BLANK	PIN_D6	VGA BLANK
VGA_HS	PIN_A7	VGA H_SYNC
VGA_VS	PIN_D8	VGA V_SYNC
VGA_SYNC	PIN_B7	VGA SYNC

Hình 2.37 Mapped pins giữa ADV7123 và FPGA

2.4.8 Audio CODEC 24-bit

Bo mạch DE2 cung cấp việc xử lý âm thanh chất lượng cao 24 bit thông qua một chip Wolfson WM8731 audio CODEC (encoder/DECcoder). Con chip này hỗ trợ ngõ vào microphone, ngõ vào và ngõ ra có khả năng điều chỉnh tốc độ lấy mẫu từ 8 kHz đến 96 kHz. Chip WM8731 được kết nối với các pin trên FPGA Cyclone II thông qua giao thức giao tiếp bus nối tiếp I2C. Sơ đồ mạch xử lý âm thanh được thể hiện trên Hình 2.38, và việc gán pin trên FPGA được liệt kê trong bảng Hình 2.39. Thông tin chi tiết về cách thức sử dụng chip WM8731 CODEC được trình bày trên datasheet của nó và ta cũng có thể tìm thấy trên website của nhà sản xuất.



Hình 2.38 Mạch thiết kế của Audio Codec

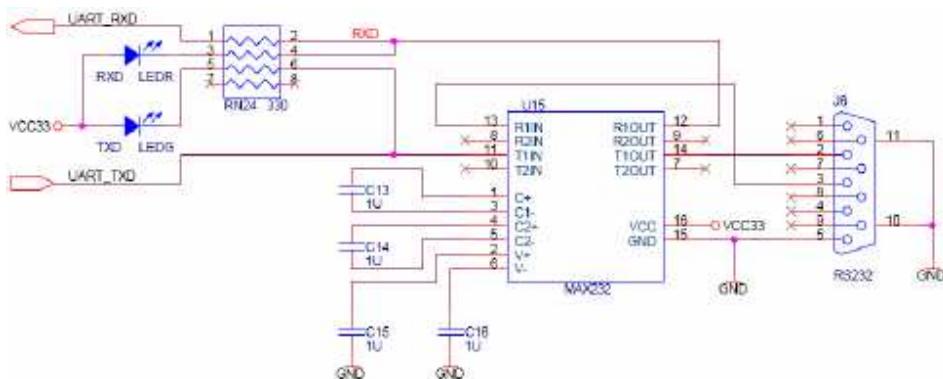
Signal Name	FPGA Pin No.	Description
AUD_ADCLRCK	PIN_C5	Audio CODEC ADC LR Clock
AUD_ADCDAT	PIN_B5	Audio CODEC ADC Data
AUD_DACLRCK	PIN_C6	Audio CODEC DAC LR Clock
AUD_DACDAT	PIN_A4	Audio CODEC DAC Data
AUD_XCK	PIN_A5	Audio CODEC Chip Clock
AUD_BCLK	PIN_B4	Audio CODEC Bit-Stream Clock
I2C_SCLK	PIN_A6	I2C Data
I2C_SDAT	PIN_B6	I2C Clock

Hình 2.39 Mapped pins giữa Audio Codec và FPGA

2.4.9 Cổng nối tiếp RS-232

Bo mạch DE2 sử dụng chip truyền nhận dữ liệu MAX232 và cổng giao tiếp D-SUB 9 pin cho việc giao tiếp RS-232. Thông tin chi tiết về cách thức sử dụng

chip MAX232 cũng như cổng giao tiếp D-SUB được trình bày trên datasheet của nó và ta cũng có thể tìm thấy trên website của nhà sản xuất. Hình 2.40 mô tả mạch kết nối cổng nối tiếp RS-232, và việc gán pin cho FPGA Cyclone II được liệt kê trong Hình 2.41



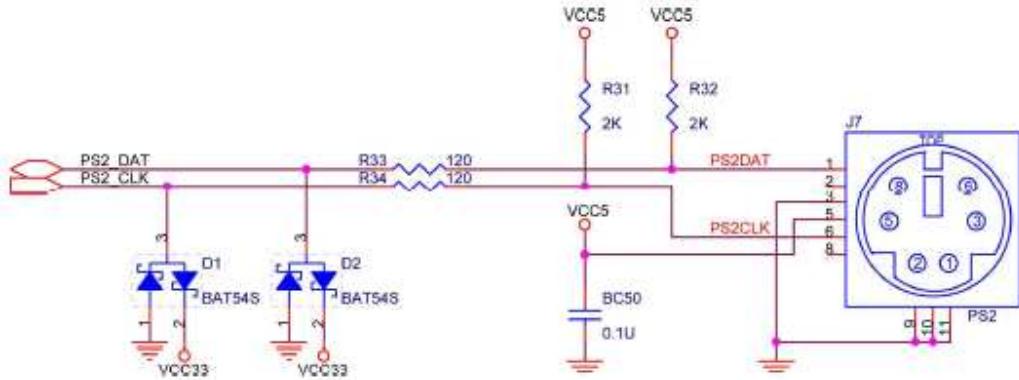
Hình 2.40 Mạch thiết kế giao tiếp giữa RS-232 và FPGA

Signal Name	FPGA Pin No.	Description
UART_RXD	PIN_C25	UART Receiver
UART_TXD	PIN_B25	UART Transmitter

Hình 2.41 Mapped pins giữa RS-232 và FPGA

2.4.10 Cổng nối tiếp PS/2

Bo mạch DE2 có một giao tiếp chuẩn PS/2 và một cổng kết nối PS/2 cho bàn phím hoặc con chuột máy tính. Hình 2.42 mô tả mạch kết nối của PS/2. Hướng dẫn sử dụng chuột PS/2 hay bàn phím PS/2 có thể được tìm thấy trên rất nhiều website. Việc gán pin giữa cổng kết nối PS/2 được chỉ ra trên Hình 2.43.



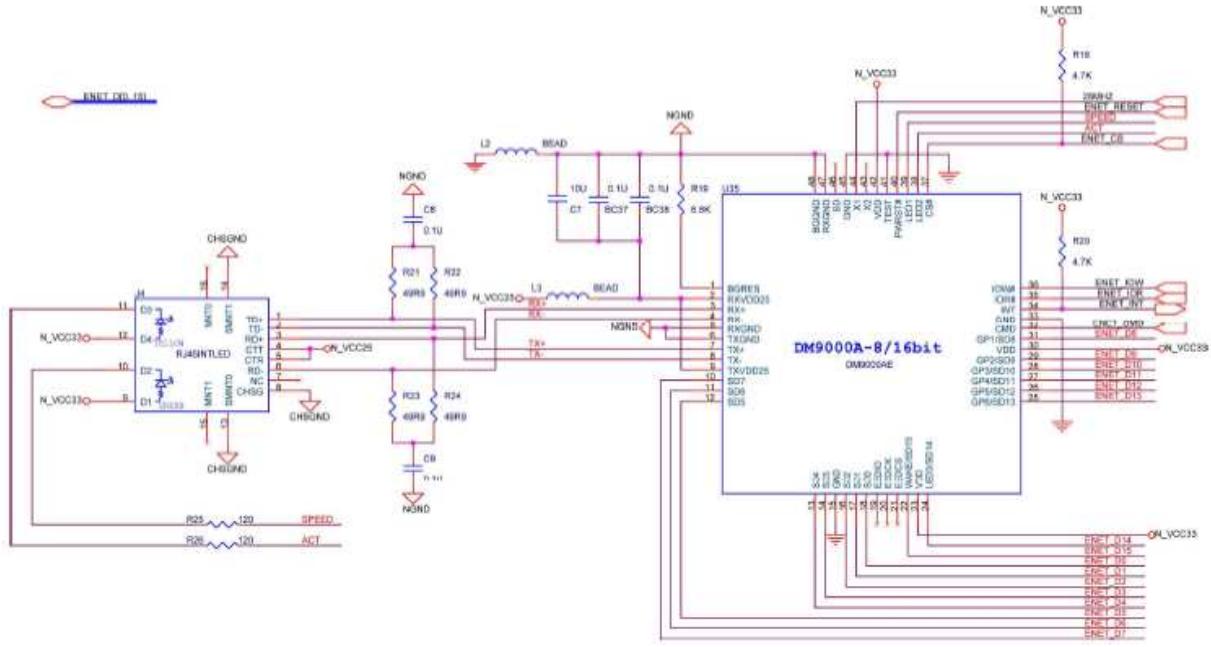
Hình 2.42 Mạch thiết kế giao tiếp giữa cổng PS/2 và FPGA

Signal Name	FPGA Pin No.	Description
PS2_CLK	PIN_D26	PS/2 Clock
PS2_DAT	PIN_C24	PS/2 Data

Hình 2.43 Mapped pins giữa cổng PS/2 và FPGA

2.4.11 Mạch điều khiển mạng Fast Ethernet

Bo mạch DE2 cung cấp việc hỗ trợ kết nối Ethernet thông qua chip điều khiển Davicom DM9000A Fast Ethernet. Chip DM9000A bao gồm một giao tiếp với một vi xử lý thông thường, một SRAM 16 Kbytes, một khối điều khiển truy xuất truyền thông (MAC), và một bộ truyền nhận dữ liệu 10/100M PHY. Hình 2.44 mô tả mạch thực hiện việc giao tiếp Fast Ethernet. Hình 2.45 liệt kê việc gán pin giữa chip DM9000A và FPGA Cyclone II. Thông tin chi tiết về cách thức sử dụng chip DM9000A được trình bày trên datasheet của nó và ta cũng có thể tìm thấy trên website của nhà sản xuất.



Hình 2.44 Mạch thiết kế giao tiếp giữa DM9000A và FPGA

Signal Name	FPGA Pin No.	Description
ENET_DATA[0]	PIN_D17	DM9000A DATA[0]
ENET_DATA[1]	PIN_C17	DM9000A DATA[1]
ENET_DATA[2]	PIN_B18	DM9000A DATA[2]
ENET_DATA[3]	PIN_A18	DM9000A DATA[3]
ENET_DATA[4]	PIN_B17	DM9000A DATA[4]
ENET_DATA[5]	PIN_A17	DM9000A DATA[5]
ENET_DATA[6]	PIN_B16	DM9000A DATA[6]
ENET_DATA[7]	PIN_B15	DM9000A DATA[7]
ENET_DATA[8]	PIN_B20	DM9000A DATA[8]
ENET_DATA[9]	PIN_A20	DM9000A DATA[9]
ENET_DATA[10]	PIN_C19	DM9000A DATA[10]
ENET_DATA[11]	PIN_D19	DM9000A DATA[11]
ENET_DATA[12]	PIN_B19	DM9000A DATA[12]

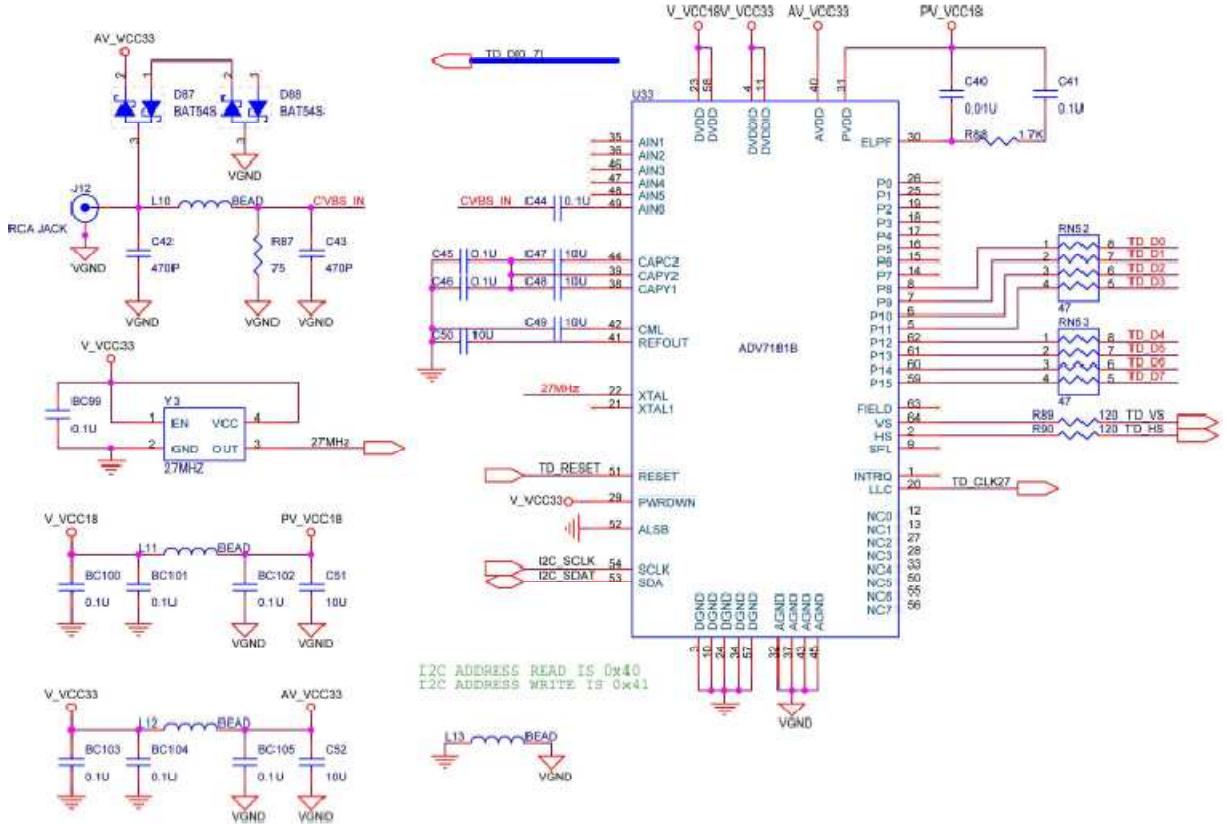
ENET_DATA[13]	PIN_A19	DM9000A DATA[13]
ENET_DATA[14]	PIN_E18	DM9000A DATA[14]
ENET_DATA[15]	PIN_D18	DM9000A DATA[15]
ENET_CLK	PIN_B24	DM9000A Clock 25 MHz
ENET_CMD	PIN_A21	DM9000A Command/Data Select, 0 = Command, 1 = Data
ENET_CS_N	PIN_A23	DM9000A Chip Select
ENET_INT	PIN_B21	DM9000A Interrupt
ENET_RD_N	PIN_A22	DM9000A Read
ENET_WR_N	PIN_B22	DM9000A Write
ENET_RST_N	PIN_B23	DM9000A Reset

Hình 2.45 Mapped pins giữa DM9000A và FPGA

2.4.12 TV Decoder

Bo mạch DE2 được trang bị một chip mã hóa thiết bị tương tự TV ADV7181. Chip ADV7181 là một mạch mã hóa video tích hợp có chức năng dò tìm và chuyển đổi một tín hiệu analog (tương tự) truyền hình có giải nền chuẩn (NTSC, PAL, SECAM) sang tín hiệu dữ liệu digital (số) 4:2:2 có khả năng tương thích với CCIR601/CCIR656 16bit/8bit. Chip ADV7181 tương thích được với nhiều thiết bị Video khác nhau bao gồm đầu DVD, thiết bị truyền thông cũng như camera theo dõi.

Những giá trị của những thanh ghi trong chip mã hóa TV có thể được lập trình thông qua giao tiếp bus nối tiếp I2C được kết nối với FPGA Cyclone II như trong Hình 2.46. Việc gán pin được liệt kê trong Hình 2.47. Thông tin chi tiết về cách thức sử dụng chip ADV7181 được trình bày trên datasheet của nó và ta cũng có thể tìm thấy trên website của nhà sản xuất.



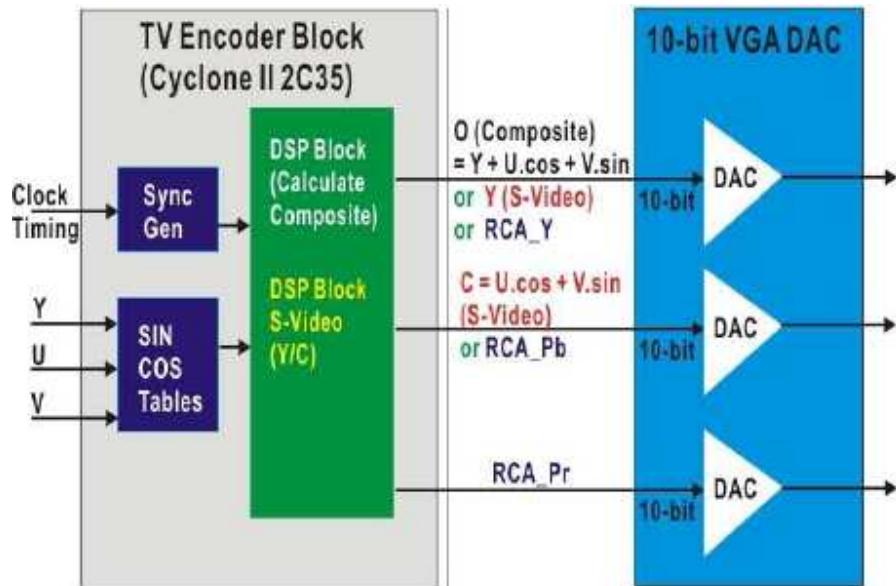
Hình 2.46 Mạch thiết kế giao tiếp giữa ADV7181 và FPGA

Signal Name	FPGA Pin No.	Description
TD_DATA[0]	PIN_J9	TV Decoder Data[0]
TD_DATA[1]	PIN_E8	TV Decoder Data[1]
TD_DATA[2]	PIN_H8	TV Decoder Data[2]
TD_DATA[3]	PIN_H10	TV Decoder Data[3]
TD_DATA[4]	PIN_G9	TV Decoder Data[4]
TD_DATA[5]	PIN_F9	TV Decoder Data[5]
TD_DATA[6]	PIN_D7	TV Decoder Data[6]
TD_DATA[7]	PIN_C7	TV Decoder Data[7]
TD_HS	PIN_D5	TV Decoder H_SYNC
TD_VS	PIN_K9	TV Decoder V_SYNC
TD_CLK27	PIN_C16	TV Decoder Clock Input.
TD_RESET	PIN_C4	TV Decoder Reset
I2C_SCLK	PIN_A6	I2C Data
I2C_SDAT	PIN_B6	I2C Clock

Hình 2.47 Mapped pins giữa ADV7181 và FPGA

2.4.13 TV Encoder

Mặc dù bo mạch DE2 không có chip TV Encoder, nhưng ta có thể sử dụng chip ADV7123 (chip ADC 10 bit tốc độ cao) để thực thi một khối TV encoder với chất lượng cao trong đó phần xử lý tín hiệu số được thực hiện trong FPGA Cyclone II. Hình 2.48 mô tả sơ đồ khối để thực thi một TV encoder

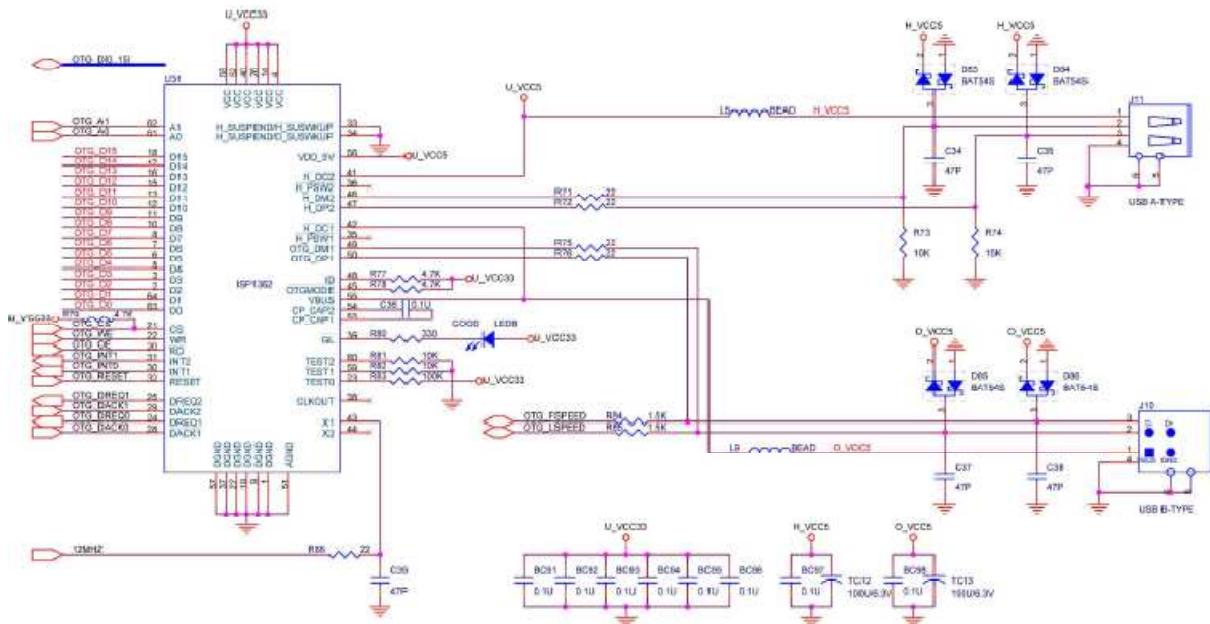


Hình 2.48 TV encoder ADV7123 và FPGA

2.4.14 USB Host and Device

Bo mạch DE2 cung cấp đồng thời hai giao tiếp USB host và device bằng việc sử dụng một chip điều khiển USB ISP1362 của Philips. Bộ điều khiển host và device tương thích với chuẩn giao tiếp USB 2.0, hỗ trợ việc truyền dữ liệu với tốc độ cao (12 Mbit/s) và tốc độ thấp (1.5 Mbit/s). Hình 2.49 mô tả sơ đồ mạch kết nối USB. Hình 2.50 liệt kê việc gán pin kết nối từ chip ISP1362 đến FPGA Cyclone II.

Thông tin chi tiết về cách thức sử dụng chip ISP1362 được trình bày trên datasheet của nó và ta cũng có thể tìm thấy trên website của nhà sản xuất. Phần thách thức nhất của một ứng dụng USB đó là luôn cần một phần mềm điều khiển.



Hình 2.49 Mạch thiết kế giao tiếp USB giữa chip ISP1362 và FPGA

Signal Name	FPGA Pin No.	Description
OTG_ADDR[0]	PIN_K7	ISP1362 Address[0]
OTG_ADDR[1]	PIN_F2	ISP1362 Address[1]
OTG_DATA[0]	PIN_F4	ISP1362 Data[0]
OTG_DATA[1]	PIN_D2	ISP1362 Data[1]
OTG_DATA[2]	PIN_D1	ISP1362 Data[2]
OTG_DATA[3]	PIN_F7	ISP1362 Data[3]
OTG_DATA[4]	PIN_J5	ISP1362 Data[4]
OTG_DATA[5]	PIN_J8	ISP1362 Data[5]
OTG_DATA[6]	PIN_J7	ISP1362 Data[6]
OTG_DATA[7]	PIN_H6	ISP1362 Data[7]
OTG_DATA[8]	PIN_E2	ISP1362 Data[8]
OTG_DATA[9]	PIN_E1	ISP1362 Data[9]
OTG_DATA[10]	PIN_K6	ISP1362 Data[10]
OTG_DATA[11]	PIN_K5	ISP1362 Data[11]
OTG_DATA[12]	PIN_G4	ISP1362 Data[12]
OTG_DATA[13]	PIN_G3	ISP1362 Data[13]

OTG_DATA[14]	PIN_J6	ISP1362 Data[14]
OTG_DATA[15]	PIN_K8	ISP1362 Data[15]
OTG_CS_N	PIN_F1	ISP1362 Chip Select
OTG_RD_N	PIN_G2	ISP1362 Read
OTG_WR_N	PIN_G1	ISP1362 Write
OTG_RST_N	PIN_G5	ISP1362 Reset
OTG_INT0	PIN_B3	ISP1362 Interrupt 0
OTG_INT1	PIN_C3	ISP1362 Interrupt 1
OTG_DACK0_N	PIN_C2	ISP1362 DMA Acknowledge 0
OTG_DACK1_N	PIN_B2	ISP1362 DMA Acknowledge 1
OTG_DREQ0	PIN_F6	ISP1362 DMA Request 0
OTG_DREQ1	PIN_E5	ISP1362 DMA Request 1
OTG_FSPEED	PIN_F3	USB Full Speed, 0 = Enable, Z = Disable
OTG_LSPEED	PIN_G6	USB Low Speed, 0 = Enable, Z = Disable

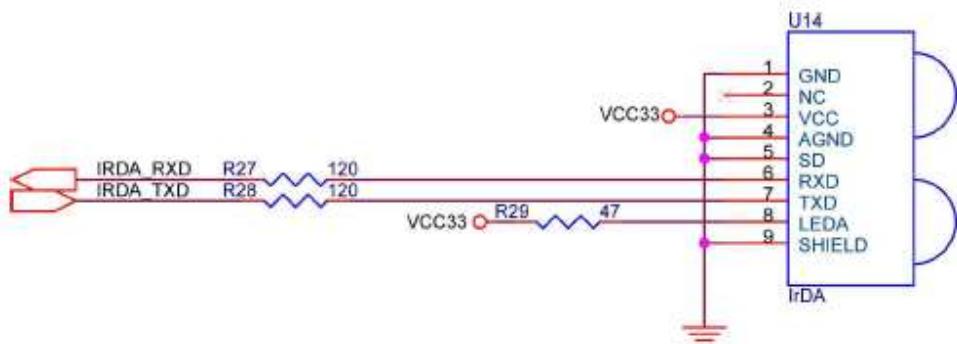
Hình 2.50 Mapped pins giữa ISP1362 và FPGA

2.4.15 Cổng hồng ngoại

Bo mạch DE2 cung cấp một giao tiếp truyền thông đơn giản không dây sử dụng bộ truyền phát hồng ngoại công suất thấp Agilent HSDL-3201. Thông tin chi tiết về cách thức sử dụng chip Agilent HSDL-3201 được trình bày trên datasheet của nó và ta cũng có thể tìm thấy trên website của nhà sản xuất. Chú ý rằng tốc độ truyền cao nhất được hỗ trợ là 115.2 Kbit/s và cả hai phía nhận và thu phải sử dụng tốc độ truyền như nhau. Hình 2.51 mô tả mạch kết nối của giao tiếp hồng ngoại. Để hiểu rõ thêm về cách thức truyền và nhận dữ liệu dung cổng hồng ngoại, ta có thể vào website sau để tham khảo:

http://techtrain.microchip.com/webseminars/documents/IrDA_BW.pdf.

Việc kết nối pin giữa cổng hồng ngoại và FPGA Cyclone II được liệt kê trong Hình 2.52



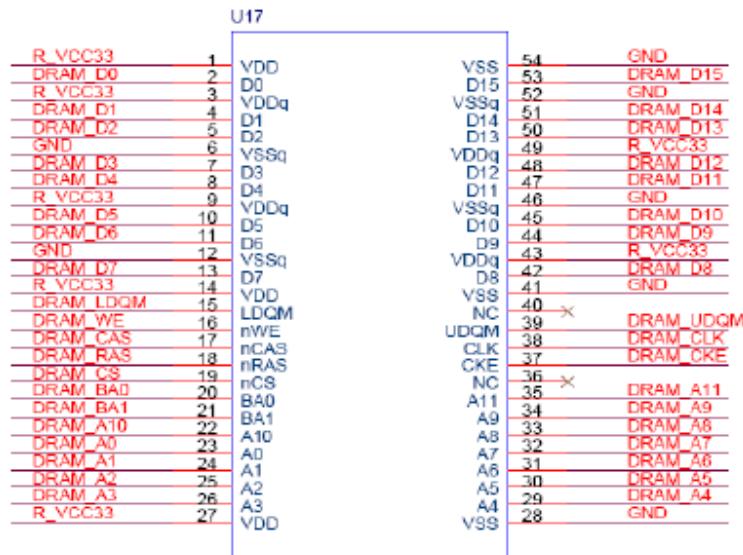
Hình 2.51 Mạch giao tiếp giữa cổng hồng ngoại và FPGA

Signal Name	FPGA Pin No.	Description
IRDA_TXD	PIN_AE24	IRDA Transmitter
IRDA_RXD	PIN_AE25	IRDA Receiver

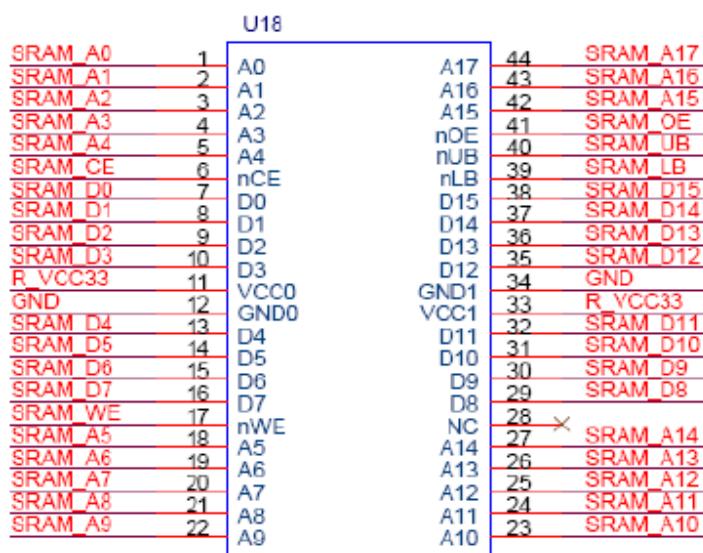
Hình 2.52 Mapped pins giữa cổng hồng ngoại và FPGA

2.4.16 Bộ nhớ SDRAM/SRAM/Flash

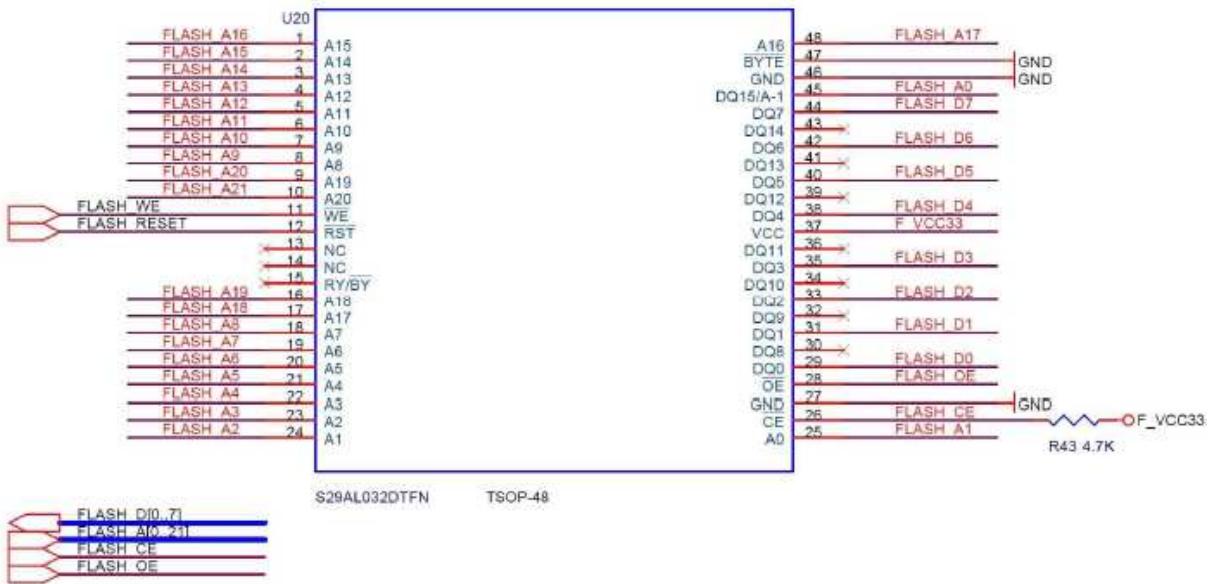
Bo mạch DE2 cung cấp một bộ nhớ SDRAM 8 Mbyte, một bộ nhớ SRAM 512 Kbyte và một bộ nhớ Flash 4 Mbyte (1 Mbyte trên một số board mạch). Hình 2.53, Hình 2.54 và Hình 2.55 mô tả sơ đồ mạch kết nối của mỗi loại bộ nhớ. Việc gán pin kết nối giữa FPGA Cyclone II với mỗi loại bộ nhớ được liệt kê trong các Hình 2.56, Hình 2.57 và Hình 2.58. Datasheet của mỗi loại bộ nhớ có thể tìm thấy dễ dàng trên các website.



Hình 2.53 Mạch giao tiếp thiết kế giữa DRAM và FPGA



Hình 2.54 Mạch giao tiếp thiết kế giữa SRAM và FPGA



Hình 2.55 Mạch giao tiếp thiết kế giữa FLASH và FPGA

Signal Name	FPGA Pin No.	Description
DRAM_ADDR[0]	PIN_T6	SDRAM Address[0]
DRAM_ADDR[1]	PIN_V4	SDRAM Address[1]
DRAM_ADDR[2]	PIN_V3	SDRAM Address[2]
DRAM_ADDR[3]	PIN_W2	SDRAM Address[3]
DRAM_ADDR[4]	PIN_W1	SDRAM Address[4]
DRAM_ADDR[5]	PIN_U6	SDRAM Address[5]
DRAM_ADDR[6]	PIN_U7	SDRAM Address[6]
DRAM_ADDR[7]	PIN_U5	SDRAM Address[7]
DRAM_ADDR[8]	PIN_W4	SDRAM Address[8]
DRAM_ADDR[9]	PIN_W3	SDRAM Address[9]
DRAM_ADDR[10]	PIN_Y1	SDRAM Address[10]
DRAM_ADDR[11]	PIN_V5	SDRAM Address[11]
DRAM_DQ[0]	PIN_V6	SDRAM Data[0]
DRAM_DQ[1]	PIN_AA2	SDRAM Data[1]
DRAM_DQ[2]	PIN_AA1	SDRAM Data[2]

DRAM_DQ[3]	PIN_Y3	SDRAM Data[3]
DRAM_DQ[4]	PIN_Y4	SDRAM Data[4]
DRAM_DQ[5]	PIN_R8	SDRAM Data[5]
DRAM_DQ[6]	PIN_T8	SDRAM Data[6]
DRAM_DQ[7]	PIN_V7	SDRAM Data[7]
DRAM_DQ[8]	PIN_W6	SDRAM Data[8]
DRAM_DQ[9]	PIN_AB2	SDRAM Data[9]
DRAM_DQ[10]	PIN_AB1	SDRAM Data[10]
DRAM_DQ[11]	PIN_AA4	SDRAM Data[11]
DRAM_DQ[12]	PIN_AA3	SDRAM Data[12]
DRAM_DQ[13]	PIN_AC2	SDRAM Data[13]
DRAM_DQ[14]	PIN_AC1	SDRAM Data[14]
DRAM_DQ[15]	PIN_AA5	SDRAM Data[15]
DRAM_BA_0	PIN_AE2	SDRAM Bank Address[0]
DRAM_BA_1	PIN_AE3	SDRAM Bank Address[1]
DRAM_LDQM	PIN_AD2	SDRAM Low-byte Data Mask
DRAM_UDQM	PIN_Y5	SDRAM High-byte Data Mask
DRAM_RAS_N	PIN_AB4	SDRAM Row Address Strobe
DRAM_CAS_N	PIN_AB3	SDRAM Column Address Strobe
DRAM_CKE	PIN_AA6	SDRAM Clock Enable
DRAM_CLK	PIN_AA7	SDRAM Clock
DRAM_WE_N	PIN_AD3	SDRAM Write Enable
DRAM_CS_N	PIN_AC3	SDRAM Chip Select

Hình 2.56 Mapped pins giữa SDRAM và FPGA

Signal Name	FPGA Pin No.	Description
SRAM_ADDR[0]	PIN_AE4	SRAM Address[0]
SRAM_ADDR[1]	PIN_AF4	SRAM Address[1]
SRAM_ADDR[2]	PIN_AC5	SRAM Address[2]
SRAM_ADDR[3]	PIN_AC6	SRAM Address[3]
SRAM_ADDR[4]	PIN_AD4	SRAM Address[4]
SRAM_ADDR[5]	PIN_AD5	SRAM Address[5]
SRAM_ADDR[6]	PIN_AE5	SRAM Address[6]
SRAM_ADDR[7]	PIN_AF5	SRAM Address[7]
SRAM_ADDR[8]	PIN_AD6	SRAM Address[8]
SRAM_ADDR[9]	PIN_AD7	SRAM Address[9]
SRAM_ADDR[10]	PIN_V10	SRAM Address[10]
SRAM_ADDR[11]	PIN_V9	SRAM Address[11]
SRAM_ADDR[12]	PIN_AC7	SRAM Address[12]
SRAM_ADDR[13]	PIN_W8	SRAM Address[13]
SRAM_ADDR[14]	PIN_W10	SRAM Address[14]
SRAM_ADDR[15]	PIN_Y10	SRAM Address[15]
SRAM_ADDR[16]	PIN_AB8	SRAM Address[16]
SRAM_ADDR[17]	PIN_AC8	SRAM Address[17]
SRAM_DQ[0]	PIN_AD8	SRAM Data[0]
SRAM_DQ[1]	PIN_AE6	SRAM Data[1]
SRAM_DQ[2]	PIN_AF6	SRAM Data[2]
SRAM_DQ[3]	PIN_AA9	SRAM Data[3]
SRAM_DQ[4]	PIN_AA10	SRAM Data[4]
SRAM_DQ[5]	PIN_AB10	SRAM Data[5]
SRAM_DQ[6]	PIN_AA11	SRAM Data[6]

SRAM_DQ[7]	PIN_Y11	SRAM Data[7]
SRAM_DQ[8]	PIN_AE7	SRAM Data[8]
SRAM_DQ[9]	PIN_AF7	SRAM Data[9]
SRAM_DQ[10]	PIN_AE8	SRAM Data[10]
SRAM_DQ[11]	PIN_AF8	SRAM Data[11]
SRAM_DQ[12]	PIN_W11	SRAM Data[12]
SRAM_DQ[13]	PIN_W12	SRAM Data[13]
SRAM_DQ[14]	PIN_AC9	SRAM Data[14]
SRAM_DQ[15]	PIN_AC10	SRAM Data[15]
SRAM_WE_N	PIN_AE10	SRAM Write Enable
SRAM_OE_N	PIN_AD10	SRAM Output Enable
SRAM_UB_N	PIN_AF9	SRAM High-byte Data Mask
SRAM_LB_N	PIN_AE9	SRAM Low-byte Data Mask
SRAM_CE_N	PIN_AC11	SRAM Chip Enable

Hình 2.57 Mapped pins giữa SRAM và FPGA

Signal Name	FPGA Pin No.	Description
FL_ADDR[0]	PIN_AC18	FLASH Address[0]
FL_ADDR[1]	PIN_AB18	FLASH Address[1]
FL_ADDR[2]	PIN_AE19	FLASH Address[2]
FL_ADDR[3]	PIN_AF19	FLASH Address[3]
FL_ADDR[4]	PIN_AE18	FLASH Address[4]
FL_ADDR[5]	PIN_AF18	FLASH Address[5]
FL_ADDR[6]	PIN_Y16	FLASH Address[6]
FL_ADDR[7]	PIN_AA16	FLASH Address[7]
FL_ADDR[8]	PIN_AD17	FLASH Address[8]
FL_ADDR[9]	PIN_AC17	FLASH Address[9]

FL_ADDR[10]	PIN_AE17	FLASH Address[10]
FL_ADDR[11]	PIN_AF17	FLASH Address[11]
FL_ADDR[12]	PIN_W16	FLASH Address[12]
FL_ADDR[13]	PIN_W15	FLASH Address[13]
FL_ADDR[14]	PIN_AC16	FLASH Address[14]
FL_ADDR[15]	PIN_AD16	FLASH Address[15]
FL_ADDR[16]	PIN_AE16	FLASH Address[16]
FL_ADDR[17]	PIN_AC15	FLASH Address[17]
FL_ADDR[18]	PIN_AB15	FLASH Address[18]
FL_ADDR[19]	PIN_AA15	FLASH Address[19]
FL_ADDR[20]	PIN_Y15	FLASH Address[20]
FL_ADDR[21]	PIN_Y14	FLASH Address[21]
FL_DQ[0]	PIN_AD19	FLASH Data[0]
FL_DQ[1]	PIN_AC19	FLASH Data[1]
FL_DQ[2]	PIN_AF20	FLASH Data[2]
FL_DQ[3]	PIN_AE20	FLASH Data[3]
FL_DQ[4]	PIN_AB20	FLASH Data[4]
FL_DQ[5]	PIN_AC20	FLASH Data[5]
FL_DQ[6]	PIN_AF21	FLASH Data[6]
FL_DQ[7]	PIN_AE21	FLASH Data[7]
FL_CE_N	PIN_V17	FLASH Chip Enable
FL_OE_N	PIN_W17	FLASH Output Enable
FL_RST_N	PIN_AA18	FLASH Reset
FL_WE_N	PIN_AA17	FLASH Write Enable

Hình 2.58 Mapped pins giữa FLASH và FPGA

Chương 3. Hướng dẫn cài đặt và sử dụng phần mềm Control Panel để điều khiển kit DE2

3.1 Hướng dẫn cài đặt Control Panel điều khiển Kit DE2

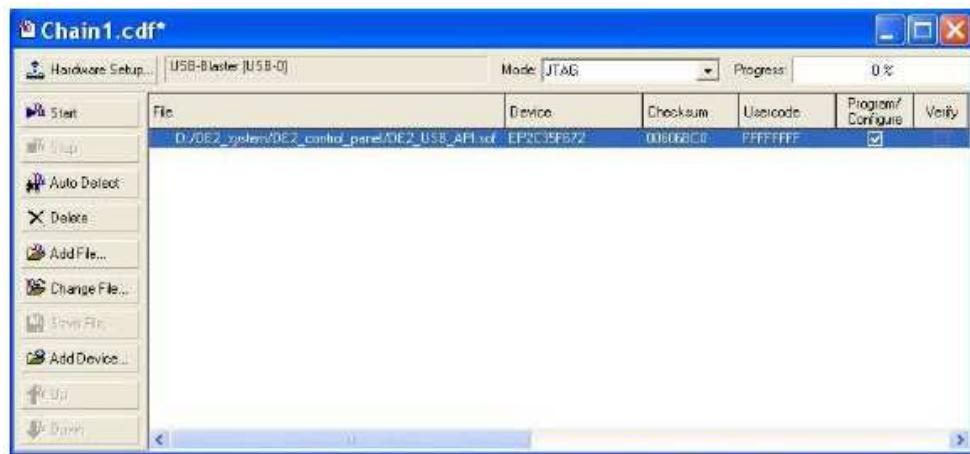
Trình ứng dụng Control Panel đi kèm với Kit DE2 cho phép người sử dụng điều khiển tất cả những linh kiện trên Kit DE2 thông qua cổng USB từ một computer chứa trình ứng dụng Control Panel trên. Trong phần này, ta sẽ mô tả một số chức năng cơ bản của trình ứng dụng Control Panel, sau đó ta sẽ mô tả cấu trúc của nó dưới dạng sơ đồ khôi, và cuối cùng là mô tả những khả năng của nó.

Để cài đặt trình điều khiển Control Panel, Altera cung cấp cho người sử dụng 2 file sau đây:

- i. DE2_USB_API.sof
- ii. DE2_control_panel.exe

Để kích hoạt trình ứng dụng Control Panel, ta cần thực hiện những bước sau:

- 1- Kết nối cáp USB đến cổng USB Blaster. Cung cấp nguồn 9V. Bật nguồn lên vị trí ON.
- 2- Chuyển switch **RUN/PROG** ở vị trí **RUN**.
- 3- Mở phần mềm Quartus II .
- 4- Chọn **Tools → Programmer**, ta sẽ được hình sau:



Hình 3.1 Giao diện cho việc cấu hình thiết kế lên FPGA

- 5- Nhấn chọn **Add File**, chỉ đường dẫn đến file cấu hình **DE2_USB_API.sof**
- 6- Nhấn chọn ở cột **Program/Configure**.
- 7- Nhấn **Start** để nạp file cấu hình DE2_USB_API.sof xuống FPGA.

Để thực thi file **DE2_control_panel.exe** trên computer bằng cách nhấp đúp lên biểu tượng của file. Sau khi chạy xong, ta sẽ thấy một giao diện như sau:

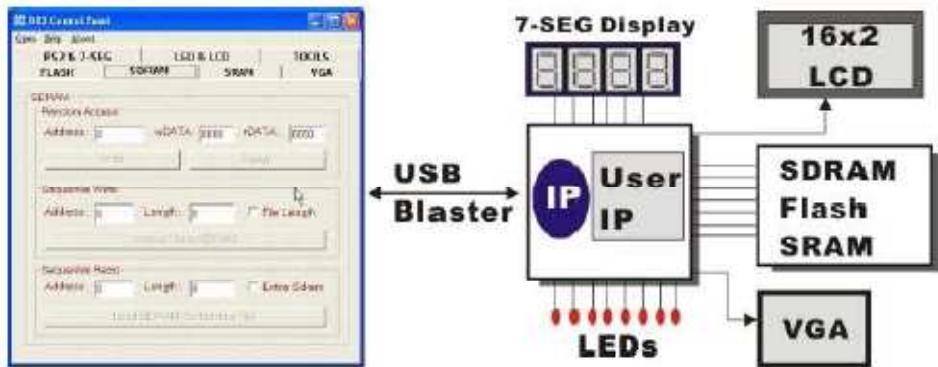


Hình 3.2 Giao diện Control Panel

Trên Control Panel, ta mở cổng USB bằng cách nhấn : **Open → Open USB Port 0**. Trình ứng dụng DE2 Control Panel sẽ liệt kê tất cả những cổng USB kết nối với Kit DE2. Trình ứng dụng này có thể điều khiển 4 Kit DE2 cùng một lúc thông qua USB links. Chú ý, trên trình ứng dụng DE2 Control Panel sẽ xuất hiện cổng USB cho đến khi ta đóng cổng này lại; khi cổng USB chưa được đóng lại thì ta không thể dùng Quartus II để nạp file câu lệnh DE2_USB_API.sof xuống FPGA.

Trình ứng dụng DE2 Control Panel bây giờ đã sẵn sàng cho việc điều khiển; tiến hành thí nghiệm bằng cách thiết lập giá trị cho một vài LEDs bảy đoạn và quan sát kết quả trên Kit DE2.

Hoạt động của trình ứng dụng DE2 Control Panel được minh họa như hình dưới đây:



Hình 3.3 Giao tiếp giữa Control Panel và các thiết bị ngoại vi trên FPGA

Trong đó IP thực thi chức năng điều khiển. IP này được thiết kế trên FPGA. Nó liên lạc với trình ứng dụng DE2 Control Panel trên computer thông qua cổng USB Blaster. Giao diện đồ họa được sử dụng để truyền lệnh xuống mạch điều khiển. Công việc của IP là quản lý tất cả những yêu cầu và thực thi việc truyền dữ liệu giữa máy tính và Kit DE2.

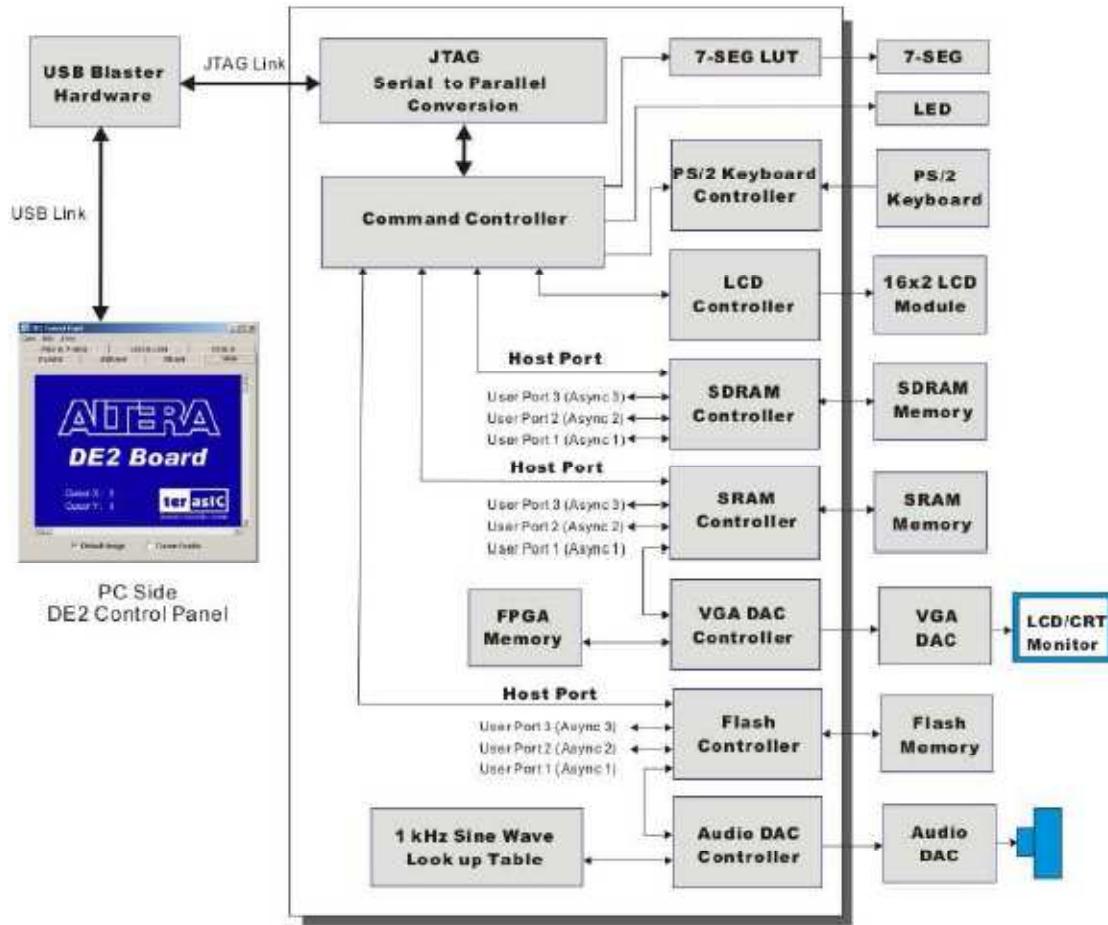
Trình ứng dụng DE2 Control Panel có thể được dùng để thay đổi giá trị hiển thị trên LEDs bảy đoạn, bật LEDs, giao tiếp với bàn phím PS/2, đọc và ghi SRAM, Flash Memory và SDRAM, tải hình ảnh lên màn hình VGA, tải nhạc vào bộ nhớ và nghe nhạc qua audio DAC. Đặc tính đọc và ghi một byte hay toàn bộ file từ hay đến Flash Memory cho phép người sử dụng phát triển nhiều ứng dụng multimedia (Flash Audio Player, Flash Picture Viewer) mà không cần quan tâm đến cách tạo ra một Flash Memory Programmer.

3.2 Tổng quan về cấu trúc và hoạt động của Control Panel

DE2 Control Panel giao tiếp với một module điện tử (thiết kế bằng ngôn ngữ phần cứng như Verilog hay VHDL) đã được biên dịch và nạp vào Chip Cyclone II FPGA. Vì Altera cung cấp cho chúng ta những module trên dưới dạng ngôn ngữ

Verilog, do đó ta hoàn toàn có thể dựa vào đó để chỉnh sửa và thay đổi chức năng hoạt động của Control Panel, tất nhiên ta phải nắm vững ngôn ngữ Verilog.

Hình dưới mô tả cấu trúc của Control Panel. Mỗi thiết bị ngõ vào hoặc ngõ ra đều được điều khiển bởi một trình điều khiển được thiết kế bằng ngôn ngữ Verilog và được nạp lên FPGA (DE2_USB_API.sof). Việc giao tiếp giữa những trình điều khiển này với PC được thực hiện thông qua USB Blaster. Nhìn trên hình ta thấy khối Command Controller, chức năng của khói này là biên dịch những lệnh nhận từ PC và thực thi những tác vụ thích hợp. Những khói trình điều khiển cho SDRAM, SRAM và Flash Memory có ba chân “user-selectable” bắt đồng bộ cùng với một chân “Host” để kết nối đến khói Command Controller. Kết nối giữa trình điều khiển VGA DAC đến bộ nhớ của FPGA cho phép hiển thị một hình ảnh mặc định (Tiger) đã lưu sẵn trong bộ nhớ của FPGA. Kết nối giữa trình điều khiển Audio DAC tới một “lookup table” trên FPGA để tạo ra một tín hiệu âm thanh để test có tần số 1KHz.



Hình 3.4 Sơ đồ khái giao tiếp giữa Control Panel và các thiết bị ngoại vi

Ta có thể kết nối những module của mình đến một trong những “User Ports” của trình điều khiển SRAM/SDRAM/Flash memory, sau đó ta có thể tải dữ liệu nhị phân vào trong SRAM/SDRAM/Flash memory. Khi dữ liệu đã được tải vào trong SRAM/SDRAM/Flash memory, ta có thể thiết lập lại cấu hình cho các khái trình điều khiển SRAM/SDRAM/Flash memory để những khái này có thể đọc/ghi dữ liệu của SRAM/SDRAM/Flash memory thông qua “User Ports” (Để thiết lập lại cấu hình cho những khái này đòi hỏi ta phải có kiến thức về Verilog).

3.3 Hướng dẫn sử dụng Control Panel

3.3.1 Điều khiển LEDs, LEDs bảy đoạn, LCD

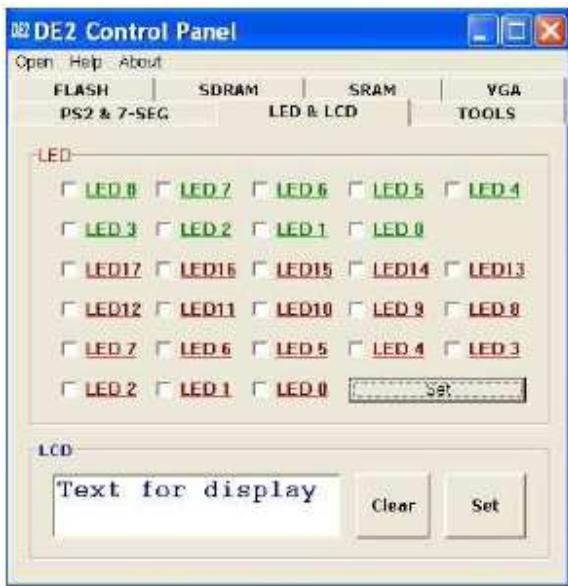
Chức năng cơ bản của Control Panel là cho phép thiết lập những giá trị hiển thị trên LEDs, LEDs bảy đoạn và LCD.

Trên cửa sổ Control Panel ở hình dưới đây, chọn tab **PS2 & 7-SEG** những giá trị được thể hiện trên LEDs bảy đoạn (HEX7-HEX0) có thể được nhập vào từ những ô tương ứng. Để những giá trị vừa nhập trên hiển thị trên LED bảy đoạn bằng cách nhấn vào nút **Set**. Để hiển thị các ký tự trên LCD, ta có thể sử dụng bàn phím được kết nối với Kit DE2 thông qua cổng PS/2, sau đó ta có thể nhập ký tự từ bàn phím này để hiển thị lên LCD.



Hình 3.5 Giao diện Control Panel điều khiển LEDs 7 đoạn

- 1- Chọn tab LED & LCD ta sẽ có cửa sổ như hình dưới



Hình 3.6 Giao diện Control Panel điều khiển LEDs đơn

- 2- Ta có thể bật sáng một LED bất kì bằng cách chọn vào LED đó sau đó nhấn nút **Set**. Ta cũng có thể hiển thị những ký tự lên LCD bằng cách nhập những ký tự đó lên hộp LCD trên cửa sổ Control Panel, sau đó nhấn nút **Set**.

Việc thiết lập giá trị hiển thị lên những thiết bị hiển thị đơn giản trên thì cũng không cần đến một trình ứng dụng như Control Panel, tuy nhiên với trình ứng dụng Control Panel, ta sẽ có một công cụ đơn giản để kiểm tra hoạt động của từng linh kiện trên Kit DE2, đồng thời nó cũng được dùng để debug trong quá trình thiết kế.

3.3.2 Truy xuất bộ nhớ SDRAM/SRAM

Ta có thể truy xuất dữ liệu như đọc dữ liệu từ bộ nhớ hay ghi dữ liệu vào bộ nhớ (SDRAM/SRAM) trên Kit DE2. Dưới đây ta sẽ mô tả cách thức truy xuất SDRAM thông qua Control Panel (cách thức này cũng tương tự đối với SRAM).

- Chọn tab **SDRAM**, ta sẽ nhìn thấy một cửa sổ như hình dưới



Hình 3.7 Giao diện Control Panel điều khiển SDRAM

- ✚ Với Control Panel, ta có thể truy xuất một từ 16 bits hoặc truy xuất một dãy những từ ở những ô nhớ liên tiếp.
- ✚ Để ghi một từ 16 bits (Random access), ta nhập địa chỉ ô nhớ cần ghi vào ô **Address**, tiếp đến nhập data cần ghi vào ô **wDATA**, sau đó nhấn nút **Write**. Trong hình trên, ta muốn ghi giá trị dưới dạng hexadecimal 6CA vào bộ nhớ ở địa chỉ 200. Sau khi ta nhấn nút **Write** thì giá trị 6CA đã được ghi vào ô nhớ có địa chỉ 200 của bộ nhớ.
- ✚ Để đọc một từ 16 bits (Random access), ta nhập địa chỉ ô nhớ cần đọc ra, sau đó nhấn nút **Read**. Trong hình trên, ta muốn đọc giá trị từ ô nhớ có địa chỉ 200. Sau khi ta nhấn nút **Read** thì giá trị 6CA (giả sử giá trị này đã được ghi vào từ thao tác ghi) sẽ hiển thị trên ô **rDATA**.

■ Để ghi một chuỗi những kí tự liên tiếp hay nội dung của một file vào SDRAM (Sequential Write), ta thực hiện những thao tác sau:

1. Nhập địa chỉ của ô nhớ bắt đầu để lưu trữ nội dung của một file vào ô **Address**.
2. Nhập chiều dài của nội dung file cần ghi vào hộp **Length**. Nếu ta muốn ghi toàn bộ file vào trong SDRAM ta chỉ việc chọn cho ô **File Length** mà không cần Nhập chiều dài của nội dung file cần ghi vào ô **Length**.
3. Chỉ đường dẫn của file data cần ghi bằng cách nhấn **Write a File to SDRAM** và xác định đường dẫn cho file đó.

■ Control Panel cũng hỗ trợ ghi file có nội dung được định dạng Hexa. File định dạng Hexa là file chứa những kí tự ASCII để biểu diễn những giá trị Hexadecimal. Giả sử, ta có một file chứa dòng những kí tự ASCII sau :

0123456789ABCDEF

■ Dòng kí tự trên xác định bốn giá trị 16-bit : 0123, 4567, 89AB, CDEF. Những giá trị này sẽ được ghi theo tuần tự vào SDRAM.

■ Để đọc một chuỗi những kí tự liên tiếp hay nội dung của một file từ SDRAM (Sequential Read), ta thực hiện những thao tác sau:

1. Nhập địa chỉ của ô nhớ bắt đầu của chuỗi ô nhớ mà ta muốn đọc ra vào ô **Address**.
2. Nhập chiều dài của nội dung file cần đọc (số byte) vào hộp **Length**. Nếu ta muốn đọc toàn bộ nội dung trong SDRAM (tất cả

8 Mbytes) ta chỉ việc chọn cho ô **Entire SDRAM** mà không cần nhập chiều dài của nội dung file cần đọc vào ô **Length**.

3. Chỉ đường dẫn của file để thể hiện những giá trị muốn đọc từ SDRAM ra bằng cách nhấn **Load SDRAM Content to a File** và xác định đường dẫn cho file đó. Sau đó ta có thể xem nội dung vừa đọc ra từ SDRAM trên file này.

3.3.3 Truy xuất bộ nhớ Flash (Flash memory)

Control Panel cho phép ta truy xuất (ghi/đọc) dữ liệu của Flash memory trên Kit DE2. Ta có thể sử dụng Control Panel để thực hiện các thao tác sau:

- ✚ Xóa toàn bộ bộ nhớ Flash
- ✚ Ghi một byte lên bộ nhớ Flash
- ✚ Đọc một byte từ bộ nhớ Flash
- ✚ Ghi một file binary lên bộ nhớ Flash
- ✚ Đọc nội dung của bộ nhớ Flash ra một file.

Chú ý: Dưới đây là những đặc tính của bộ nhớ Flash

- ✚ Dung lượng bộ nhớ Flash là 4Mx8bits
- ✚ Ta phải xóa toàn bộ bộ nhớ Flash trước khi ghi dữ liệu lên nó. (Số lần xóa bị giới hạn bởi nhà sản xuất, do đó người sử dụng phải nhận thức được số lần có thể xóa bộ nhớ Flash mà không khiến nó bị hỏng).
- ✚ Thời gian để xóa toàn bộ bộ nhớ Flash là khoảng 20s. Do đó không được đóng (ngắt nguồn) Kit DE2 trong suốt quá trình xóa bộ nhớ Flash.

Để truy xuất bộ nhớ Flash thông qua Control Panel, ta chọn tab FLASH trên cửa sổ Control Panel, một cửa sổ như hình dưới sẽ xuất hiện



Hình 3.8 Giao diện Control Panel điều khiển FLASH

Để ghi một byte dữ liệu vào một vị trí ngẫu nhiên trên bộ nhớ Flash, ta cần thực hiện những bước sau:

1. Nhấn nút Chip Erase. Quá trình xóa được thực hiện trong khoảng 20s.
2. Nhập địa chỉ của ô nhớ mà ta muốn ghi một byte dữ liệu vào trên ô **Address**.
3. Nhập byte dữ liệu cần ghi vào ô **wDATA**. Sau đó nhấn nút **Write**

Để đọc một byte dữ liệu vào một vị trí ngẫu nhiên trên bộ nhớ Flash, ta cần thực hiện những bước sau:

1. Nhập địa chỉ của ô nhớ mà ta muốn đọc một byte dữ liệu ra trên ô **Address**.
2. Sau đó nhấn nút **Read**. Byte dữ liệu đọc ra sẽ được hiển thị trên ô **rDATA**.

Để ghi một file dữ liệu (Sequential Write) vào trong bộ nhớ Flash, ta cần thực hiện những bước sau:

1. Nhập địa chỉ của ô nhớ bắt đầu để lưu trữ nội dung của một file vào ô **Address**.
2. Nhập chiều dài của nội dung file cần ghi vào hộp **Length**. Nếu ta muốn ghi toàn bộ file vào trong bộ nhớ Flash ta chỉ việc chọn cho ô **File Length** mà không cần nhập chiều dài của nội dung file cần ghi vào ô **Length**.
3. Chỉ đường dẫn của file data cần ghi bằng cách nhấn **Write a File to Flash** và xác định đường dẫn cho file đó.

Để đọc một chuỗi những ký tự liên tiếp hay nội dung của một file từ bộ nhớ Flash (Sequential Read), ta thực hiện những thao tác sau:

1. Nhập địa chỉ của ô nhớ bắt đầu của chuỗi ô nhớ mà ta muốn đọc ra vào ô **Address**.
2. Nhập chiều dài của nội dung file cần đọc (số byte) vào hộp **Length**. Nếu ta muốn đọc toàn bộ nội dung trong bộ nhớ Flash ta chỉ việc chọn cho ô **Entire Flash** mà không cần nhập chiều dài của nội dung file cần đọc vào ô **Length**.
3. Chỉ đường dẫn của file để thể hiện những giá trị muốn đọc từ bộ nhớ Flash ra bằng cách nhấn **Load Flash Content to a File** và xác định đường dẫn cho file đó. Sau đó ta có thể xem nội dung vừa đọc ra từ bộ nhớ Flash trên file này.

3.3.4 TOOLS – Multi-Port SRAM/SDRAM/Flash Controller

Chọn Tab TOOLS trên cửa sổ Control Panel sẽ cho phép ta chọn “User Ports”. Chúng ta sẽ một ví dụ cụ thể bằng việc thực thi một Flash Music Player. Ta tải một file nhạc vào trong Flash memory. “User Port 1” của trình điều khiển Flash memory được dùng để truyền file nhạc đến trình điều khiển Audio DAC và xuất ra output.

Để thực hiện ví dụ trên ta thực thi theo các bước sau:

1. Xóa bộ nhớ Flash . Sau đó ghi file nhạc (music.wav) lên bộ nhớ Flash (Trình bày trong phần 3.3.3)
2. Trên Control Panel, chọn Tab TOOLS, một cửa sổ như hình dưới sẽ xuất hiện.



Hình 3.9 Giao diện Control Panel điều khiển Multi-Ports

3. Chọn cổng Asynchronous 1 cho Flash Multiplexer và nhấn nút **Configure** để kích hoạt cổng trên. Ta cần nhấn nút **Configure** để cho phép kết nối từ Flash Memory đến Asynchronous Port 1 của trình điều khiển Flash.

4. Tắt SW1 (vị trí DOWN) và bật SW0 (vị trí UP).
5. Cắm headphone vào audio output, ta có thể nghe nhạc được phát ra từ khối mạch Audio DAC.

Chú ý rằng, Asynchronous Port 1 được kết nối đến Audio DAC như hình trên. Khi ta chọn Asynchronous Port 1 và nhấn nút **Configure**, trình điều khiển Audio DAC sẽ giao tiếp trực tiếp với bộ nhớ Flash. Trong ví dụ trên, khôi module Verilog AUDIO_DAC có nhiệm vụ đọc nội dung của bộ nhớ Flash và truyền nó đến Chip Audio bên ngoài FPGA.

3.3.5 VGA Display Control

Control Panel cung cấp một công cụ kết hợp với IP cho phép ta hiện thị hình ảnh thông qua cổng ngõ ra VGA. Dưới đây là những bước dùng để minh họa cách thức hiện thị một bức ảnh lên màn hình VGA.

Chọn tab VGA trên cửa sổ Control Panel, một cửa sổ như hình dưới sẽ xuất hiện.



Hình 3.10 Giao diện Control Panel điều khiển VGA

1. Đánh dấu vào hai ô Default Image và Cursor Enable

- Kết nối màn hình VGA đến kit DE2. Ta sẽ nhìn thấy một màn hình mặc định (do ta đã chọn ở ô **Default Image**) như hình mặc định như hình trên. Trên màn hình bao gồm một con trỏ (do ta đã chọn ở ô **Cursor Enable**) mà vị trí của nó được điều khiển bởi thanh cuộn X/Y trên Control Panel.

Hình ảnh xuất hiện ở màn hình trên được lưu trữ trong một vùng nhớ M4K trên Cyclon II FPGA. Nó sẽ được tải đến vùng nhớ M4K dưới định dạng MIF/Hex (Intel) trong suốt quá trình ta thiết lập cấu hình mặc định cho nó.

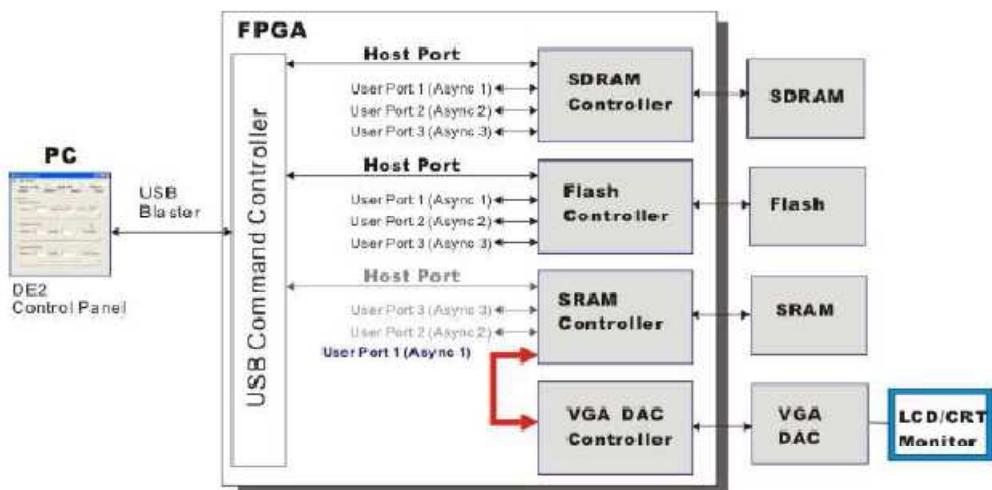
Tiếp theo ta sẽ trình bày cách hiện thị một hình ảnh bất kì ta mong muốn lên màn hình VGA.

- Ta có thể lấy một hình ảnh có định dạng pitutre.dat (để minh họa cho phần trình bày này ta sẽ lấy hình ảnh có sẵn trong thư mục DE2_demonstrations/picture/picture.dat trên đĩa CD DE2 System).
- Tải file picture.dat vào trong SRAM (đã trình bày ở phần Đọc/Ghi SRAM)
- Chọn tab TOOLS trên Control Panel, trên tab TOOLS ta chọn Asynchronous 1 cho mục SRAM multiplexer như hình dưới.



Hình 3.11 Giao diện Control Panel điều khiển Multi-ports

4. Nhấn nút **Configure** để kích hoạt hoạt động của multi-ports. Sau khi kích hoạt, FPGA sẽ được cấu hình lại như hình dưới đây



Hình 3.12 Cấu hình trên FPGA của Multi-ports

- Chọn tab VGA trên Control Panel và không chọn hộp Default Image nữa. Trên màn hình VGA (như hình dưới) sẽ xuất hiện hình ảnh của file picture.dat mà đã được lưu trữ trong SRAM.

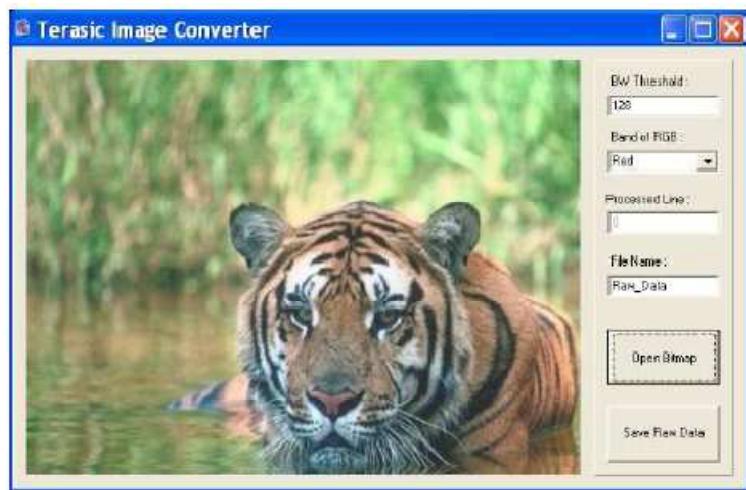


Hình 3.13 Màn hình VGA

Ta có thể hiển thị bất kì hình ảnh nào bằng việc tải file hình ảnh đó vào trong SRAM hoặc vùng nhớ M4K trong Cyclone II FPGA. Tuy nhiên định dạng của file hình ảnh này phải được chuyển sang định dạng bitmap (*file.dat*), để chuyển sang định dạng bitmap ta thực hiện những bước sau:

- Mở một hình ảnh lên một phần mềm hay công cụ xử lý ảnh nào đó chẳng hạn như Paint trên Window.
- Điều chỉnh độ phân giải của ảnh gốc để đạt được độ phân giải 640x480. Sau đó lưu lại dưới định dạng Windows Bitmap format (.BMP)

3. Thực thi chương trình thực thi ImgConv.exe (có sẵn trong thư mục DE2_control_panel/ImgConv.exe), một công cụ biến đổi ảnh cho Kit DE2 sẽ xuất hiện như hình dưới:



Hình 3.14 Trình biến đổi ảnh

4. Nhấn chọn **Open Bitmap** và chọn file hình ảnh có độ phân giải 640x480 ở trên.
5. Khi việc xử lí file hoàn thành, nhấn chọn nút **Save Raw Data**, một file có định dạng Raw_Data_Gray.dat sẽ được tạo ra và lưu trữ trong cùng thư mục với thư mục chứa ảnh gốc. Tất nhiên ta có thể đặt tên khác cho file hình ảnh mới bằng cách thay đổi nó trong trường File Name trên cửa sổ hình trên.

Sau khi có file Raw_Data_Gray.dat, ta đã có thể tải trực tiếp nó lên SRAM trên Kit DE2 và hiện thị lên màn hình VGA như đã trình bày phần trên.

Công cụ ImgConv trên cũng cho phép ta tạo ra Raw_Data_BW.data (và định dạng .TXT của nó) với hình ảnh đen trắng. Việc điều chỉnh mức độ màu sắc được định nghĩa trong BW Threshold dưới đây:

Image Source	R/G/B Band Filter	B&W Threshold Filter	Output Result (640x480)
Color Picture	R/G/B	N/A	Raw_Data_Gray
Color Picture	R/G/B (optional)	BW Threshold	Raw_Data_BW + Raw_Data_BW.txt
Grayscale Picture	N/A	N/A	Raw_Data_Gray
Grayscale Picture	N/A	BW Threshold	Raw_Data_BW + Raw_Data_BW.txt

Hình 3.15 Giá trị ngưỡng của ảnh

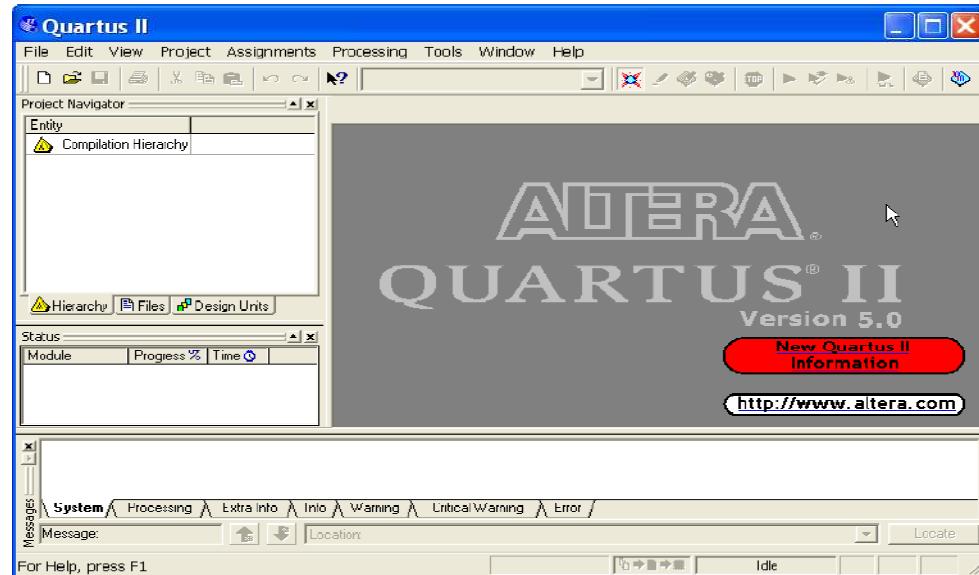
Ta có thể điều chỉnh màu sắc cho hình ảnh bằng cách thay đổi nhưng thông số trong những trường **BW Threshold** và **Band of RGB** trên cửa sổ ImvConv.

Chương 4. Hướng dẫn thiết kế và thực hành môn học Hệ thống số trên Kit DE2

4.1 Hướng dẫn thực hành

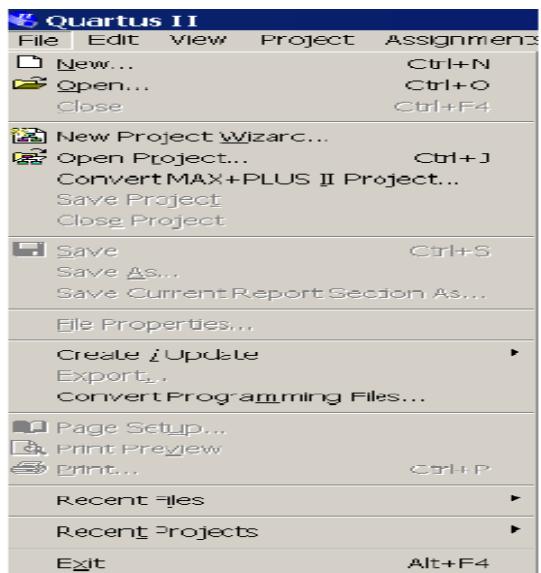
4.1.1 Tạo một project trên Quartus II

Bước 1. Start → Programs → Altera → Quartus II 7.2 → Quartus II 7.2 (32 -Bit)



Hình 4.1 Màn hình chính của Quartus

Bước 2. Nhấn tab **File** trên màn hình chính



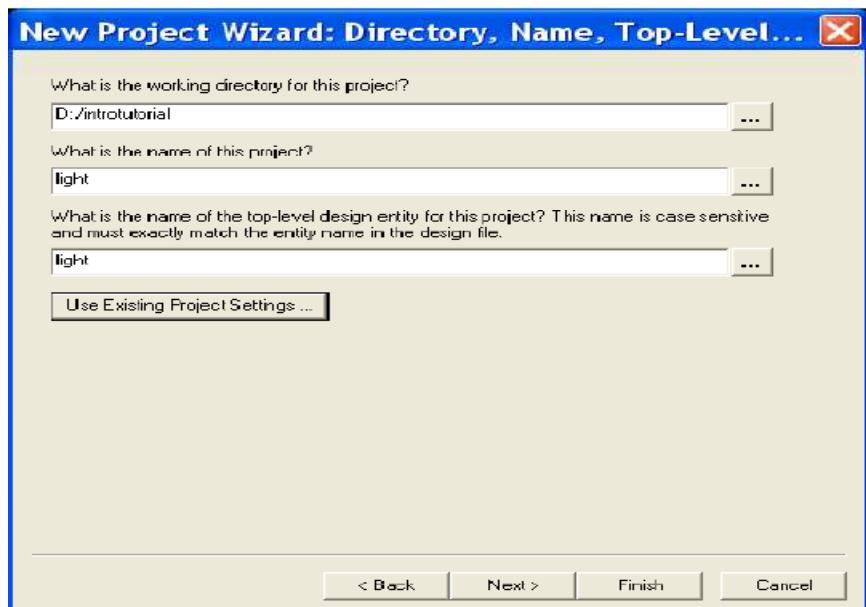
Hình 4.2 Tab File

Bước 3. Mở một project mới : **File ➔ New Project Wizard...**



Hình 4.3 Tạo project

Bước 4. Nhấn Next



Hình 4.4 Chỉ đường dẫn và tên project

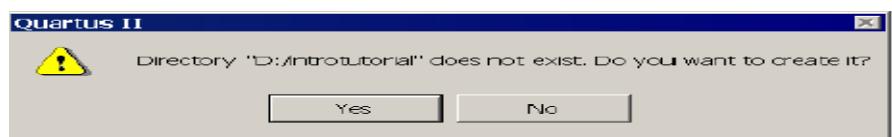
Bước 5. Nhập đường dẫn thư mục của project (có thể tạo trước hoặc nếu chưa tạo sẽ được tự động tạo).

Bước 6. Nhập tên của project.

Bước 7. Nhập top-level của thiết kế cho project (nên cho giống tên của project).

Bước 8. Nhấn **Next**

Bước 9. Nếu đường dẫn thư mục của project chưa được tạo trước :



Hình 4.5 Đường dẫn chưa tồn tại

Bước 10. Nhấn **Yes**



Hình 4.6 Add các file sử dụng trong project

Bước 11. Nhấn Next

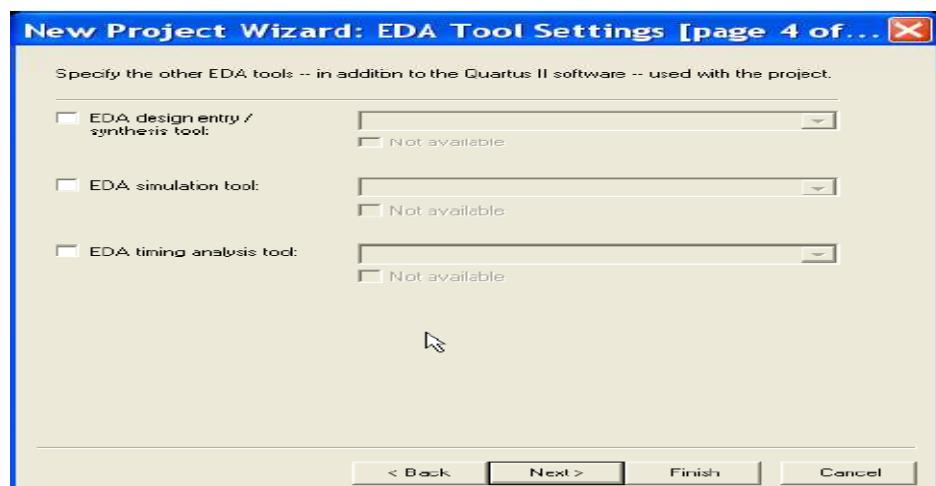


Hình 4.7 Chọn thiết bị FPGA

Bước 12. Chọn Family : Cyclone II

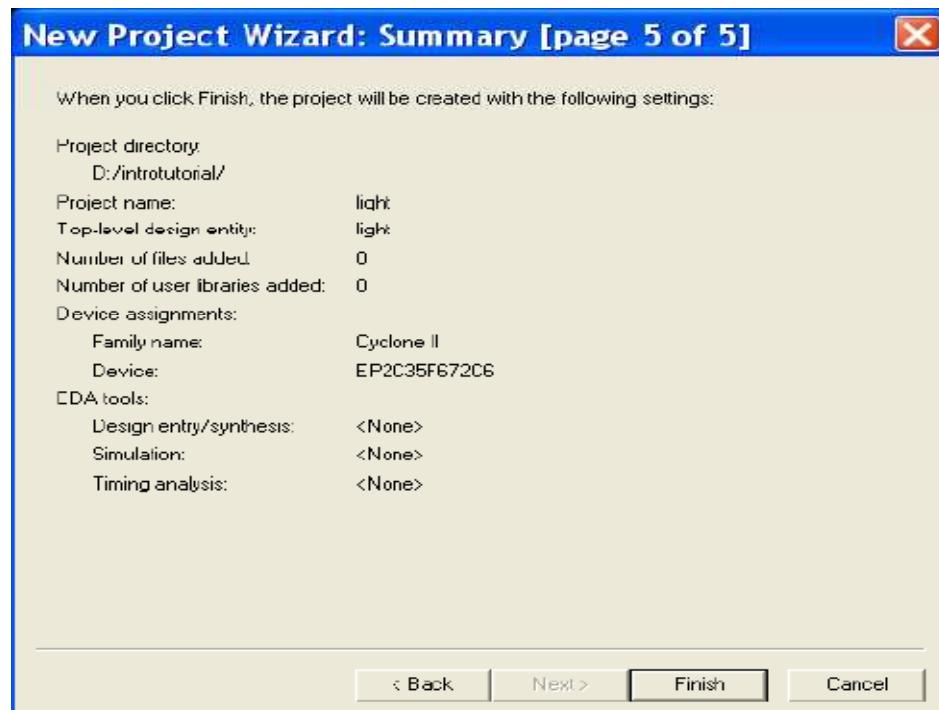
Bước 13. Chọn Available devices : **EP2C35F672C6** (Hộ của Chip FPGA Cyclone II trên Kit DE2).

Bước 14. Nhấn **Next**



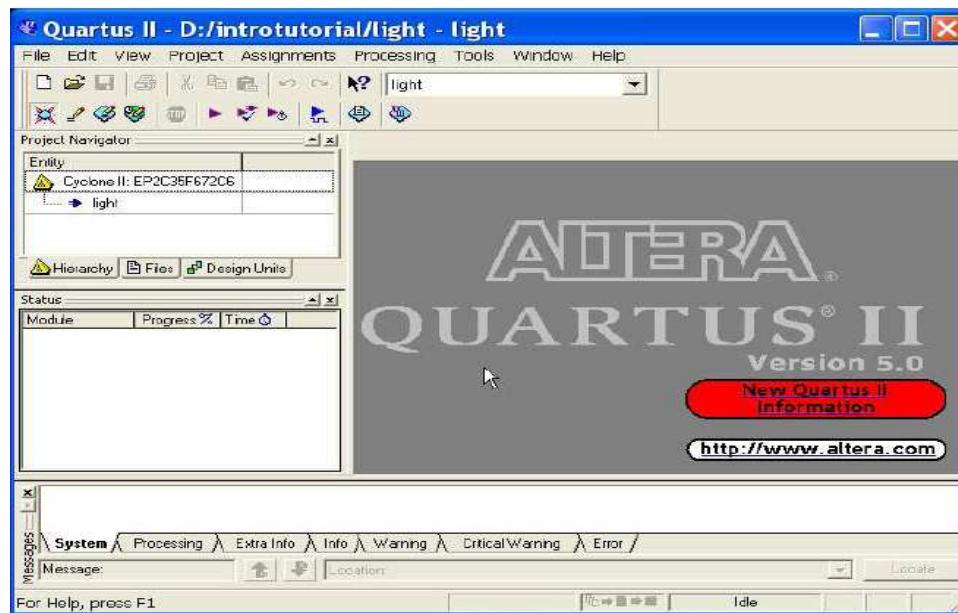
Hình 4.8 Thiết lập EDA tool

Bước 15. Nhấn **Next**



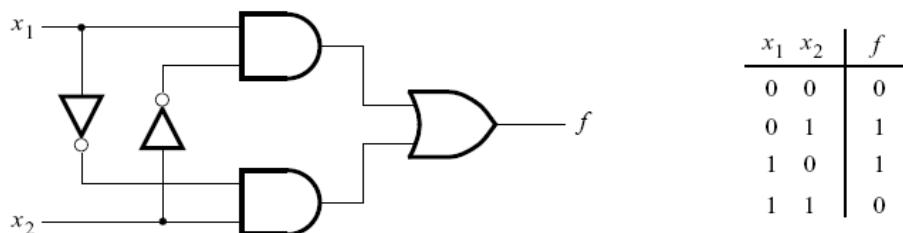
Hình 4.9 Hoàn thành việc tạo project

Bước 16. Nhấn **Finish** để trở về màn hình chính.



Hình 4.10 Màn hình chính sau khi tạo project hoàn thành

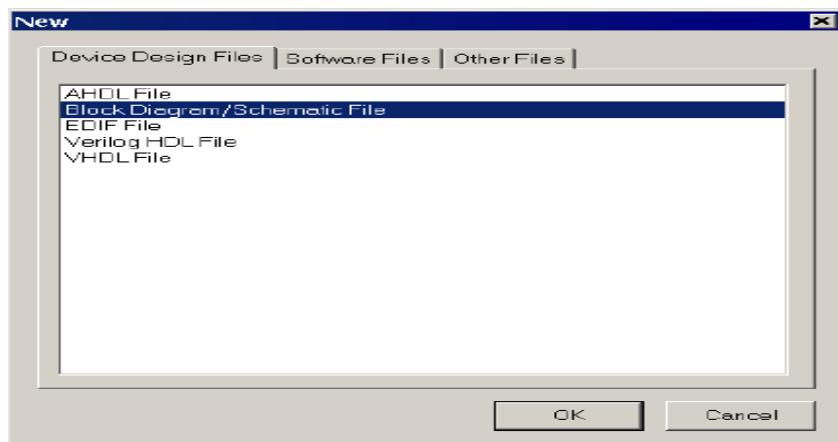
4.1.2 Thiết kế một mạch điện đơn giản (công XOR) dùng Schematic trên Quartus II:



Hình 4.11 Thiết kế một mạch số đơn giản

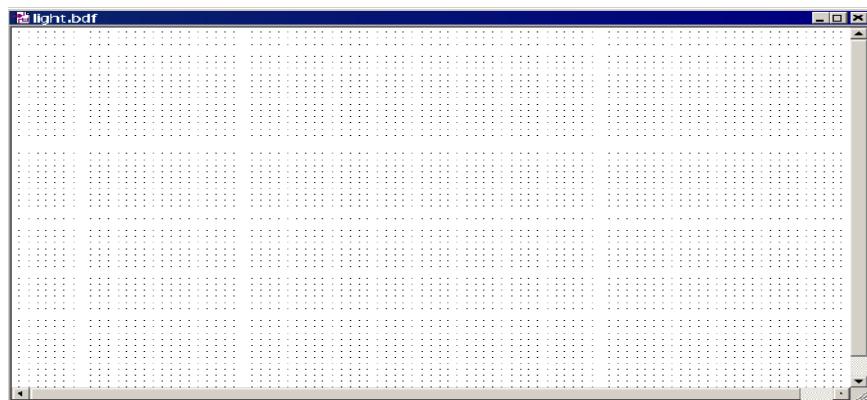
4.1.2.1 Mở trình thiết kế sử dụng schematic

Bước 1. Mở File ➔ New



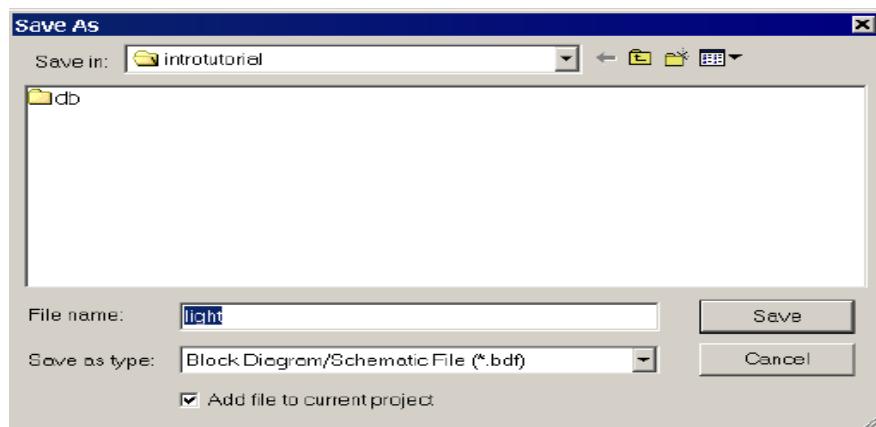
Hình 4.12 Chọn công cụ thiết kế

Bước 2. Chọn Block Diagram/Schematic File



Hình 4.13 Cửa sổ thiết kế mạch số

Bước 3. Save as file : **File → Save as**



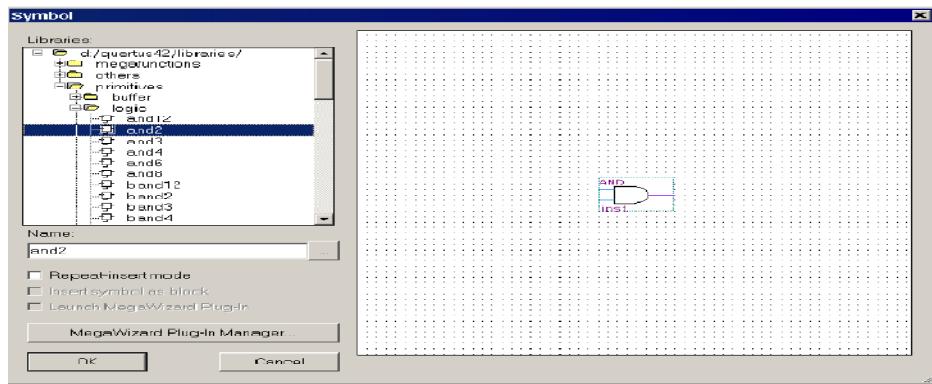
Hình 4.14 Lưu thiết kế

4.1.2.2 Thiết kế mạch hệ thống số

Bước 1. Chọn và nhập cổng Logic

Graphic Editor cung cấp một số thư viện chứa những linh kiện điện tử, cho phép người sử dụng chọn và nhập vào schematic. Nhập đúp lên khoảng trống bên

trong cửa sổ Graphic Editor hoặc nhấp lên biểu tượng  trong thanh công cụ. Một cửa sổ như hình dưới xuất hiện



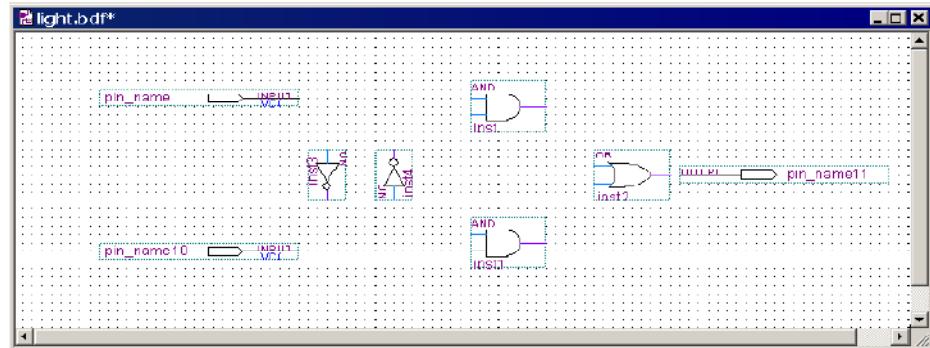
Hình 4.15 Chọn linh kiện

Từ cửa sổ này, ta có thể tìm và chọn những linh kiện hay cổng logic mà ta muốn để nhập vào cửa sổ Graphic Editor bằng cách sau khi chọn linh kiện thì ta nhấp nút **OK**. Thí dụ ta muốn nhập một cổng AND 2 ngõ vào, ta sẽ tìm và chọn and2 từ Library, sau đó nhấn **OK**, ta sẽ được một biểu tượng cổng AND2 xuất hiện trên cửa sổ Graphic Editor. Sử dụng chuột để di chuyển linh kiện đến vị trí mong muốn bằng cách nhấn chuột lên linh kiện và kéo rồi nhấp chuột để đặt nó xuống vị trí mới. Nếu muốn nhập một cổng AND2 lần thứ hai, ta có thể làm như cách trên hoặc có thể copy từ biểu tượng đã có sẵn trên cửa sổ bằng cách nhấp phải chuột, kéo rê chuột để tạo ra một biểu tượng thứ hai. Ta cũng có thể xoay biểu tượng của linh kiện bằng việc sử dụng biểu tượng  trên thanh công cụ.

Bước 2. Gán ngõ vào và ngõ ra cho linh kiện:

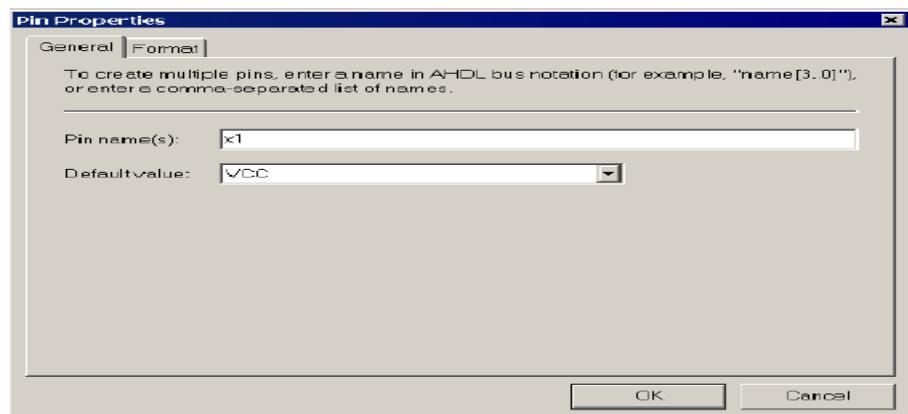
Sau khi đã nhập linh kiện vào trong cửa sổ Graphic Editor, ta phải gán ngõ vào và ngõ ra cho linh kiện trong mạch điện. Qui trình cũng tương tự như tìm và nhập linh kiện, nhưng biểu tượng ngõ vào hay ngõ ra sẽ được tìm thấy trong thư

viện **primitives/pin**. Trong hình dưới, ta sẽ nhìn thấy biểu tượng của ngõ vào và ngõ ra được gán vào chân của linh kiện.



Hình 4.16 Các linh kiện đã được chọn

Sau khi gán ngõ vào và ngõ ra cho linh kiện, ta phải đặt tên cho chúng. Để đặt tên, ta nhấp đúp vào từ **pin_name** của ngõ vào hay ngõ ra. Một hộp thoại như hình sau sẽ xuất hiện



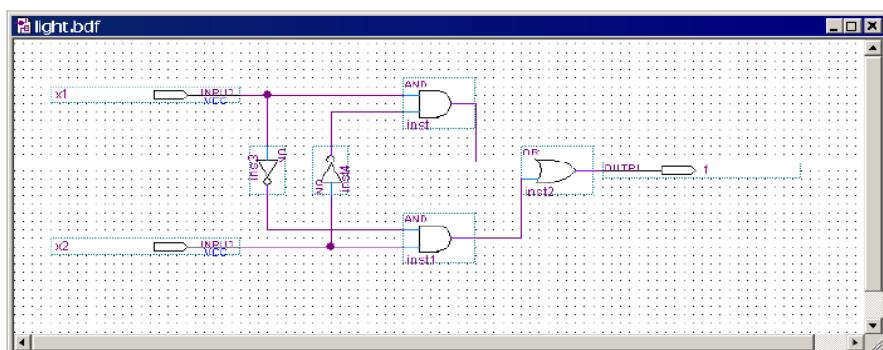
Hình 4.17 Đặt tên pin cho thiết kế

Nhập tên cho chân linh kiện vào ô **Pin name(s)**, rồi nhấn **OK**.

Bước 3. Kết nối linh kiện

Những linh kiện trong mạch phải được kết nối bằng dây điện. Nhấn chọn biểu tượng  trên thanh công cụ để kích hoạt **Orthogonal Node Tool**. Di chuyển con trỏ đến đầu của chân linh kiện, nhấn và giữ chuột trái và kéo cho đến khi đường dây chạm vào chân của linh kiện nào mà mình muốn kết nối tới. Chú ý, dấu chấm đen nhỏ thể hiện cho sự kết nối giữa hai đường dây dẫn.

Với qui trình tương tự, ta sẽ kết nối cho toàn bộ mạch điện sao cho đúng với chức năng hoạt động mà ta mong muốn. Nếu trong quá trình kết nối dây, ta kết nối sai dây dẫn nào đó, ta có thể xóa dây dẫn đó đi bằng cách nhấn chọn dây dẫn đó rồi nhấn phím **Delete (Del)** trên bàn phím. Sau khi hoàn thành kết nối dây, ta nhấn biểu tượng  để kích hoạt chức năng Select and Smart Drawing Tool. Bây giờ ta có thể sắp đặt lại vị trí của mạch điện sao cho dễ nhìn bằng cách chọn linh kiện hoặc dây dẫn và di chuyển chúng đến một vị trí thích hợp hơn. Thí dụ, hình dưới đây là một mạch điện hoàn chỉnh:



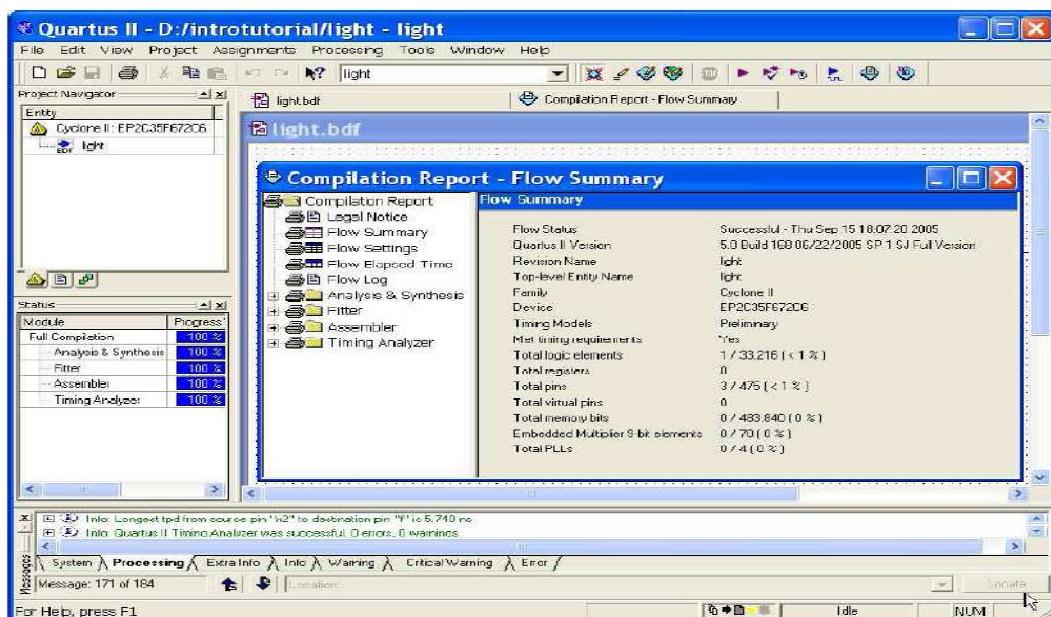
Hình 4.18 Thiết kế hoàn chỉnh

Sau khi mạch điện được hoàn chỉnh, ta nhớ lưu lại. Một file thiết kế sẽ được lưu dưới định dạng .bdf. Thi dụ **light.bdf**

4.1.2.3 Trình biên dịch

Với dữ liệu vào là file định dạng .bdf (light.bdf), nhiều công cụ trong phần mềm Quartus II được dùng để phân tích, tổng hợp mạch đã được thiết kế ở phần trên, rồi sau đó sẽ tạo ra một file thực thi dùng để nạp lên FPGA. Những công cụ được sử dụng trong quá trình này được gọi là trình biên dịch. Để thực thi quá trình biên dịch, ta thực hiện các bước sau:

Bước 1. Chọn: **Processing → Start Compilation** hoặc nhấn chọn biểu tượng trên thanh công cụ. Sau khi quá trình biên dịch được hoàn tất, một bảng báo cáo được tạo ra như hình dưới



Hình 4.19 Cửa sổ trình biên dịch report

Bước 2. Để xem lại quá trình biên dịch, ta chọn : **Processing → Compilation Report** hoặc nhấn chọn biểu tượng trên thanh công cụ.

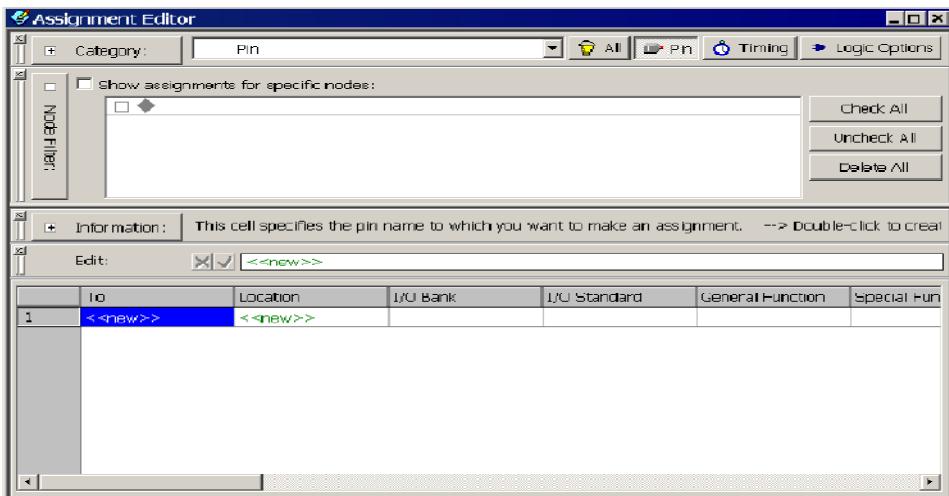
4.1.2.4 Message window

Phần mềm Quartus II sẽ hiển thị thông tin trong suốt quá trình biên dịch trên cửa sổ **Message** widow. Nếu sơ đồ mạch điện được thiết kế trong phần Graphic Editor hoàn toàn đúng, thì một thông báo “The compilation was successful” được hiện thị. Trong trường hợp quá trình biên dịch xuất hiện lỗi thì có nghĩa đã có lỗi xảy ra trong quá trình thiết kế trên Graphic Editor. Mỗi thông báo tương ứng với một lỗi được tìm thấy sẽ xuất hiện trên cửa sổ **Message**. Nhấp đúp vào thông báo lỗi đó ta sẽ biết rõ hơn về lỗi đã xảy ra trên mạch điện. Tương tự, trình biên dịch cũng thông báo một số cảnh báo “Warning”. Ngoài ra ta cũng có thể tìm hiểu thêm thông tin về lỗi cũng như cảnh báo bằng cách nhấn chọn vào thông báo đó rồi nhấn phím F1 trên bàn phím.

4.1.3 Gán pin

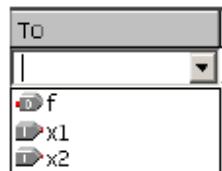
Vì ta chưa thực hiện gán pin trên FPGA cho linh kiện trong mạch điện đã thiết kế ở trên nên khi thực hiện biên dịch thì trình biên dịch Quartus II đã gán chân của linh kiện với pin của FPGA một cách ngẫu nhiên. Tuy nhiên, giả sử trong thiết kế cổng XOR đơn giản ở trên, sau khi thiết kế được biên dịch và nạp lên FPGA, ta muốn hai ngõ vào x1, x2 được điều khiển bởi hai switch SW0 và SW1 còn kết quả ngõ ra f sẽ được thể hiện trên led LEDG0 (SW0, SW1, LEDG0 được ghi trên Kit). Mặt khác ta biết switch SW0 được kết nối cố định với pin N25 của FPGA, tương tự vậy switch SW1 được kết nối cố định với pin N25 của FPGA và led LEDG0 được kết nối cố định với pin AE22 của FPGA. Để thực hiện được điều đó ta phải gán chân linh kiện trên mạch (x1, x2, f) với pin tương ứng trên FPGA (N25, N26, AE22). Để gán pin ta thực hiện các bước sau

Bước 1. Chọn **Assignments > Pins**, một cửa sổ như hình dưới sẽ xuất hiện



Hình 4.20 Cửa sổ mapped pin giữa thiết kế và FPGA

Bước 2. Trong mục **Category** chọn **Pin**. Nhấp đúp lên mục <<new>> trong cột **To**. Một cửa sổ như hình dưới xuất hiện



Hình 4.21 Cửa sổ gán pin

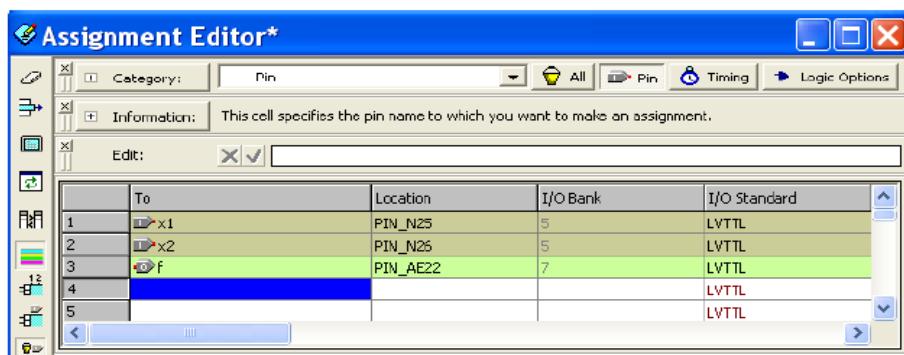
Bước 3. Nhấn chọn x1 để gán pin trước, tiếp đến nhấp đúp lên mục ngay bên phải của x1 trong cột **Location**, một cửa sổ như hình dưới sẽ xuất hiện

Location	I/O Bank	I/O Standard	General Function	Special Fun
PIN_N1	I/O Bank 2	Dedicated Clock	CLK1, LVDSCLK0n, Input	
PIN_N2	I/O Bank 2	Dedicated Clock	CLK0, LVDSCLK0p, Input	
PIN_N9	I/O Bank 2	Row I/O	LVDS31p	
PIN_N18	I/O Bank 5	Row I/O	LVDS110p	
PIN_N20	I/O Bank 5	Row I/O	LVDS124p	
PIN_N23	I/O Bank 5	Row I/O	LVDS126p, DPCLK7/DQ50R/CQ1R	
PIN_N24	I/O Bank 5	Row I/O	LVDS126n	
PIN_N25	I/O Bank 5	Dedicated Clock	CLK4, LVDSCLK2p, Input	
PIN_N26	I/O Bank 5	Dedicated Clock	CLK5, LVDSCLK2n, Input	
PIN_P1	I/O Bank 1	Dedicated Clock	CLK3, LVDSCLK1n, Input	
PIN_P2	I/O Bank 1	Dedicated Clock	CLK2, LVDSCLK1p, Input	
PIN_P3	I/O Bank 1	Row I/O	LVDS25p, DPCLK1/DQ51L/CQ1L#	
PIN_P4	I/O Bank 1	Row I/O	LVDS26n	
PIN_P6	I/O Bank 1	Row I/O	LVDS22n	
PIN_P7	I/O Bank 1	Row I/O	LVDS22p	
PIN_P9	I/O Bank 2	Row I/O	LVDS31n	
PIN_P17	I/O Bank 6	Row I/O	LVDS130n	
PIN_P18	I/O Bank 5	Row I/O	LVDS110n	
PIN_P23	I/O Bank 6	Row I/O	LVDS127p, DPCLK6/DQ51R/CQ1R#	
PIN_P24	I/O Bank 6	Row I/O	LVDS127n	

Hình 4.22 Cửa sổ liệt kê danh sách pin của FPGA

Bước 4. Ta nhấp chọn PIN_N25.

Bước 5. Tương tự, ta gán pin cho chân ngõ vào x2 tới pin PIN_N26, và chân ngõ ra f tới pin PIN_AE22. Sau khi gán pin hoàn tất, ta sẽ được như hình dưới



Hình 4.23 Cửa sổ sau gán pin

Bước 6. Lưu lại kết quả gán pin: **File ➔ Save**

Bước 7. Ta phải biên dịch lại thiết kế ở trên với kết quả gán pin này vì như ta đã nói ở trên, vì quá trình biên dịch ở trên, trình biên dịch Quartus

II chỉ gán pin một cách ngẫu nhiên nên sẽ không đúng với yêu cầu thiết kế của ta, do đó ta phải gán lại pin cho đúng với yêu cầu rồi phải chạy lại quá trình biên dịch. Lúc này trình biên dịch Quartus II sẽ sử dụng những pin mà ta đã gán cho chân của mạch điện trong thiết kế để phân tích, tổng hợp và tạo ra một file để thực thi việc nạp xuống cho FPGA.

Ngoài ra ta cũng có một cách khác để gán pins cho design, đặc biệt là rất hữu ích trong thiết kế mà có nhiều chân, ta không thể ngồi gán pin cho từng chân được vì sẽ tốn nhiều thời gian, Quartus II cung cấp một phương pháp giúp ta gán nhiều pin vào hoặc gỡ nhiều pin ra cùng một lúc bằng một file có định dạng đặc biệt dùng cho mục đích này đó là định dạng .CSV. Format của file này như sau

- Nếu ta dùng file text để tạo file này, thì đơn giản ta chỉ cần nhập theo mẫu sau

To, Location

x1, PIN_N25

x2, PIN_N26

f, PIN_AE22

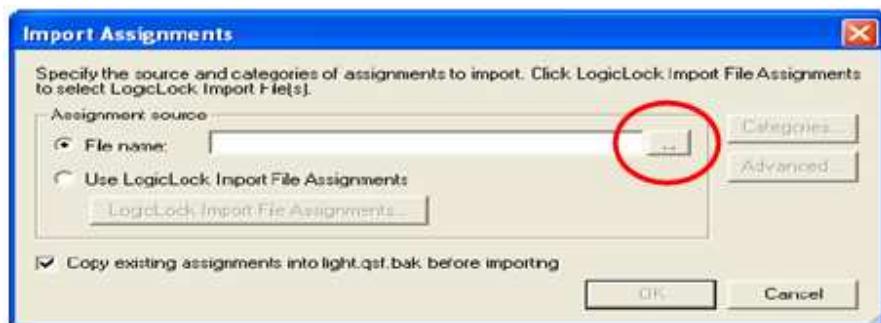
- Nếu ta dùng Microsoft Excel, thì ta sẽ có format như sau:

	A	B
1	To	Location
2	x1	PIN_N25
3	x2	PIN_N26
4	f	PIN_AE22

Hình 4.24 Dùng Microsoft Excel để tạo file gán pin

- Sau khi tạo file có format như trên, ta sẽ thực hiện việc gán pin như sau

Bước 1. Chọn **Assignments** -> **Import Assignments**, một hộp thoại như hình dưới xuất hiện



Hình 4.25 Import file gán pin

Bước 2. Click **button ...**, chỉ đường dẫn của file ta vừa tạo ở trên. Rồi nhấn **OK**.

Để thuận tiện cho người sử dụng Altera đã cung cấp một file CSV có tên DE2_pin_assignments, file này liệt kê tất cả các pin của FPGA, có format như sau:

	A	B	C
7	To	Location	
8	SW[0]	PIN_N25	
9	SW[1]	PIN_N26	
10	SW[2]	PIN_P25	
11	SW[3]	PIN_AE14	
12	SW[4]	PIN_AF14	
13	SW[5]	PIN_AD13	
14	SW[6]	PIN_AC13	
15	SW[7]	PIN_C13	
16	SW[8]	PIN_B13	

Hình 4.26 File gán pin tạo sẵn bởi Altera

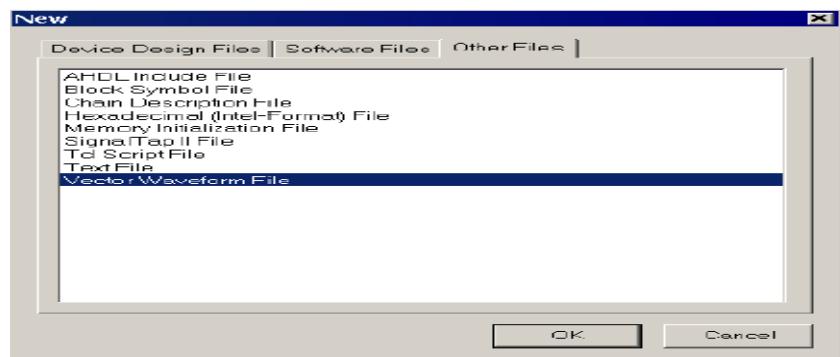
Nếu ta muốn sử dụng file có sẵn này vào việc gán pin cho thiết kế của ta thì một yêu cầu bắt buộc khi ta đặt tên cho chân linh kiện phải trùng với tên trong cột **To** của file này. Thí dụ, nếu ta muốn hai chân ngõ vào của cổng XOR được điều khiển bởi hai Switch 0 và Switch 1 trên Kit DE2 thì ta phải đặt tên cho hai chân này lần lượt là SW[0], SW[1] như trong cột **To** của file này. Do đó ta phải tham khảo file này trước khi đặt tên cho chân linh kiện để khi gán pin ta sẽ rất thuận tiện đó là không phải tạo file.csv nữa mà chỉ cần Import file có sẵn này vào thôi.

Sau khi gán pin xong, ta biên dịch lại.

- Re-compiling design : **Processing → Start Compilation**
- Review Compilation report : **Processing → Compilation Report**

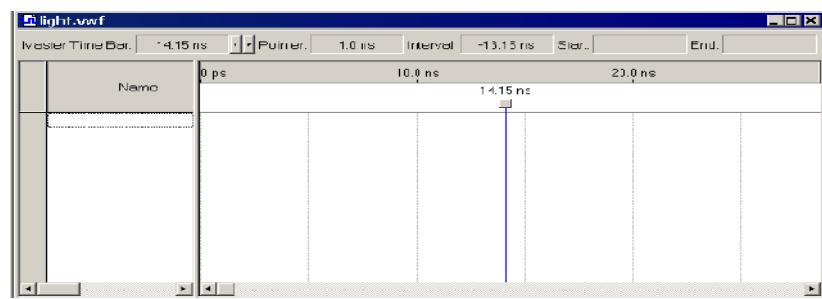
4.1.4 Mô phỏng mạch đã thiết kế :

Bước 1. Tạo input waveform : **File → New → Other Files → Vector Waveform File**



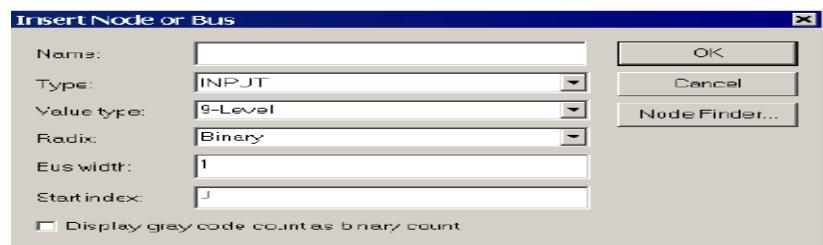
Hình 4.27 Tạo waveform

Bước 2. Nhấn **OK**



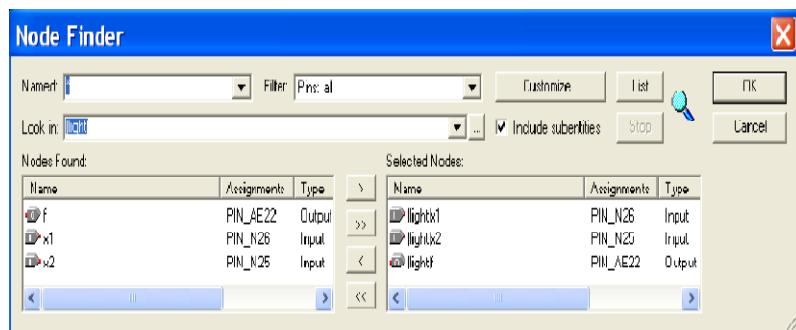
Hình 4.28 Cửa sổ tạo waveform

- Bước 3. Chọn thời gian thực hiện mô phỏng : **Edit → End Time**
- Bước 4. Nhập thời gian thực hiện mô phỏng.
- Bước 5. Fit window : **View → Fit in Window**
- Bước 6. Tạo waveform cho inputs : **Edit → Insert Node or Bus**



Hình 4.29 Nhập tên signal của thiết kế

Bước 7. Chọn Node Finder



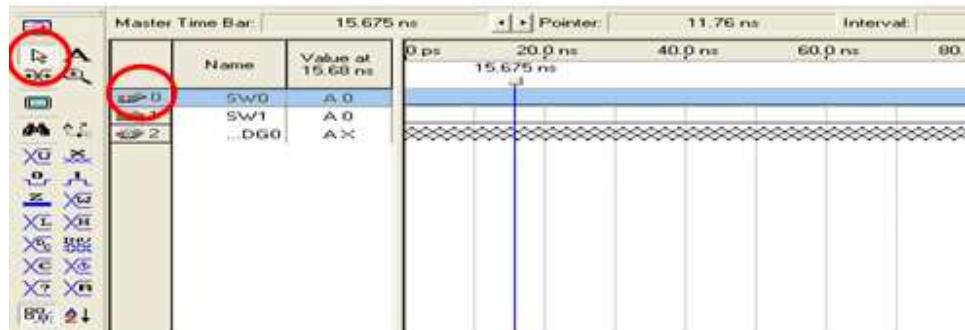
Hình 4.30 Dùng chức năng Node Finder

Bước 8. Chọn Filter : Pins : all

Bước 9. Nhấn button List

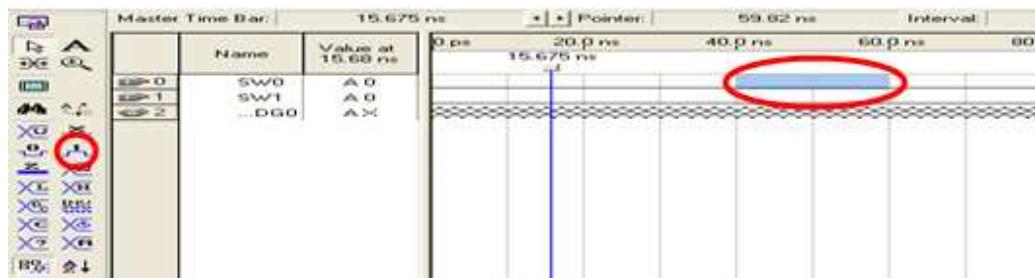
Bước 10. Chọn signal bên **Nodes found** ; nhấn >> để chuyển sang bên **Selected Nodes**

Bước 11. Nhấn OK



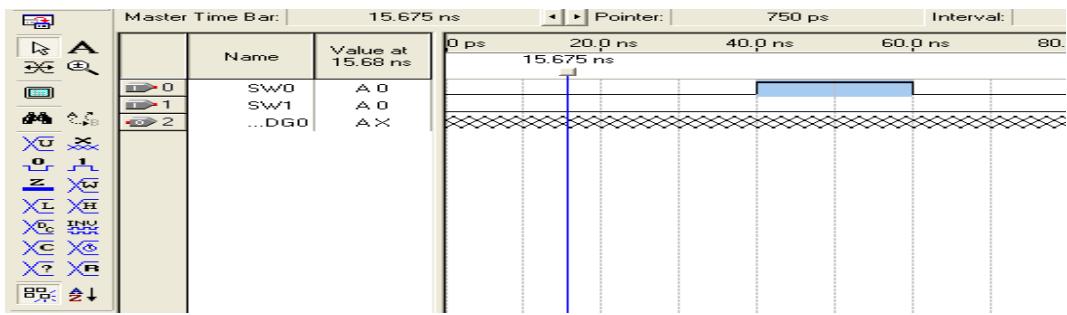
Hình 4.31 Tạo input waveform

- Bước 12. Chọn một input signal bằng cách nhấp chuột vào signal đó .
- Bước 13. Chọn biểu tượng mũi tên con trỏ
- Bước 14. Di chuyển con trỏ sang màn hình waveform .
- Bước 15. Nhấn và giữ chuột và kéo rê (left) trong một khoảng thời gian (giả sử ta muốn trong khoảng thời gian từ 40ns -> 60 ns , SW0 signal có giá trị “1”, thì ta nhấn , giữ và rê chuột trong khoảng thời gian từ 40ns -> 60ns.



Hình 4.32 Tạo mức logic "1"

- Bước 16. Nhấn button “1” phía bên trái màn hình

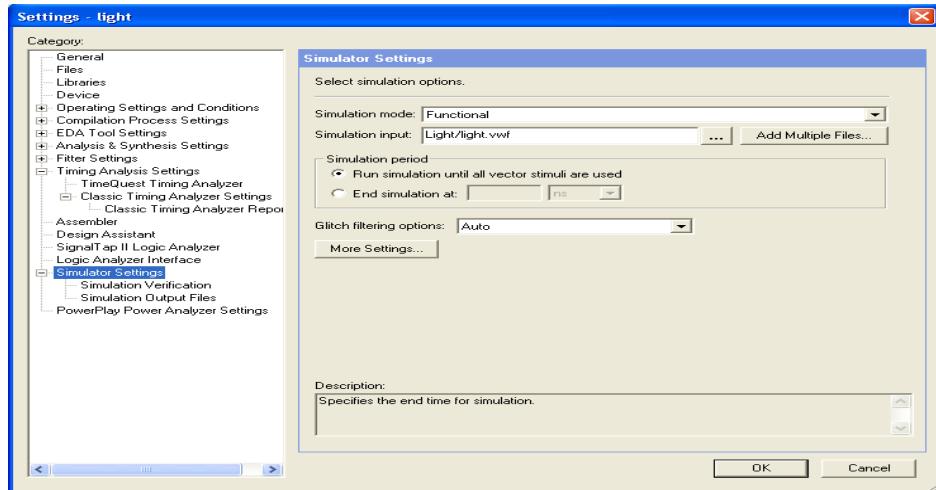


Hình 4.33 Mức logic "1" đã được tạo

Bước 17. Tương tự cho những tín hiệu inputs khác, không tạo waveform cho outputs (XXX).

Bước 18. Save File Waveform : **File ➔ Save As**

Bước 19. Thiết lập thực hiện mô phỏng : **Assignments ➔ Setting**



Hình 4.34 Thiết lập chế độ simulation

Bước 20. Chọn **Simulator Settings**

Bước 21. Chọn **Simulation mode : Functional / Timing**

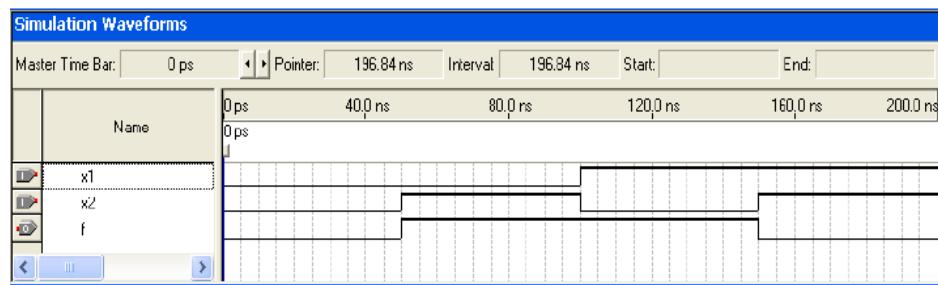
Bước 22. Chỉ đường dẫn của input waveform vừa tạo.

Bước 23. Nhấn **OK**

Bước 24. Tạo simulation netlist : **Processing → Generate Functional Simulation Netlist**

Bước 25. Chạy mô phỏng : **Processing → Start Simulation.**

Bước 26. Quan sát waveform của Output và debug nếu có lỗi.



Hình 4.35 Waveform sau khi chạy mô phỏng

4.1.5 Programming mạch đã thiết kế lên FPGA :

Bước 1. Kết nối Kit DE2 với máy tính qua cổng USB-Blaster (phải cài đặt driver trước).

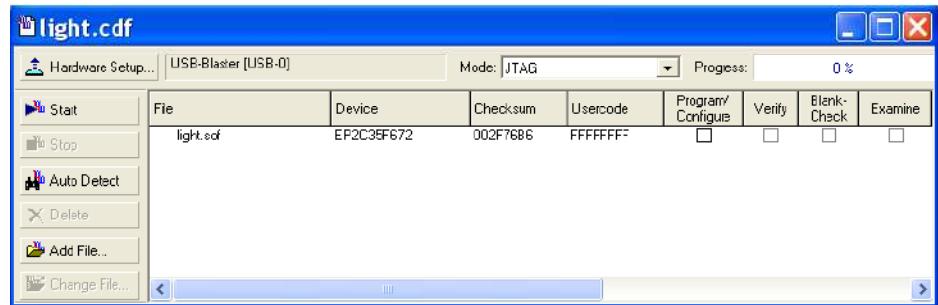
Bước 2. Bật nguồn Kit DE2.

Có 2 mode cho việc programming : JTAG và Active Serial modes

JTAG mode

Bước 3. Trên Kit DE2 , chuyển Switch **RUN/PROG** về vị trí **RUN**

Bước 4. Trên màn hình chính Quantus II, chọn **Tools → Programmer**



Hình 4.36 Nạp thiế́ kέ lên FPGA

Bước 5. Nhấn **Hardware Setup**, chọn **USB-Blaster[USB-0]** (Chú ý : phải cài đặt driver cho USB-Blaster trước).



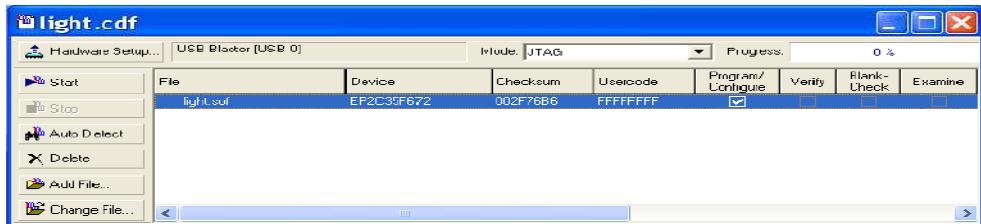
Hình 4.37 Thiết lập công giao tiếp giữa kit DE2 và Computer

Bước 6. Nhấn **Close**

Bước 7. Chọn **Mode JTAG**

Bước 8. Nhấn **Add File**, chỉ đường dẫn đến **File .sof** (được tạo ra khi chạy Compilation).

Bước 9. Check box **Program/Configure**



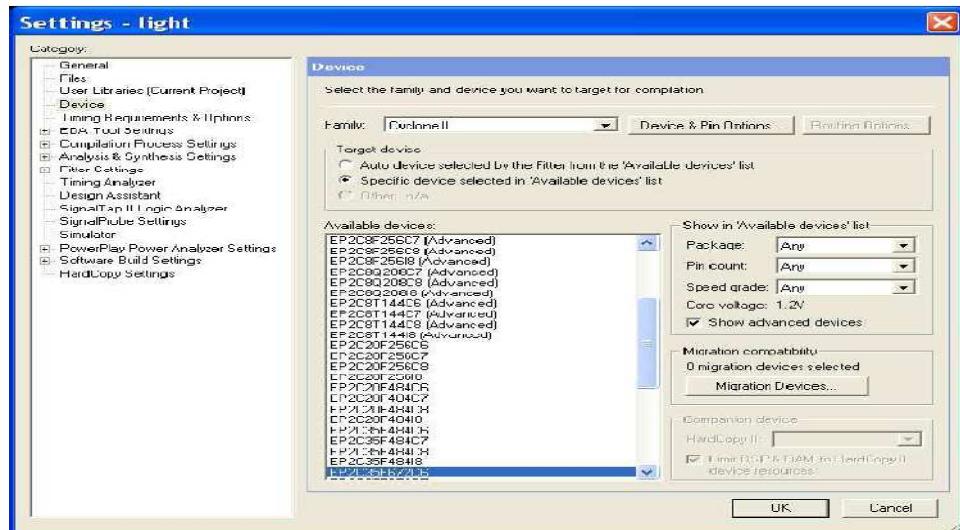
Hình 4.38 Chọn cấu hình nạp thiết kế

Bước 10. Nhấn Start.

Bước 11. Quan sát trên Kit DE2, switch SW0, SW1 và quan sát LED.

■ Active Serial Mode :

Bước 12. Chọn Assignments → Devide



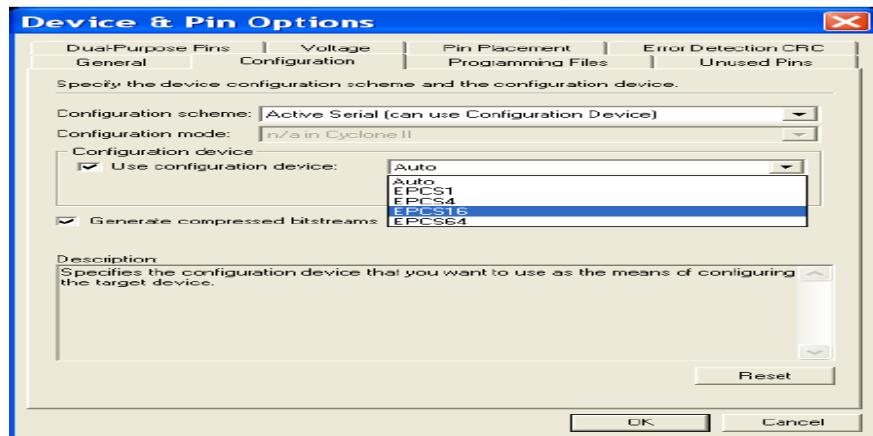
Hình 4.39 Thiết kế chế độ nạp lên FPGA bằng AS mode

Bước 13. Chọn Family : Cyclone II

Bước 14. Chọn Available devices : EP2C35F672C6

Bước 15. Nhấn Device & Pin Option

Bước 16. Chọn Tab Configuration



Hình 4.40 Chọn loại ROM tương ứng

Bước 17. Chọn Configuration device : EPCS64 (hỗ trợ EPPROM trên Kit DE2 , dùng để lưu chương trình để nạp cho FPGA mỗi khi power on).

Bước 18. Tương tự JTAG ở những bước kế tiếp :

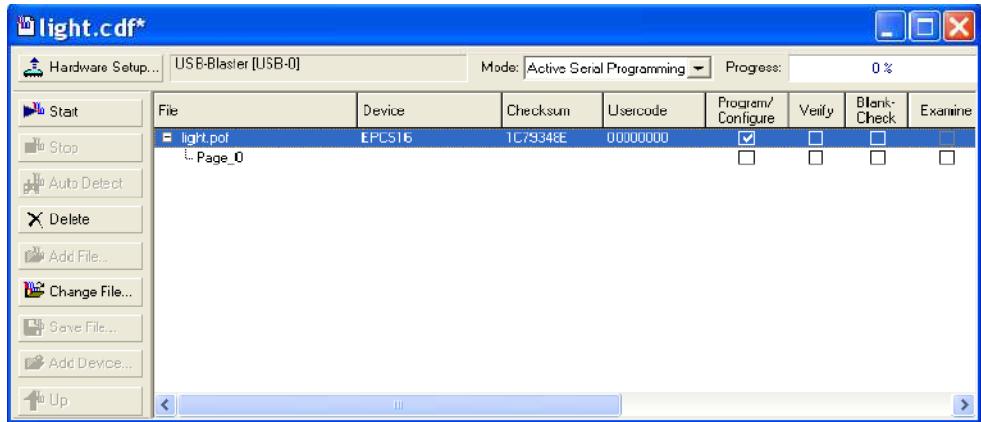
Bước 19. Trên Kit DE2 , chuyển Switch **RUN/PROG** về vị trí **RUN**

Bước 20. Trên màn hình chính Quantus II, chọn **Tools ➔ Programmer**

Bước 21. Chọn **Hardware Setup : USB-Blaster[USB-0]**

Bước 22. Chọn **Mode : Active Serial Programming**

Bước 23. Nhấn **Add File**, chỉ đường dẫn đến **File .pof** (File được tạo ra trong quá trình chạy Compilation).



Hình 4.41 Chọn file thiết kế .pof

- Bước 24. Check box **Program/Configure**.
- Bước 25. Nhấn **Start** để programming chương trình cho EPPROM.
- Bước 26. Nhấn Phím **Restart** trên Kit DE2,
- Bước 27. Quan sát trên Kit DE2, switch SW0, SW1 và quan sát LED.

4.2 Nội dung thực hành môn Hệ thống số

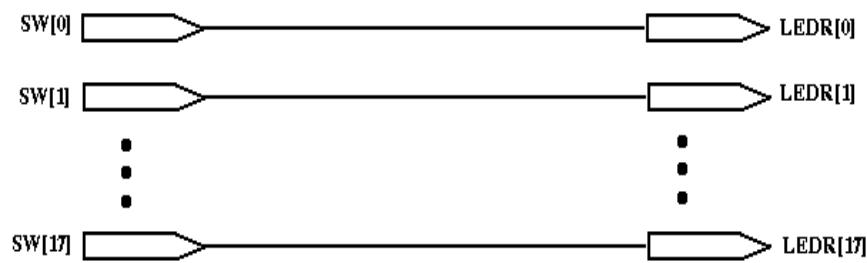
4.2.1 Bài thực hành số 1 – Switchs, Lights, Multiplexers

Mục đích của Lab 1: Học cách kết nối đơn giản những ngõ vào và ngõ ra của linh kiện đến FPGA và thiết kế một số mạch điện đơn giản sử dụng những linh kiện trên Kit DE2 như là ngõ vào và ngõ ra của mạch thiết kế. Trong Lab này, ta sẽ sử dụng Switch SW17-SW0 trên Kit DE2 như là ngõ vào của mạch và sử dụng LED và LED bảy đoạn như là ngõ ra của mạch. Để làm tốt Lab1, sinh viên cần phải nắm trước ở nhà về cách thiết kế, biên dịch và mô phỏng một mạch điện đơn giản trên Quartus II.

4.2.1.1 Phần 1

Từng bước thực hiện:

- Bước 1. Tạo một project Quartus mới, đặt tên: user_dir/lab1/lab1_par1
- Bước 2. Thiết kế một mạch như sau:

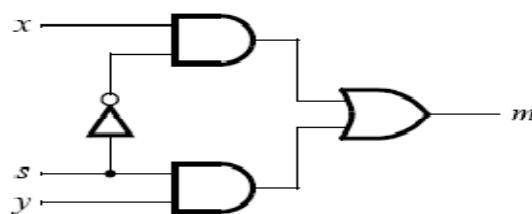


Hình 4.42 Thiết kế đơn giản

- Bước 3. Gán pin cho mạch trên
 - Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.
 - Bước 5. Tạo file Vector Waveform (.vWF) và chạy mô phỏng để kiểm tra hoạt động của mạch.
 - Bước 6. Nạp file thực thi lên FPGA. Kiểm tra hoạt động của mạch.
- Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm → Tính vắng buổi đó)
- Đọc và thực hiện các bước từ 1 đến 5 ở nhà.

4.2.1.2 Phản 2

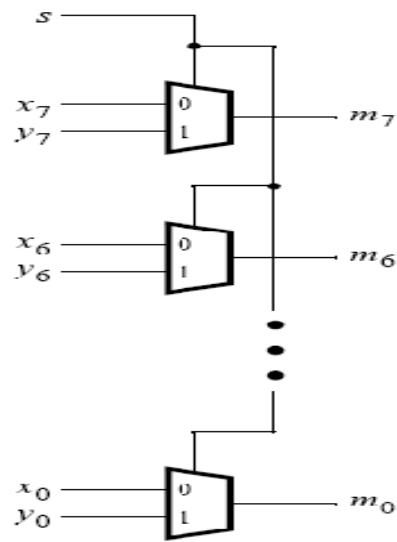
1. Cho mạch sau:



Hình 4.43 Mạch số đơn giản

- Bước 1. Nêu hoạt động của mạch trên và viết bảng sự thật cho mạch.

2. Cho mạch sau:

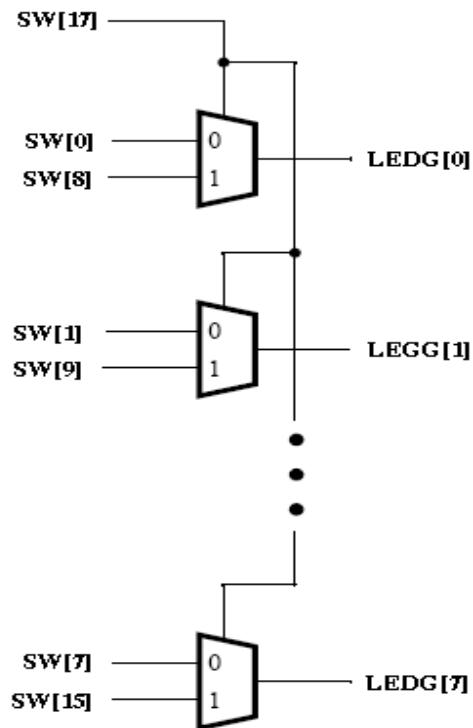


Hình 4.44 Mạch gồm 8 MUX 2-1

Bước 2. Nêu hoạt động của mạch trên

Bước 3. Tạo một project Quartus mới, đặt tên:
user_dir/lab1/lab1_part2

Bước 4. Thiết kế một mạch như sau:

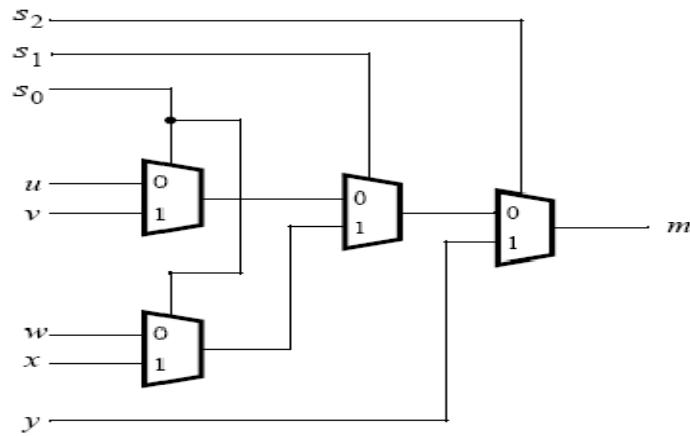


Hình 4.45 Mạch 8 MUX 2-1 với SW và LED

- Bước 5. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.
 - Bước 6. Tạo file Vector Waveform (.vWF) và chạy mô phỏng để kiểm tra hoạt động của mạch.
 - Bước 7. Nạp file thực thi lên FPGA. Kiểm tra hoạt động của mạch.
- Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm → Tính vắng buổi đó)
- Đọc và thực hiện các bước từ 1 đến 6 ở nhà.

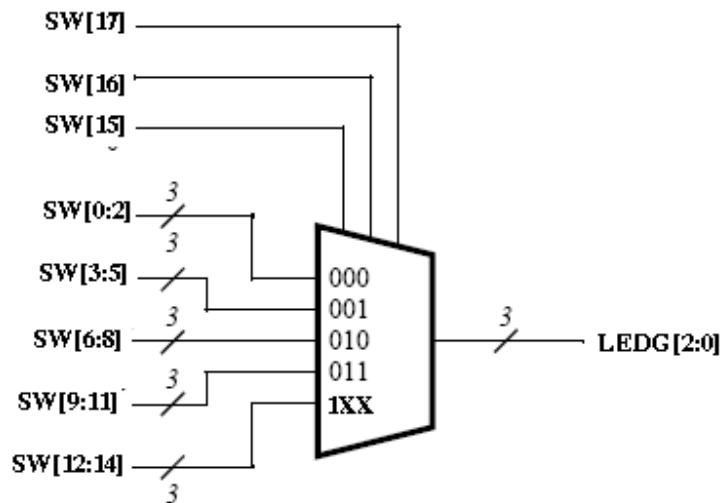
4.2.1.3 Phản 3

Cho mạch sau:



Hình 4.46 Mạch chọn kênh

- Bước 1. Nêu hoạt động của mạch trên và viết bảng sự thật cho mạch.
- Bước 2. Tạo một project Quartus mới, đặt tên: user_dir/lab1/lab1_part3
- Bước 3. Thiết kế một mạch như sau:



Hình 4.47 Mạch chọn kênh 3 input

- Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.
- Bước 5. Tạo file Vector Waveform (.vWF) và chạy mô phỏng để kiểm tra hoạt động của mạch.

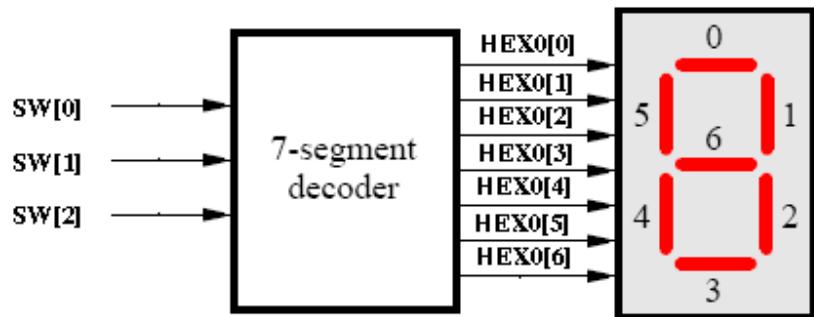
Bước 6. Nạp file thực thi lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm → Tính vắng buổi đó)

- Đọc và thực hiện các bước từ 1 đến 5 ở nhà.

4.2.1.4 Phần 4

Cho mạch sau:



Hình 4.48 Mạch giải mã HEX

Dưới đây là bảng sự thật của mạch giải mã cho Led 7 đoạn trên dùng để hiện thị một trong 4 ký tự H, E, L, O.

SW[2:0]	Character	HEX0[0]	HEX0[1]	HEX0[2]	HEX0[3]	HEX0[4]	HEX0[5]	HEX0[6]
000	H	?	?	?	?	?	?	?
001	E	?	?	?	?	?	?	?
010	L	?	?	?	?	?	?	?
011	O	?	?	?	?	?	?	?
100	Off	1	1	1	1	1	1	1
101	Off	1	1	1	1	1	1	1
110	Off	1	1	1	1	1	1	1
111	Off	1	1	1	1	1	1	1

Hình 4.49 Bảng giải mã

Bước 1. Sinh viên hoàn thành bảng sự thật trên. (Chú ý: Các đoạn LED tích cực mức thấp).

Bước 2. Dựa vào bảng sự thật, thiết kế mạch giải mã cho LED 7 đoạn trên.

Bước 3. Tạo một project Quartus mới, đặt tên:
user_dir/lab1/lab1_part4

Bước 4. Thiết kế mạch như hình trên.

Bước 5. Biên dịch để phân tích, tổng hợp và tạo ra file .sof

Bước 6. Tạo file Vector Waveform (.vWF) và chạy mô phỏng để kiểm tra hoạt động của mạch.

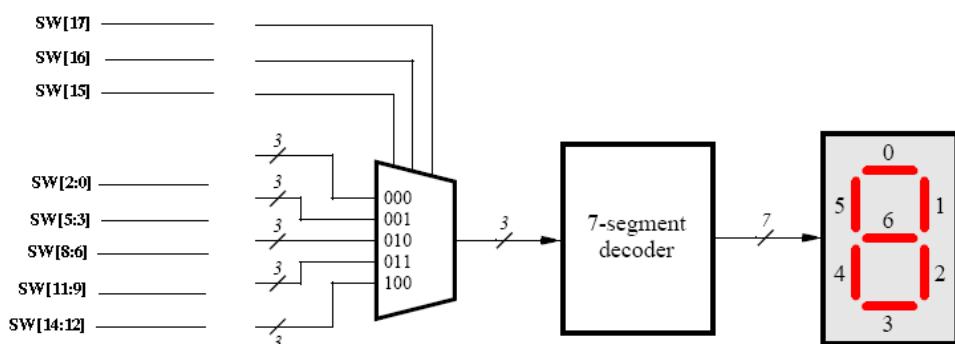
Bước 7. Nạp file thực thi lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

■ Đọc và thực hiện các bước từ 1 đến 6 ở nhà.

4.2.1.5 Phần 5

Cho dữ liệu mạch điện như sau:



Hình 4.50 Mạch chọn kênh và hiển thị

- 💡 Dựa vào những dữ kiện đã cho ở hình trên và kiến thức đã nắm được từ Part1 đến Part4, hãy thiết kế mạch điện hoạt động theo bảng sự thật sau (Chú ý: có thể thiết kế bằng nhiều cách, có thể sử dụng hết các SW hoặc sử dụng một số SW tùy vào cách thiết kế)

SW_{17}	SW_{16}	SW_{15}	LED0	LED1	LED2	LED3	LED4
000			H	E	L	L	O
001			E	L	L	O	H
010			L	L	O	H	E
011			L	O	H	E	L
100			O	H	E	L	L

Hình 4.51 Mode hiển thị

4.2.2 Bài thực hành số 2 – Số và cách hiển thị

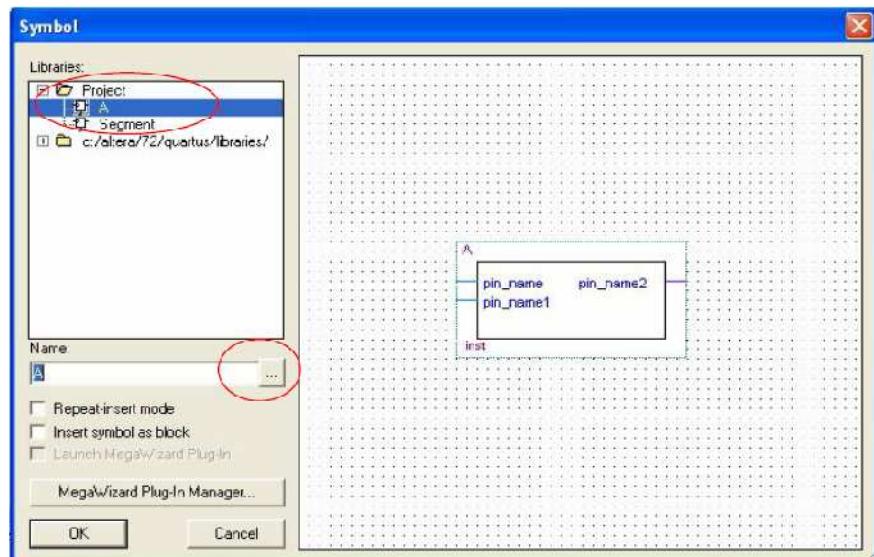
Mục đích lab 2: Thiết kế mạch chuyển từ số nhị phân sang thập phân và mạch cộng số BCD

Hướng dẫn cách đóng gói và tái sử dụng 1 mạch đã thiết kế

Bước 1. Tạo 1 file .bdf, vẽ mạch cần sử dụng lại trên đó.

Bước 2. Chọn **File ➔ Create / Update ➔ Create symbol file for current file**, tạo ra file .bsf

Bước 3. Sau khi tạo ra file .bsf, sử dụng lại mạch này bằng cách chọn **Symbol Tool**



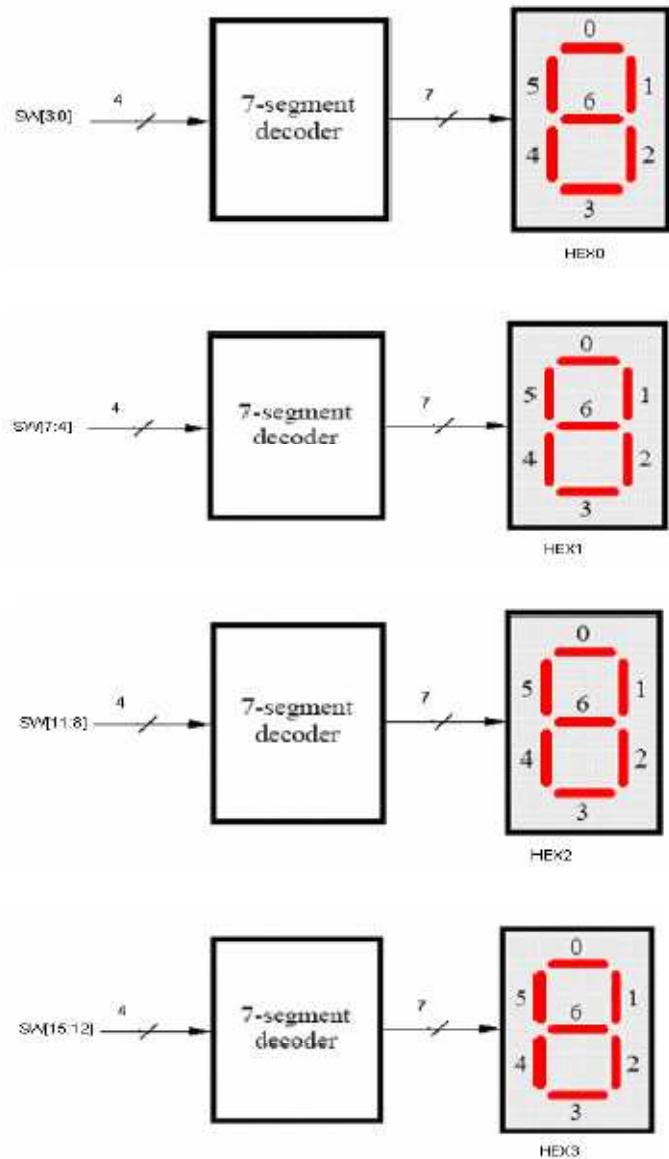
Hình 4.52 Cửa sổ tạo Symbol

Nếu mạch đóng gói sử dụng trên cùng project làm việc, ta có thể thấy symbol tương ứng trong mục Project.

Nếu mạch đóng gói sử dụng không cùng project làm việc, ta có thể lấy symbol bằng cách chỉ đường dẫn đến file .bsf tương ứng.

4.2.2.1 Phản 1

Sử dụng 16 Switch và 4 đèn Led 7 đoạn thiết kế mạch như hình Hình 4.53. Lưu ý mỗi Led 7 đoạn chỉ hiển thị giá trị từ 0 đến 9, các giá trị từ 1010 đến 1111 sẽ không cần xem xét.



Hình 4.53 Mạch giải mã hiện thị HEXA

Từng bước thực hiện:

- Bước 1. Tạo project Quartus mới, đặt tên `user_dir/lab2/part1`
- Bước 2. Thiết kế mạch như Hình 4.53
- Bước 3. Gán pin cho mạch hình trên
- Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Tạo file Vector Waveform (.vwf) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 6. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

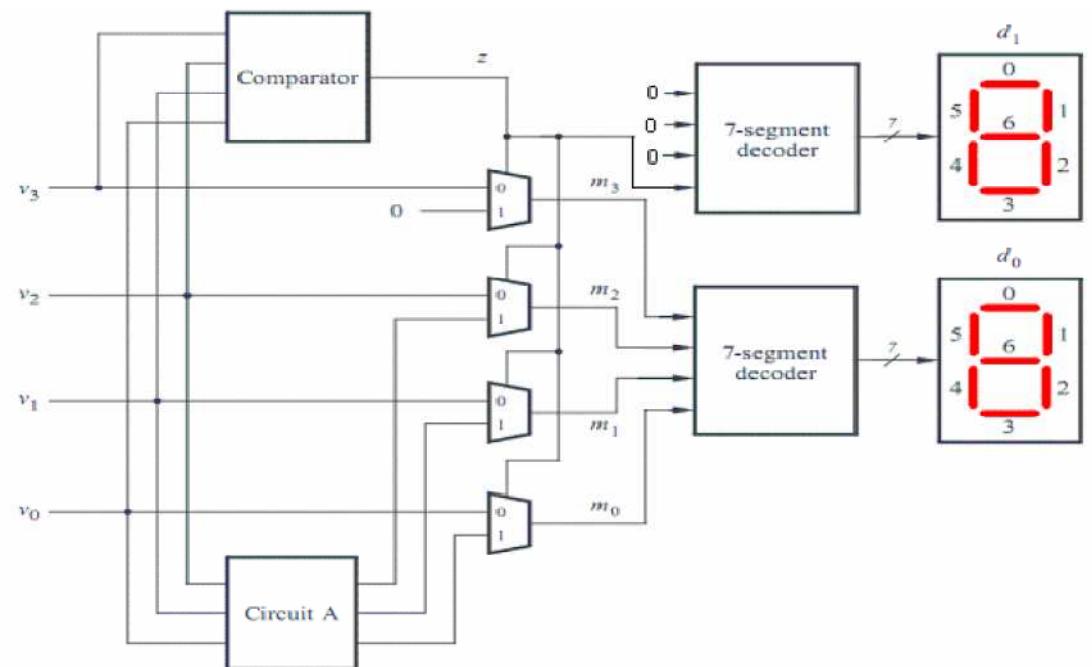
Chú ý: Sinh viên chuẩn bị trước những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ tính vắng buổi đó)

➡ Đọc và thực hiện các bước từ 1 đến 5 ở nhà

4.2.2.2 Phản 2

Phản I đã chuyển từ số nhị phân 4 bit sang số thập phân dùng 1 đèn led 7 đoạn. Yêu cầu của Phản II là sử dụng 2 led 7 đoạn để có thể biểu diễn thêm các số thập phân từ 10 đến 15.

Một phần thiết kế của mạch này được gợi ý như hình Hình 4.54



Hình 4.54 Mạch hiển thị từ 0 đến 15

Mạch hình 2 bao gồm 1 khối comparator để kiểm tra khi nào giá trị của số nhị phân 4 bit (v3v2v1v0) lớn hơn 9 và sử dụng output của comparator này điều khiển led 7 đoạn.

Từng bước thực hiện:

Bước 1. Tạo project Quartus mới, đặt tên user_dir/lab2/part2

Bước 2. Thiết kế mạch như hình Hình 4.54

Bước 3. Mô tả hoạt động của mạch trên

Bước 4. Thiết kế các khối Comparator , Circuit A

Bước 5. Gán pin cho mạch trên

- Input v3v2v1v0 được gán tới SW[3:0]

- Output 2 Led 7 đoạn được gán tới HEX0 và HEX1

Bước 6. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 7. Tạo file Vector Waveform (.vwf) và chạy mô phỏng để kiểm tra hoạt động của mạch.

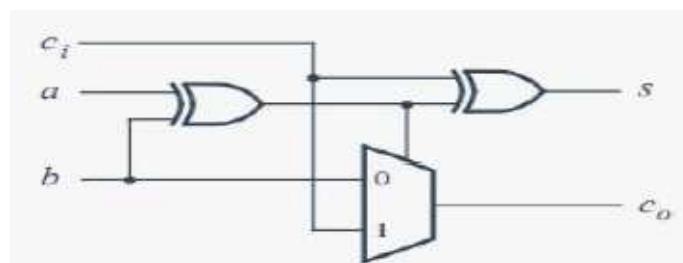
Bước 8. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý: Sinh viên chuẩn bị trước những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

 Đọc và thực hiện các bước từ 1 đến 7

4.2.2.3 Phần 3

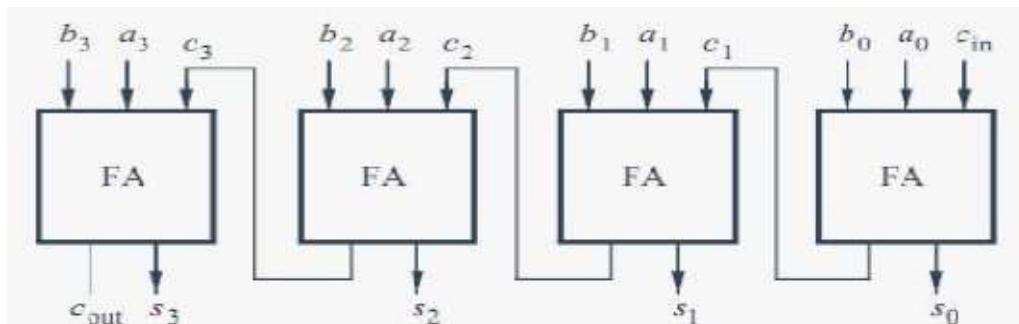
1. Cho mạch sau



Hình 4.55 Mạch FA

Bước 1. Nêu hoạt động của mạch trên và viết bảng sự thật cho mạch

2. Cho mạch sau



Hình 4.56 Mạch cộng FA 4 bit

Trong hình Hình 4.56, FA là 1 khối thể hiện cho mạch ở hình Hình 4.55.

Bước 2. Nêu ý nghĩa hoạt động của mạch này.

Bước 3. Tạo project Quartus mới, đặt tên user_dir/lab2/part3

Bước 4. Thiết kế mạch như hình Hình 4.55, sau đó sử dụng mạch này để thiết kế mạch như Hình 4.56

Bước 5. Gán pin

- Input a3a2a1a0 được gán tới SW[3:0], input b3b2b1b0 được gán tới SW[7:4], input cin được gán tới SW[8].
- Output s3s2s1s0 và cout lần lượt được gán tới LEDG[3:0] và LEDG[4].
- Kết nối các SW ra LEDR để kiểm tra giá trị nhập vào.

Bước 6. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 7. Tạo file Vector Waveform (.vWF) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 8. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Sinh viên chuẩn bị trước những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

 Thực hiện các bước từ 1 đến 6

4.2.2.4 Phần 4

Thiết kế mạch cộng 2 số BCD

Bước 1. Tạo project Quartus mới, đặt tên user_dir/lab2/part4

Bước 2. Thiết kế mạch cộng 2 số BCD với input của mạch là 2 số BCD
một chữ số A, B và 1 bit cin.

Bước 3. Gán pin

- Sử dụng SW[3:0] cho A, SW[7:4] cho B và SW[8] cho cin. Kết nối các SW này ra LEDR để kiểm tra kết quả. Giá trị BCD của A được hiển thị trên HEX6, của B hiển thị trên HEX4.
- Output là số BCD 2 chữ số S1S0.
- Sử dụng các LEDG để hiển thị kết quả của 4 bit tổng và số dư. Giá trị BCD của S1S0 được hiển thị trên HEX1 và HEX0

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Tạo file Vector Waveform (.vwf) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 6. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Sinh viên chuẩn bị trước những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ tính vắng buổi đó)

 Thực hiện các bước từ 1 đến 5

4.2.2.5 Phần 5

Thiết kế mạch cộng 2 số BCD hai chữ số A_1A_0 và B_1B_0 , tổng là số BCD ba chữ số $S_2S_1S_0$.

Bước 1. Tạo project Quartus mới, đặt tên user_dir/lab2/part5

Bước 2. Thiết kế mạch cộng 2 số BCD hai chữ số với input của mạch là 2 số BCD hai chữ số A1A0 và B1B0.

Bước 3. Gán pin

- Sử dụng SW[15:8] cho A1A0, SW[7:0] cho B1B0. Kết nối các SW này ra LEDR để kiểm tra kết quả. Giá trị BCD của A1A0 được hiển thị trên HEX7 và HEX6, của B1B0 hiển thị trên HEX5 và HEX4.
- Output là số BCD 3 chữ số S1S0 S2.
- Giá trị BCD của S1S0 S2 được hiển thị trên HEX2, HEX1 và HEX0

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Tạo file Vector Waveform (.vwf) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 6. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Sinh viên chuẩn bị trước những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

 Thực hiện các bước từ 1 đến 5

4.2.3 Bài thực hành số 3 – Latch, Flip-flop, Register

Mục đích: Hiểu hoạt động, thiết kế Latch, Flip-flop và Register.

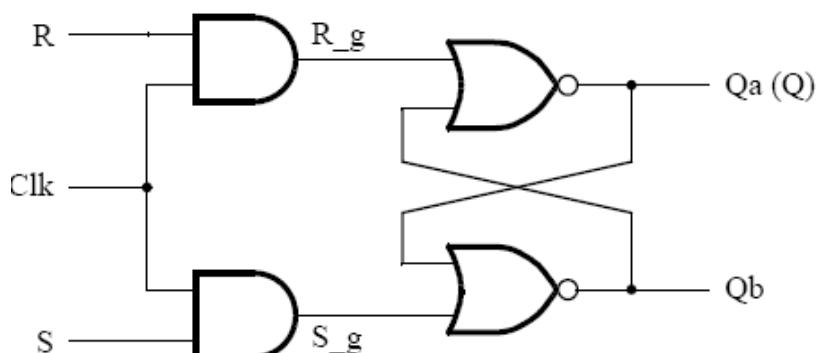
4.2.3.1 Phần 1

Thiết kế một mạch điện hết sức đơn giản như sau:

Từng bước thực hiện:

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab3/lab3_part1

Bước 2. Thiết kế một mạch như sau:



Hình 4.57 Mạch latch

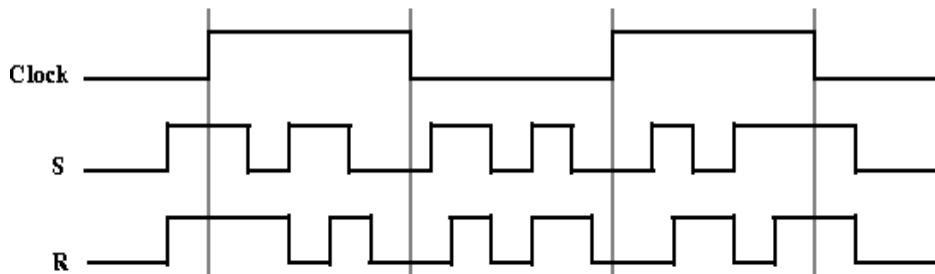
Bước 3. Gán pin cho mạch trên với SW[0] làm xung CLK, SW[1] điều khiển input R, SW[2] điều khiển input S, gán output Qa tới LEDR0, Qb tới LEDR1

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Tạo file Vector Waveform (.vWF) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 6. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

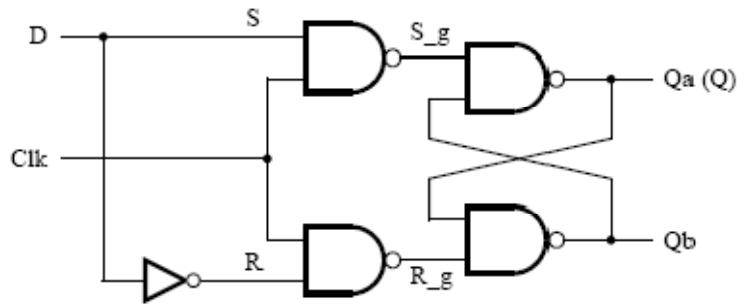


Hình 4.58 Giản đồ xung input của mạch latch RS

- Vẽ giản đồ xung với inputs Clock, S, R như hình trên ➔ output Qa tương ứng.
- Đọc và thực hiện các bước 1 đến 5 ở nhà.

4.2.3.2 Phần 2

Cho mạch sau:



Hình 4.59 mạch D latch

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab3/lab3_part2

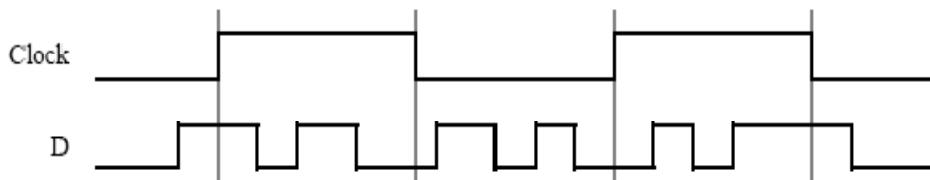
Bước 2. Gán pin cho mạch trên với SW[0] làm xung CLK, SW[1] điều khiển input D, gán output Q tới LEDR0.

Bước 3. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 4. Tạo file Vector Waveform (.vwf) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 5. Nạp file thực thi lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)



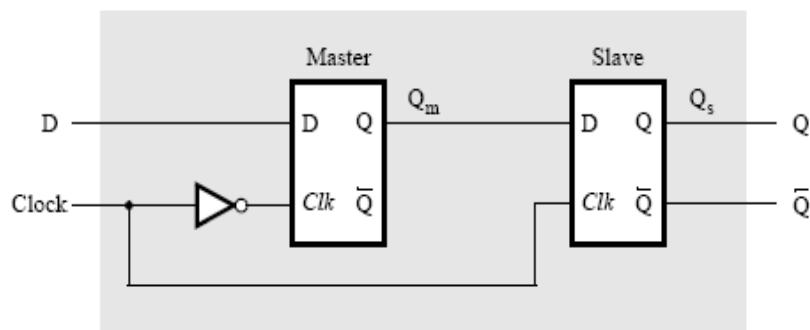
Hình 4.60 Giản đồ xung input của D latch

└ Vẽ giản đồ xung với inputs Clock, D như hình trên ➔ output Qa tương ứng.

└ Đọc và thực hiện các bước 1 đến 4 ở nhà.

4.2.3.3 Phản 3

Cho một D flip-flop Master-Slaver sau, với Master và Slave là hai D latch:



Hình 4.61 Mạch D-Flipflop

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab3/lab3_part3

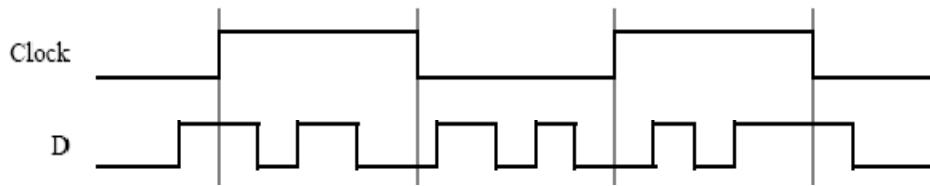
Bước 2. Gán pin cho mạch trên với SW[0] làm xung CLK, SW[1] điều
khiển input D, gán output Q tới LEDR0.

Bước 3. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 4. Tạo file Vector Waveform (.vwf) và chạy mô phỏng để kiểm tra
hoạt động của mạch.

Bước 5. Nạp file thực thi lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có
bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

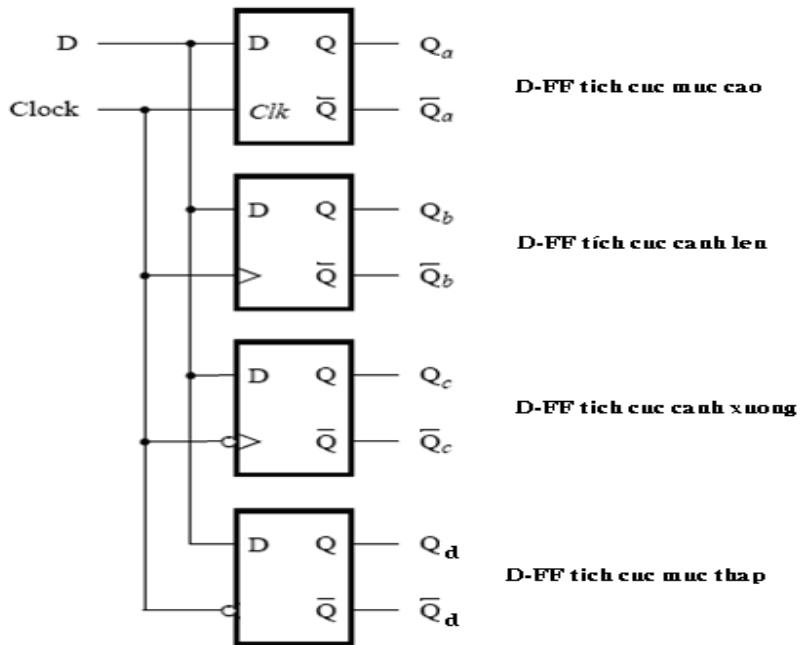


Hình 4.62 Giản đồ xung input D Flipflop

- Vẽ giản đồ xung với inputs Clock, D như hình trên \Rightarrow output Q tương ứng.
- Nếu sự khác nhau giữa D flip-flop Master-Slaver với D latch, tại sao ta phải sử dụng D flip-flop Master-Slaver.
- Đọc và thực hiện các bước 1 đến 4 ở nhà.

4.2.3.4 Phản 4

Cho mạch sau với một D-FF tích cực mức cao, một D-FF tích cực cạnh lên, một D-FF tích cực cạnh xuống và một D-FF tích cực mức thấp. Thiết kế mạch tạo D-FF tích cực cạnh lên và D-FF tích cực cạnh xuống và D-FF tích cực mức thấp (có thể tham khảo trên Internet), sau đó thiết kế mạch như hình dưới:



Hình 4.63 Latch và Flipflop

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab3/lab3_part4

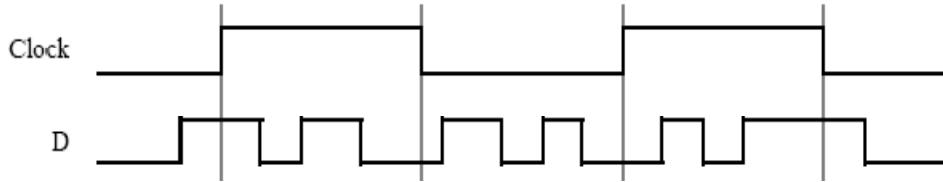
Bước 2. Gán pin cho mạch trên với SW[0] làm xung CLK, SW[1] điều khiển input D, gán output Qa tới LEDR0, Qb tới LEDR1, Qc tới LEDR2, Qd tới LEDR3.

Bước 3. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 4. Tạo file Vector Waveform (.vwf) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 5. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)



Hình 4.64 Giản đồ xung input

- ✚ Với giản đồ xung input Clock, D như trên hãy vẽ tín hiệu xung ngõ ra Qa, Qb, Qc và Qd.
- ✚ Đọc và thực hiện các bước 1 đến 4 ở nhà.

4.2.3.5 Phần 5

Thiết kế một mạch hiển thị LED 7 đoạn với yêu cầu như sau:

- ❖ Hiển thị số Hexa A : 1357 lên 4 LED HEX[3:0], với số 1357 được nhập từ từ 16 switchs SW[15:0]
- ❖ Hiển thị số Hexa B : 2468 lên 4 LED HEX[7:4], với số 2468 cũng được nhập từ từ 16 switchs SW[15:0]
- ❖ Điều khiển SW để số A hiển thị trước, sau đó điều khiển SW để hiển thị số B trong khi số A vẫn được giữ không đổi.
- ❖ Mạch có thể Reset lại để tất cả LED HEX[7:0] trở về 0. Và nhập lại giá trị bất kì cho A và B.

Gợi ý: Chốt giá trị inputs của số A vào các mạch latches. Lấy nút nhấn KEY0 như là tín hiệu CLK và KEY1 là tín hiệu RESET bắt đồng bộ.

Chú ý: Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm → Tính vắng buổi đó)

- ✚ Thiết kế sơ đồ khối của mạch (trên bài chuẩn bị)
- ✚ Thiết kế mạch chi tiết trên QuartusII.

- Tạo file Vector Waveform (.vWF) và chạy mô phỏng để kiểm tra hoạt động của mạch.
- Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

4.2.4 Bài thực hành số 4 – Bộ đếm (Counters)

Mục đích: tìm hiểu hoạt động và thiết kế bộ đếm

4.2.4.1 Phần 1

Tìm hiểu về bộ đếm tuần tự (hay còn gọi là bộ đếm không đồng bộ hoặc nối tiếp) sử dụng T flipflop

Từng bước thực hiện

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab4/lab4_part1

Bước 2. Thiết kế 3 bộ đếm với yêu cầu như sau:

- ❖ Bộ đếm thứ 1: $M = 8$ (M là hệ số đếm hay còn gọi là dung lượng bộ đếm), đếm từ 0 đến 7
- ❖ Bộ đếm thứ 2: Bộ đếm ngược từ 7 tới 0
- ❖ Bộ đếm thứ 3: $M = 6$, đếm từ 0 đến 5

Bước 3. Gán pin cho từng mạch trên

- KEY0 cho Clock và SW1, SW0 cho Enable và Reset input của các flipflop.
- LED 7 đoạn hiển thị kết quả dưới dạng số thập phân.

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof

Bước 5. Tạo file Vector Waveform (.vwf) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 6. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Sinh viên thực hiện trước các công việc sau (không bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

─ ┌─┐ Từ bước 1 đến bước 5

4.2.4.2 Phần 2

Tìm hiểu về bộ đếm song song (hay còn gọi là bộ đếm đồng bộ).

Từng bước thực hiện

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab4/lab4_part2

Bước 2. Thiết kế các bộ đếm với yêu cầu như sau:

- ❖ Bộ đếm thứ 1: $M = 16$, đếm từ 0 đến 15 (sử dụng T_flipflop)
- ❖ Bộ đếm thứ 2: $M = 5$, đếm từ 0 đến 4 (sử dụng JK_flipflop)

Bước 3. Gán pin cho từng mạch trên

- KEY0 cho Clock và SW1, SW0 cho Enable và Reset input của các flipflop.

- LED 7 đoạn hiển thị kết quả dưới dạng số thập phân.

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof

Bước 5. Tạo file Vector Waveform (.vWF) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 6. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Sinh viên thực hiện trước các công việc sau (không bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

■ Thực hiện các công việc từ bước 1 đến bước 5

■ Nhận xét ưu khuyết điểm của bộ đếm tuần tự và bộ đếm song song

4.2.4.3 Phần 3

Sử dụng xung clock 1Mhz được cung cấp (file kèm) để thiết kế mạch như sau: dùng 8 LED 7 đoạn (HEX7 tới HEX0) hiển thị chữ HELLO sao cho cứ sau khoảng thời gian xấp xỉ 1s, chữ sẽ được di chuyển từ phải sang trái như bảng sau:

Chu kì clock	Giá trị hiển thị
0	H E L L O
1	H E L L O
2	H E L L O
3	H E L L O
4	E L L O H
5	L L O H E
6	L O H E L
7	O H E L L
8	H E L L O
...	

Hình 4.65 Bảng nội dung hiển thị

Sinh viên thực hiện trước các công việc sau (không bài chuẩn bị không được vào lớp làm thí nghiệm ➔ tính vắng buổi đó)

- ─ Thiết kế mạch, gán pin, biên dịch, chạy mô phỏng trên Quartus

4.2.5 Bài thực hành số 5 – Adder, Subtractor, Multiplier of two signed numbers in 2's-complement form

Mục đích: Tìm hiểu phương pháp thiết kế một số mạch số học như là mạch cộng, mạch trừ và mạch nhân 2 số có dấu dưới dạng bù -2.

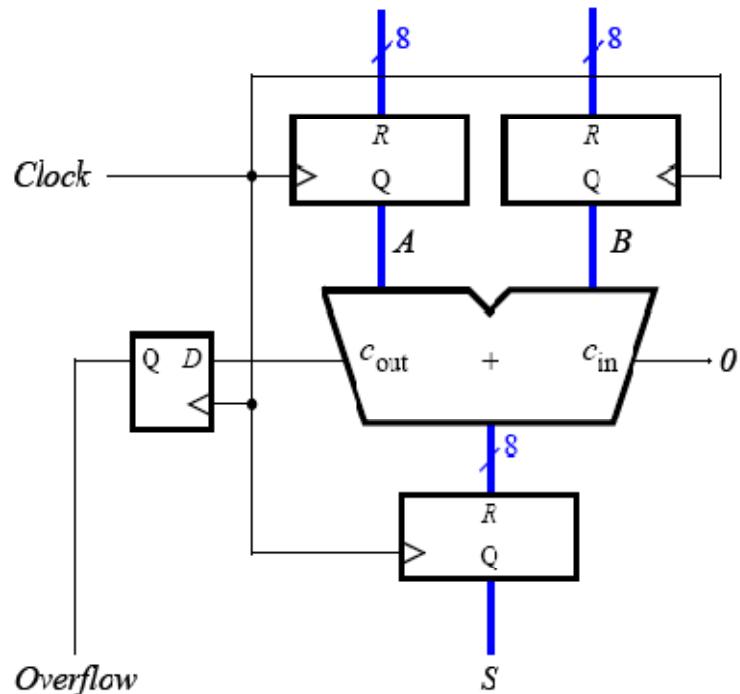
4.2.5.1 Phần 1

Thiết kế một mạch cộng 2 số 8 bit có dấu dạng bù -2.

Từng bước thực hiện:

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab5/lab5_part1

Bước 2. Dựa vào mạch hình dưới, thiết kế mạch cộng 2 số 8 bit có dấu
dạng bù -2.



Hình 4.66 Mạch cộng hai số có dấu bù 2

Bước 3. Gán pin cho mạch trên như sau:

- SW[15:8] => A[7:0]
- SW[7:0] => B[7:0]
- KEY0 là tín hiệu xung Clock
- LEDR[7:0] => S[7:0]
- LEDG8 => Overflow ; Overflow = 1 khi S nằm ngoài giá trị biểu diễn của số có dấu 8 bit dạng bù -2.
- HEX[7:6] hiện thị giá trị hexadecimal của số A.
- HEX[5:4] hiện thị giá trị hexadecimal của số B.
- HEX[1:0] hiện thị giá trị hexadecimal của số S.

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Tạo file Vector Waveform (.vWF) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 6. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm → Tính vắng buổi đó)

➡ Đọc và thực hiện các bước từ 1 đến 5 ở nhà.

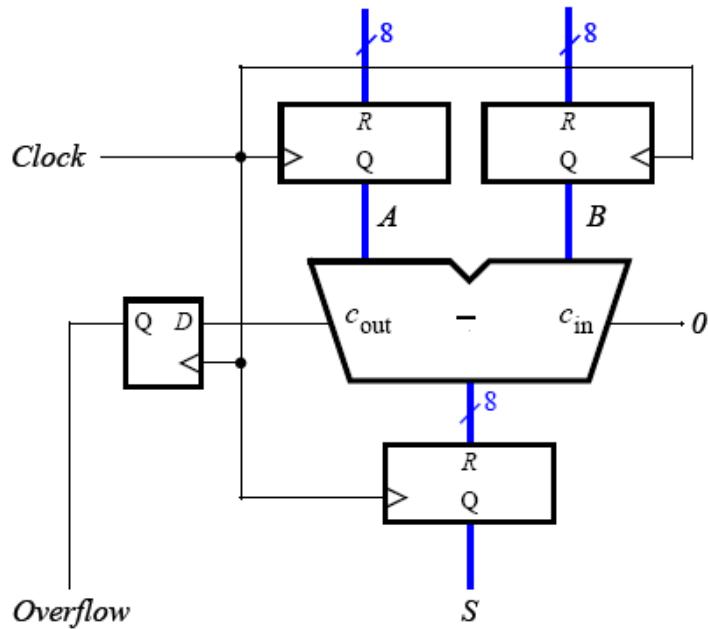
4.2.5.2 Phần 2

Thiết kế một mạch trừ 2 số 8 bit có dấu dạng bù -2.

Từng bước thực hiện:

Bước 1. Tạo một project Quartus mới, đặt tên: user_dir/lab5/lab5_part2

Bước 2. Dựa vào mạch hình dưới, thiết kế mạch trừ 2 số 8 bit có dấu dạng bù -2.



Hình 4.67 Mạch cộng hai số có dấu bù 2

Bước 3. Gán pin cho mạch trên như sau:

- SW[15:8] => A[7:0]
- SW[7:0] => B[7:0]
- KEY0 là tín hiệu xung Clock
- LEDR[7:0] => S[7:0]
- LEDG8 => Overflow; Overflow = 1 khi S nằm ngoài giá trị biểu diễn của số có dấu 8 bit dạng bù -2.
- HEX[7:6] hiện thị giá trị hexadecimal của số A.
- HEX[5:4] hiện thị giá trị hexadecimal của số B.
- HEX[1:0] hiện thị giá trị hexadecimal của số S.

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Tạo file Vector Waveform (.vWF) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 6. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý: Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

- Đọc và thực hiện các bước từ 1 đến 5 ở nhà.

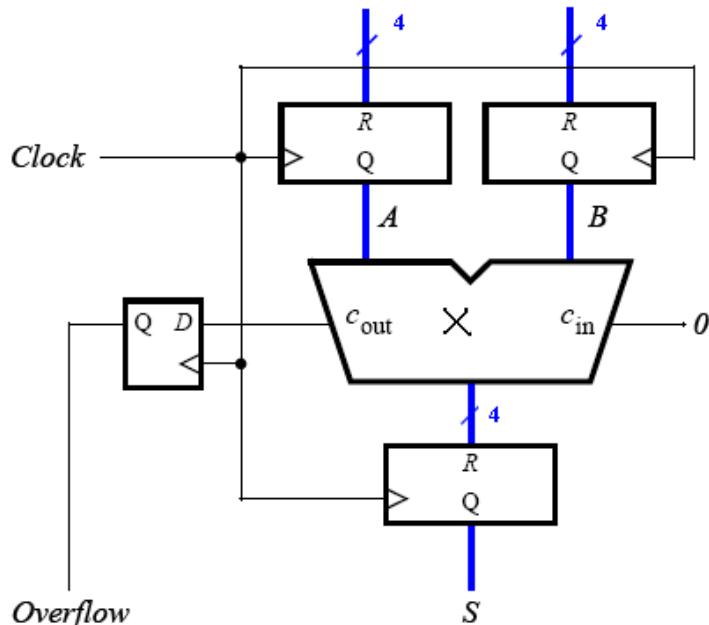
4.2.5.3 Phần 3

Thiết kế một mạch nhân 2 số 4 bit có dấu dạng bù -2.

Từng bước thực hiện:

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab5/lab5_part3

Bước 2. Dựa vào mạch hình dưới, thiết kế mạch nhân 2 số 4 bit có dấu dạng bù -2.



Hình 4.68 Mạch nhân hai số có dấu bù 2

Bước 3. Gán pin cho mạch trên như sau:

- SW[11:8] => A[3:0]
- SW[3:0] => B[3:0]
- KEY0 là tín hiệu xung Clock
- LEDR[3:0] => S[3:0]
- LEDG8 => Overflow; Overflow = 1 khi S nằm ngoài giá trị biểu diễn của số có dấu 4 bit dạng bù -2.
- HEX[6] hiện thị giá trị hexadecimal của số A.
- HEX[4] hiện thị giá trị hexadecimal của số B.
- HEX[0] hiện thị giá trị hexadecimal của số S.

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Tạo file Vector Waveform (.vWF) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 6. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm → Tính vắng buổi đó)

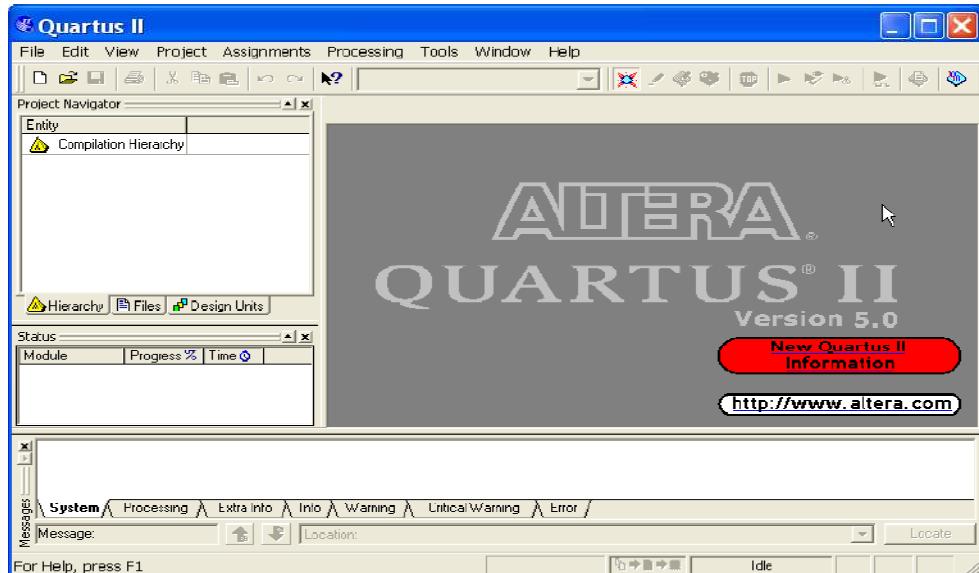
 Đọc và thực hiện các bước từ 1 đến 5 ở nhà.

Chương 5. Hướng dẫn thiết kế và thực hành môn Thiết kế mạch dùng Verilog HDL trên Kit DE2

5.1 Hướng dẫn thực hành

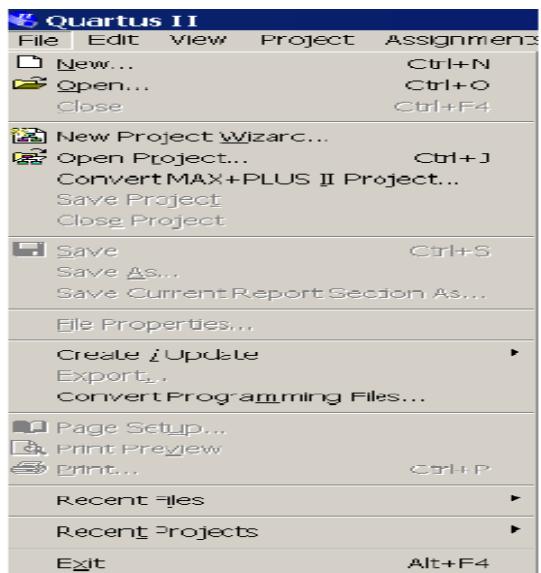
5.1.1 Tạo một project trên Quartus II

Bước 1. Start → Programs → Altera → Quartus II 7.2 → Quartus II 7.2 (32 -Bit)



Hình 5.1 Màn hình chính Quartus

Bước 2. Nhấn tab **File** trên màn hình chính



Hình 5.2 Tab File

Bước 3. Mở một project mới : **File ➔ New Project Wizard...**



Hình 5.3 Tạo project mới

Bước 4. Nhấn **Next**



Hình 5.4 Nhập đường dẫn và tên project

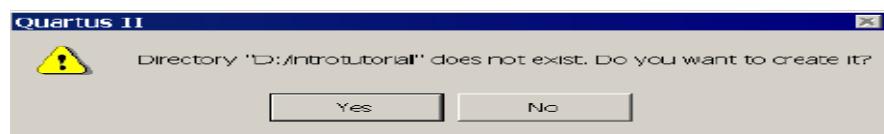
Bước 5. Nhập đường dẫn thư mục của project (có thể tạo trước hoặc nếu chưa tạo sẽ được tự động tạo).

Bước 6. Nhập tên của project.

Bước 7. Nhập top-level của thiết kế cho project (nên cho giống tên của project).

Bước 8. Nhấn **Next**

Bước 9. Nếu đường dẫn thư mục của project chưa được tạo trước :



Hình 5.5 Đường dẫn chưa tồn tại

Bước 10. Nhấn Yes



Hình 5.6 Add các file liên quan đến project

Bước 11. Nhấn Next

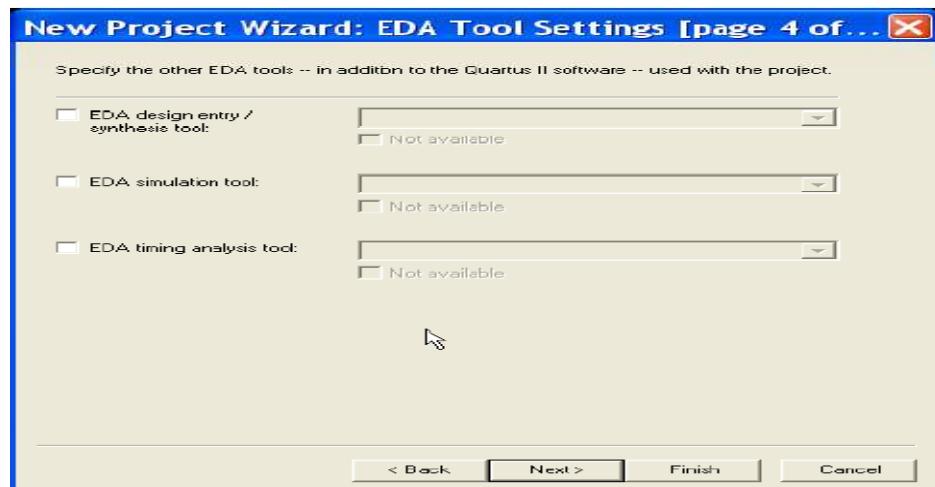


Hình 5.7 Chọn tên FPGA

Bước 12. Chọn Family : **Cyclone II**

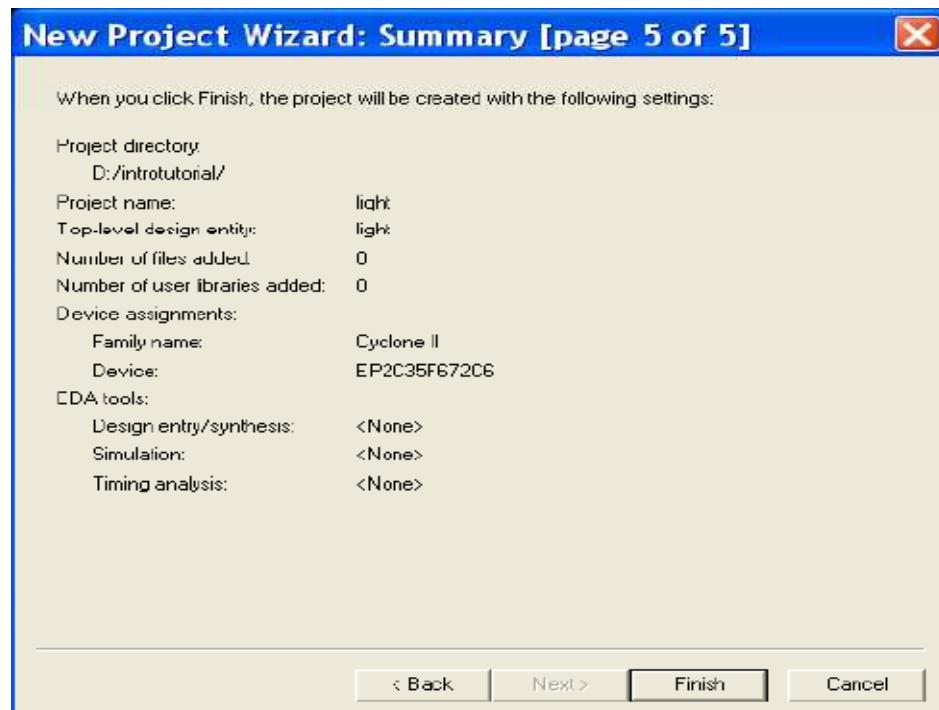
Bước 13. Chọn Available devices : **EP2C35F672C6** (Họ của Chip FPGA Cyclone II trên Kit DE2).

Bước 14. Nhấn **Next**



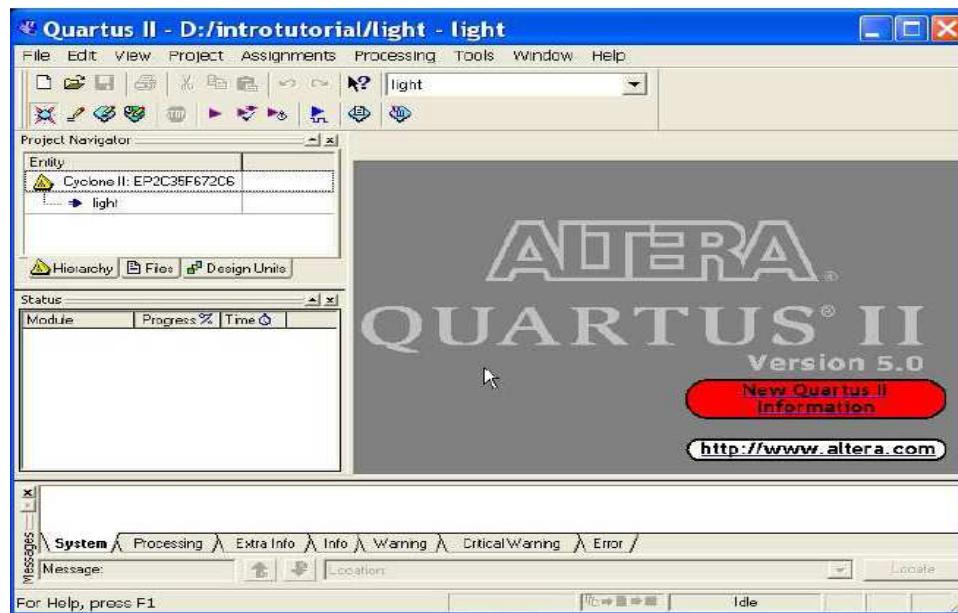
Hình 5.8 Thiết lập thông số EDA

Bước 15. Nhấn **Next**



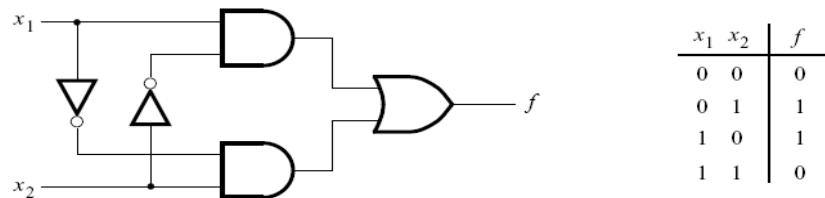
Hình 5.9 Project mới được tạo

Bước 16. Nhấn **Finish** để trở về màn hình chính.



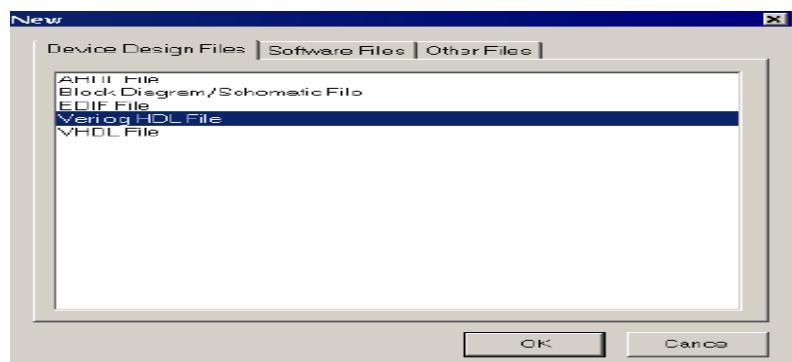
Hình 5.10 Cửa sổ Quartus sau khi project mới được tạo

5.1.2 Thiết kế một mạch điện đơn giản (cổng XOR) dùng Verilog trên Quartus II:



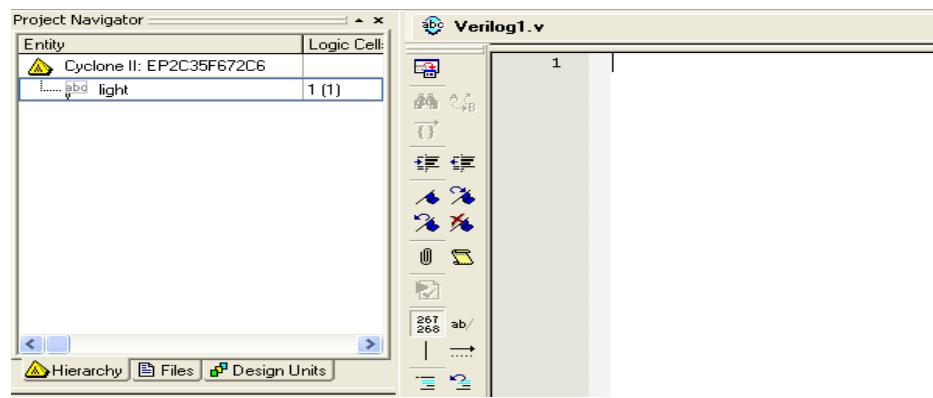
Hình 5.11 Mạch số đơn giản

Bước 1. Mở File ➔ New



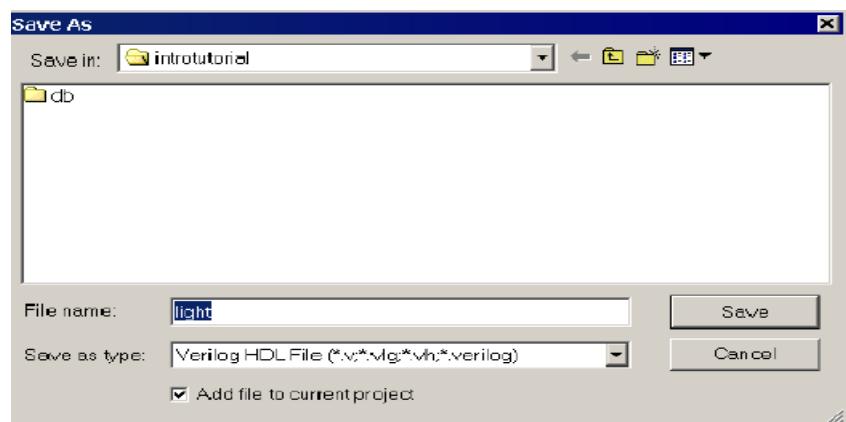
Hình 5.12 Chọn môi trường thiết kế Verilog

Bước 2. Chọn Verilog HDL File



Hình 5.13 Cửa sổ thiết kế Verilog

Bước 3. Save as file : File → Save as



Hình 5.14 Lưu thiết kế

Bước 4. Mô tả thiết kế bằng ngôn ngữ Verilog cho cổng XOR vào cửa sổ Text Editor. Nhớ phải đặt tên của top-level module phải giống tên của Project. (Trong thí dụ này là “light”).

```

module light (SW0, SW1, LEDGO);
    input SW0, SW1;
    output LEDGO;
    assign LEDGO = (SW0 & ~SW1) | (~SW0 & SW1);
endmodule

```

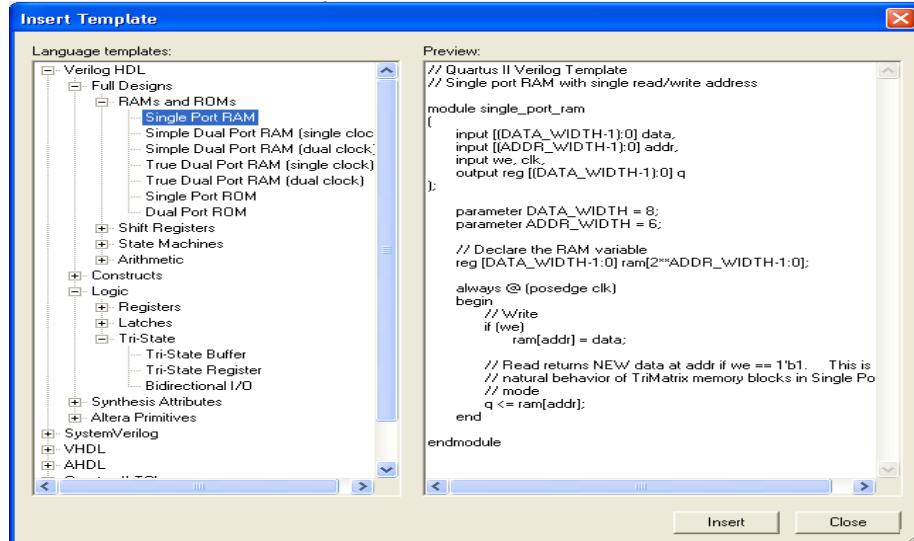
Hình 5.15 Mô tả thiết kế

Bước 5. Save : File → Save

Một số cách tạo Verilog khác :

Dùng Verilog template :

Edit → Insert Template → Verilog HDL

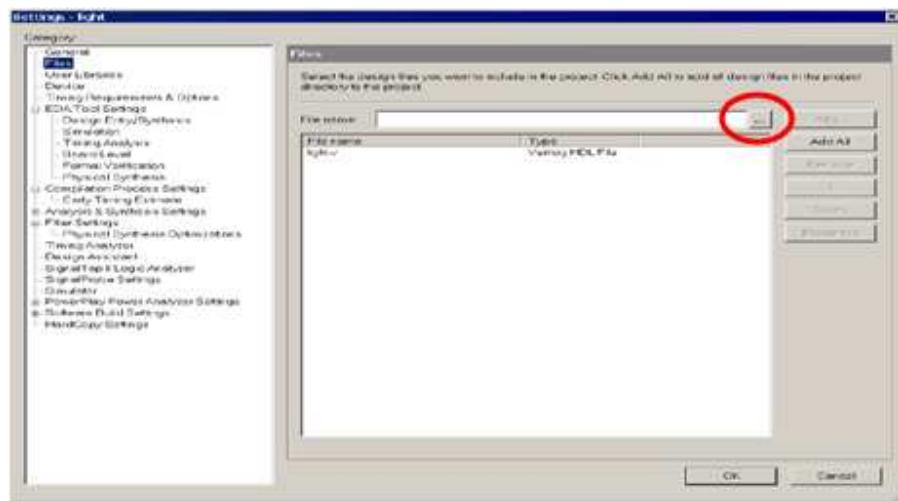


Hình 5.16 Chọn template Verilog

- ⊕ Đưa File vào project (có thể đưa một hoặc nhiều File vào Project, chẳng hạn như File chứa top-level module và những Files chứa sub-level module). Các bước thực hiện

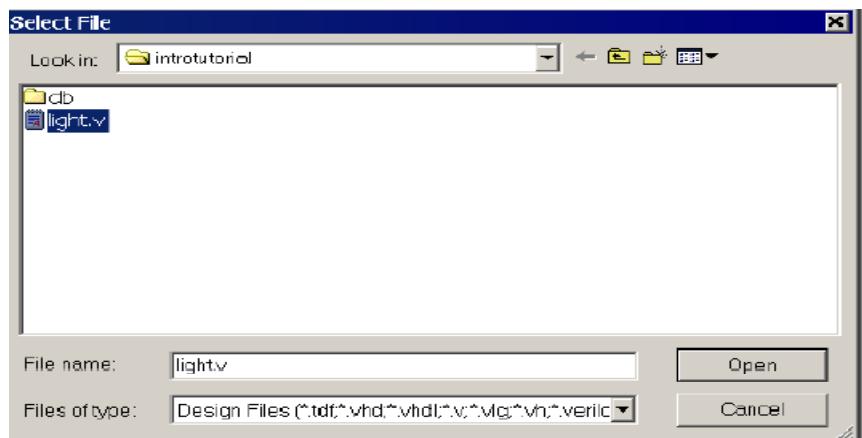
Bước 1. **Assignments → Settings**

Bước 2. Chọn **Category : Files**



Hình 5.17 Add file Verilog liên quan

Bước 3. Nhấn vào **button ...**



Hình 5.18 Chỉ đường dẫn

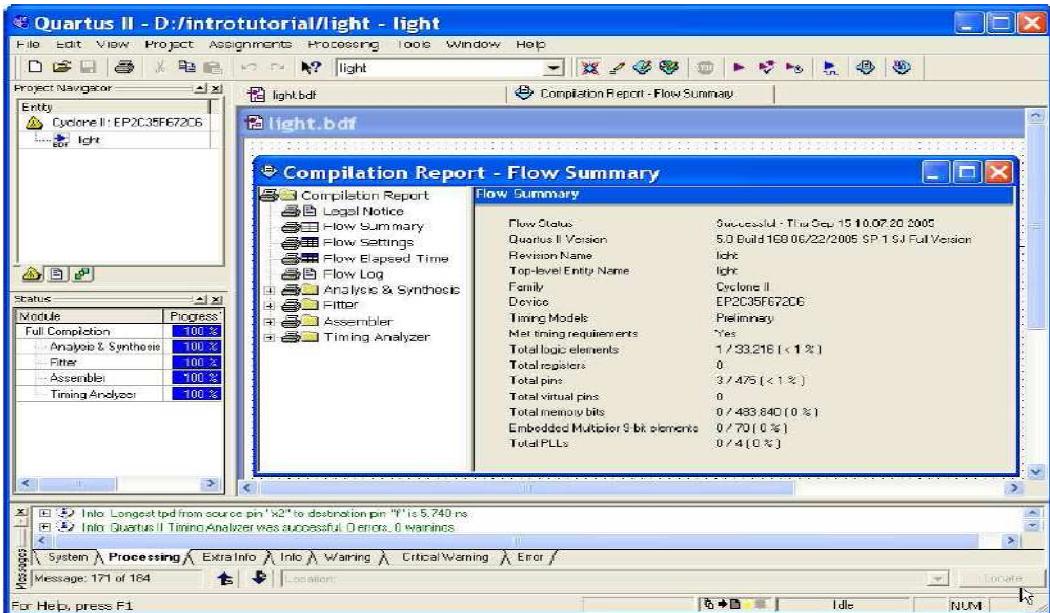
Bước 4. Chỉ đường dẫn của những Files verilog .v cần đưa vào Project.

5.1.2.1 Trình biên dịch

Với dữ liệu vào là file định dạng .v (light.v), nhiều công cụ trong phần mềm Quartus II được dùng để phân tích, tổng hợp mạch đã được thiết kế ở phần trên, rồi sau đó sẽ tạo ra một file thực thi dùng để nạp lên FPGA. Những công cụ được sử dụng trong quá trình này được gọi là trình biên dịch. Để thực thi quá trình biên dịch, ta thực hiện các bước sau:

Bước 1. Chọn: **Processing → Start Compilation** hoặc nhấn chọn biểu

tượng trên thanh công cụ. Sau khi quá trình biên dịch được hoàn tất, một bảng báo cáo được tạo ra như hình dưới



Hình 5.19 Cửa sổ sau quá trình biên dịch

Bước 2. Để xem lại quá trình biên dịch, ta chọn : **Processing ➔ Compilation Report** hoặc nhấn chọn biểu tượng trên thanh công cụ.

5.1.2.2 Message window

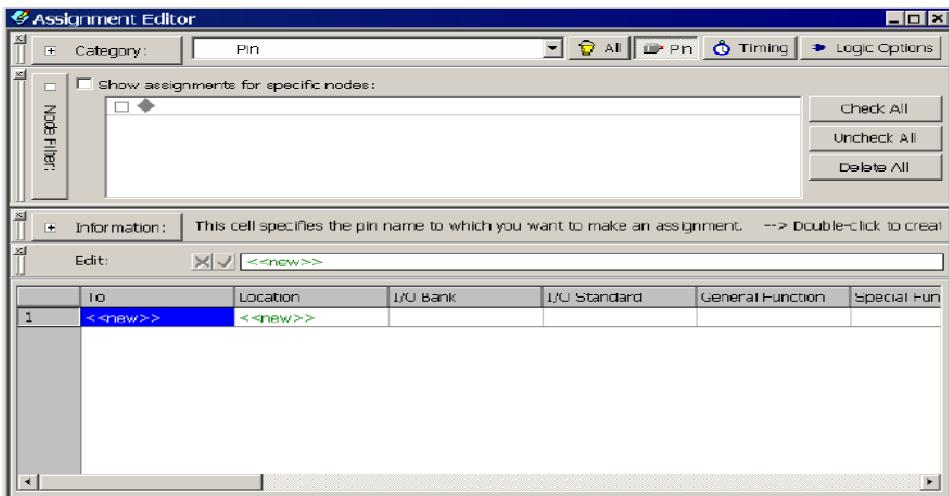
Phần mềm Quartus II sẽ hiển thị thông tin trong suốt quá trình biên dịch trên cửa sổ **Message** widow. Nếu sơ đồ mạch điện được thiết kế trong phần Graphic Editor hoàn toàn đúng, thì một thông báo “The compilation was successful” được hiện thị. Trong trường hợp quá trình biên dịch xuất hiện lỗi thì có nghĩa đã có lỗi xảy ra trong quá trình thiết kế trên Graphic Editor. Mỗi thông báo tương ứng với một lỗi được tìm thấy sẽ xuất hiện trên cửa sổ **Message**. Nhấp đúp vào thông báo lỗi đó ta sẽ biết rõ hơn về lỗi đã xảy ra trên mạch điện. Tương tự, trình biên dịch cũng thông báo một số cảnh báo “Warning”. Ngoài ra ta cũng có

thể tìm hiểu thêm thông tin về lỗi cũng như cảnh báo bằng cách nhấn chọn vào thông báo đó rồi nhấn phím F1 trên bàn phím.

5.1.3 Gán pin

Vì ta chưa thực hiện gán pin trên FPGA cho linh kiện trong mạch điện đã thiết kế ở trên nên khi thực hiện biên dịch thì trình biên dịch Quartus II đã gán chân của linh kiện với pin của FPGA một cách ngẫu nhiên. Tuy nhiên, giả sử trong thiết kế cổng XOR đơn giản ở trên, sau khi thiết kế được biên dịch và nạp lên FPGA, ta muốn hai ngõ vào x1, x2 được điều khiển bởi hai switch SW0 và SW1 còn kết quả ngõ ra f sẽ được thể hiện trên led LEDG0 (SW0, SW1, LEDG0 được ghi trên Kit). Mặt khác ta biết switch SW0 được kết nối cố định với pin N25 của FPGA, tương tự vậy switch SW1 được kết nối cố định với pin N26 của FPGA và led LEDG0 được kết nối cố định với pin AE22 của FPGA. Để thực hiện được điều đó ta phải gán chân linh kiện trên mạch (x1, x2, f) với pin tương ứng trên FPGA (N25, N26, AE22). Để gán pin ta thực hiện các bước sau

Bước 1. Chọn **Assignments > Pins**, một cửa sổ như hình dưới sẽ xuất hiện



Hình 5.20 Cửa sổ mapped pin giữa thiết kế và FPGA

Bước 2. Trong mục **Category** chọn **Pin**. Nhấp đúp lên mục <<new>> trong cột **To**. Một cửa sổ như hình dưới xuất hiện



Hình 5.21 Cửa sổ gán pin

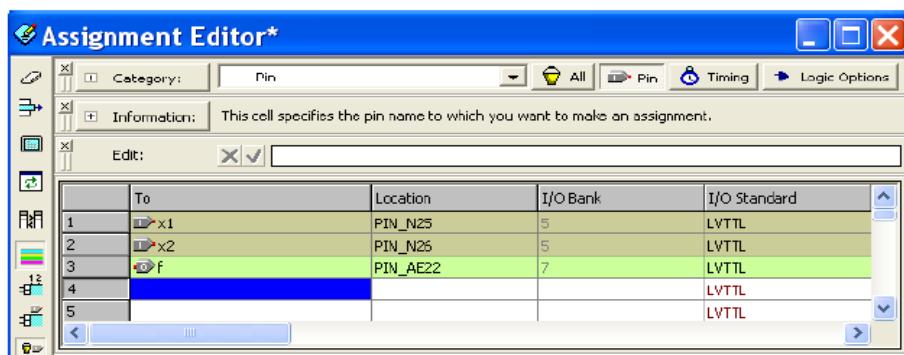
Bước 3. Nhấn chọn x1 để gán pin trước, tiếp đến nhấp đúp lên mục ngay bên phải của x1 trong cột **Location**, một cửa sổ như hình dưới sẽ xuất hiện

Location	I/O Bank	I/O Standard	General Function	Special Fun
		LVTTL		
PIN_N1	I/O Bank 2	Dedicated Clock	CLK1, LVDSCLK0n, Input	
PIN_N2	I/O Bank 2	Dedicated Clock	CLK0, LVDSCLK0p, Input	
PIN_N9	I/O Bank 2	Row I/O	LVDS31p	
PIN_N18	I/O Bank 5	Row I/O	LVDS110p	
PIN_N20	I/O Bank 5	Row I/O	LVDS124p	
PIN_N23	I/O Bank 5	Row I/O	LVDS126p, DPCLK7/DQ50R/CQ1R	
PIN_N24	I/O Bank 5	Row I/O	LVDS126n	
PIN_N25	I/O Bank 5	Dedicated Clock	CLK4, LVDSCLK2p, Input	
PIN_N26	I/O Bank 5	Dedicated Clock	CLK5, LVDSCLK2n, Input	
PIN_P1	I/O Bank 1	Dedicated Clock	CLK3, LVDSCLK1n, Input	
PIN_P2	I/O Bank 1	Dedicated Clock	CLK2, LVDSCLK1p, Input	
PIN_P3	I/O Bank 1	Row I/O	LVDS25p, DPCLK1/DQ51L/CQ1L#	
PIN_P4	I/O Bank 1	Row I/O	LVDS26n	
PIN_P6	I/O Bank 1	Row I/O	LVDS22n	
PIN_P7	I/O Bank 1	Row I/O	LVDS22p	
PIN_P9	I/O Bank 2	Row I/O	LVDS31n	
PIN_P17	I/O Bank 6	Row I/O	LVDS130n	
PIN_P18	I/O Bank 5	Row I/O	LVDS110n	
PIN_P23	I/O Bank 6	Row I/O	LVDS127p, DPCLK6/DQ51R/CQ1R#	
PIN_P24	I/O Bank 6	Row I/O	LVDS127n	

Hình 5.22 Cửa sổ liệt kê danh sách pin của FPGA

Bước 4. Ta nhấp chọn PIN_N25.

Bước 5. Tương tự, ta gán pin cho chân ngõ vào x2 tới pin PIN_N26, và chân ngõ ra f tới pin PIN_AE22. Sau khi gán pin hoàn tất, ta sẽ được như hình dưới



Hình 5.23 Cửa sổ sau gán pin

Bước 6. Lưu lại kết quả gán pin: **File ➔ Save**

Bước 7. Ta phải biên dịch lại thiết kế ở trên với kết quả gán pin này vì như ta đã nói ở trên, vì quá trình biên dịch ở trên, trình biên dịch Quartus

II chỉ gán pin một cách ngẫu nhiên nên sẽ không đúng với yêu cầu thiết kế của ta, do đó ta phải gán lại pin cho đúng với yêu cầu rồi phải chạy lại quá trình biên dịch. Lúc này trình biên dịch Quartus II sẽ sử dụng những pin mà ta đã gán cho chân của mạch điện trong thiết kế để phân tích, tổng hợp và tạo ra một file để thực thi việc nạp xuống cho FPGA.

Ngoài ra ta cũng có một cách khác để gán pins cho design, đặc biệt là rất hữu ích trong thiết kế mà có nhiều chân, ta không thể ngồi gán pin cho từng chân được vì sẽ tốn nhiều thời gian, Quartus II cung cấp một phương pháp giúp ta gán nhiều pin vào hoặc gỡ nhiều pin ra cùng một lúc bằng một file có định dạng đặc biệt dùng cho mục đích này đó là định dạng .CSV. Format của file này như sau

- Nếu ta dùng file text để tạo file này, thì đơn giản ta chỉ cần nhập theo mẫu sau

To, Location

x1, PIN_N25

x2, PIN_N26

f, PIN_AE22

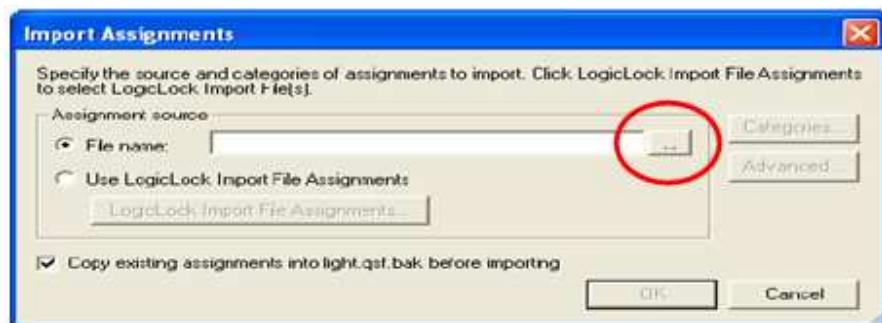
- Nếu ta dùng Microsoft Excel, thì ta sẽ có format như sau:

	A	B
1	To	Location
2	x1	PIN_N25
3	x2	PIN_N26
4	f	PIN_AE22

Hình 5.24 Dùng Microsoft Excel để tạo file gán pin

- Sau khi tạo file có format như trên, ta sẽ thực hiện việc gán pin như sau

Bước 8. Chọn **Assignments** -> **Import Assignments**, một hộp thoại như hình dưới xuất hiện



Hình 5.25 Import file gán pin

Bước 9. Click **button ...**, chỉ đường dẫn của file ta vừa tạo ở trên. Rồi nhấn **OK**.

Để thuận tiện cho người sử dụng Altera đã cung cấp một file CSV có tên DE2_pin_assignments, file này liệt kê tất cả các pin của FPGA, có format như sau:

	A	B	C
7	To	Location	
8	SW[0]	PIN_N25	
9	SW[1]	PIN_N26	
10	SW[2]	PIN_P25	
11	SW[3]	PIN_AE14	
12	SW[4]	PIN_AF14	
13	SW[5]	PIN_AD13	
14	SW[6]	PIN_AC13	
15	SW[7]	PIN_C13	
16	SW[8]	PIN_B13	

Hình 5.26 File gán pin tạo sẵn bởi Altera

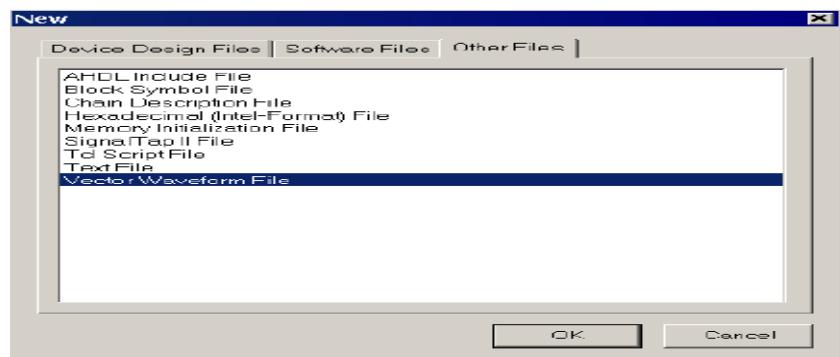
Nếu ta muốn sử dụng file có sẵn này vào việc gán pin cho thiết kế của ta thì một yêu cầu bắt buộc khi ta đặt tên cho chân linh kiện phải trùng với tên trong cột **To** của file này. Thí dụ, nếu ta muốn hai chân ngõ vào của cổng XOR được điều khiển bởi hai Switch 0 và Switch 1 trên Kit DE2 thì ta phải đặt tên cho hai chân này lần lượt là SW[0], SW[1] như trong cột **To** của file này. Do đó ta phải tham khảo file này trước khi đặt tên cho chân linh kiện để khi gán pin ta sẽ rất thuận tiện đó là không phải tạo file.csv nữa mà chỉ cần Import file có sẵn này vào thôi.

Sau khi gán pin xong, ta biên dịch lại.

- ─ Re-compiling design : **Processing → Start Compilation**
- ─ Review Compilation report : **Processing → Compilation Report**

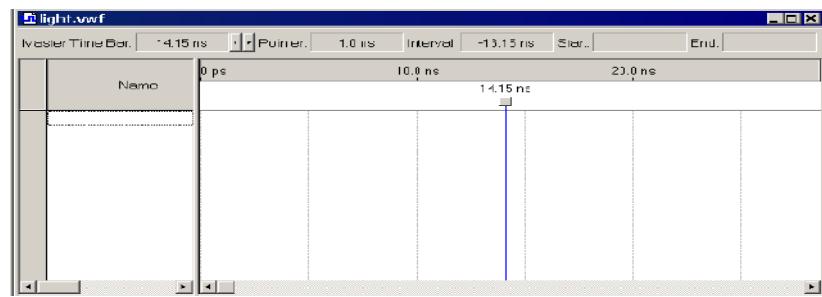
5.1.4 Mô phỏng mạch đã thiết kế :

Bước 1. Tạo input waveform : **File → New → Other Files → Vector Waveform File**



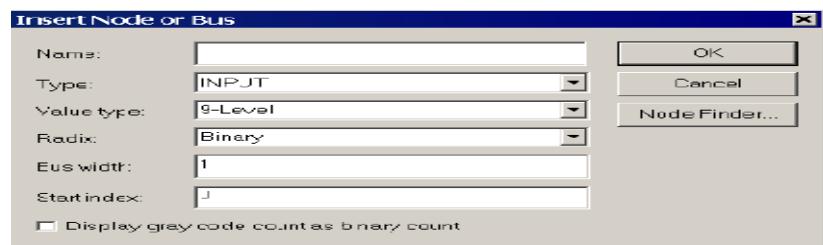
Hình 5.27 Tạo waveform

Bước 2. Nhấn **OK**



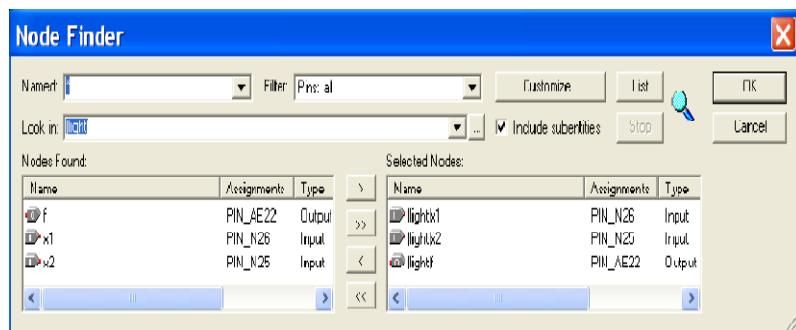
Hình 5.28 Cửa sổ tạo waveform

- Bước 3. Chọn thời gian thực hiện mô phỏng : **Edit → End Time**
- Bước 4. Nhập thời gian thực hiện mô phỏng.
- Bước 5. Fit window : **View → Fit in Window**
- Bước 6. Tạo waveform cho inputs : **Edit → Insert Node or Bus**



Hình 5.29 Nhập tên signal của thiết kế

Bước 7. Chọn Node Finder



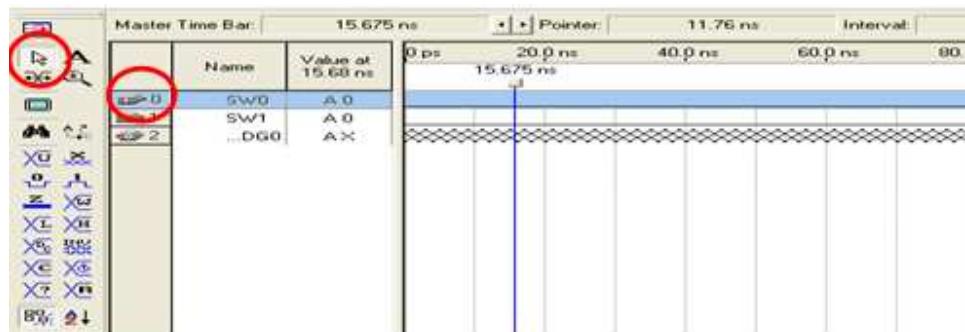
Hình 5.30 Dùng chức năng Node Finder

Bước 8. Chọn Filter : Pins : all

Bước 9. Nhấn button List

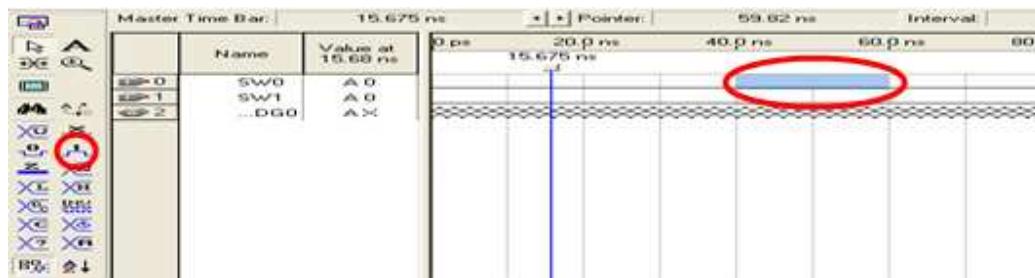
Bước 10. Chọn signal bên **Nodes found** ; nhấn >> để chuyển sang bên **Selected Nodes**

Bước 11. Nhấn OK



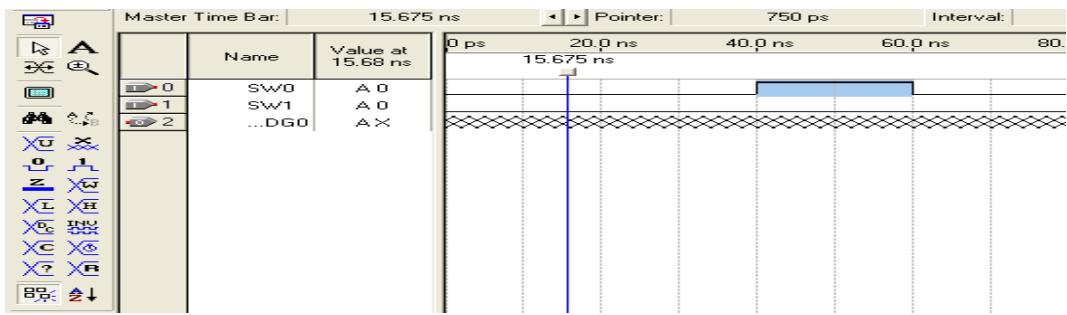
Hình 5.31 Tạo input waveform

- Bước 12. Chọn một input signal bằng cách nhấp chuột vào signal đó .
- Bước 13. Chọn biểu tượng mũi tên con trỏ
- Bước 14. Di chuyển con trỏ sang màn hình waveform .
- Bước 15. Nhấn và giữ chuột và kéo rê (left) trong một khoảng thời gian (giả sử ta muốn trong khoảng thời gian từ 40ns -> 60 ns , SW0 signal có giá trị “1”, thì ta nhấn , giữ và rê chuột trong khoảng thời gian từ 40ns -> 60ns.



Hình 5.32 Tạo mức logic "1"

- Bước 16. Nhấn button “1” phía bên trái màn hình

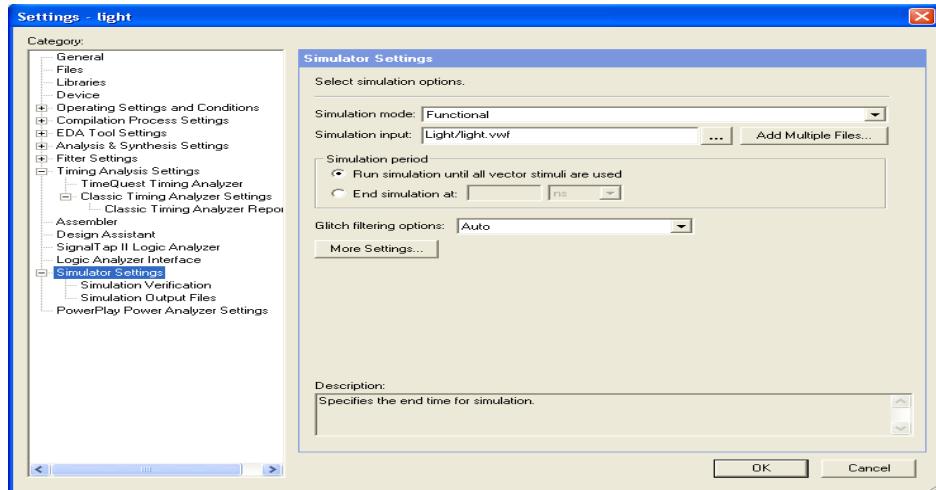


Hình 5.33 Mức logic "1" đã được tạo

Bước 17. Tương tự cho những tín hiệu inputs khác, không tạo waveform cho outputs (XXX).

Bước 18. Save File Waveform : **File ➔ Save As**

Bước 19. Thiết lập thực hiện mô phỏng : **Assignments ➔ Setting**



Hình 5.34 Thiết lập chế độ simulation

Bước 20. Chọn **Simulator Settings**

Bước 21. Chọn **Simulation mode : Functional / Timing**

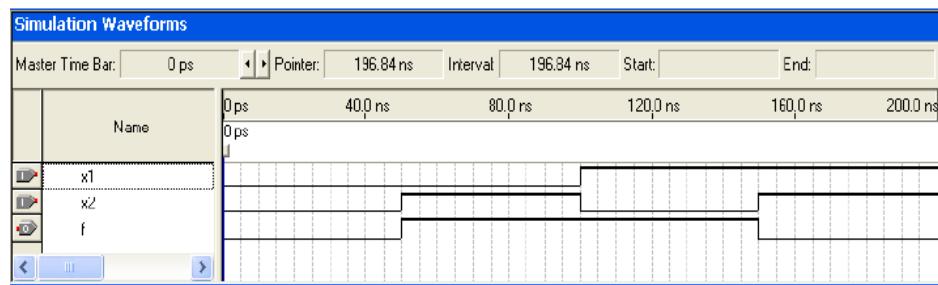
Bước 22. Chỉ đường dẫn của input waveform vừa tạo.

Bước 23. Nhấn **OK**

Bước 24. Tạo simulation netlist : **Processing → Generate Functional Simulation Netlist**

Bước 25. Chạy mô phỏng : **Processing → Start Simulation.**

Bước 26. Quan sát waveform của Output và debug nếu có lỗi.



Hình 5.35 Waveform sau khi chạy mô phỏng

5.1.5 Programming mạch đã thiết kế lên FPGA :

Bước 1. Kết nối Kit DE2 với máy tính qua cổng USB-Blaster (phải cài đặt driver trước).

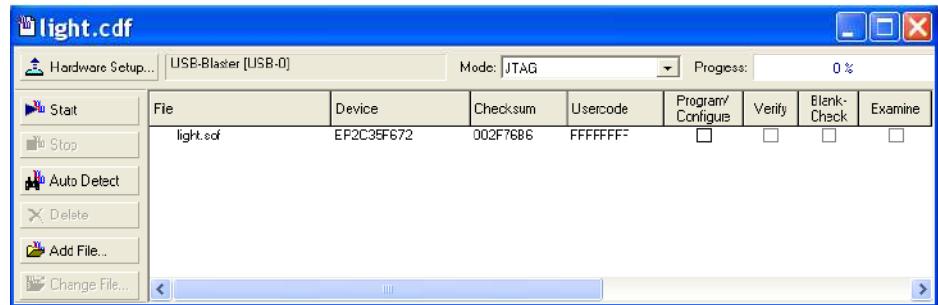
Bước 2. Bật nguồn Kit DE2.

Có 2 mode cho việc programming : JTAG và Active Serial modes

JTAG mode

Bước 3. Trên Kit DE2 , chuyển Switch **RUN/PROG** về vị trí **RUN**

Bước 4. Trên màn hình chính Quantus II, chọn **Tools → Programmer**



Hình 5.36 Nạp thiế́ kέ lên FPGA

Bước 5. Nhấn **Hardware Setup**, chọn **USB-Blaster[USB-0]** (Chú ý : phải cài đặt driver cho USB-Blaster trước).



Hình 5.37 Thiết lập công giao tiếp giữa kit DE2 và Computer

Bước 6. Nhấn **Close**

Bước 7. Chọn **Mode JTAG**

Bước 8. Nhấn **Add File**, chỉ đường dẫn đến **File .sof** (được tạo ra khi chạy Compilation).

Bước 9. Check box **Program/Configure**



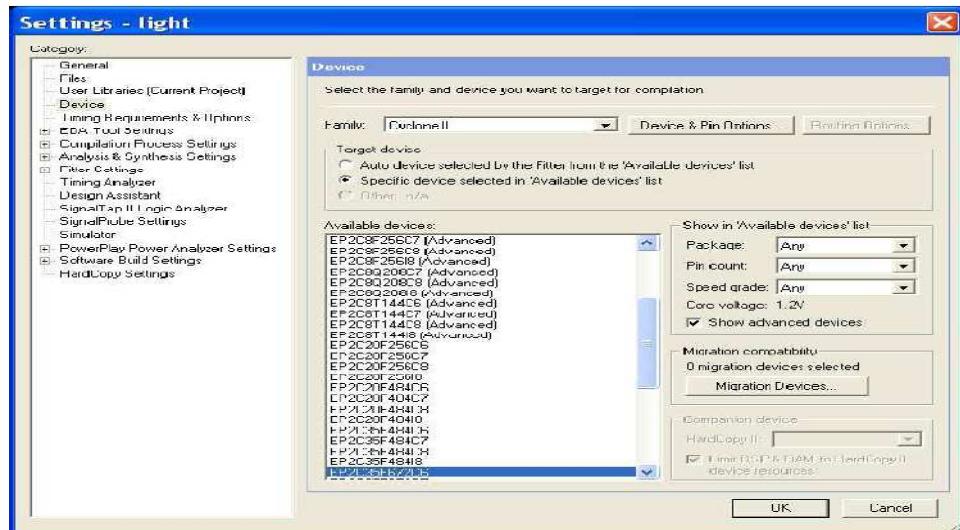
Hình 5.38 Chọn cấu hình nạp thiết kế

Bước 10. Nhấn Start.

Bước 11. Quan sát trên Kit DE2, switch SW0, SW1 và quan sát LED.

■ Active Serial Mode :

Bước 3. Chọn Assignments → Devide



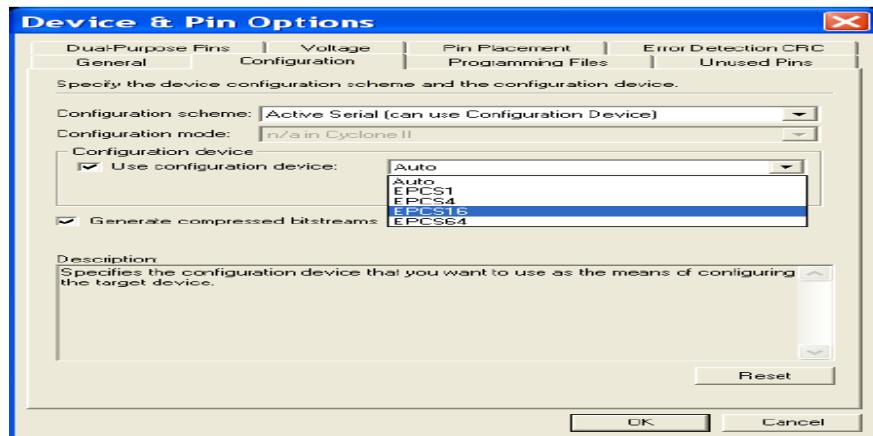
Hình 5.39 Thiết kế chế độ nạp lên FPGA bằng AS mode

Bước 4. Chọn Family : Cyclone II

Bước 5. Chọn Available devices : EP2C35F672C6

Bước 6. Nhấn Device & Pin Option

Bước 7. Chọn Tab Configuration



Hình 5.40 Chọn loại ROM tương ứng

Bước 8. Chọn Configuration device : **EPCS64** (hỗ trợ EEPROM trên Kit

DE2 , dùng để lưu chương trình để nạp cho FPGA mỗi khi power on).

Bước 9. Tương tự JTAG ở những bước kế tiếp :

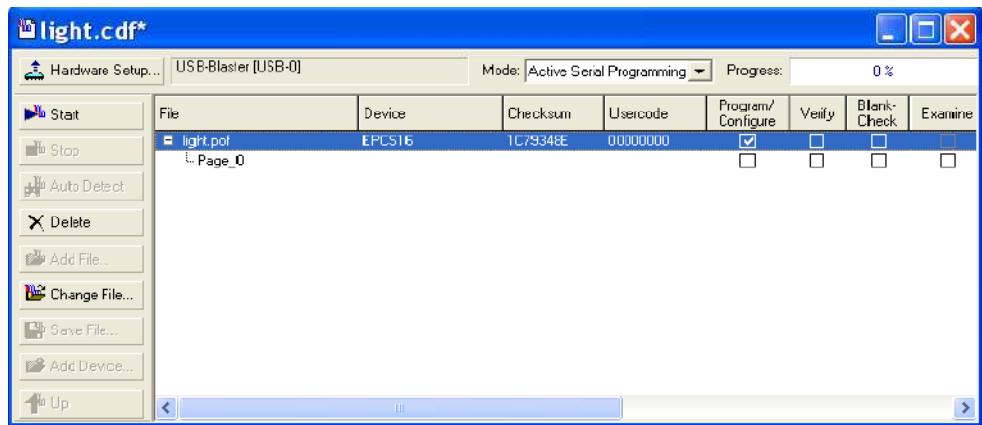
Bước 10. Trên Kit DE2 , chuyển Switch **RUN/PROG** về vị trí **RUN**

Bước 11. Trên màn hình chính Quantus II, chọn **Tools ➔ Programmer**

Bước 12. Chọn **Hardware Setup : USB-Blaster[USB-0]**

Bước 13. Chọn **Mode : Active Serial Programming**

Bước 14. Nhấn **Add File**, chỉ đường dẫn đến **File .pof** (File được tạo ra trong quá trình chạy Compilation).



Hình 5.41 Chọn file thiết kế .pof

- Bước 15. Check box **Program/Configure**.
- Bước 16. Nhấn **Start** để programming chương trình cho EPPROM.
- Bước 17. Nhấn Phím **Restart** trên Kit DE2.
- Bước 18. Quan sát trên Kit DE2, switch SW0, SW1 và quan sát LED.

5.2 Nội dung thực hành môn Thiết kế mạch với Verilog HDL

5.2.1 Bài thực hành số 1 – Thiết kế mạch tổ hợp và mạch tuần tự đơn giản

Mục đích:

- Thực hành thiết kế mạch tổ hợp dùng để nhân hai số không dấu 4 bits.
- Thực hành thiết kế mạch tuần tự dùng thanh ghi và bộ đếm.

5.2.1.1 Phần 1

Thiết kế một mạch nhân hai số 4 bits

Các bước thực hiện

- Bước 1. Tạo một project mới, đặt tên:
user_dir/lab1/lab1_part1

Bước 2. Mô tả thiết kế một mạch nhân hai số 4 bit sử dụng ngôn ngữ Verilog.

Bước 3. Gán pin như sau

- Dùng switches SW11- 8 để nhập số A và switches SW3-0 để nhập số B. Giá trị của A và B được hiển thị trên Led 7 đoạn HEX6 và HEX4 dưới dạng số Hexa. Kết quả phép nhân $C=A*B$ được hiển thị trên HEX1 và HEX0 cũng dưới dạng Hexa.

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Tạo file Vector Waveform (.vWF) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 6. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

➡ Đọc và thực hiện các bước từ 1 đến 5 ở nhà.

5.2.1.2 Phần 2

Mở rộng mạch nhân sang 4 bit sang mạch nhân 8 bit. Dùng SW15-8 để nhập số A và switches SW7-0 để nhập số B. . Giá trị của A và B được hiển thị tương ứng trên các Led 7 đoạn HEX7-6 và HEX5-4 dưới dạng số Hexa. Kết quả phép nhân $C=A*B$ được hiển thị trên HEX3-0 cũng dưới dạng Hexa.

Các bước thực hiện

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab1/lab1_part2

Bước 2. Mô tả thiết kế một mạch nhân hai số 8 bit sử dụng ngôn ngữ Verilog.

Bước 3. Gán pin như sau

➤ Dùng switches SW15-8 để nhập số A và switches SW7-0 để nhập số B. Giá trị của A và B được hiển thị trên Led 7 đoạn HEX7-6 và HEX5-4 dưới dạng số Hexa. Kết quả phép nhân $C=A*B$ được hiển thị trên HEX3-0 cũng dưới dạng Hexa.

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Tạo file Vector Waveform (.vWF) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 6. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

─► Đọc và thực hiện các bước từ 1 đến 5 ở nhà.

Part III

Ta muốn hiển thị một giá trị hexadecimal của một số A 16 bit trên bốn Led 7 đoạn HEX7-4 và hiển thị số B 16 bit trên bốn Led 7 đoạn HEX3-0. Giá trị của A và B đều được cung cấp bởi SW15-0. Muốn làm được điều này thì trước hết ta phải nhập giá trị A từ SW15-0 và phải lưu trữ giá trị này vào thanh ghi trước khi nhập giá trị mới cho B.

Các bước thực hiện

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab1/lab1_part3

Bước 2. Mô tả thiết kế bằng ngôn ngữ Verilog để có thể thực hiện được chức năng trên.

Bước 3. Gán pin như sau

- Dùng switches SW15- 0 để nhập số A và B. Giá trị của A và B được hiển thị trên Led 7 đoạn HEX7-4 và HEX3-0 dưới dạng số Hexa.

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Tạo file Vector Waveform (.vWF) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 6. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

 Đọc và thực hiện các bước từ 1 đến 5 ở nhà.

5.2.1.3 Phần 4

Sử dụng ngôn ngữ Verilog để mô tả thiết kế của một mạch có chức năng là hiển thị lần lượt các số 0 đến 9 trên Led 7 đoạn HEX0. Mỗi số sẽ được hiển thị trong khoảng thời gian là 1 giây. Sử dụng một bộ đếm để xác định khoảng thời gian 1 giây đó. Xung Clock sử dụng cho bộ đếm là 50 MHz từ Board DE2.

Các bước thực hiện

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab1/lab1_part4

Bước 2. Mô tả thiết kế bằng ngôn ngữ Verilog để có thể thực hiện được chức năng trên.

Bước 3. Gán pin như sau

- Sử dụng Led 7 đoạn HEX0 để hiển thị số.
- CLK_50 gán cho xung clock của counter.

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Tạo file Vector Waveform (.vWF) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 6. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm → Tính vắng buổi đó)

- ─  Đọc và thực hiện các bước từ 1 đến 5 ở nhà.

5.2.2 Bài thực hành số 2 – Thực hành tìm hiểu thiết kế latches, flip-flops và counters.

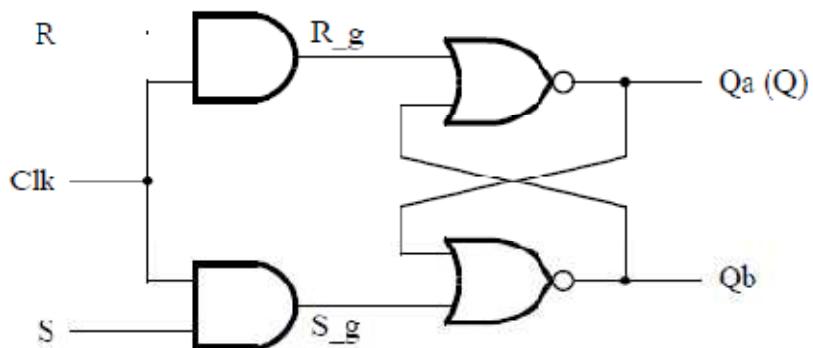
Mục đích:

- ─  Thực hành và tìm hiểu phương pháp thiết kế latches, flip-flops và counter.

5.2.2.1 Phần 1

Trong Hình 5.42 mô tả một mạch latch RS. Phía dưới Hình 5.42 thể hiện hai phong cách code Verilog có thể dùng để mô tả mạch latch RS trên. Phần a mô tả

latch bằng việc gọi và kết nối các cổng logic lại với nhau, còn trong phần b thì sử dụng biểu thức logic để tạo ra latch. Còn nếu latch này được thực thi dùng bảng lookup tables 4 ngõ vào (LUTs) thì chỉ cần một bảng lookup tables là đủ, như trình bày trong hình Hình 5.43 a.



Hình 5.42 Một mạch latch RS

// RS latch

```
module part1 (Clk, R, S, Q);
input Clk, R, S;
output Q;
    wire R_g, S_g, Qa, Qb /*synthesis keep */;
    and (R_g, R, Clk);
    and (S_g, S, Clk);
    nor (Qa, R_g, Qb);
    nor (Qb, S_g, Qa);
    assign Q = Qa;
endmodule
```

a. Gọi và kết nối các cổng logic để tạo thành mạch RS latch.

```

// RS latch

module part1 (Clk, R, S, Q);

input Clk, R, S;
output Q;

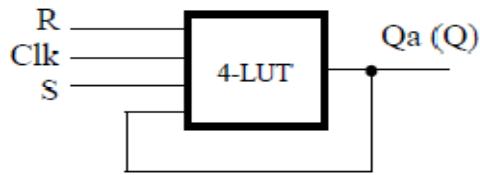
wire R_g, S_g, Qa, Qb /*synthesis keep */;
assign R_g = R & Clk;
assign S_g = S & Clk;
assign Qa = !(R_g / Qb);
assign Qb = !(S_g / Qa);
assign Q = Qa;

endmodule

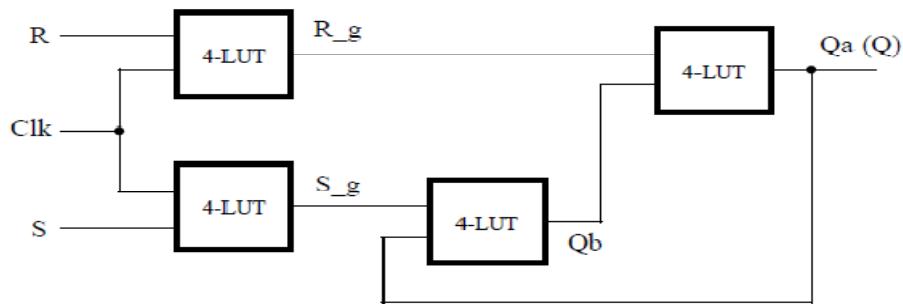
```

b. Dùng các biểu thức logic để mô tả mạch latch RS.

Vì mạch latch có thể thiết kế mà chỉ cần sử dụng một LUT 4 input như trong Hình 5.43 (tất nhiên ở đây ta đang nói complier cho một FPGA nào đó sẽ sử dụng LUT để synthesize ra mạch chứ không phải là dùng các cổng logic ghép lại), tuy nhiên việc thực thi này không cho phép quan sát được các tín hiệu bên trong của nó khi chạy mô phỏng, chẳng hạn như R_g và S_g, bởi vì chúng không được cung cấp như là output của LUT. Để giữ các tín hiệu bên trong này khi thiết kế, ta cần phải sử dụng hướng dẫn trình biên dịch (*compiler directive*) trong code mô tả thiết kế. Trong hình 2, một hướng dẫn trình biên dịch /*synthesis keep */ được thêm vào để hướng dẫn trình biên dịch Quartus II biết rằng cần phải sử dụng những cổng logic độc lập cho mỗi tín hiệu R_g, S_g, Qa, Qb, nhờ vậy mà ta có thể quan sát được các tín hiệu bên trong mạch latch. Sau quá trình biên dịch code mô tả Verilog, ta sẽ có một mạch với bốn LUT 4 input được mô tả trong hình Hình 5.43 b.



(a) Sử dụng một LUT 4 inputs để tạo thành RS latch



(b) Sử dụng bốn LUT 4 inputs để tạo thành RS latch.

Hình 5.43 Mạch thực hiện trên FPGA cho mạch RS latch

Các bước thực hiện

Bước 1. Tạo một project Quartus mới, đặt tên: user_dir/lab2/lab2_part1

Bước 2. Mô tả thiết kế mạch RS latch như trên Hình 5.42 (có thể sử dụng một trong hai cách a hoặc b, đều tạo ra mạch giống nhau)

Bước 3. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 4. Quan sát mạch được tạo ra từ mô tả Verilog trên dùng công cụ Technology Viewer và so sánh với mạch trên hình Hình 5.43 b.

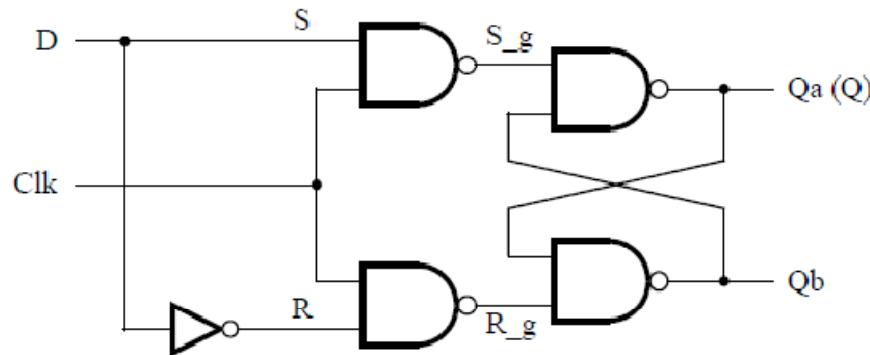
Bước 5. Tạo file Vector Waveform (.vwf) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

➡ Đọc và thực hiện các bước từ 1 đến 5 ở nhà.

5.2.2.2 Phản 2

Thiết kế mạch D latch như trên Hình 4



Hình 5.44 Mạch D latch

Các bước thực hiện

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab2/lab2_part2

Bước 2. Dùng Verilog HDL mô tả thiết kế mạch D latch theo phong cách code b. Dùng hướng dẫn trình biên dịch /*synthesis keep */ để đảm bảo rằng những phần tử logic độc lập được dùng để thực thi các tín hiệu R, S_g, R_g, Qa, và Qb.

Bước 3. Gán pin như sau

➤ Dùng switches SW0 để điều khiển tín hiệu D và switches SW1 để làm tín hiệu xung Clock. LEDR0 được gán đến Q.

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Quan sát mạch được tạo ra từ mô tả Verilog trên dùng công cụ Technology Viewer.

Bước 6. Tạo file Vector Waveform (.vwf) và chạy mô phỏng để kiểm tra hoạt động của mạch.

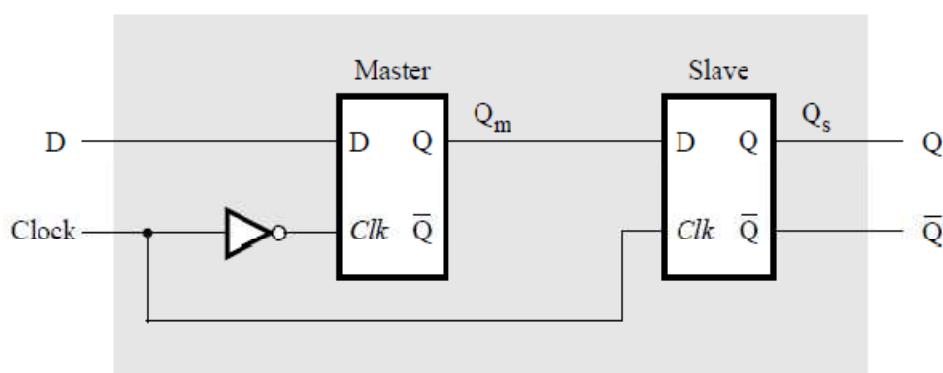
Bước 7. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

- ─ Đọc và thực hiện các bước từ 1 đến 6 ở nhà.

5.2.2.3 Phần 3

Thiết kế mạch master-slave D flip-flop như trên Hình 5.45.



Hình 5.45 Mạch master-slave D flipflop

Các bước thực hiện

- Bước 1. Tạo một project Quartus mới, đặt tên: user_dir/lab2/lab2_part3
- Bước 2. Sử dụng Verilog HDL mô tả thiết kế mạch master-slave flip-flop trên, gọi (instantiate) module D latch trên phần 2 để thực thi D flip-flop này.

Bước 3. Gán pin như sau

- Dùng switches SW0 để điều khiển tín hiệu D và switches SW1 để làm tín hiệu xung Clock. LEDR0 được gán đến Q.

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Quan sát mạch được tạo ra từ mô tả Verilog trên dùng công cụ Technology Viewer.

Bước 6. Tạo file Vector Waveform (.vww) và chạy mô phỏng để kiểm tra hoạt động của mạch.

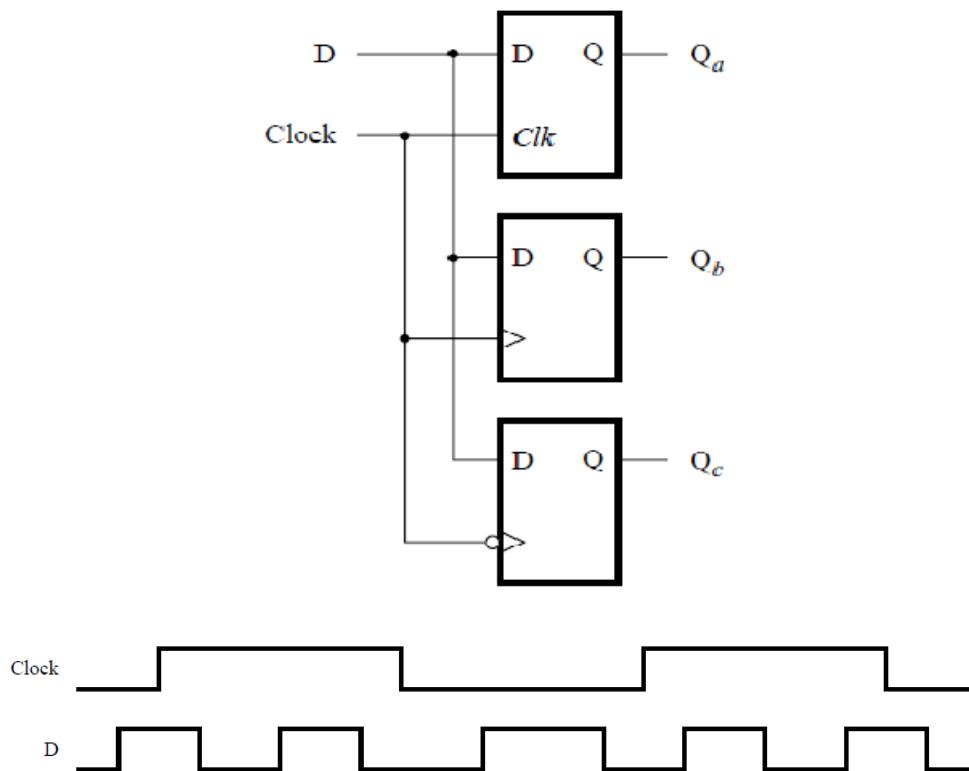
Bước 7. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

➡ Đọc và thực hiện các bước từ 1 đến 6 ở nhà.

5.2.2.4 Phần 4

Thiết kế một mạch như trên Hình 5.46 với 3 phần tử nhớ khác nhau: một D latch, một D flip-flop kích cạnh lên, và một D flip-flop kích cạnh xuống.



Hình 5.46 Mạch và dạng sóng ngõ vào cho phần 4

Các bước thực hiện

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab2/lab2_part4

Bước 2. Dùng Verilog HDL mô tả thiết kế mạch mạch trên bằng cách gọi (instantiate) ba phần tử nhớ từ phần 2 và 3. Trong phần này, ta không sử dụng hướng dẫn trình biên dịch /*synthesis keep */ như phần 1 đến 3. Đoạn code dưới thể hiện một phong cách khác để mô tả D latch trong Hình 5.44, đó là sử dụng mô hình hành vi (behavioral style). Ta cũng có thể sử dụng mô hình hành vi này để mô tả mạch trong Hình 5.46.

```
module D_latch (input D, Clk, output Q);
    reg Q;
    always @ (D or Clk)
        if (Clk)
            Q = D;
endmodule
```

Code sử dụng mô hình hành vi để mô tả D latch

Bước 3. Gán pin như sau

- Dùng switches SW0 để điều khiển tín hiệu D và switches SW1 để làm tín hiệu xung Clock. LEDR0 được gán đến Qa, LEDR1 được gán đến Qb, LEDR2 được gán đến Qc.

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Quan sát mạch được tạo ra từ mô tả Verilog trên dùng công cụ Technology Viewer.

Bước 6. Tạo file Vector Waveform (.vwf) và chạy mô phỏng để kiểm tra hoạt động của mạch.

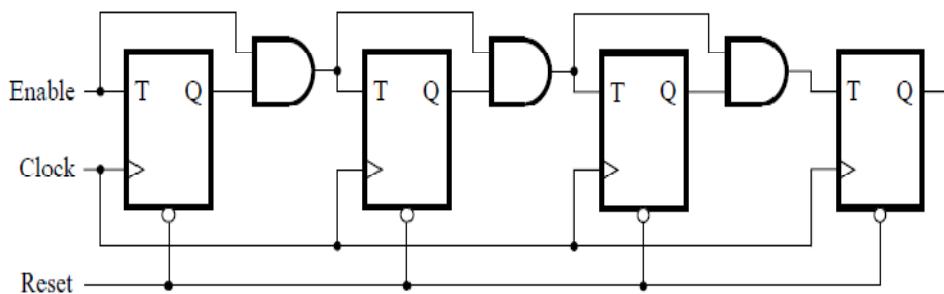
Bước 7. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

➡ Đọc và thực hiện các bước từ 1 đến 6 ở nhà.

5.2.2.5 Phần 5

Thiết kế một mạch đếm đồng bộ 16 bit sử dụng T flip-flop dựa trên như mạch đếm đồng bộ 4 bit sử dụng T flip-flop trên Hình 5.47. Bộ đếm tăng thêm 1 tại mỗi cạnh lên xung clock khi mà tín hiệu Enable được tích cực. Bộ đếm được Reset về 0 khi tín hiệu Reset tích cực.



Hình 5.47 Một bộ đếm đồng bộ 4 bit

Các bước thực hiện

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab3/lab3_part5

Bước 2. Dùng Verilog HDL mô tả thiết kế bộ đếm đồng bộ 16 bit dùng 2 cách: một là dựa trên mạch ở Hình 5.47, hai là mô tả mạch sử dụng mô hình hành vi (behavioral model).

Bước 3. Gán pin như sau

- Dùng switches SW0 để điều khiển tín hiệu Enable và switches SW1 để làm tín hiệu xung Reset.
- KEY0 làm tín hiệu xung Clock.
- Bốn led 7 đoạn HEX3-0 để hiện thị giá trị đếm theo mã hexa.

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Quan sát mạch được tạo ra từ mô tả Verilog trên dùng công cụ Technology Viewer. Thiết kế này tốn mát bao nhiêu phần tử logic (LEs)? Tần số lớn nhất mà mạch đã thiết kế có thể hoạt động ? So sánh khi thiết kế theo hai cách trên.

Bước 6. Tạo file Vector Waveform (.vWF) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 7. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm → Tính vắng buổi đó)

 Đọc và thực hiện các bước từ 1 đến 6 ở nhà.

5.2.3 Bài thực hành số 3 – Thiết kế hệ thống sử dụng xung Clock thời gian thực

Mục đích:

- Thực hành thiết kế và sử dụng xung Clock thời gian thực

5.2.3.1 Phần 1

Thiết kế một mạch đếm ba kí số BCD. Hiển thị nội dung của bộ đếm lên ba led 7 đoạn HEX2-0. Tạo một tín hiệu điều khiển từ xung Clock 50 MHz lấy từ board DE2, tín hiệu điều khiển này dùng điều khiển để tăng bộ đếm lên 1 sau một khoảng 1 giây. Dùng phím nhấn KEY0 để reset bộ đếm về 0.

Các bước thực hiện

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab3/lab3_part1

Bước 2. Mô tả thiết kế một mạch thực hiện chức năng trên sử dụng Verilog

Bước 3. Gán pin như sau

- Ngõ ra bộ đếm BCD nối đến 3 led 7 đoạn HEX2-0
- Chân Reset_n nối đến KEY0

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Tạo file Vector Waveform (.vWF) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 6. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm → Tính vắng buổi đó)

➡ Đọc và thực hiện các bước từ 1 đến 5 ở nhà.

5.2.3.2 Phần 2

Thiết kế và thực thi một mạch hoạt động như một đồng hồ chỉ thời gian. Nó hiển thị giờ (từ 0 đến 23) trên led 7 đoạn HEX7-6, hiển thị phút (từ 0 đến 60) trên HEX5-4 và giây (từ 0 đến 60) trên HEX3-2. Dùng switches SW15-0 để thiết lập giá trị giờ, phút.

Các bước thực hiện

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab3/lab3_part2

Bước 2. Mô tả thiết kế một mạch thực hiện chức năng trên sử dụng Verilog

Bước 3. Gán pin như sau

- Ngõ ra chỉ giờ nối đến 2 led 7 đoạn HEX7-6
- Ngõ ra chỉ phút nối đến 2 led 7 đoạn HEX5-4
- Ngõ ra chỉ giây nối đến 2 led 7 đoạn HEX3-2
- SW15-0 nối đến chân thiết lập giá trị giờ, phút cho đồng hồ.

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Tạo file Vector Waveform (.vWF) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 6. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm → Tính vắng buổi đó)

➡ Đọc và thực hiện các bước từ 1 đến 5 ở nhà.

5.2.4 Bài thực hành số 4 – Thực hành tìm hiểu thiết kế sử dụng State machine

Mục đích:

- ─ Thực hành và tìm hiểu thiết kế hệ thống sử dụng State machines.

5.2.4.1 Phần 1

Thiết kế một mạch sử dụng lưu đồ máy trạng thái hữu hạn (FSM) để kiểm tra xem nếu ngõ vào input w mà có giá trị bằng 0 hay bằng 1 trong bốn chu kì liên tục thì ngõ ra z sẽ được bật lên 1.

Các bước thực hiện

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab4/lab4_part1

Bước 2. Sử dụng mô hình máy trạng thái trong Verilog để mô tả thiết kế cho mạch có chức năng như trên.

Bước 3. Gán pin

- SW0 gán đến chân input w.
- LEDG0 gán đến chân output z
- KEY0 gán đến chân xung Clock

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Quan sát mạch được tạo ra từ mô tả Verilog trên dùng công cụ Technology Viewer.

Bước 6. Tạo file Vector Waveform (.vwf) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

- Nạp file thực thi .sof lên FPGA và quan sát hoạt động của mạch
- Đọc và thực hiện các bước từ 1 đến 6 ở nhà.

5.2.4.2 Phần 2

Thiết kế một mạch đếm modulo-10 hoạt động như sau, ngõ ra bộ đếm được trả về 0 bởi ngõ vào Reset, hai ngõ vào w1, w0 dùng để điều khiển việc đếm. Nếu $w_1w_0 = 00$, bộ đếm sẽ giữ nguyên giá trị, nếu $w_1w_0 = 01$, bộ đếm tăng mỗi lần lên 1, $w_1w_0 = 10$, bộ đếm tăng mỗi lần lên 2, $w_1w_0 = 11$, bộ đếm tăng mỗi giảm đi 1. Tất cả sự thay đổi diễn ra tại cạnh lên xung Clock.

Các bước thực hiện

- Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab4/lab4_part2
- Bước 2. Dùng mô hình máy trạng thái State machine trong Verilog HDL mô tả thiết kế mạch có chức năng như trên
- Bước 3. Gán pin như sau
 - Dùng switches SW0, SW1 để điều khiển tín hiệu w0, w1.
 - KEY0 để điều khiển xung Clock.
 - Ngõ ra bộ đếm được hiện thị trên Led 7 đoạn HEX0 dưới dạng số HEX.
- Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.
- Bước 5. Quan sát mạch được tạo ra từ mô tả Verilog trên dùng công cụ Technology Viewer.
- Bước 6. Tạo file Vector Waveform (.vwf) và chạy mô phỏng để kiểm tra hoạt động của mạch.
- Bước 7. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

- ─  Đọc và thực hiện các bước từ 1 đến 6 ở nhà.

5.2.4.3 Phần 3

Trong phần thiết kế này gồm hai bước:

- ❖ Thiết kế mạch hiển thị từ “HELLO” lên Led 7 đoạn, sau đó cho những ký tự này dịch từ trái sang phải (từ HEX0 đến HEX7) trong mỗi khoảng thời gian là một giây.
- ❖ Cải tiến thiết kế trên để có thể sử dụng hai input KEY3 và KEY2 điều khiển tốc độ dịch của các ký tự từ trái sang phải. Nếu KEY2 được nhấn, tốc độ dịch chuyển của các ký tự tăng lên hai lần. Nếu KEY3 được nhấn, tốc độ dịch chuyển của các ký tự giảm đi hai lần.

Các bước thực hiện

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab3/lab3_part3

Bước 2. Sử dụng Verilog HDL mô tả thiết kế mạch thực hiện được chức năng đầu tiên của mạch. Mô phỏng và chạy thử để kiểm tra mạch, nếu đúng thì tiếp tục sử dụng mô hình máy trạng thái trong Verilog để cải tiến mạch thực hiện được chức năng thứ hai.

Bước 3. Gán pin như sau

- Dùng HEX0-HEX7 để hiện ký tự.
- Dùng KEY2, KEY3 làm chân input để điều khiển tốc độ dịch.
- Dùng xung Clock_50 để tạo các khoảng delay.

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Quan sát mạch được tạo ra từ mô tả Verilog trên dùng công cụ Technology Viewer.

Bước 6. Tạo file Vector Waveform (.vwf) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 7. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

 Đọc và thực hiện các bước từ 1 đến 6 ở nhà.

5.2.5 Bài thực hành số 5 – Thực hành tìm hiểu phương pháp thiết kế onchip Memory trên FPGA và phương pháp sử dụng offchip Memory

Mục đích:

- ─ Trong những hệ thống máy tính, để lưu trữ chương trình và lưu trữ dữ liệu nó luôn cần một bộ nhớ với dung lượng đủ để lưu trữ. Nếu một hệ thống được thực thi trên FPGA, thì nó có thể sử dụng bộ nhớ có sẵn trên FPGA (onchip memory) do nhà sản xuất hỗ trợ sẵn hoặc ta cũng có thể thiết kế bộ nhớ ngay trên FPGA (onchip memory). Tuy nhiên, nếu dung lượng bộ nhớ trên FPGA không đủ thì một bộ nhớ bên ngoài FPGA (offchip memory) sẽ cần được sử dụng. Do đó, trong phần thực hành này ta sẽ tìm hiểu ba vấn đề sau:
 - ❖ Tìm hiểu cách thức hiện thực một memory trên FPGA (onchip memory)
 - ❖ Tìm hiểu cách thức sử dụng một offchip memory (SRAM chip trên kit DE2).
 - ❖ Chạy mô phỏng trên ModelSim

5.2.5.1 Phần 1

Altera cung cấp một thư viện có sẵn LPM chứa các thiết kế cơ bản như là adders, registers, counters và memories (Quartus II Library of Parameterized Modules). Ta có thể sử dụng các mô tả thiết kế này để thiết kế trên FPGA. Để thiết kế một bộ nhớ SRAM trên FPGA ta có thể dùng module *altsyncram* LPM do

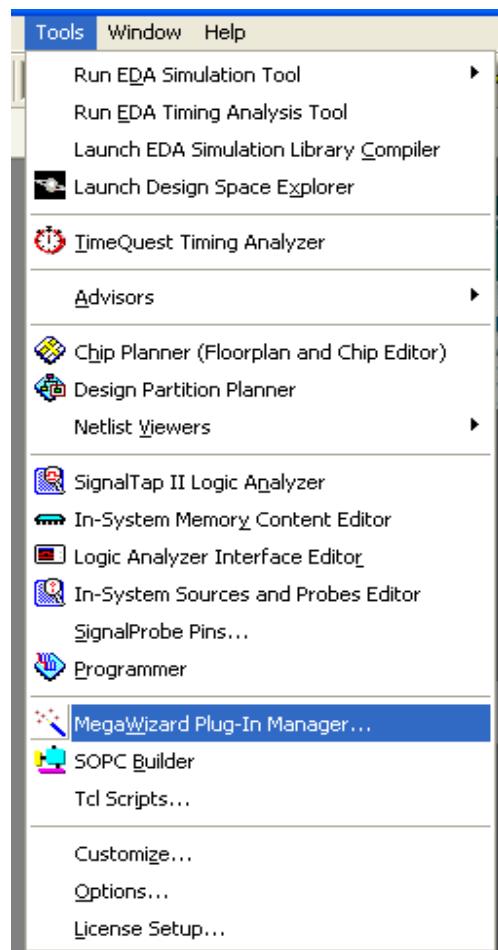
Altera cung cấp. Phần này ta sẽ thiết kế một SRAM onchip có dung lượng 32 words, mỗi word là 8 bits.

Các bước thực hiện

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab5/lab5_part1

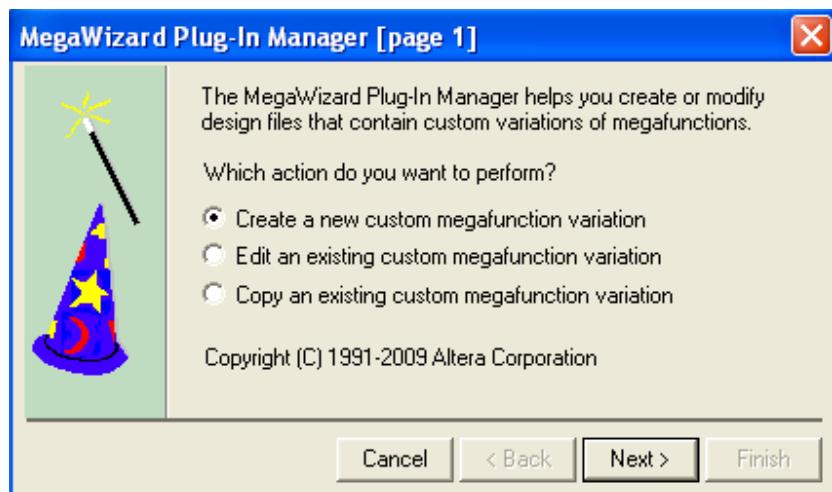
Bước 2. Sử dụng công cụ MegaWizard Plug-in Manager trên Quartus để tạo ra RAM như yêu cầu.

➤ Tools ➔ MegaWizard Plug-in Manager



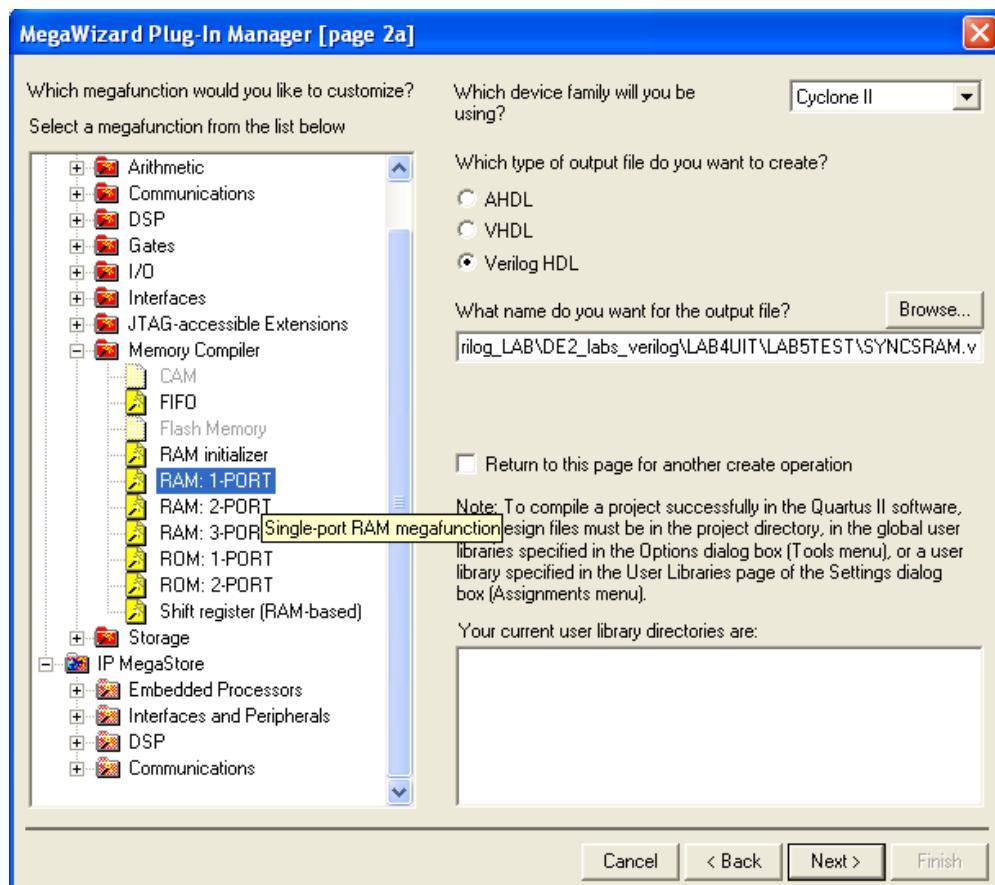
Hình 5.48 Dùng Tool tạo Mega Wizard

➤ Chọn option “**Create a new custom megafunction variation**”



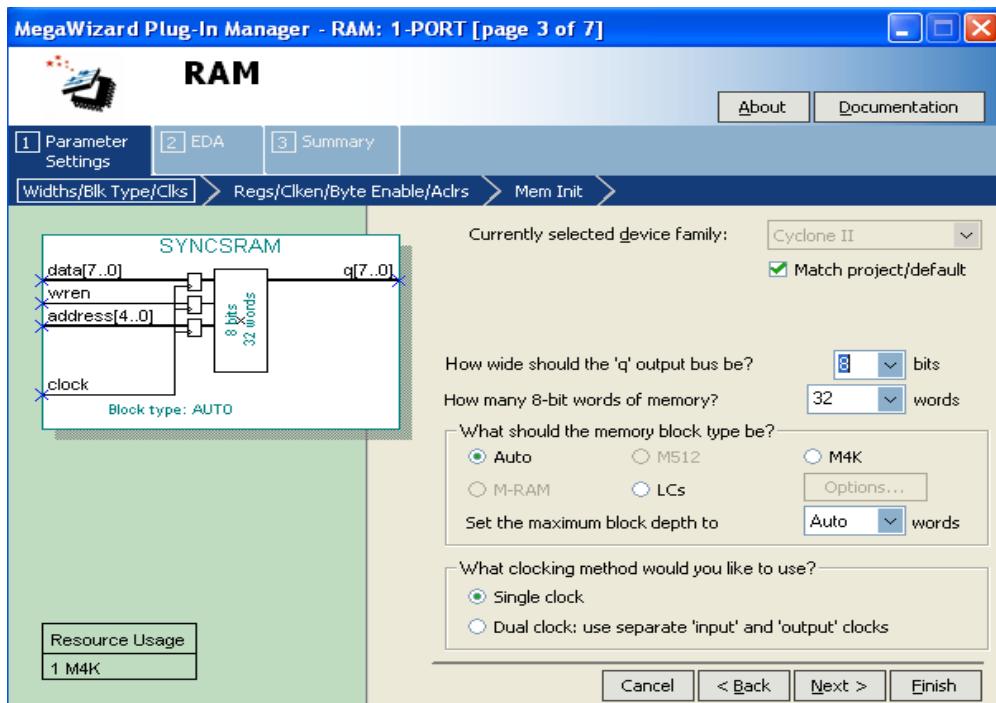
Hình 5.49 Tạo một Mega mới

- Chọn **RAM: 1-PORT** trong danh sách các megafunction
- Chọn device family: **Cyclone II**
- Chọn loại dữ liệu ngõ ra: **Verilog HDL**
- Chọn đường dẫn và tên cho file output



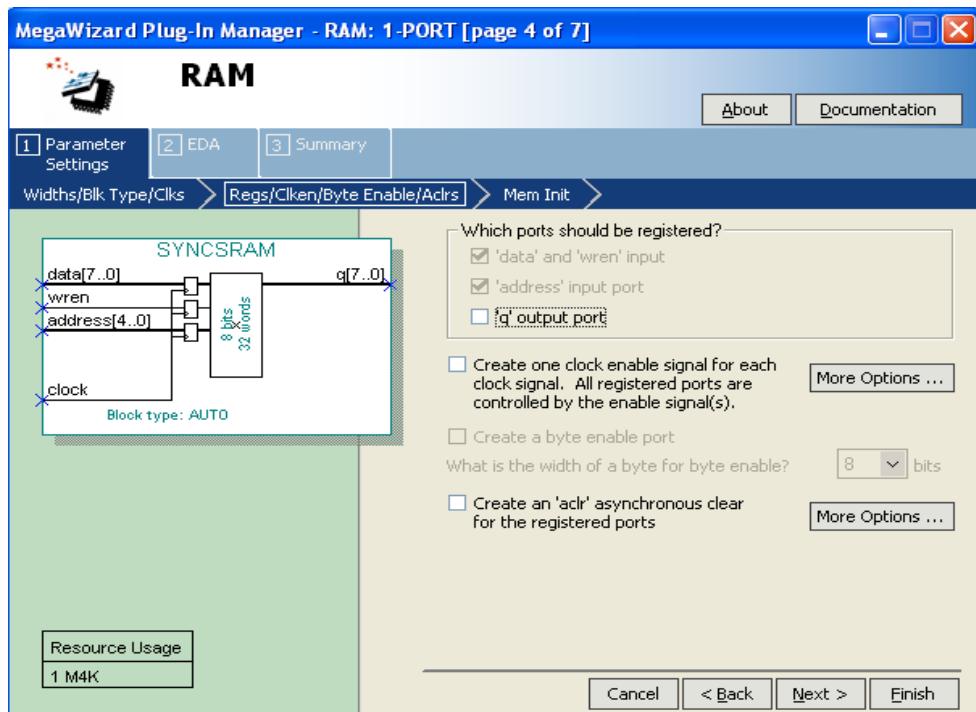
Hình 5.50 Chọn các thông số như hình vẽ

- Nhấn Next
- Nhập các thông số như yêu cầu thiết kế.



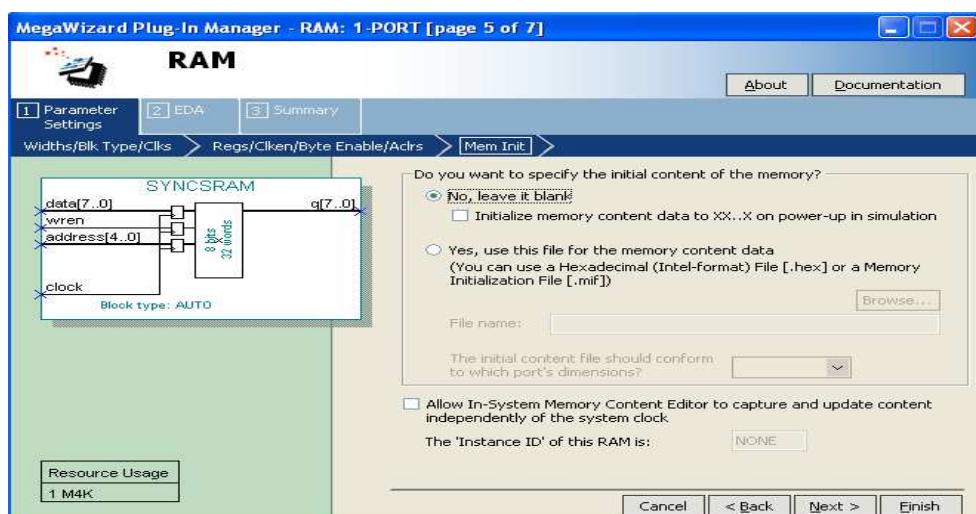
Hình 5.51 Chọn thông số như trên hình

- Nhấn **Next**
- Chọn các option như hình dưới



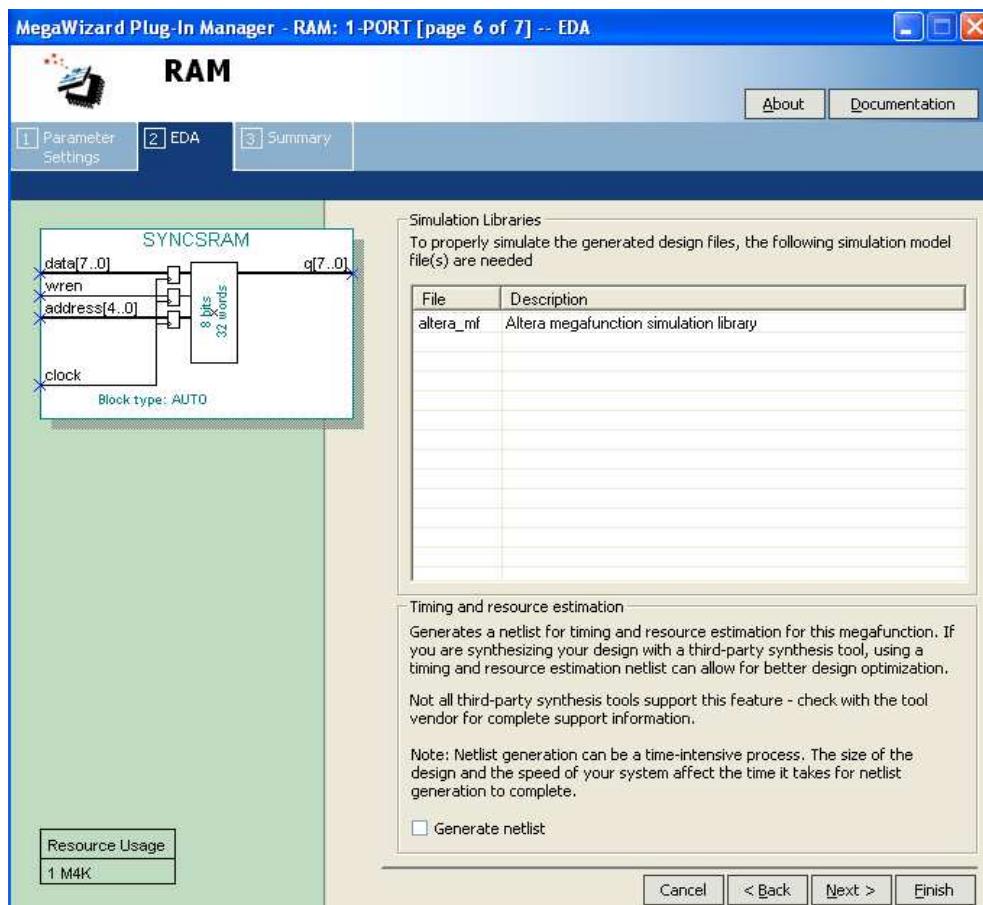
Hình 5.52 Gán registers cho inputs

- Nhấn Next
- Chọn option “**No, leave it blank**”, bởi vì ta không muốn bộ nhớ có dữ liệu sẵn.



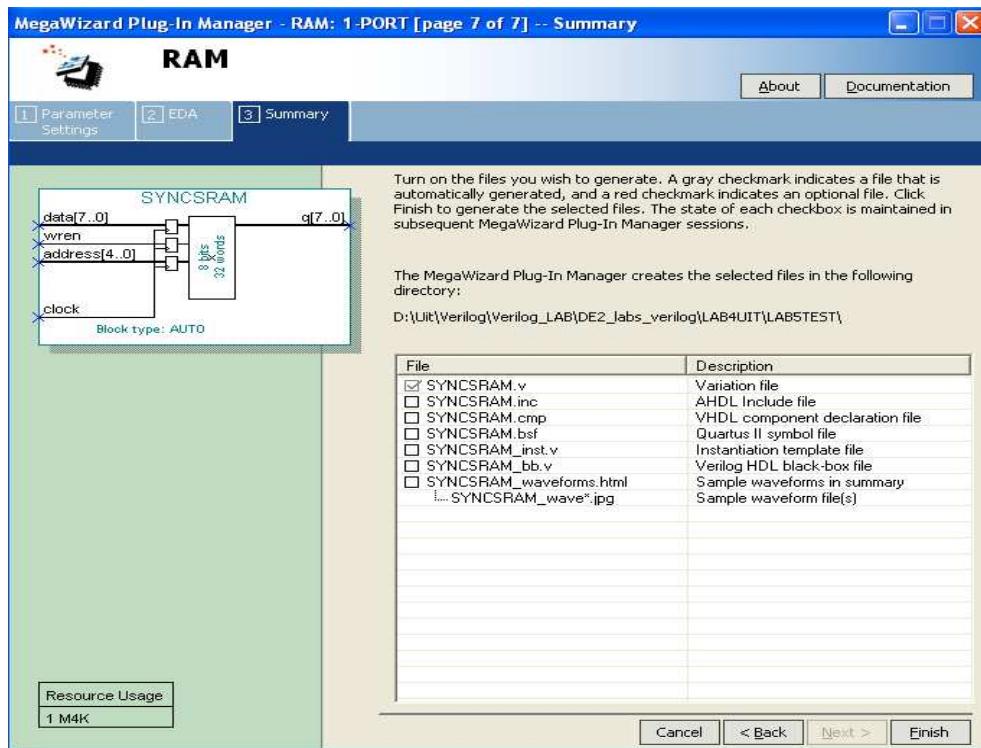
Hình 5.53 Không tạo giá trị ban đầu cho SRAM

- Nhấn Next



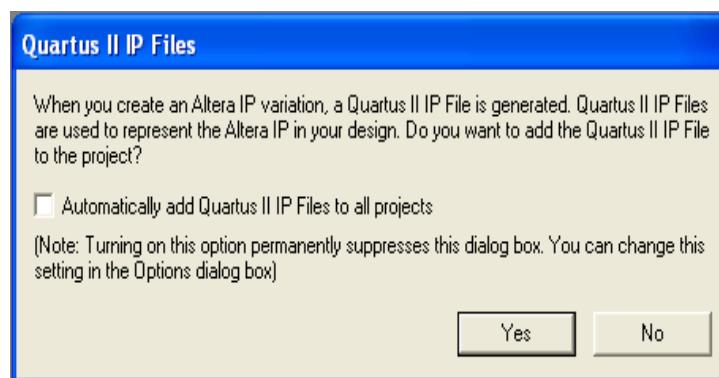
Hình 5.54 Simulation library

- Nhấn **Next**
- Chọn option tạo ra file mô tả thiết kế.v



Hình 5.55 Chọn các loại dữ liệu cần tạo ra

➤ Nhấn Next



Hình 5.56 Add dữ liệu tạo ra vào project hiện hành

➤ Nhấn Yes

- Một mô tả thiết kế verilog cho một bộ nhớ SRAM được tạo ra.

The screenshot shows the Quartus II interface with the following details:

- Title Bar:** Quartus II - D:\Uit\Verilog\Verilog_LAB\DE2_labs_verilog\LAB4UIT\LAB5TEST\Part1 Test - Part1 Test - [SYNCSRAM.v]
- File Menu:** File Edit View Project Assignments Processing Tools Window Help
- Toolbar:** Standard toolbar icons.
- Project Navigator:** Shows a project named "SYNCSRAM.qpj" containing "SYNCSRAM.v".
- Design Editor:** The main window displays the Verilog code for the "SYNCSRAM" module.
- Task List:** Shows tasks like Compile Design, Analysis & Synthesis, Filter (Place & Route), Assembler (Generate program), Classic Timing Analysis, Edit Settings, View Report, and TimeQuest Timing Analysis.

```

36 // synopsys translate_off
37 `timescale 1 ps / 1 ps
38 // synopsys translate_on
39 module SYNCRAM (
40   address,
41   clock,
42   data,
43   wren,
44   q);
45
46   input  [4:0] address;
47   input    clock;
48   input  [7:0] data;
49   input    wren;
50   output [7:0] q;
51
52   wire [7:0] sub_wire0;
53   wire [7:0] q = sub_wire0[7:0];
54
55   altsyncram altsyncram_component (
56     .wren_a (wren),
57     .clock0 (clock),
58     .address_a (address),
59     .data_a (data),
60     .q_a (sub_wire0),
61     .aclr0 (1'b0),
62     .aclr1 (1'b0),
63     .address_b (1'b1),
64     .addressstall_a (1'b0),
65     .addressstall_b (1'b0),
66     .byteena_a (1'b1),
67     .byteena_b (1'b1),
68     .clock1 (1'b1),
69     .clocken0 (1'b1),
70   );

```

Hình 5.57 Mô tả verilog của SRAM vừa tạo ra

Bước 3. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 4. Tạo file Vector Waveform (.vWF) và chạy mô phỏng để kiểm tra hoạt động của mạch (ghi dữ liệu vào SRAM, sau đó đọc các dữ liệu đã ghi xem SRAM có hoạt động đúng không).

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

➡ Đọc và thực hiện các bước từ 1 đến 4 ở nhà.

5.2.5.2 Phần 2

Phần này ta dùng Switches trên kit DE2 để ghi dữ liệu vào SRAM đã tạo ra ở phần 1. Sau đó dùng Leds 7 đoạn để hiển thị dữ liệu được đọc ra từ SRAM

Các bước thực hiện

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab5/lab5_part2

Bước 2. Add file mô tả thiết kế đã tạo ra trong phần 1 vào project. Tạo một top-module để instantiate module SRAM.

Bước 3. Gán pin như sau

- Dùng switches SW[7:0] làm tín hiệu dữ liệu ngõ vào.
- Dùng switches SW[15:11] làm tín hiệu địa chỉ.
- Dùng HEX0, HEX1 hiển thị tín hiệu dữ liệu ngõ vào.
- Dùng HEX2, HEX3 hiển thị tín hiệu địa chỉ.
- Dùng HEX4, HEX5 hiển thị tín hiệu ngõ ra.
- Dùng SW[17] làm tín hiệu WREN (khi WREN = 1, cho phép ghi, WREN = 0, cho phép đọc)
- Dùng KEY0 làm xung CLK.

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Tạo file Vector Waveform (.vWF) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 6. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm → Tính vắng buổi đó)

➡ Đọc và thực hiện các bước từ 1 đến 5 ở nhà.

5.2.5.3 Phần 3

Thay vì sử dụng các module thiết kế có sẵn trong LPM do Altera cung cấp, ta cũng có thể tự mô tả thiết kế bởi riêng mình sử dụng ngôn ngữ Verilog. Trong phần này ta sẽ tự thiết kế một bộ nhớ SRAM có dung lượng 32 words, mỗi word 8 bits như phần 1 mà không sử dụng module có sẵn. Để dễ hiểu, ta có thể xem bộ nhớ SRAM trên như là 32 thanh ghi, mỗi thanh ghi có độ dài 8 bits. Hoạt động đọc và ghi của các thanh ghi này được điều khiển bởi tín hiệu WREN. Mỗi thanh ghi được gán một địa chỉ xác định.

Các bước thực hiện

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab5/lab5_part3

Bước 2. Sử dụng Verilog HDL mô tả thiết kế bộ nhớ SRAM như ở trên.

Bước 3. Gán pin như sau

- Dùng switches SW[7:0] làm tín hiệu dữ liệu ngõ vào.
- Dùng switches SW[15:11] làm tín hiệu địa chỉ.
- Dùng HEX0, HEX1 hiển thị tín hiệu dữ liệu ngõ vào.
- Dùng HEX2, HEX3 hiển thị tín hiệu địa chỉ.
- Dùng HEX4, HEX5 hiển thị tín hiệu ngõ ra.
- Dùng SW[17] làm tín hiệu WREN (khi WREN = 1, cho phép ghi, WREN = 0, cho phép đọc)
- Dùng KEY0 làm xung CLK.

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Viết testbench và chạy mô phỏng dùng ModelSim để kiểm tra hoạt động và timing của SRAM. (Trình bày trong Chương 7.)

Bước 6. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

➡ Đọc và thực hiện các bước từ 1 đến 5 ở nhà.

5.2.5.4 Phần 4

Trên Kit DE2 có một con chip SRAM bất đồng bộ (offchip memory) có dung lượng 256 K 16-bit words (512K 8-bit words). Các pin giao tiếp SRAM gồm:

- ❖ Bus ADDRESS[17:0]
- ❖ Bus DATA [15:0], đây là bus In/Out
- ❖ Chip Enable (/CE) tích cực mức thấp.
- ❖ Write Enable (/WE): mức thấp cho phép ghi, khi đọc tín hiệu này phải ở mức cao.
- ❖ Output Enable (/OE): mức thấp thì tín hiệu ngõ ra mới valid.
- ❖ Upper Byte (/UB) và Lower Byte (/LB). Nếu cả UB và LB đều tích cực mức thấp thì 16 bits ngõ ra D[15:0] khi đọc đều valid. Nếu chỉ UB tích cực mức thấp thì chỉ 8 bits cao D[15:8] khi đọc ra sẽ valid, ngược lại nếu chỉ LB tích cực mức thấp thì chỉ 8 bits thấp D[7:0] khi đọc ra sẽ valid

Phần này ta sẽ tìm hiểu cách thức sử dụng offchip SRAM

Các bước thực hiện

Bước 1. Tạo một project Quartus mới, đặt tên:
user_dir/lab5/lab5_part4

Bước 2. Sử dụng Verilog HDL mô tả thiết kế một module giao tiếp giữa offchip SRAM và các pin cũng như LEDs.

Bước 3. Gán pin như sau

- Dùng switches SW[7:0] làm tín hiệu dữ liệu ngõ vào DATA[15:0] = {8'h00,SW[7:0]}.
- Dùng switches SW[15:8] làm tín hiệu địa chỉ (với 8 địa chỉ này sẽ giải mã ra một phần dung lượng của SRAM, ADDR[17:0] = {10'h000,SW[15:8]}).
- Dùng HEX0-HEX3 hiện thị tín hiệu ngõ ra.
- Dùng KEY[0] làm tín hiệu /OE.
- Dùng KEY[1] làm tín hiệu /WE
- Nối mức logic “0” cho cả /LB và /UB.

Bước 4. Biên dịch để phân tích, tổng hợp và tạo ra file .sof.

Bước 5. Tạo file Vector Waveform (.vWF) và chạy mô phỏng để kiểm tra hoạt động của mạch.

Bước 6. Nạp file thực thi .sof lên FPGA. Kiểm tra hoạt động của mạch.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

➡ Đọc và thực hiện các bước từ 1 đến 5 ở nhà.

END

Chương 6. Hướng dẫn thiết kế và thực hành môn Kiến trúc máy tính nâng cao

6.1 Kiến thức tổng quát về vi xử lý Nios II

6.1.1 Giới thiệu tổng quan Nios II

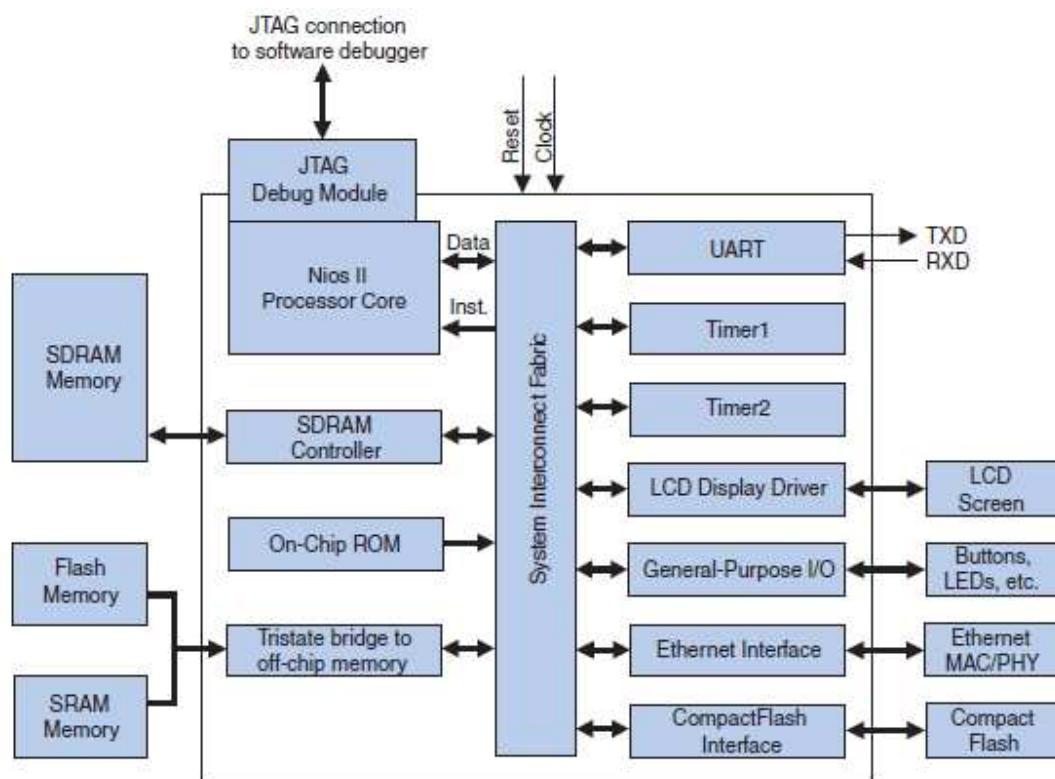
6.1.1.1 Cơ bản về bộ xử lý Nios II và hệ thống Nios II

Bộ xử lý NIOS II là một lõi xử lý RISC đa năng, cung cấp:

- Tập lệnh, đường dẫn dữ liệu và không gian địa chỉ dạng full-32-bit
- 32 thanh ghi đa dụng
- 32 nguồn ngắt ngoài
- Nhân và chia 32×32 bằng lệnh đơn cho kết quả 32 bit
- Lệnh chuyên dụng cho tính toán 64bit và 128bit của phép nhân
- Lệnh dấu chấm động cho những thao tác liên quan đến chấm động cần độ chính xác đơn.
- Giao tiếp với nhiều thiết bị ngoại vi dạng on-chip và interface của bộ nhớ và các ngoại vi dạng off-chip
- Khối trợ giúp gỡ rối có thể làm cho bộ xử lý bắt đầu, dừng, chạy từng bước lệnh hay truy vết thông qua một IDE (intergrated development environment).
- Đơn vị quản lý bộ nhớ (MMU) tùy chọn để hỗ trợ các hệ điều hành yêu cầu MMU.
- Đơn vị bảo vệ bộ nhớ (MPU) tùy chọn.
- Môi trường phát triển phần mềm dựa trên chuỗi công cụ GNU C/C++ và Eclipse IDE.
- Tích hợp với Altera's SignalTap® II Embedded Logic Analyzer, cho phép phân tích thời gian thực các lệnh và dữ liệu cùng với các tín hiệu khác trong thiết kế FPGA.

- Kiến trúc tập lệnh (ISA) tương thích trên tất cả các hệ thống xử lí Nios II.
- Hiệu suất lên đến 250 DMIPS

Một hệ thống Nios II tương đương với một vi điều khiển hoặc “máy tính trên chip”, bao gồm một bộ xử lí và kết hợp các thiết bị ngoại vi, bộ nhớ trên một chip đơn. Một hệ thống xử lí Nios II bao gồm một nhân xử lý Nios II, một tập hợp những thiết bị ngoại vi dạng on-chip, bộ nhớ on-chip và các interface tới bộ nhớ off-chip, tất cả đều được thực hiện trên một thiết bị của Altera. Giống như một họ vi điều khiển, tất cả hệ thống xử lí Nios II đều sử dụng một tập lệnh và một mô hình lập trình chung, nhất quán.



Hình 6.1 Một ví dụ về hệ thống xử lí Nios II

6.1.1.2 Những đặc trưng khác biệt của hệ thống Nios II với các vi điều khiển khác

Phần này giới thiệu những khái niệm riêng của Nios II nhằm cung cấp nền tảng để các nhà thiết kế phần cứng có thể điều chỉnh hệ thống một cách linh hoạt.

6.1.1.2.1 Bộ xử lý nhân mềm khả cấu hình

Nios II là bộ xử lý nhân mềm khả cấu hình, ngược lại với các vi điều khiển cố định sẵn. Trong nội dung này, “cấu hình được” nghĩa là có thể thêm hoặc gỡ bỏ các chức năng trên nền tảng hệ thống để đạt được mục tiêu về hiệu suất hoặc là giá thành. “Nhân mềm” có nghĩa là nhân xử lý không cố định trong silicon và có thể gắn với bất kỳ họ Altera FPGA nào.

Khả năng tái cấu hình không có nghĩa là phải cấu hình mới mỗi lần thiết kế. Altera cung cấp các thiết kế hệ thống Nios II được làm sẵn mà có thể sử dụng ngay. Nếu các thiết kế đó hợp với yêu cầu hệ thống của người sử dụng thì không cần phải thiết kế thêm. Ngoài ra các nhà thiết kế phần mềm có thể dùng chương trình mô phỏng tập lệnh của Nios II để bắt đầu viết và sửa lỗi các ứng dụng chạy trên Nios II trước khi xác định cấu hình phần cứng cuối cùng.

6.1.1.2.2 Tập hợp các ngoại vi và ánh xạ địa chỉ linh động

Tập thiết bị ngoại vi linh động là một trong những điểm khác biệt đáng chú ý giữa hệ thống xử lý Nios II với các vi điều khiển cố định. Với môi trường nhân mềm của Nios II, có thể dễ dàng xây dựng hệ thống xử lý Nios II theo yêu cầu với các tập ngoại vi chính xác đúng với các ứng dụng muốn hướng tới.

Vì sử dụng thiết bị ngoại vi linh hoạt nên sẽ có ánh xạ địa chỉ linh hoạt. Altera cung cấp các cách thức truy cập bộ nhớ và các thiết bị ngoại vi một cách tổng quát, không phụ thuộc vào vị trí địa chỉ. Vì thế các bộ ngoại vi và ánh xạ địa chỉ linh động không ảnh hưởng tới các nhà phát triển ứng dụng.

Có 2 lớp thiết bị ngoại vi: thiết bị ngoại vi chuẩn và thiết bị ngoại vi tùy chỉnh.

■ **Thiết bị ngoại vi chuẩn:** Altera cung cấp một tập hợp các thiết bị ngoại vi thông dụng, thường dùng trong các vi điều khiển như bộ đếm (timer), giao diện cho giao tiếp nối tiếp (serial communication interfaces), xuất nhập đa dụng (general-purpose I/O), bộ điều khiển SDRAM, và các giao diện bộ nhớ khác. Altera và các đơn vị liên quan vẫn đang tiếp tục phát triển danh sách các thiết bị ngoại vi này.

■ **Thiết bị ngoại vi tùy chỉnh:** Cũng có thể tạo ra các thiết bị ngoại vi tùy chỉnh theo yêu cầu của ứng dụng và tích hợp chúng vào hệ thống xử lý Nios II. Đối với các hệ thống chú trọng vào hiệu năng, phải thường xuyên thực thi một chức năng riêng biệt, đặc trưng nào đó thì kỹ thuật thông thường sử dụng là tạo thiết bị ngoại vi tùy chỉnh thực hiện chức năng đó thay vì phần mềm. Cách tiếp cận này đem lại lợi ích về hiệu năng: phần cứng thực thi nhanh hơn phần mềm; và bộ xử lý được giải phóng để thực thi các tác vụ song song khác trong khi thiết bị ngoại vi tùy chỉnh sẽ thao tác với dữ liệu.

■ **Lệnh tùy chỉnh**

Giống như thiết bị ngoại vi tùy chỉnh, lệnh cũng cho phép tùy chỉnh theo yêu cầu để tăng hiệu năng của hệ thống bằng cách nâng cấp bộ xử lý với phần cứng tùy chỉnh. Bản chất nhân mềm của Nios II cho phép tích hợp các thành phần logic tùy chỉnh vào đơn vị luận lý số học ALU. Tương tự như các lệnh có sẵn trong Nios II, lệnh tùy chỉnh có thể lấy giá trị tối đa trong 2 thanh ghi nguồn và có thể ghi lại kết quả vào thanh ghi đích tùy ý.

6.1.1.2.3 Tự động phát sinh hệ thống

Công cụ thiết kế SOPC Builder của Altera hoàn toàn tự động hóa quá trình cấu hình những đặc trưng của bộ xử lý và sinh ra một thiết kế phần cứng lập trình được trên FPGA. SOPC Builder cung cấp giao diện đồ họa cho phép cấu hình hệ xử lý Nios II với số lượng thiết bị ngoại vi hay bộ nhớ không hạn chế. Có thể tạo hệ thống xử lý hoàn chỉnh mà không cần thực hiện bất kì một phác họa hay ngôn ngữ mô tả phần cứng (HDL) nào. SOPC Builder cũng có thể nhận các file HDL, cung cấp cách đơn giản để tích hợp các thành phần logic tùy chọn vào trong hệ xử lý Nios II.

Sau quá trình phát sinh hệ thống có thể đưa thiết kế vào bo mạch và kiểm tra lỗi thực thi phần mềm trên đó. Quá trình phát triển phần mềm được thực hiện tương tự như với các bộ xử lý không tái cấu hình được.

6.1.1.3 Các loại nhân xử lý Nios II

Có ba dạng nhân đặc trưng của bộ xử lý Nios II. Tất cả các nhân đều hỗ trợ kiến trúc tập lệnh của Nios II.

- “Nios II/f Core”
- “Nios II/s Core”
- “Nios II/e Core”

Nhân Nios II/f:

Nhân Nios II/f (fast) nhanh được thiết kế cho hiệu năng thực thi cao.

Altera đã thiết kế nhân Nios II/f cho các ứng dụng yêu cầu cao về hiệu suất, nhất là với các ứng dụng có nhiều code và dữ liệu, như các hệ thống hoạt động trong một hệ điều hành chứa đầy đủ tính năng. Nhân Nios II/f có cấu hình như sau:

- Có bộ nhớ đệm lệnh và dữ liệu riêng.

- Cung cấp tùy chọn MMU để hỗ trợ các hệ điều hành cần MMU.
- Cung cấp tùy chọn MPU để hỗ trợ các hệ điều hành và môi trường thực thi cần phải bảo vệ bộ nhớ nhưng không cần bộ quản lý bộ nhớ ảo.
- Có thể có 2Gbyte không gian địa chỉ ngoài khi không có MMU và 4Gbyte khi có MMU.
- Hỗ trợ bộ nhớ tightly-coupled cho lệnh và dữ liệu.
- Thực thi tiên đoán nhánh động.
- Cung cấp tùy chọn nhân, chia, dịch bằng phần cứng để tăng tốc xử lý số học.
- Hỗ trợ lệnh tùy biến.
- Nios II/f hỗ trợ khối JTAG debug

Nhân Nios II/s:

Nios II/s (standar) chuẩn được thiết kế cho dạng core kích thước nhỏ, chuẩn.

Altera thiết kế Nios II/s cho các ứng dụng mà hiệu suất ở dạng trung bình và hiệu suất không phải là tiêu chuẩn được đòi hỏi cao nhất. Nhân Nios II/s có cấu hình như sau

- Có bộ nhớ đệm cho lệnh nhưng không có bộ nhớ đệm cho dữ liệu
- Có truy xuất lên đến 2 Gbytes không gian địa chỉ trống bên ngoài.
- Hỗ trợ bộ nhớ tightly-coupled cho lệnh
- Thực hiện tiên đoán nhánh tinh
- Cung cấp tùy chọn nhân, chia, dịch bằng phần cứng để tăng tốc xử lý số học
- Hỗ trợ thêm lệnh tùy biến.
- Hỗ trợ khối JTAG debug

Nhân Nios II/e:

Nios II/e (ecommy) kinh tế được thiết kế để đạt được khích thước core nhỏ nhất có thể.

Altera thiết kế Nios II/e với mục tiêu giảm tài nguyên sử dụng bằng mọi cách có thể trong khi vẫn duy trì được khả năng tương thích với tập lệnh Nios II. Nios II/e core chỉ bằng khoảng nửa kích thước của Nios II/s core, nhưng hiệu suất thực thi thực chất thấp hơn. Nhân Nios II/e có cấu hình như sau:

- Thực thi tối đa một lệnh mỗi sáu chu kỳ đồng hồ.
- Có thể truy xuất lên đến 2 Gbytes của địa chỉ trống bên ngoài.
- Hỗ trợ bổ sung của lệnh tùy biến.
- Hỗ trợ JTAG debug module.
- Không có bộ đệm lệnh hoặc bộ đệm dữ liệu
- Không thực hiện tiên đoán phân nhánh

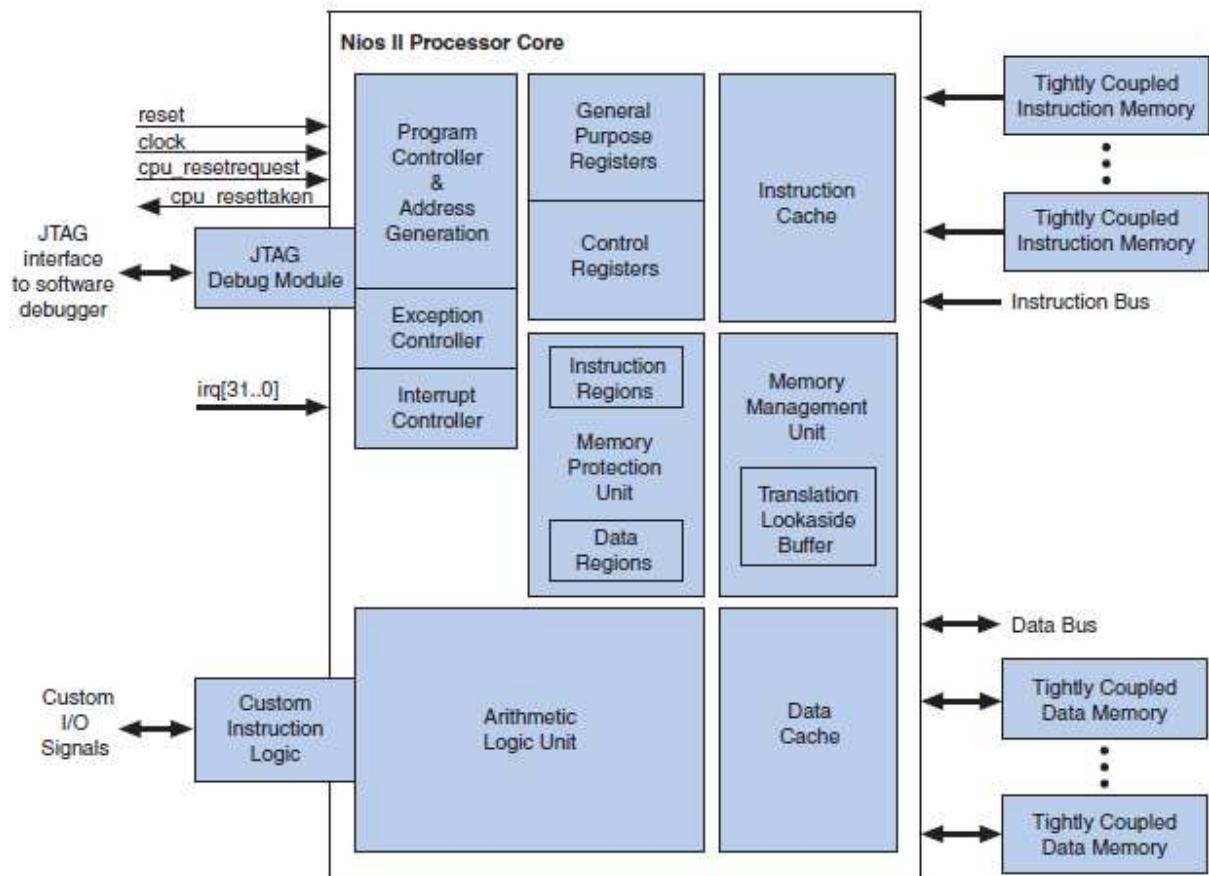
6.1.2 Kiến trúc bộ xử lý Nios II

6.1.2.1 Giới thiệu

Kiến trúc Nios II gồm các đơn vị chức năng chính:

- Tập thanh ghi
- Đơn vị luận lý số học (ALU)
- Interface cho lệnh tùy chỉnh
- Bộ điều khiển ngoại lệ
- Bộ điều khiển ngắt
- Bus lệnh
- Bus dữ liệu
- Đơn vị quản lý bộ nhớ (MMU)
- Đơn vị bảo vệ bộ nhớ (MPU)

- Bộ nhớ đệm (dữ liệu và lệnh)
- Interface cho bộ nhớ tightly-coupled cho lệnh và dữ liệu
- JTAG debug module



Hình 6.2 Sơ đồ khái niệm Nios II processor

6.1.2.2 Tập thanh ghi

Kiến trúc Nios II hỗ trợ một tập tin thanh ghi phẳng, chứa 32 thanh ghi đa dụng 32-bit, và lên đến 32 thanh ghi điều khiển 32-bit. Kiến trúc này hỗ trợ chế độ giám sát (supervisor mode) và chế độ người dùng (user mode), cho phép bảo vệ

các thanh ghi điều khiển từ các ứng dụng sai. Trong tương lai, kiến trúc Nios II cho phép bổ sung những thanh ghi chấm động.

6.1.2.3 Đơn vị luận lý số học (ALU)

ALU của Nios II hoạt động trên dữ liệu được lưu trữ trong các thanh ghi đa dụng. Các thao tác của ALU nhận một hoặc hai dữ liệu đầu vào từ thanh ghi, và lưu trữ kết quả trở lại thanh ghi. ALU hỗ trợ các thao tác được chỉ ra trong Bảng 6.1

Bảng 6.1 Thao tác hỗ trợ bởi Nios II ALU

Loại	Chi tiết
Số học	ALU hỗ trợ các thao tác cộng, trừ, nhân, và chia có dấu và không
Quan hệ	ALU hỗ trợ các thao tác ==, !=, >=, < có dấu và không dấu.
Luận lý	ALU hỗ trợ thao tác luận lý AND, OR, NOR, XOR
Dịch và xoay	ALU hỗ trợ thao tác dịch và xoay, và mỗi lệnh có thể dịch/xoay dữ liệu 0-31 bit. ALU hỗ trợ dịch phải/trái dạng số học và luận lý. ALU hỗ trợ xoay trái / phải.

Để hiện thực bất kỳ thao tác nào khác, phần mềm sẽ tính toán kết quả bằng cách kết hợp các thao tác cơ bản trong Bảng 6.1.

Ngoài ra, kiến trúc NIOS II hỗ trợ lệnh tùy chỉnh theo định nghĩa của người dùng, tham khảo thêm *NiosII Custom Instruction User Guide* của Altera.

6.1.2.4 Bộ điều khiển ngoại lệ và bộ điều khiển ngắt (Exception and Interrupt Controller)

6.1.2.4.1 Bộ điều khiển ngoại lệ (Exception Controller)

Kiến trúc Nios II cung cấp một bộ điều khiển ngoại lệ vô hướng đơn giản để xử lý tất cả loại ngoại lệ. Mỗi ngoại lệ, bao gồm cả ngắt phần cứng, làm cho bộ xử lý phải chuyển giao thực thi đến một địa chỉ ngoại lệ. Bộ điều khiển ngoại lệ tại địa

chỉ này xác định nguyên nhân gây ra ngoại lệ và thực hiện một thủ tục xử lý ngoại lệ nào đó.

Địa chỉ ngoại lệ được quy định trong SOPC Builder tại thời điểm sinh hệ thống.

6.1.2.4.2 Bộ điều khiển ngắt tích hợp (Integral Interrupt Controller)

Kiến trúc Nios II hỗ trợ 32 ngắt cứng ngoài. Bộ xử lý có 32 đầu vào yêu cầu ngắt (IRQ), từ irq0 đến irq31. Độ ưu tiên IRQ được xác định bằng phần mềm. Kiến trúc Nios II hỗ trợ dạng ngắt lồng nhau.

Phần mềm có thể cho phép hoặc không cho phép bắt cứ nguồn ngắt nào một cách riêng lẽ thông qua thanh ghi điều khiển *ienable*. Mỗi bit trong thanh ghi này tương ứng với việc cho phép hoặc không cho phép một IRQ. Phần mềm cũng có thể cho phép hoặc không cho phép ngắt toàn cục bằng cách sử dụng bit PIE của thanh ghi điều khiển *status*. Một ngắt phần cứng được tạo ra khi và chỉ khi tất cả các điều kiện sau là đúng:

- Bit PIE của thanh ghi trạng thái là 1
- Một đầu vào yêu cầu ngắt, $\text{irq}<\text{n}>$, được xác nhận
- bit tương ứng thứ n của thanh ghi *ienable* là 1

6.1.2.5 Dữ liệu và lệnh

Kiến trúc của Nios II hỗ trợ các bus dữ liệu và lệnh riêng biệt, như kiến trúc Harvard. Bus dữ liệu thông qua cổng dữ liệu chính kết nối với cả bộ nhớ và ngoại vi, trong khi đó bus lệnh thông qua cổng lệnh chính chỉ kết nối với bộ nhớ.

Cấu trúc Nios II truy xuất I/O và bộ nhớ theo dạng: cả hai bộ nhớ dữ liệu và thiết bị ngoại vi đều được ánh xạ vào không gian địa chỉ của cổng dữ liệu chính. Kiến trúc lưu trữ của Nios II theo dạng *little endian*, các từ và nửa từ được lưu trữ trong bộ nhớ với bytes cao thì lưu trữ ở địa chỉ cao.

6.1.2.6 Bộ nhớ đệm (cache)

Kiến trúc Nios II hỗ trợ bộ nhớ đệm lệnh và đệm dữ liệu. Bộ nhớ đệm nằm trên chip như là một phần không thể thiếu của nhân xử lý Nios II. Bộ nhớ đệm giúp tăng thời gian truy cập bộ nhớ cho những hệ thống xử lý Nios II mà sử dụng bộ nhớ ngoài chip tốc độ chậm như SDRAM. Việc quản lý và liên kết bộ nhớ đệm được điều khiển bằng phần mềm. Tập lệnh Nios II cung cấp một số lệnh cho việc quản lý bộ nhớ đệm

Việc dùng bộ nhớ *cache* là tùy chọn, tùy vào ứng dụng. Một nhân xử lý Nios II có thể bao gồm một, hoặc hai, hoặc không có bộ nhớ *cache* nào cả. Hơn nữa, kích thước của bộ nhớ *cache* có thể được cấu hình theo ý người sử dụng. Việc đưa bộ nhớ *cache* vào sẽ không ảnh hưởng đến chức năng của chương trình, nhưng ảnh hưởng đến tốc độ tìm nạp lệnh, đọc hoặc ghi dữ liệu của bộ xử lý.

6.1.2.7 Bộ nhớ tightly-coupled

Bộ nhớ *tightly-coupled* đảm bảo truy cập bộ nhớ với độ trễ thấp cho các ứng dụng chú trọng hiệu năng. So với bộ nhớ cache, bộ nhớ *tightly-coupled* cung cấp các lợi ích sau:

- Hiệu năng giống với bộ nhớ *cache*
- Phần mềm có thể đảm bảo code hoặc dữ liệu nào chú trọng hiệu năng sẽ được đặt trong bộ nhớ *tightly-coupled*.
- Không xảy ra những tình trạng như tải, vô hiệu hóa hay phục hồi lại bộ nhớ trong quá trình chạy

Về mặt vật lý, cổng bộ nhớ *tightly-coupled* là một cổng chính riêng biệt trên nhân xử lý Nios II, tương tự cổng lệnh hoặc cổng dữ liệu chính. Một nhân Nios II

có thể không có, hoặc một, hoặc nhiều bộ nhớ *tightly-coupled*. Kiến trúc Nios II hỗ trợ bộ nhớ *tightly-coupled* cho cả truy cập lệnh và truy cập dữ liệu.

Bộ nhớ *tightly-coupled* chiếm không gian địa chỉ thông thường, giống như các thiết bị bộ nhớ khác được kết nối thông qua cơ cấu kết nối hệ thống. Dãy địa chỉ dành cho bộ nhớ *tightly-coupled* (nếu có) được xác định tại thời điểm phát sinh hệ thống.

Phần mềm truy cập bộ nhớ *tightly-coupled* thường sử dụng lệnh dạng *load* và *store*. Từ khía cạnh của phần mềm, không thấy sự khác biệt giữa truy cập bộ nhớ *tightly-coupled* với những bộ nhớ khác.

Hiệu quả sử dụng của bộ nhớ *tightly-coupled*: Một hệ thống có thể sử dụng bộ nhớ *tightly-coupled* để đạt được hiệu suất tối đa khi truy cập vào một phần đặc biệt nào đó của code hoặc dữ liệu. Ví dụ, các ứng dụng liên quan đến interrupt có thể đặt đoạn code xử lý ngoại lệ vào bộ nhớ *tightly-coupled* để giảm thiểu độ trễ khi xảy ra interrupt. Tương tự như vậy, với các ứng dụng liên quan đến xử lý tín hiệu số (DSP) có thể đặt một bộ đệm dữ liệu vào bộ nhớ *tightly-coupled* để truy cập dữ liệu nhanh nhất có thể.

Nếu yêu cầu về bộ nhớ của ứng dụng đủ nhỏ để gắn hoàn toàn vào chip, có thể sử dụng bộ nhớ *tightly-coupled* dành riêng cho mã và dữ liệu. Với các ứng dụng lớn hơn phải lựa chọn các phần đưa vào *tightly-coupled* sao cho cân bằng giữa hiệu suất và chi phí.

6.1.2.8 Đơn vị quản lý bộ nhớ (MMU)

Nios II MMU cung cấp những chức năng sau:

- Ánh xạ địa chỉ ảo thành địa chỉ vật lý
- Bảo vệ bộ nhớ

- 32-bit địa chỉ ảo và địa chỉ vật lý, ánh xạ 4 GB không gian địa chỉ ảo vào 4 GBytes bộ nhớ vật lý.
- 4 Kbyte kích thước trang và khung.
- Dưới 512 MBytes không gian địa chỉ vật lý có sẵn cho việc truy cập trực tiếp.

Khi khởi tạo bộ xử lý Nios II, có thể chọn đưa MMU vào hay không, và với một số thông số có sẵn có thể cho phép tối ưu hóa MMU cho hệ thống khi cần.

Lưu ý: Đơn vị quản lý bộ nhớ MMU và đơn vị bảo vệ bộ nhớ MPU loại trừ lẫn nhau. Hệ thống Nios II hoặc là chứa MMU, hoặc là chứa MPU, nhưng không thể chứa cả hai trong cùng một thiết kế.

6.1.2.9 Đơn vị bảo vệ bộ nhớ (MPU)

Bộ xử lý Nios II cung cấp MPU cho hệ điều hành và môi trường thời gian thực để bảo vệ bộ nhớ nhưng không yêu cầu quản lý bộ nhớ ảo.

Nios II MPU cung cấp những tính năng và nhiệm vụ sau:

- Bảo vệ bộ nhớ
- Lê đến 32 vùng lệnh và 32 vùng dữ liệu
- Kích thước vùng dữ liệu và vùng lệnh có thể thay đổi
- Độ lớn của bộ nhớ được xác định bằng kích thước của nó hoặc giới hạn địa chỉ trên cùng.
- Cấp quyền đọc và ghi trong vùng dữ liệu
- Cấp quyền thực thi trong vùng lệnh
- Các vùng chồng chéo nhau

Khi khởi tạo bộ xử lý Nios II, có thể chọn đưa MMU vào hay không và với một số thông số có sẵn có thể cho phép tối ưu hóa MMU cho hệ thống khi cần.

Lưu ý: Đơn vị quản lý bộ nhớ MMU và đơn vị bảo vệ bộ nhớ MPU loại trừ lẫn nhau. Hệ thống Nios II hoặc là chứa MMU, hoặc là chứa MPU, nhưng không thể chứa cả hai trong cùng một thiết kế.

6.1.2.10 Khối JTAG Debug

Kiến trúc NIOS II hỗ trợ khối JTAG debug nhằm điều khiển, theo dõi bộ xử lý từ xa thông qua một máy PC nào đó. Những công cụ debug dạng phần mềm chạy trên PC kết nối với khói JTAG debug của Nios II cung cấp những tính năng và nhiệm vụ sau:

- Tải chương trình tới bộ nhớ
- Khởi động và ngừng thực thi chương trình
- Cài đặt các breakpoints và watchpoints
- Phân tích các thanh ghi và bộ nhớ
- Truy vết khi chương trình thực thi

6.1.3 Mô hình lập trình

Chương này mô tả mô hình lập trình Nios® II, bao gồm các tính năng của bộ xử lý ở cấp độ hợp ngữ. Để hiểu biết đầy đủ các nội dung của chương này đòi hỏi phải có kiến thức trước về kiến trúc máy tính, hệ điều hành, quản lý bộ nhớ và bộ nhớ ảo, quản lý tiến trình, xử lý ngoại lệ, và tập lệnh, ...

6.1.3.1 Các chế độ hoạt động Nios II

Các chế độ hoạt động kiểm soát cách thức bộ xử lý hoạt động, quản lý bộ nhớ hệ thống và truy xuất ngoại vi. Kiến trúc Nios II hỗ trợ các chế độ hoạt động sau:

- Chế độ giám sát (supervisor mode)

- Chế độ người dùng (user mode)

- Supervisor mode:

Supervisor mode không hạn chế hoạt động của bộ xử lý. Bộ xử lý có thể thực hiện bất kỳ hoạt động nào mà kiến trúc Nios II cung cấp. Bất kì lệnh nào cũng có thể được thực thi, và bất kì hoạt động nhập xuất đều có thể được khởi tạo, và bất kỳ vùng nhớ nào đều có thể truy cập được.

Các hệ điều hành và phần mềm hệ thống khác đều chạy trong supervisor mode. Trong các hệ thống có MMU, ứng dụng chạy trong user mode và hệ điều hành chạy trong supervisor mode nhằm kiểm soát sự truy cập của ứng dụng vào bộ nhớ và các thiết bị ngoại vi. Trong các hệ thống có MPU, phần mềm hệ thống kiểm soát chế độ mà ứng dụng chạy trong đó. Trong hệ thống Nios II mà không có MMU hoặc MPU, tất cả các mã ứng dụng và mã hệ thống đều chạy trong supervisor mode.

- User mode:

User mode sẵn sàng chỉ khi trong thiết kế của bộ xử lý Nios II có MMU hoặc MPU. User mode thông thường chỉ để hỗ trợ các hệ điều hành vì các hệ điều hành đều chạy các ứng dụng ở user mode. Các khả năng của bộ xử lý trong user mode là tập con trong supervisor mode, tức chỉ một tập con các lệnh trong tập lệnh của Nios II có thể sử dụng trong user mode.

Hệ điều hành quyết định vùng bộ nhớ nào có thể truy cập bởi các ứng dụng trong user mode. Việc cố gắng truy cập vào các vị trí ô nhớ mà các ứng dụng trong user mode không được phép sẽ gây ra một ngoại lệ. Code chạy trong user mode sử dụng lời gọi hệ thống để yêu cầu hệ điều hành thực hiện các hoạt động I/O, quản lý bộ nhớ, hay truy cập đến các chức năng hệ thống khác trong supervisor mode

6.1.3.2 Các thanh ghi đa dụng

Kiến trúc Nios II cung cấp 32 thanh ghi đa dụng 32-bit, từ r0 đến r31, như trong Bảng 6.2

Bảng 6.2 Các thanh ghi đa dụng của NIOS II

Thanh ghi	Tên	Chức năng	Thanh ghi	Tên	Chức năng
r0	Zero	0x00000000	r16		
r1	at	Assembler temporary	r17		
r2		Giá trị trả về	r18		
r3		Giá trị trả về	r19		
r4		Đối số thanh ghi	r20		
r5		Đối số thanh ghi	r21		
r6		Đối số thanh ghi	r22		
r7		Đối số thanh ghi	r23		
r8		caller saved register	r24	et	Exception temporary
r9		caller saved register	r25	bt	Breakpoint temporary (1)
r10		caller saved register	r26	gp	Con trỏ toàn cục
r11		caller saved register	r27	sp	Con trỏ stack
r12		caller saved register	r28	fp	Con trỏ frame
r13		caller saved register	r29	ea	Địa chỉ trả về của ngoại lệ
r14		caller saved register	r30	ba	Địa chỉ trả về Breakpoint (1)
r15		caller saved register	r31	ra	Địa chỉ trả về

Ghi chú Bảng 6.2:

- (1) Thanh ghi này được dành riêng cho JTAG debug module.

6.1.3.3 Các thanh ghi điều khiển

Các thanh ghi điều khiển cho biết trạng thái và thay đổi hành vi của bộ xử lý. Cách thức truy cập của chúng khác với thanh ghi đa dụng. Các lệnh đặc biệt *rdctl* và *wrctl* là các lệnh duy nhất dùng để đọc, ghi vào thanh ghi điều khiển và chúng chỉ hoạt động được trong chế độ giám sát (supervisor mode).

Lưu ý: Khi ghi lên một thanh ghi điều khiển, tất cả các bit không được định nghĩa phải mang giá trị 0.

Kiến trúc Nios II hỗ trợ lên đến 32 thanh ghi điều khiển. Bảng 6.3 cho biết chi tiết về các thanh ghi điều khiển.

Bảng 6.3 Các bit và tên của thanh ghi điều khiển

Register	Tên	Nội dung
0	status	
1	estatus	
2	estatus	
3	ienable	Những bit cho phép ngắt
4	ipending	Những bit chờ ngắt
5	cpuid	ID của bộ xử lý
6	reserved	Dự trữ
7	exception	
8	pteaddr (1)	
9	tlbacc (1)	
10	tlbmisc (1)	
11	reserved	Dự trữ
12	badaddr	
13	config (2)	
14	mpubase (2)	
15	mpuacc (2)	
16-31	reserved	Dự trữ

Ghi chú cho Bảng 6.3:

- (1) Sử dụng chỉ khi có MMU. Nếu không sẽ trở thành thanh ghi lưu trữ.

(2) Sử dụng chỉ khi có MPU. Nếu không sẽ trở thành thanh ghi lưu trữ.

✚ Một vài thanh ghi điều khiển thông dụng

❖ Thanh ghi status

Giá trị của thanh ghi *status* sẽ điều khiển trạng thái của bộ xử lý NIOS II. Tất cả các bit trạng thái sẽ trở về 0 khi bộ xử lý khởi động lại. Một số bit được dùng riêng và chỉ thích hợp cho một số tính năng của bộ xử lý. Bảng 6.4 cho thấy cách bố trí của thanh ghi này.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																															

Bảng 6.4 Cho biết chi tiết các trường được định nghĩa trong thanh ghi status

Bit	Mô tả	Truy cập	Reset	Tương thích
EH <i>(1)</i>	EH là bit xử lý ngoại lệ. Bộ xử lý bật EH lên 1 khi có ngoại lệ. Khi đã xử lý xong ngoại lệ, phần mềm sẽ xóa EH về 0. Trong hệ thống không có MMU, EH luôn bằng 0. MMU sử dụng EH để xác định xem ngoại lệ TLB miss vừa xảy ra là TLB miss tốc độ cao hay TLB miss kép.	Đọc /Ghi	0	Chỉ với MMU
U <i>(1)</i>	U là bit cho user mode. Khi U = 1 bộ xử lý hoạt động ở user mode. Khi U = 0 bộ xử lý sẽ hoạt động ở supervisor mode. Trong các hệ thống không có MMU thì U luôn bằng 0.	Đọc/Ghi	0	Với MMU hoặc MPU
PIE	PIE là bit cho phép ngắt của bộ xử lý. Khi PIE = 0, các ngắt sẽ bị bỏ qua. Khi PIE = 1 có thể ngắt hay không tùy thuộc vào giá trị của thanh ghi <i>ienable</i> .	Đọc/Ghi	0	Tất cả

Ghi chú:

- (1) EH và U cùng là bit 1 thì không được phép và có thể sẽ cho ra kết quả sai.

❖ Thanh ghi estatus

Thanh ghi estatus làm nhiệm vụ lưu giữ một bản sao của thanh ghi *status* trong suốt quá trình xử lý ngoại lệ dạng non-break. Bảng 6.5 cho thấy cách bố trí của thanh ghi *estatus*.

Bảng 6.5 Các trường của thanh ghi điều khiển estatus

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																															

Tên các trường trong *estatus* giống các trường trong *status* ngoại trừ việc thêm tiền tố “E”.

Bộ xử lý ngoại lệ có thể kiểm tra *estatus* để xác định trạng thái trước ngoại lệ của bộ xử lý. Khi một ngoại lệ đã được xử lý xong, lệnh *eret* thực hiện làm cho bộ xử lý sao chép nội dung từ *estatus* sang *status*, qua đó khôi phục giá trị của *status* trước khi ngoại lệ xảy ra.

Đọc thêm phần “Tiến trình xử lý ngoại lệ” để biết thêm chi tiết

❖ Thanh ghi *bstatus*

Thanh ghi *bstatus* giữ bản sao lưu của thanh ghi *status* trong quá trình xử lý ngoại lệ dạng break. Bảng 6.6 cho thấy cách bố trí của *bstatus*

Bảng 6.6 Các trường của thanh ghi điều khiển *bstatus*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																															

Tên các trường trong *bstatus* giống các trường trong *status* ngoại trừ việc thêm tiền tố ”B”.

Khi một break xuất hiện, giá trị của *status* được sao lưu vào *bstatus*. Bằng cách sử dụng *bstatus*, debugger có thể phục hồi lại giá trị trước khi ngắt của *status*. Lệnh *bret* thực hiện làm cho bộ xử lý sao chép giá trị từ *bstatus* vào lại *status*.

❖ Thanh ghi *ienable*

Thanh ghi *ienable* dùng để điều khiển các ngắt cứng ngoài. Mỗi bit của *ienable* tương ứng với một đầu vào interrupt, từ *irq0* đến *irq31*. Nói cách khác nếu bit n của *ienable* có giá trị 1 thì interrupt thứ n được phép, nếu giá trị 0 có nghĩa là không được phép.

❖ Thanh ghi *ipending*

Thanh ghi *ipending* cho biết tín hiệu interrupt nào đang tác động vào bộ xử lý. Nói cách khác, nếu bit thứ n có giá trị 1 thì interrupt thứ n được khẳng định đang tác động. Lưu ý, thao tác ghi tới thanh ghi *ipending* không có tác dụng

❖ Thanh ghi *cupid*

Thanh ghi *cupid* nắm giữ một hằng số dùng phân biệt các bộ xử lý trong hệ thống đa xử lý, hay còn gọi là nắm giữ các ID của các bộ xử lý. Giá trị *cupid* được xác định ở thời điểm phát sinh hệ thống và được đảm bảo là mỗi bộ xử lý trong hệ thống có một ID duy nhất. Lưu ý, thao tác ghi tới thanh ghi *cupid* không có tác dụng.

6.1.3.4 Tiến trình xử lý ngoại lệ

Ngoại lệ là quá trình chuyển quyền điều khiển ra khỏi luồng thực thi bình thường của chương trình, được tạo ra bởi một sự kiện nào đó, bên trong hay bên ngoài bộ xử lý mà đòi hỏi phải được chú ý ngay lập tức. Xử lý ngoại lệ là ứng phó với một ngoại lệ và sau đó đưa hệ thống trở lại trạng thái trước khi ngoại lệ xảy ra.

Các ngoại lệ trong Nios II thuộc một trong các loại sau:

- Ngoại lệ khởi động lại (Reset exceptions)
- Ngoại lệ break (Break exceptions)
- Ngoại lệ ngắt (Interrupt exceptions)
- Ngoại lệ liên quan đến lệnh (Instruction-related exceptions)

Ngoại lệ khởi động lại (Reset exceptions)

Khi một tín hiệu reset bộ xử lý được đưa vào, bộ xử lý Nios II thực hiện các bước sau:

- Xóa thanh ghi *status* thành 0x0. Việc xoá thanh ghi *status* sẽ vô hiệu hóa các interrupt cứng. Nếu có MMU hoặc MPU, xóa thanh ghi *status* sẽ buộc bộ xử lý chuyển sang supervisor mode.
- Làm mất hiệu lực dòng cache lệnh tương ứng với vector reset. Việc này được thực hiện nhằm đảm bảo rằng lệnh nạp cho code reset đến từ vùng nhớ không cache
- Bắt đầu thực thi bộ điều khiển reset, được cấp phát tại vector reset.

Lưu ý nội dung của những vùng nhớ đệm là không xác định sau khi reset. Do đó bộ điều khiển reset phải ngay lập tức khởi tạo lại vùng cache lệnh. Tiếp theo, hoặc là bộ xử lý reset hoặc là một hàm tiếp theo nào đó phải tiến hành khởi tạo lại vùng cache dữ liệu.

Ngoại lệ break (Break exceptions)

Ngoại lệ dạng break là quá trình chuyển quyền điều khiển ra khỏi luồng thực thi bình thường của chương trình cho mục đích debug. Công cụ gỡ rối mềm có thể lấy quyền kiểm soát của bộ vi xử lý Nios II thông qua khói JTAG debug.

Xử lý break nghĩa là bằng cách sử dụng công cụ gỡ rối bằng phần mềm cài đặt tính năng gỡ lỗi và chẩn đoán, như các điểm ngắt (breakpoint) và điểm nhìn (watchpoint). Xử lý break là một dạng của xử lý ngoại lệ, nhưng cơ chế của break thì độc lập với quá trình xử lý ngoại lệ thông thường.

Bộ xử lý rơi vào trạng thái xử lý break trong các điều kiện sau đây:

- Bộ xử lý thực thi lệnh break mà thường được xem như một break mềm
- JTAG debug module xác nhận một ngắt cứng.

Một break xảy ra, bộ xử lý phải thực hiện các bước sau :

- Lưu trữ nội dung thanh ghi *status* vào thanh ghi *bstatus*.
- Xóa trường *status.PIE* thành 0, vô hiệu hóa các ngắt ngoài.
- Ghi địa chỉ lệnh theo sau break vào thanh ghi *ba* (*r30*).
- Xóa trường *status.U* thành 0, buộc bộ xử lý vào chế độ supervisor mode khi hệ thống chưa MMU hoặc MPU.
- Đặt trường *status.EH* thành 1, chỉ ra bộ xử lý đang nắm giữ ngoại lệ khi hệ thống có chưa MMU.
- Chuyển quyền thực thi cho bộ xử lý break. Địa chỉ của bộ xử lý break được lưu trữ trong vector break, được xác định tại thời điểm phát sinh hệ thống.

Thanh ghi *bstatus* và các thanh ghi đa dụng *bt* (*r25*), *ba* (*r30*) được dành cho debug. Lưu ý là việc code có thể sử dụng được các thanh ghi này, nhưng debug có thể làm cho các giá trị trong các thanh ghi này bị ghi đè. Bộ xử lý break có thể sử dụng *bt* (*r25*) để giúp lưu các thanh ghi thêm vào.

Sau khi xử lý một break, bộ xử lý break giải phóng quyền điều khiển bộ xử lý trả về luồng thực thi ban đầu bằng cách thực hiện lệnh *bret*. Lệnh *bret* này phục hồi lại giá trị của thanh ghi *status* bằng cách sao chép nội dung thanh ghi *bstatus* về *status* và trả về chương trình thực thi ban đầu tại địa chỉ được lưu trong thanh ghi *ba* (*r30*). Ngoài thanh ghi *bt*, tất cả các thanh ghi đều được đảm bảo trả lại giá trị trước khi break xảy ra.

Ngoại lệ ngắt (Interrupt exceptions)

Tài nguyên bên ngoài như thiết bị ngoại vi có thể tạo một interrupt cứng bằng cách tác động đến một trong 32 đầu vào interrupt của bộ xử lý, tính từ *irq0* qua *irq31*. Một ngắt cứng được tạo ra khi và chỉ khi cả ba điều kiện sau là đúng:

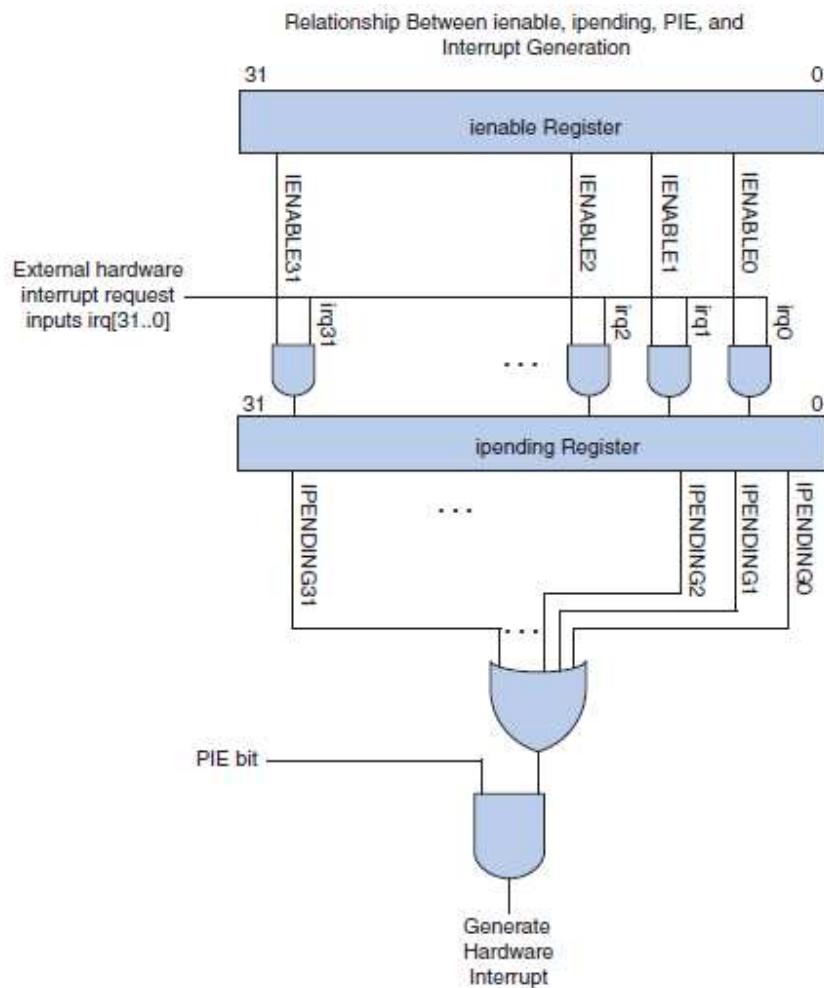
- Bit *PIE* của SR bằng một.

- Một đầu vào interrupt được tác động lên 1 (*irqn*)
- Bit thứ n tương ứng của thanh ghi *ienable* bằng 1.

Trong thời gian xử lý ngắt cứng, bộ xử lý xóa bit PIE về 0, không cho phép thêm các interrupt, sau đó thực hiện các bước khác để xử lý ngắt.

Giá trị của thanh ghi điều khiển *ipending* chỉ ra interrupt nào đang chờ giải quyết. Hình 3-1 cho thấy mối quan hệ giữa *ipending*, *ienable*, *PIE*, và sự phát sinh của một interrupt.

Nios II có một đoạn chương trình con dùng để xác định xem trong số những ngắt đang chờ giải quyết, ngắt nào có độ ưu tiên cao nhất, và sau đó chuyển quyền điều khiển cho đoạn chương trình xử lý ngắt đó (còn gọi là ISR _ interrupt service routine).



Hình 6.3 Mạch mô tả phần cứng các thanh ghi phục vụ ngắt

✚ Ngoại lệ liên quan lệnh (Instruction-related exceptions)

Những ngoại lệ liên quan đến lệnh xảy ra suốt trong quá trình thực thi lệnh Nios II. Các trường hợp xảy ra ngoại lệ dạng này:

- Lệnh trap: Khi một chương trình cấp phát lệnh trap, nó sinh ra một trap exception mềm. Một chương trình phát sinh trap khi chương trình yêu cầu phục vụ bởi hệ điều hành. Bộ xử lý ngoại lệ tổng quát cho hệ điều hành sẽ xác định nguyên nhân cho trap và phản ứng lại một cách thích hợp.

- Lệnh break: Lệnh break được coi là gây ra một ngoại lệ break mềm
- Lệnh chưa hiện thực (unimplemented): Khi bộ xử lý phát một lệnh hợp lệ mà chưa được hiện thực trong phần cứng, một ngoại lệ cho lệnh chưa được hiện thực sẽ phát sinh. Bộ xử lý ngoại lệ tổng quát sẽ xác định lệnh nào tạo ra ngoại lệ. Nếu lệnh không được hiện thực trong phần cứng, quyền điều khiển được chuyển cho một thủ tục xử lý ngoại lệ, thủ tục này có thể chọn lựa để mô phỏng lệnh bằng phần mềm.
- Lệnh không hợp lệ: Lệnh không hợp lệ là các lệnh có trường opcode hoặc opcode mở rộng không xác định. Bộ xử lý Nios II có thể kiểm tra để biết các lệnh không hợp lệ và tạo ra ngoại lệ khi gặp chúng. Khi hệ thống có chứa MMU hoặc MPU thì việc kiểm tra các lệnh không hợp lệ luôn thực hiện. Khi hệ thống không có MMU hoặc MPU, việc có kiểm tra các lệnh bất hợp lệ hay không là tùy ý.
- Lệnh dành riêng chế độ supervisor mode: Khi hệ thống chứa MMU hoặc MPU và bộ xử lý đang trong chế độ user mode (status.U = 1), việc thực thi lệnh dành riêng cho supervisor mode sẽ gây ra một ngoại lệ. Các lệnh dành riêng cho supervisor mode là *initd*, *initi*, *eret*, *bret*, *rdctl*, và *wrctl*.
- Địa chỉ lệnh dành riêng chế độ supervisor mode: Khi hệ thống chứa MMU và bộ xử lý đang trong chế độ user mode (status.U = 1), việc cố gắng truy cập vào một vùng địa chỉ lệnh dành riêng cho chế độ supervisor mode sẽ gây ra một ngoại lệ. Ngoại lệ này được hiện thực chỉ trong bộ xử lý Nios II có MMU.
- Địa chỉ dữ liệu dành riêng chế độ supervisor mode: Khi hệ thống có chứa MMU và bộ xử lý đang trong chế độ user mode (status.U = 1),

việc cố gắng truy cập vào một vùng địa chỉ dữ liệu dành riêng cho supervisor mode sẽ gây ra một ngoại lệ. Các lệnh có thể gây ra ngoại lệ dạng này tất cả các lệnh dạng load, dạng store và *flushda*. Ngoại lệ này được hiện thực chỉ trong bộ xử lý Nios II có chứa MMU.

Ngoài ra còn các ngoại lệ khác như:

- Địa chỉ dữ liệu sai vị trí
- Địa chỉ đích sai vị trí
- Lỗi chia
- Vi phạm vùng MPU
- ...

Phần trên diễn tả các loại ngoại lệ được định nghĩa bởi kiến trúc Nios II. Tuy nhiên, có thể sẽ có một số ngoại lệ khác không có trong danh sách trên. Vì thế bộ xử lý ngoại lệ phải cung cấp một cơ chế xử lý an toàn (như phát ra một cảnh báo) trong trường hợp nó không thể xác định được nguyên nhân gây ngoại lệ.

6.1.3.5 Truy xuất bộ nhớ và ngoại vi

Địa chỉ Nios II là 32 bit, cho phép truy xuất lên đến 4 Gb không gian địa chỉ. Hệ thống xử lý core Nios II không có MMU giới hạn địa chỉ đến 31 bit hoặc thấp hơn, có MMU hỗ trợ đầy đủ 32 bit địa chỉ.

Thiết bị ngoại vi, bộ nhớ dữ liệu và bộ nhớ chương trình được đưa vào trong cùng một không gian địa chỉ. Địa chỉ được cấp cho bộ nhớ và thiết bị ngoại vi được xác định tại thời điểm phát sinh hệ thống. Đọc hoặc viết tới một địa chỉ mà không phải là vùng được cấp phát cho bộ nhớ hoặc thiết bị ngoại vi sẽ đưa ra một kết quả không xác định.

Bus dữ liệu của bộ xử lý rộng 32 bits. Các lệnh có thể dùng đọc/ghi dữ liệu theo byte, nửa từ (16-bit), hoặc một từ (32-bit).

Kiến trúc Nios II là little endian.

Kiến trúc Nios II hỗ trợ địa chỉ theo dạng thanh ghi+giá trị tức thời.

Nios II chúa vùng cache lệnh và cache dữ liệu. Việc quản lý cache được hiện thực trong phần mềm bằng cách sử dụng những lệnh dành riêng cho cache như lệnh yêu cầu khởi tạo cache, làm sạch cache khi cần thiết hay bỏ qua không cache, ...

Nios II cung cấp một vài cách để bỏ qua cache như sau :

- Khi không có MMU, bit thứ 31 của địa chỉ được dùng để điều khiển việc bỏ cache.
- Khi MMU hiện hành, cache được điều khiển bởi MMU.
- Một vài lệnh bỏ qua cache như *ldwio* và *stwio*.

6.1.3.6 Tập lệnh Nios II

Phần này giới thiệu các lệnh của Nios II, được phân nhóm theo từng dạng hoạt động của chúng

6.1.3.6.1 Nhóm lệnh chuyển dữ liệu

Kiến trúc Nios II là kiến trúc load-store. Lệnh load (lấy dữ liệu) và store (lưu trữ dữ liệu) nắm giữ tất cả các hoạt động di chuyển dữ liệu giữa các thanh ghi, bộ nhớ và các thiết bị ngoại vi. Bảng 6.7 liệt kê các lệnh dạng load và store (32 bit)

Bảng 6.7 Nhóm lệnh chuyển dữ liệu

Lệnh	Mô tả
ldw stw	Lệnh ldw/stw dùng để lấy/lưu trữ dữ liệu 32-bit (một từ hay một word) từ/tới bộ nhớ. Địa chỉ cần lấy hay cần lưu trữ dữ liệu là tổng của giá trị chưa trong thanh ghi và một giá trị tức thời chưa trong lệnh. Việc chuyển dữ liệu bộ nhớ có thể đệm (cached hoặc buffered) để tăng tốc độ truy xuất. Dữ liệu chuyển cho thiết bị ngoại vi I/O phải sử dụng ldwio và stwio
ldwio stwio	Lệnh ldwio/stwio dùng để lấy/lưu trữ dữ liệu 32-bit từ/tới thiết bị ngoại vi không đệm.
ldb ldbu stb ldh ldhu sth	ldb, ldbu, ldh và ldhu lấy một byte hoặc nửa-word từ bộ nhớ đến thanh ghi. Ldb và ldh mở rộng có dấu dữ liệu lấy được lên 32-bits, ldbu và ldbhu mở rộng không dấu dữ liệu lấy được lên 32-bits stb và sth lần lượt là lưu trữ giá trị theo byte và nửa-word Có thể truy xuất bộ nhớ có thể đệm (cached hoặc buffered) để tăng tốc thực thi. Chuyển dữ liệu tới/từ ngoại vi, sử dụng lệnh có “io” ở cuối.
ldbio ldbuio stbio ldhio ldhuio sthio	Thực hiện việc lấy/lưu trữ byte hay nửa-word dữ liệu từ/tới thiết bị ngoại vi, không có đệm.

6.1.3.6.2 Nhóm lệnh số học và logic

Nhóm lệnh logic hỗ trợ cho các hoạt động: and, or, xor, và nor. Nhóm lệnh số học hỗ trợ thao tác: cộng, trừ, nhân, chia. Tham khảo

Bảng 6.8.

Bảng 6.8 Lệnh logic và số học

Lệnh	Mô tả
and or xor nor	Đây là những phép toán logic 32-bit chuẩn. Những phép toán này lấy giá trị từ hai thanh ghi và kết hợp từng bit trên chúng để sinh ra một kết quả vào thanh ghi thứ ba.
andi ori xori	Phép toán là phiên bản tức thời của lệnh and, or và xor. Giá trị 16-bit của số tức thời được mở rộng không dấu thành 32-bit, và sau đó kết hợp với giá trị trong thanh ghi để tạo ra kết quả. Trong những phiên bản này của and, or và xor, giá trị 16-bit của số tức thời được dịch trái 16 bit tạo thành một toán hạng 32-bit
andhi orhi xorhi	Trong những phiên bản này của and, or và xor, giá trị 16-bit của số tức thời được dịch trái 16 bit tạo thành một toán hạng 32-bit. Andhi orhi xorhi
add sub mul div divu	Đây là những phép toán số học 32-bit chuẩn. Các phép toán này lấy giá trị đầu vào từ hai thanh ghi kết quả tính toán được lưu lại trong thanh ghi thứ ba. Add sub mul div divu
addi subi muli	Đây là 1 phiên bản tức thời của lệnh add, sub, và mul. Addi subi muli
mulss mulxuu	Những lệnh này có thể truy xuất trên 32-bit của phép toán nhân 32x32
mulxsu	Lệnh này được dùng trong tính toán kết quả 128-bit của phép nhân 64x64 có dấu. mulxsu

6.1.3.6.3 Nhóm các lệnh di chuyển dữ liệu

Nhóm lệnh này thực hiện các thao tác chép giá trị của một thanh ghi hoặc một số tức thời tới một thanh ghi khác. Tham khảo Bảng 6.9

Bảng 6.9 Các lệnh di chuyển

Lệnh	Mô tả
mov	mov chép giá trị của 1 thanh ghi đến thanh ghi khác.
movhi	movi đưa một giá trị tức thời 16-bit có dấu đến thanh ghi và mở rộng có dấu giá trị đó thành 32-bit.
movi	movui và movhi lần lượt đưa một giá trị tức thời 16-bit vào nửa phần thấp hoặc nửa phần cao của thanh ghi. Mỗi nửa phần trong thanh ghi là 16-bit, số 0 được chèn vào những bit còn lại.
movui	movia dùng để đưa một giá trị địa chỉ vào thanh ghi.
movia	

6.1.3.6.4 Nhóm lệnh so sánh

Nhóm lệnh này thực hiện việc so sánh hai thanh ghi hoặc thanh ghi với một giá trị tức thời; kết quả là 1(nếu đúng) hoặc 0 (nếu sai) được ghi ra thanh ghi kết quả. Tham khảo Bảng 6.10

Bảng 6.10 Các lệnh so sánh

Lệnh	Mô tả
cmpeq	==
cmpne	!=
cmpge	signed >=
cmpgeu	unsigned >=
cmpgt	signed >
cmpgtu	unsigned >
cmple	unsigned <=
cmpleu	unsigned <=
cmplt	signed <
cmpltu	unsigned <
cmpeqi cmpnei cmpgei cmpgeui cmpgti cmpgtui cmplei cmpleui cmplti cmpltui	Lệnh này là phiên bản tức thời của phép toán so sánh, so sánh giá trị của thanh ghi và một số tức thời 16-bit. Với những phép toán có dấu, giá trị tức thời 16-bit được mở rộng có dấu lên 32-bit. Với những phép toán không dấu, giá trị tức thời 16-bit được mở rộng lên 32-bit bằng cách làm đầy 16 bit cao với số 0

6.1.3.6.5 Nhóm lệnh dịch và xoay

Nhóm lệnh này thực hiện các phép toán dịch và xoay. Số bit để xoay hoặc dịch được chỉ rõ trong một thanh ghi hoặc một số tức thời. Tham khảo Bảng 6.11

Bảng 6.11 Lệnh dịch và xoay

Lệnh	Mô tả
rol ror roli	Lệnh rol và ror thực hiện phép xoay trái. Lệnh roli sử dụng một số tức thời để chỉ số bit xoay. Lệnh rol sử dụng giá trị chứa trong thanh ghi để chỉ số bit xoay. Lệnh ror yêu thực hiện phép xoay phải. Không có phiên bản tức thời của ror, roli có thể được dùng thay thế.
sll slli sra srl srai srli	Lệnh sll, slli, srl, srli thực hiện phép dịch trái và phải logic, thêm 0 vào những vị trí dịch trống. Lệnh sra và srai thực hiện phép dịch phải số học, lặp lại bit đầu trong bit có trọng số lớn nhất. slli, srl và srai sử dụng một số tức thời để chỉ rõ số bit dịch.

6.1.3.6.6 Nhóm lệnh điều khiển chương trình

Kiến trúc Nios II hỗ trợ nhóm lệnh nhảy và gọi hàm không điều kiện được liệt kê trong Bảng 6.12

Bảng 6.12 Nhóm lệnh nhảy và gọi hàm không điều kiện

Lệnh	Mô tả
call	Lệnh này gọi một đoạn chương trình con sử dụng một số tức thời như là một địa chỉ trực tiếp của đoạn chương trình con đó và địa chỉ trả về được lưu trữ trong thanh ghi ra.
callr	Lệnh này gọi một đoạn chương trình con tại một địa chỉ tuyệt đối được chứa trong một thanh ghi lưu địa chỉ trả về trong thanh ghi ra.
ret	Lệnh ret được sử dụng để trả về chương trình chính từ đoạn chương trình con được gọi bởi call hoặc callr. ret sẽ lấy và thực thi lệnh ở ngay địa chỉ được chứa trong thanh ghi ra.
jmp	Lệnh jmp nhảy đến một địa chỉ tuyệt đối chứa trong thanh ghi.
jmpi	Lệnh jumi nhảy đến một địa chỉ tuyệt đối theo một số tức thời đưa ra.

Kiến trúc Nios II hỗ trợ nhóm lệnh nhảy có điều kiện được liệt kê trong Bảng 6.13. Nhóm lệnh này so sánh các giá trị chứa trong thanh ghi và nhảy nếu kết quả so sánh là đúng

Bảng 6.13 Nhóm lệnh nhảy có điều kiện

Lệnh	Mô tả
bge	
bgeu	
bgt	
bgtu	
ble	
bleu	
blt	
bltu	
beq	
bne	

6.1.3.6.7 Nhóm các lệnh điều khiển khác

Bảng 6.14 Nhóm lệnh điều khiển khác

Lệnh	Mô tả
trap eret	Lệnh trap và eret tạo ra và trả về từ những ngoại lệ. Cặp lệnh này tương tự với cặp call/ret, nhưng được sử dụng cho exception. <ul style="list-style-type: none"> - trap: lưu thanh ghi status vào thanh ghi estatus, lưu địa chỉ trả về vào thanh ghi ea, và sau đó chuyển sự thực thi đến một bộ xử lý ngoại lệ. - eret: trả về chương trình chính từ tiến trình xử lý ngoại lệ bằng cách khôi phục lại status từ estatus và thực thi lệnh tại địa chỉ trong ea.
break bret	Lệnh break và bret tạo ra và trả về từ những break. break và bret được sử dụng dành riêng cho những công cụ debug phần mềm. Lập trình viên không bao giờ sử dụng những lệnh này trong code ứng dụng.
rdctl wrctl	Những lệnh này đọc và ghi thanh ghi điều khiển. Giá trị được đọc từ hoặc lưu trữ tới một thanh ghi đa dụng.
flushd flushda flushi initd	Lệnh này được sử dụng để quản lý vùng nhớ đệm dữ liệu và lệnh.
flushp	Lệnh này xóa tất cả các lệnh nạp trước trong pipeline.
sync	Lệnh này chắc chắn rằng tất cả các hoạt động được cập nhật trước đây đã hoàn thành trước khi thực thi những hoạt động load và store sau.

6.1.3.7 Các kiểu dữ liệu

Các kiểu dữ liệu của Nios II trình bày trong

Bảng 6.15

Bảng 6.15 Các kiểu dữ liệu Nios II

Kiểu	Kích thước (Bytes)	Biểu diễn
Char, signed char	1	Bù hai (ASCII)
Unsigned char	1	Nhi phân (ASCII)
Short, signed short	2	Bù hai
Unsigned short	2	Nhi phân
Int, signed int	4	Bù hai
Unsigned int	4	Nhi phân
Long, signed long	4	Bù hai
Unsigned long	4	Nhi phân
Float	4	IEEE
Double	8	IEEE
Pointer	4	Nhi phân
Long long	8	Bù hai
Unsigned long long	8	Nhi phân

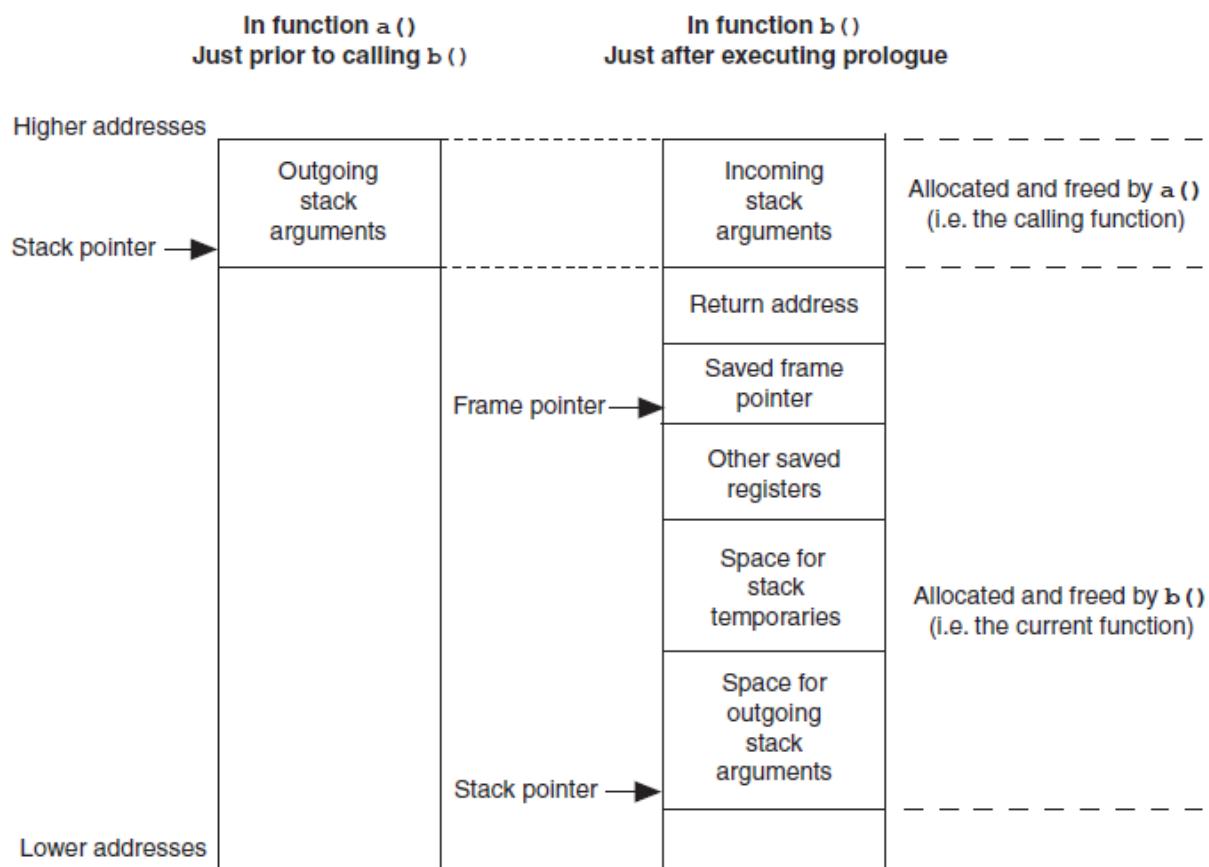
6.1.3.8 Stacks trong Nios II

Kiến trúc stack trong Nios II phát triển theo hướng xuống, hai con trỏ cần quan tâm khi làm việc với stack là con trỏ stack (stack pointer) và con trỏ khung (frame pointer). Stack pointer chỉ tới khe (slot) được dùng cuối cùng. Frame pointer trỏ đến frame pointer được lưu gần đỉnh của stack.

Frame pointer được cung cấp để hỗ trợ cho công việc debug. Nếu không cần debug, frame pointer có thể bỏ ra khỏi code. Dùng tùy chọn `-fomit-frame-pointer` cho trình biên dịch, thanh ghi fp sẽ được xem như thanh ghi tạm.

Trình biên dịch chịu trách nhiệm lưu trữ những thanh ghi mà cần được lưu trong một hàm. Nếu có những thanh ghi như vậy, chúng sẽ được lưu trong stack, từ

địa chỉ cao đến thấp theo thứ tự sau: ra, fp, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, gp, và sp. Không gian stack không được cấp cho các thanh ghi mà không được lưu. Hình 6.4 trình bày một ví dụ hàm *a()* gọi hàm *b()*, và stack được thể hiện trước lời gọi và sau dẫn nhập của cấu trúc khung hiện tại. Trong trường hợp này, hàm *a()* gọi hàm *b()*, và stack được thể hiện trước và ngay sau lời gọi hàm.



Hình 6.4 Cấu trúc của Stack Pointer

6.1.3.9 Chi tiết từ lệnh Nios II

Có 3 loại dạng từ lệnh trong Nios II: loại I, loại R, loại J.

6.1.3.9.1 Từ lệnh loại I

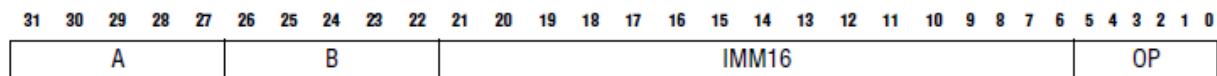
Đặc tính của dạng từ lệnh loại I là giá trị tức thời được nhúng trong từ lệnh.

Từ lệnh dạng I chứa các trường:

- Trường OP: 6-bit opcode lệnh
- Trường A và B: 5-bit chứa chỉ số thanh ghi A và B
- Trường IMM16: 16-bit dữ liệu tức thời

Trong hầu hết trường hợp, trường A và IMM16 là đầu vào của lệnh, và trường B là thanh ghi chứa kết quả. Lệnh loại I bao gồm những phép toán số học và logic như addi và andi; phép toán rẽ nhánh; các hoạt động nạp và lưu và lệnh quản lý cache.

Định dạng lệnh loại I như sau:



Hình 6.5 Định dạng từ lệnh loại I

6.1.3.9.2 Từ lệnh loại R

Đặc tính của dạng từ lệnh loại R là tất cả những đôi số và kết quả là thanh ghi. Loại lệnh R chứa các trường:

- Trường OP: 6-bit opcode
- Trường A, B, C: 5-bit chỉ số của các thanh ghi A, B, C
- Trường OPX: 11-bit opcode mở rộng

Trong hầu hết trường hợp, trường A và B là các đầu vào của lệnh, và trường C là thanh ghi kết quả. Một vài lệnh loại R đưa vào một giá trị tức thời trong trường OPX.

Lệnh loại R bao gồm các phép toán số học và logic như add và nor; phép toán so sánh như cmpeq và cmplt; lệnh tùy chỉnh; và những phép toán khác mà chỉ thao tác trên thanh ghi.

Định dạng lệnh loại R như sau:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A		B		C																											OP

Hình 6.6 Định dạng từ lệnh loại R

6.1.3.9.3 Từ lệnh loại J

Lệnh loại J chứa :

- ─ Trường OP: 6-bit opcode
- ─ Trường IMMED26: 26-bit dữ liệu tức thời

Lệnh loại J chỉ thao tác trên số tức thời, như như call và jmp, chuyển quyền thực thi đến bất cứ đâu trong phạm vi 256 MByte

Định dạng lệnh loại J như sau:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IMMED26																													OP		

Hình 6.7 Định dạng từ lệnh loại J

Bảng 6.16 Mã hóa trường OP của các từ lệnh

O	Lệnh	O	Lệnh	O	Lệnh	O	Lệnh
0x00	call	0x10	cmplti	0x20	cmpeqi	0x3	cmpltui
0x01	jmpi	0x11		0x21		0x3	
0x02		0x12		0x22		0x3	custom
0x03	ldbu	0x13	initda	0x23	ldbuio	0x3	initd
0x04	addi	0x14	ori	0x24	muli	0x3	orhi
0x05	stb	0x15	stw	0x25	stbio	0x3	stwio
0x06	br	0x16	blt	0x26	beq	0x3	bltu
0x07	ldb	0x17	ldw	0x27	ldbio	0x3	ldwio
0x08	cmpgei	0x18	cmpnei	0x28	cmpgeui	0x3	
0x09		0x19		0x29		0x3	
0x0A		0x1A		0x2A		0x3	R-type
OP	Lệnh	OP	Lệnh	OP	Lệnh	OP	Lệnh
0x0B	ldhu	0x1B	flushda	0x2B	ldhuio	0x3	flushhd
0x0C	andi	0x1C	xori	0x2C	andhi	0x3	xorhi
0x0D	sth	0x1D		0x2D	sthio	0x3	
0x0E	bge	0x1E	bne	0x2E	bgeu	0x3	
0x0F	ldh	0x1F		0x2F	ldhio	0x3	

Bảng 6.17 Mã hóa trường OPX của các từ lệnh loại R

OP	Lệnh	OPX	Lệnh	OPX	Lệnh	0	Lệnh
0x00		0x10	cmplt	0x20	cmpeq	0x30	cmpltu
0x01	eret	0x11		0x21		0x31	add
0x02	roli	0x12	slli	0x22		0x32	
0x03	rol	0x13	sll	0x23		0x33	
0x04	flushp	0x14		0x24	divu	0x34	break
0x05	ret	0x15		0x25	div	0x35	
0x06	nor	0x16	or	0x26	rdctl	0x36	sync
0x07	mulxuu	0x17	mulxsu	0x27	mul	0x37	
0x08	cmpge	0x18	cmpne	0x28	cmpgeu	0x38	
0x09	bret	0x19		0x29	initi	0x39	sub
0x0		0x1A	srl	0x2		0x3A	srai
0x0	ror	0x1B	srl	0x2		0x3B	sra
0x0	flushi	0x1C	nextpc	0x2		0x3C	
0x0	jmp	0x1D	callr	0x2	trap	0x3D	
0x0E	and	0x1E	xor	0x2E	wrctl	0x3E	
0x0F		0x1F	mulxss	0x2F		0x3F	

6.1.3.10 Lệnh giả

Những lệnh giả được sử dụng trong code như những lệnh assembly thông thường. Mỗi lệnh giả có thể có tên khác, nhưng thực chất là sử dụng một lệnh assembly đương lượng ngoại trừ lệnh movie. Movie được hiện thực với hai lệnh tương đương. Lệnh giả chỉ có giá trị cho lập trình, trong mã máy, nó chỉ được thể hiện bằng các lệnh tương đương thực chất của nó.

Bảng 6.18 Danh sách các lệnh và lệnh tương đương

bgtu rA, rB, label	bltu rB, rA, label
ble rA, rB, label	bge rB, rA, label
bleu rA, rB, label	bgeu rB, rA, label
cmpgt rC, rA, rB	cmplt rC, rB, rA
cmpgti rB, rA, IMMED	cmpgei rB, rA, (IMMED+1)
cmpgtu rC, rA, rB	cmpltu rC, rB, rA
cmpgtui rB, rA, IMMED	cmpgeui rB, rA, (IMMED+1)
cmple rC, rA, rB	cmpge rC, rB, rA
cmplei rB, rA, IMMED	cmplti rB, rA, (IMMED+1)
cmpleu rC, rA, rB	cmpgeu rC, rB, rA
cmpleui rB, rA, IMMED	cmpltui rB, rA, (IMMED+1)
mov rC, rA	add rC, rA, r0
movhi rB, IMMED	orhi rB, r0, IMMED
movi rB, IMMED	addi, rB, r0, IMMED
movia rB, label	orhi rB, r0, %hiadj(label) addi, rB, r0, %lo(label)
movui rB, IMMED	ori rB, r0, IMMED
nop	add r0, r0, r0
subi rB, rA, IMMED	addi rB, rA, (-IMMED)

6.1.3.11 Các Macros

Assembler của Nios II cung cấp các macro để rút trích nửa-word từ những nhãn và từ những giá trị tức thời 32-bit. Những macro trả về giá trị có dấu 16-bit hoặc giá trị không dấu 16-bit tùy thuộc vào nơi nào chúng được sử dụng. Khi sử

dụng với một lệnh đòi hỏi một giá trị tức thời có dấu 16-bit, những macro sẽ trả về một giá trị từ -32768 đến 32767. Khi sử dụng với một lệnh cần giá trị tức thời không dấu 16-bit, những macro này sẽ trả về một giá trị từ 0 đến 65535.

Bảng 6.19 Danh sách các macro hiện hành

Macro	Mô	Phép
%lo(immed32)	Rút trích giá trị từ bits 15 tới bit 0 của số tức thời 32 bit	immed32 & 0xffff
%hi(immed32)	Rút trích giá trị từ bits 16 tới 31 của số tức thời 32 bit	(immed32 >> 16) & 0xffff
%hiadj(immed32)	Rút trích giá trị từ bits 15 tới 31 của số tức thời 32 bit và cộng với bit 15 của số tức thời	((immed32 >> 16) & 0xffff) + ((immed32 >> 15) & 0x1)
%gprel(immed32)	Thay thế địa chỉ immed32 với một offset từ global pointer.	immed32 - _gp

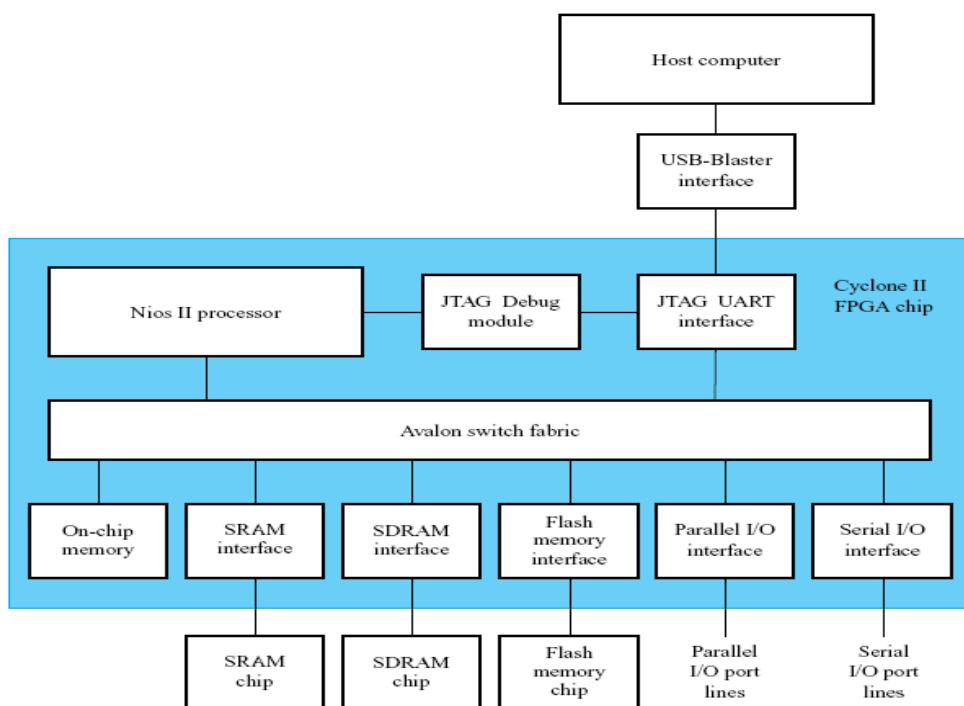
6.2 Hướng dẫn thực hành trên vi xử lí Nios II

6.2.1 Nios II System :

Nios II system là một users system được thiết kế trên Cyclone II FPGA bằng một công cụ trên Quartus II, đó là SOPC Builder. Tuy nhiên về mặt lý thuyết thì users có thể tự tạo một system trên FPGA nhưng điều đó sẽ tốn thời gian và công sức.

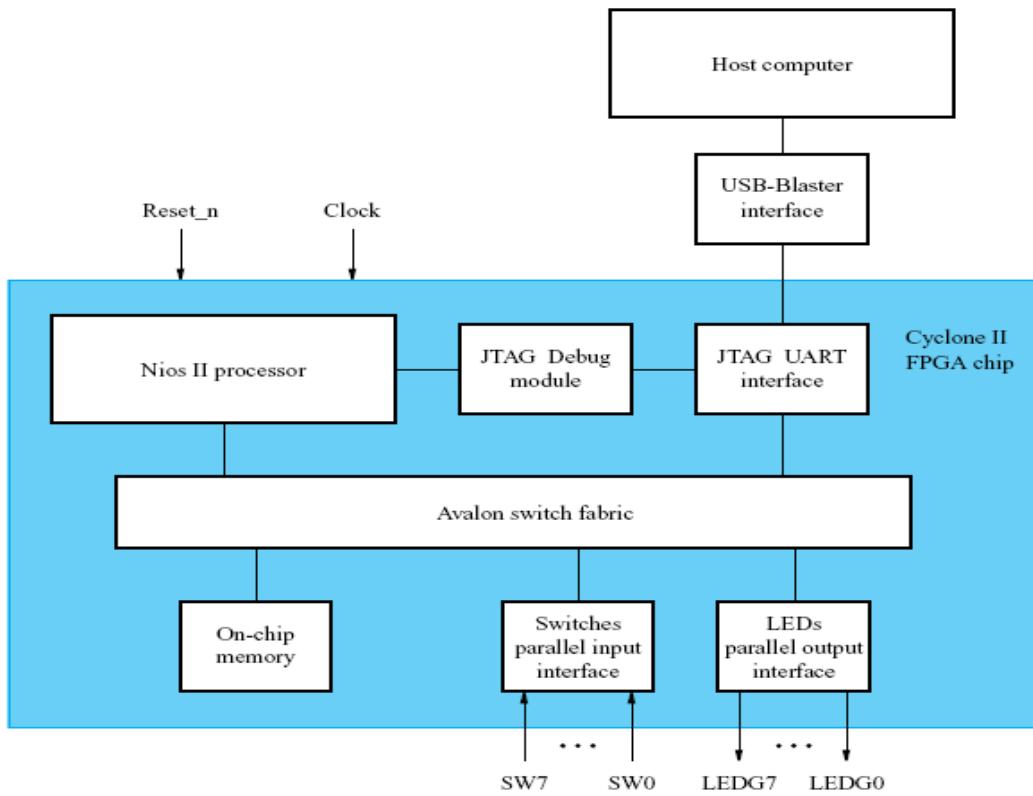
Nios II system với Nios II processor đóng vai trò trung tâm. Về mặt ý nghĩa nào đó thì Nios II processor giống như Chip vi điều khiển 8951 mà chúng ta đã được học trong môn Vi xử lí.

Dưới đây là một thí dụ về một Nios II system.



Hình 6.8 Nios system

Dưới đây là một Nios II system đơn giản hơn, chúng ta cũng sử dụng Nios II system này làm thí dụ minh họa cho toàn bộ hướng dẫn này.



Hình 6.9 Nios system đơn giản

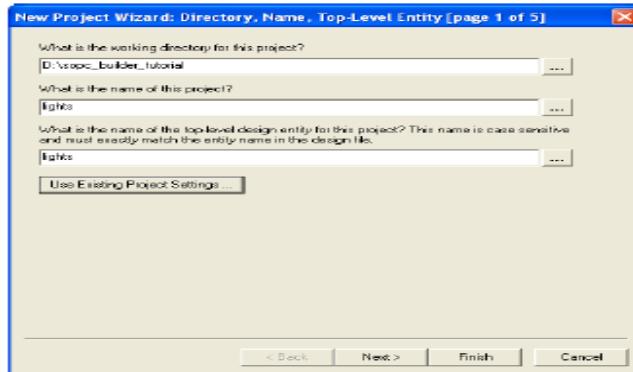
Những khối trong vùng màu xanh liên kết với nhau tạo nên một system.

Ta sẽ thiết kế từng module trên chip FPGA. Phương pháp thông thường là ta sẽ thiết kế từng module dùng Verilog hay VHDL hoặc schematic, rồi liên kết chúng lại thành một system. Nhưng như ta đã trình bày, điều này sẽ tốn rất nhiều thời gian và công sức, đặc biệt đối với những người mới học. Tuy nhiên Quartus II có support cho chúng ta một công cụ với những modules đã được build sẵn. Đó là SOPC Builder. Nhiệm vụ của chúng ta đơn giản là chọn ra những module nào cần thiết cho thiết kế của mình để liên kết chúng lại.

6.2.2 Mở một project mới

Tương tự như mở một project bình thường (đã được trình bày trong phần các phần mềm chúng ta sẽ không trình bày lại). Giả sử ta chọn đường dẫn thư mục chứa

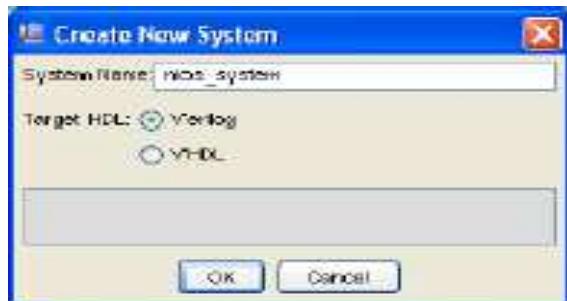
project là : D:\sopc_builder_tutorial ; tên project : lights; tên top-level của project : lights (nhớ khi chúng ta tạo verilog cho top-level thì bắt buộc phải trùng tên với tên đã chọn ở đây).



Hình 6.10 Tạo project

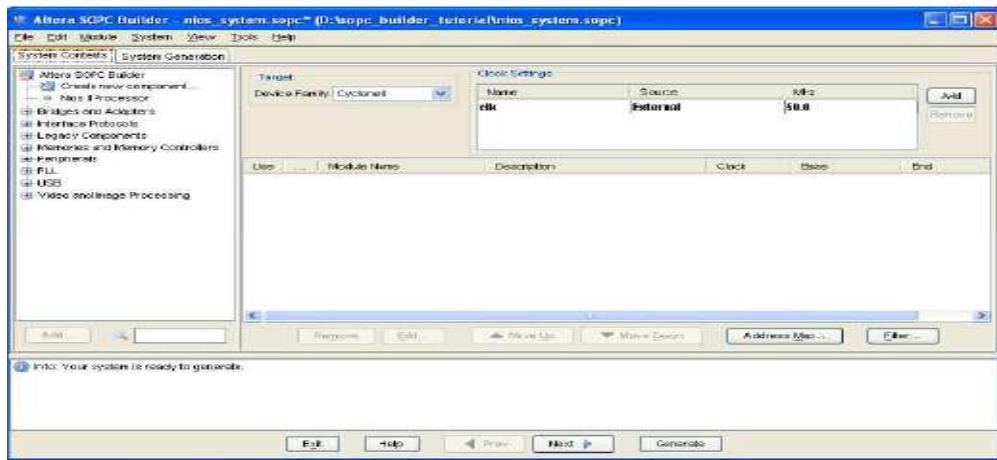
Sau khi đã tạo xong project, ta tiếp tục thực hiện các bước sau

- Bước 1. Chọn **Tool → SOPC Builder**
- Bước 2. Nhập tên system chúng ta muốn gọi , giả sử nios_system.
- Bước 3. Chọn **Verilog** (ta sẽ tìm hiểu thiết kế bằng Verilog trước , VHDL sẽ tương tự).



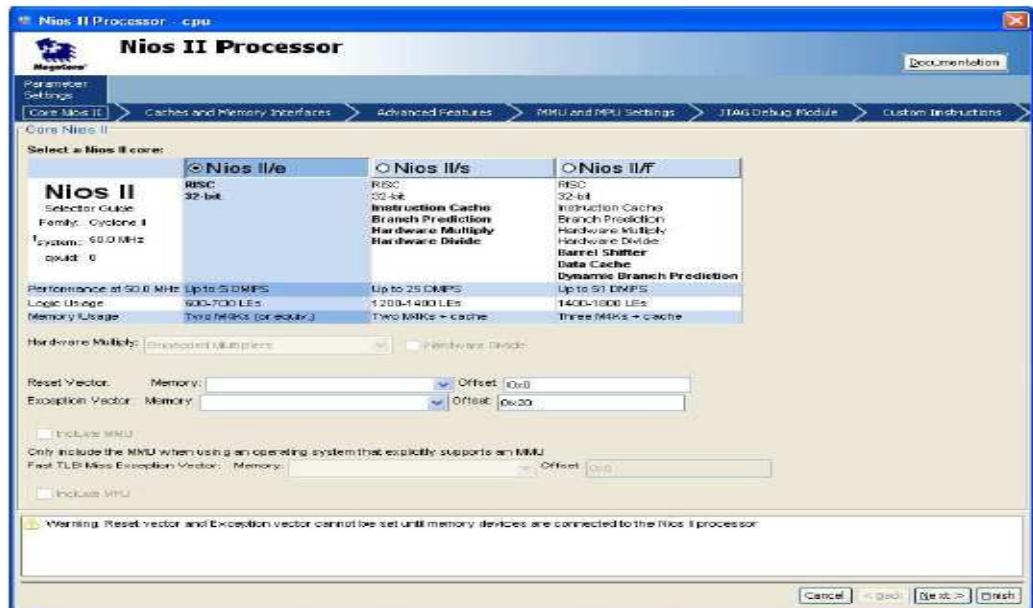
Hình 6.11 Đặt tên cho Nios system

- Bước 4. Nhấn **OK**



Hình 6.12 Cửa sổ SOPC

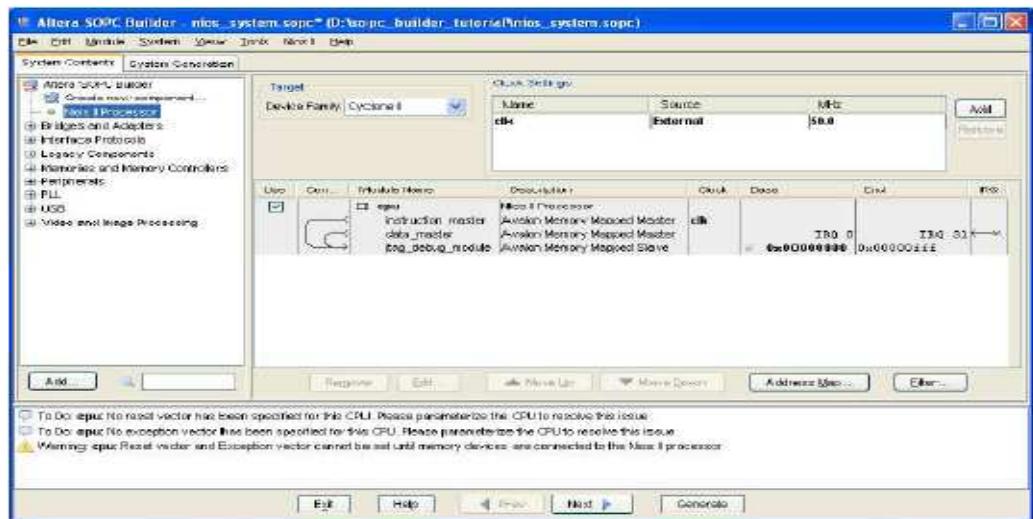
- Bước 5. Chọn Device Family : Cyclone II
- Bước 6. Trên Tab System Contens, chọn Nios II processor.
- Bước 7. Nhấn Add



Hình 6.13 Chọn processor

Bước 8. Chọn Nios II/e (simplest and economical).

Bước 9. Nhấn Finish



Hình 6.14 Quay về SOPC Builder

Bước 10. Tạo một memory on Chip : Chọn **Memories and Memory Controllers** → **On- Chip** → **On-Chip Memory (RAM or ROM)**

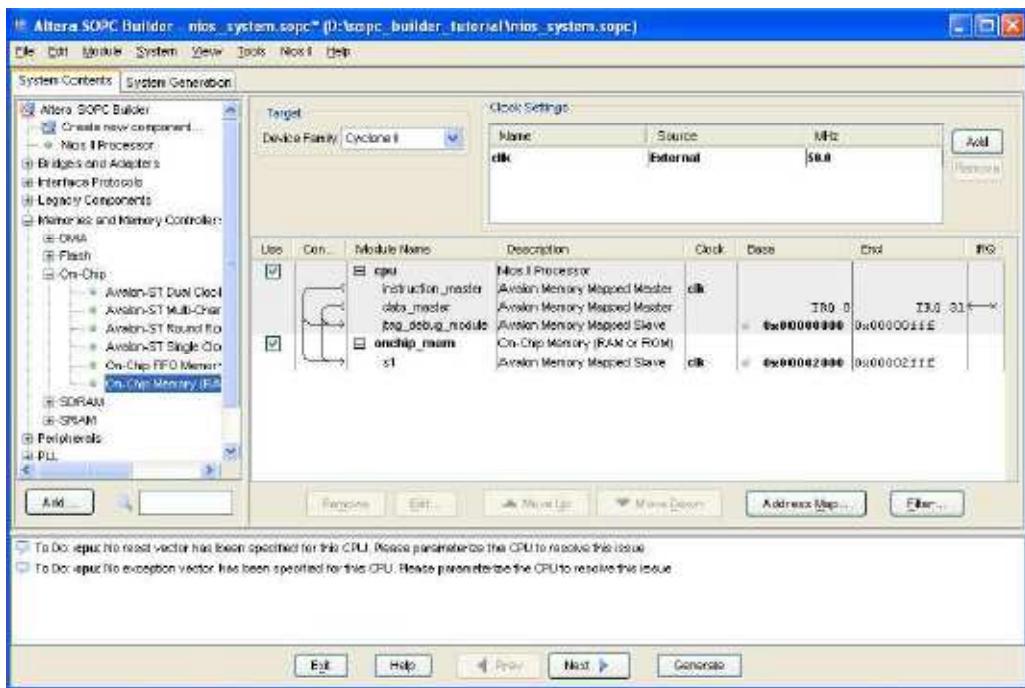
Bước 11. Nhấn **Add**



Hình 6.15 Chọn On-Chip memory

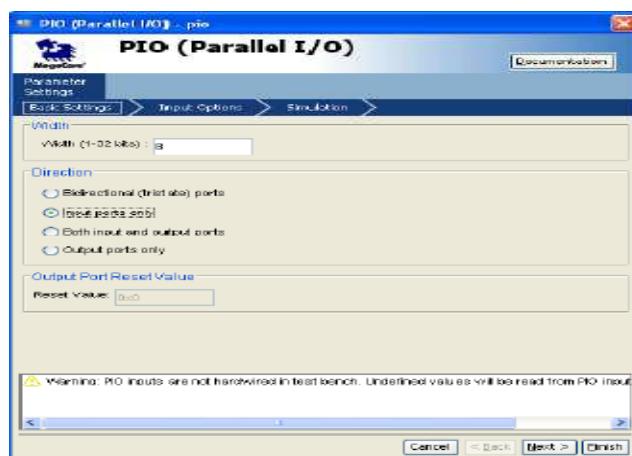
Bước 12. Chọn **Memory Width 32 bits, Memory size 4 Kbytes.**

Bước 13. Nhấn **Finish**



Hình 6.16 Quay về SOPC Builder

Bước 14. Tạo Input Parallel I/O Interface : Peripherals → Microcontroller Peripherals → PIO (Parallel I/O) ; Nhấn Add

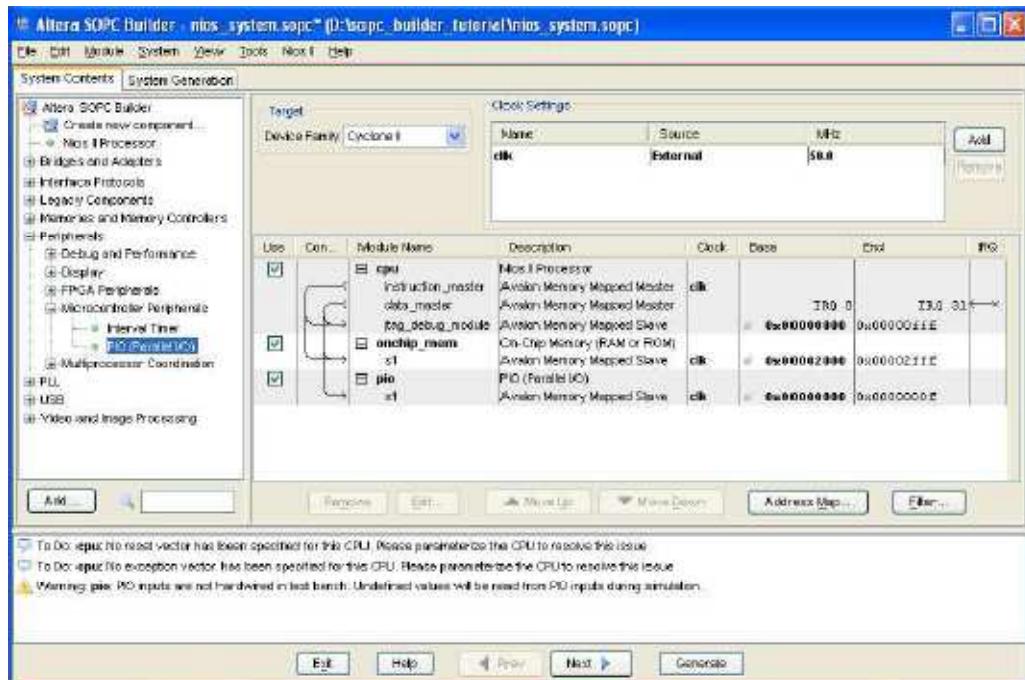


Hình 6.17 Chọn PIO

Bước 15. Chọn Width of Port : 8 bits

Bước 16. Direction : Input ports only

Bước 17. Nhấn Finish



Hình 6.18 Quay về SOPC Builder

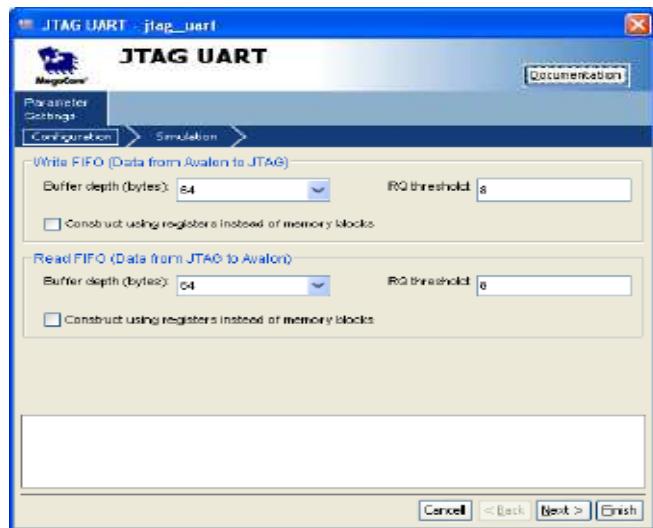
Bước 18. Tương tự tạo Output Parallel I/O Interface : Peripherals → Microcontroller Peripherals → PIO (Parallel I/O) ; Nhấn Add

Bước 19. Chọn Width of Port : 8 bits

Bước 20. Direction : Output ports only

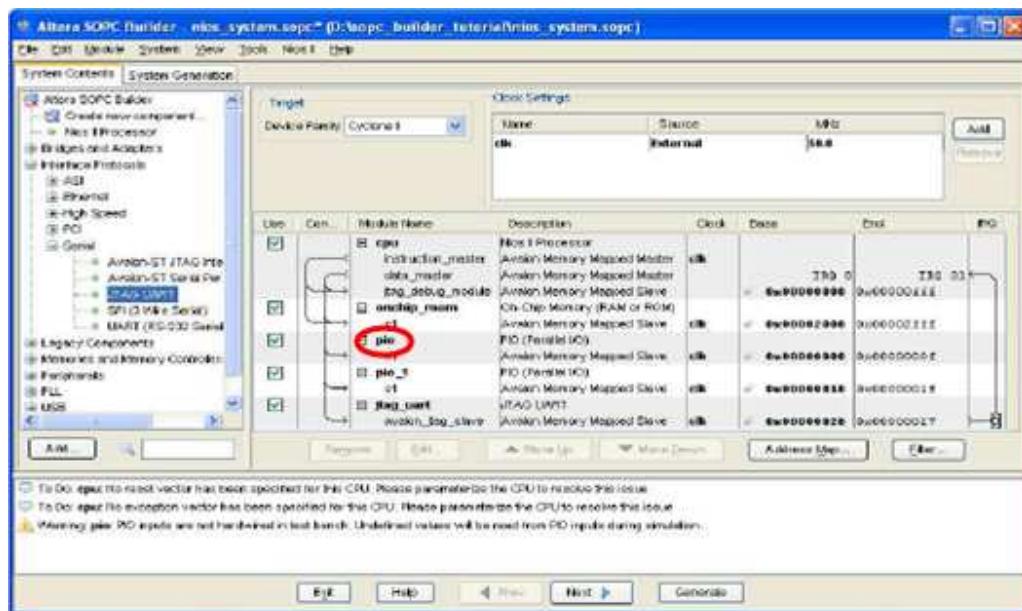
Bước 21. Nhấn Finish

Bước 22. Ta cần phải tạo một JTAG UART Interface để tạo giao tiếp giữa host computer với Nios II system : Chọn Interface Protocols → Serial → JTAG UART ; Nhấn Add



Hình 6.19 Chọn JTAG UART

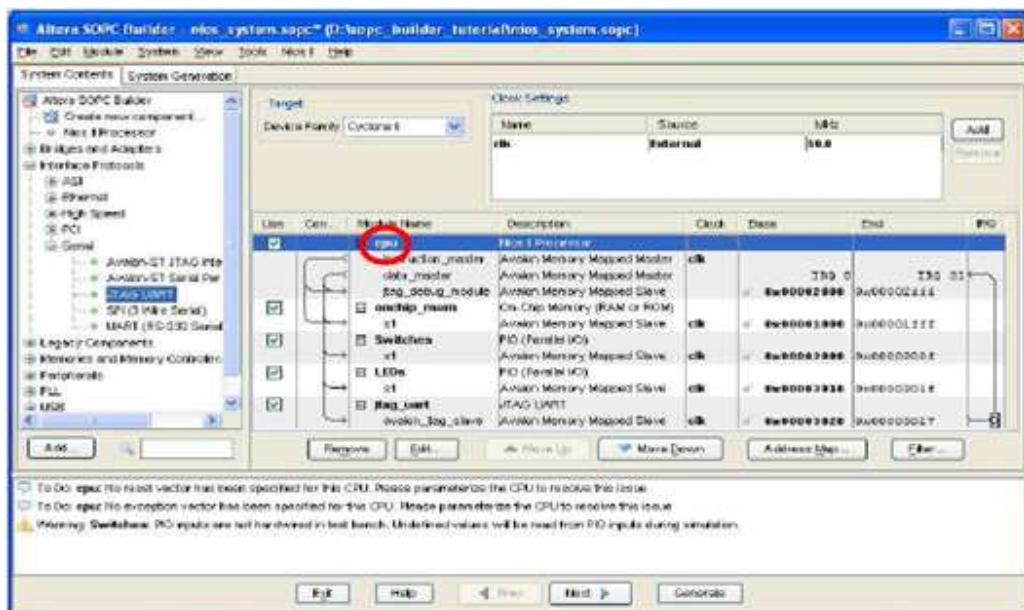
Bước 23. Nhấn Finish.



Hình 6.20 Quay về SOPC Builder

Bước 24. Ta có thể thay đổi tên của Module bằng cách nhấp chuột phải vào Default Module Name và thay đổi tên chúng.

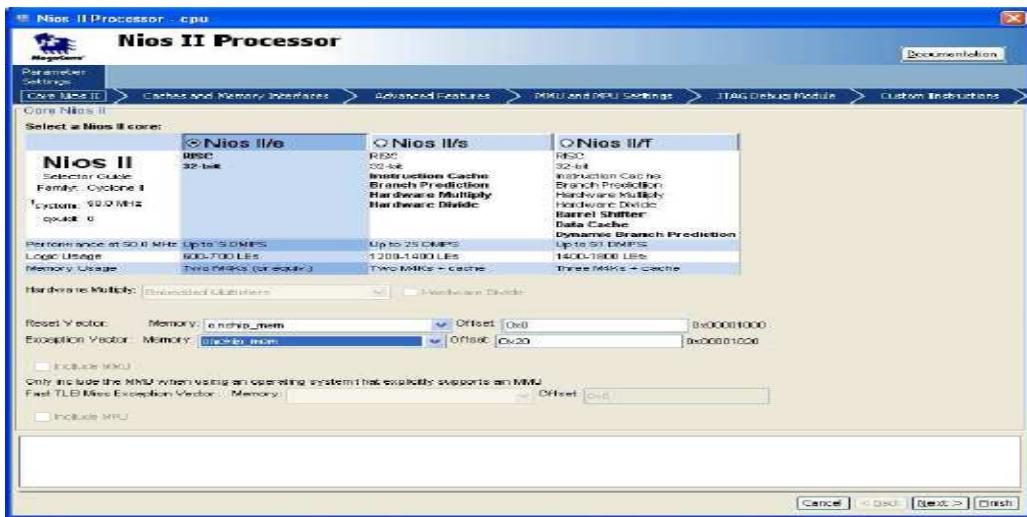
Bước 25. Chọn System → Auto-Assign Base Addresses



Hình 6.21 Thay đổi đặc tính cho Processor

Một processor sẽ có tín hiệu Reset. Việc reset processor được thực hiện bởi vector reset, vector reset là địa chỉ của memory mà processor sẽ tìm đến để thực hiện lệnh kế tiếp khi reset xảy ra. Tương tự cho việc interrupt cũng sẽ có vector exception, vector exception là địa chỉ của memory mà processor sẽ nhảy tới khi một interrupt xảy ra. Hai vectors này được khai báo như sau :

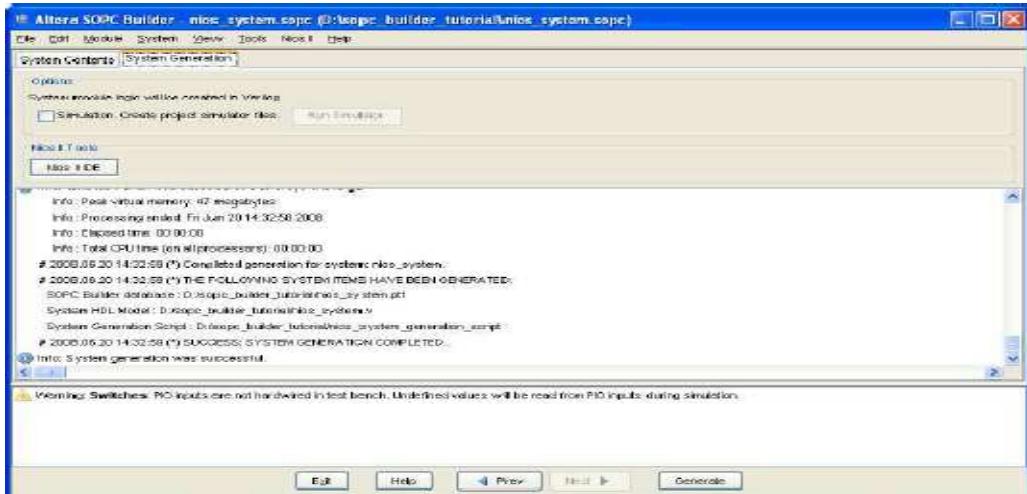
Bước 26. Nhấn chuột phải lên Module Name **cpu**, chọn **Edit**.



Hình 6.22 Thay đổi Exception Vector

Bước 27. Chọn onchip_mem cho cả 2 vector : **Reset Vector** và **Exception Vector**. Nhấn **Finish**

Bước 28. Sau khi đã chọn và khai báo đầy đủ những modules cần thiết cho system của chúng ta, chúng ta có thể tạo system : Chọn **System Generation Tab**



Hình 6.23 Tạo verilog files cho nios system

- Bước 29. **Turn-off Simulation – Create simulator project files**
- Bước 30. Nhấn **Generate**
- Bước 31. Khi một message “**Success : System Generation Completed**”.
- Nhấn **Exit**. Nios II system của chúng ta đã được tạo.
- Bước 32. Sau khi Nios II system đã tạo xong, chúng ta mở thư mục chứa project mà lúc đầu chúng ta đã khai báo, chúng ta sẽ thấy một số file.v đã được tạo. Đó chính là những module mà ta đã sử dụng SOPC Builder để tạo ra với top-level module là nios_system.v và một file nios_system.ptf (dùng cho quá trình download chương trình assembler lên system trên FPGA).
- Bước 33. Bước tiếp theo là chúng ta tích hợp nó trên FPGA dùng QuartusII (tương tự như đã hướng dẫn trong phần thực hành môn học Verilog).
- Bước 34. Tạo một file lights.v chứa module mang tên “lights” (tên top-module này phải giống với tên top-module mà ta đã khai báo trong lúc tạo project).

```

module lights (SW, KEY, CLOCK_50, LEDG);
    input [7:0];
    input [0:0] KEY;
    input CLOCK_50;
    output [7:0] LEDG;
    nios_systemNiosII(CLOCK_50, KEY[0], LEDG, SW);
endmodule

```

Hình 6.24 Top-level module

Bước 35. Add file lights.v cũng như những file.v (nios_system) mà đã được tạo bởi SOPC Builder vào project trên Quartus II (phần này đã trình bày trong phần hướng dẫn thực hành môn học Verilog).

Bước 36. Sau đó assign pins cho những signals SW[0:7], KEY, CLOCK_50, LEDG (phần này đã trình bày trong phần hướng dẫn thực hành môn học Verilog).

Bước 37. Compiling design (phần này đã trình bày trong phần hướng dẫn thực hành môn học Verilog). File lights_time_limited.sof được tạo ra.

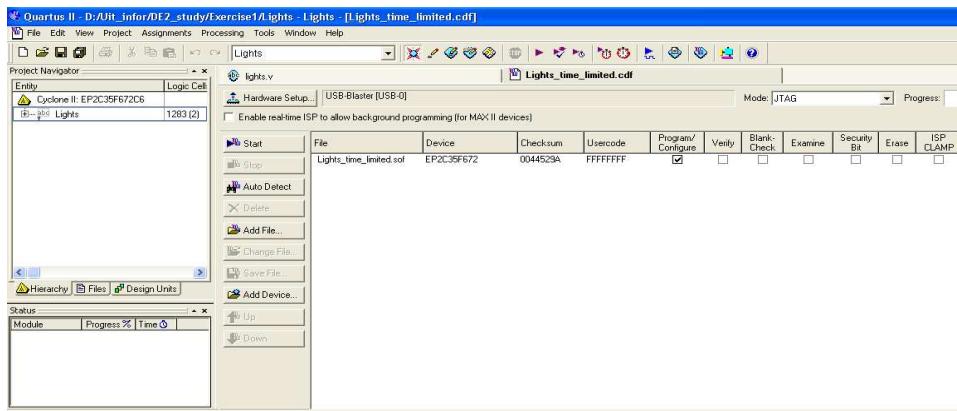
Bước 38. Tiếp theo là programming và configure project lên FPGA. (phần này đã trình bày trong phần hướng dẫn thực hành môn học Verilog) :

Bước 39. **Tools ➔ Programmer**



Hình 6.25 Programmer

Bước 40. Nhấn **OK**



Hình 6.26 Cửa sổ programmer

Bước 41. **Add File**, chỉ đường dẫn đến file lights_time_limited.sof.

Bước 42. **Nhấn Start**



Hình 6.27 Giữ cửa sổ này trong suốt quá trình nạp FPGA

Bước 43. Giữ Status này trong suốt quá trình chạy Altera Monitor Program (trình bày ở dưới).

Như vậy chúng ta đã thiết kế xong một system trên FPGA. System này có một processor là NiosII processor (tương đối giống như vi điều khiển C8951 mà chúng ta đã được học trong môn Vi điều khiển), nhưng ở đây ta làm việc trên Vi xử lí.

Để thiết kế một ứng dụng trên system này ta sẽ sử dụng Assembler language hoặc C language.

Giờ ta sẽ thiết kế một ứng dụng đơn giản , đó là dùng 8 Switches để điều khiển 8 Leds tương ứng. Các bước thực hiện:

Bước 1. Tạo một lights.s dùng ngôn ngữ assembler như sau :

```
.include "nios_macros.s"

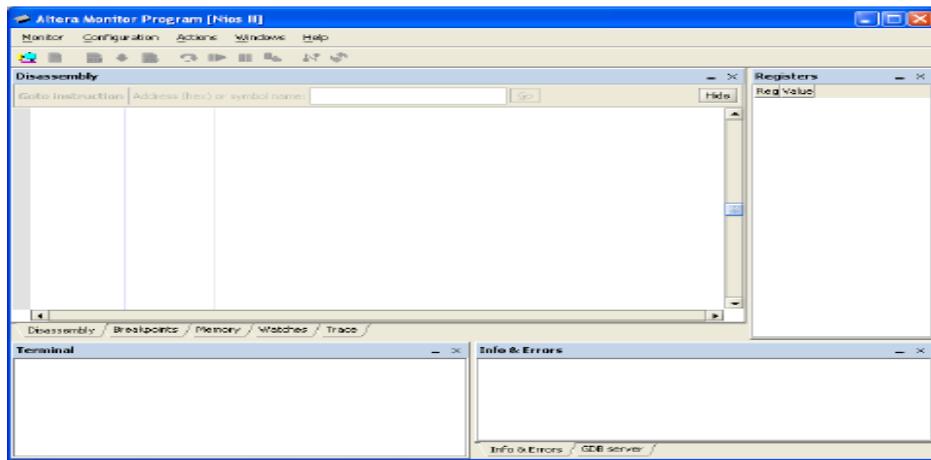
.equ   Switches, 0x00003000
.equ   LEDs, 0x00003010

.global _start
_start:
        movia r2, Switches
        movia r3, LEDs
loop:   ldbio  r4, 0(r2)
        stbio  r4, 0(r3)
        br     loop
```

Hình 6.28 Chương trình assemble đơn giản

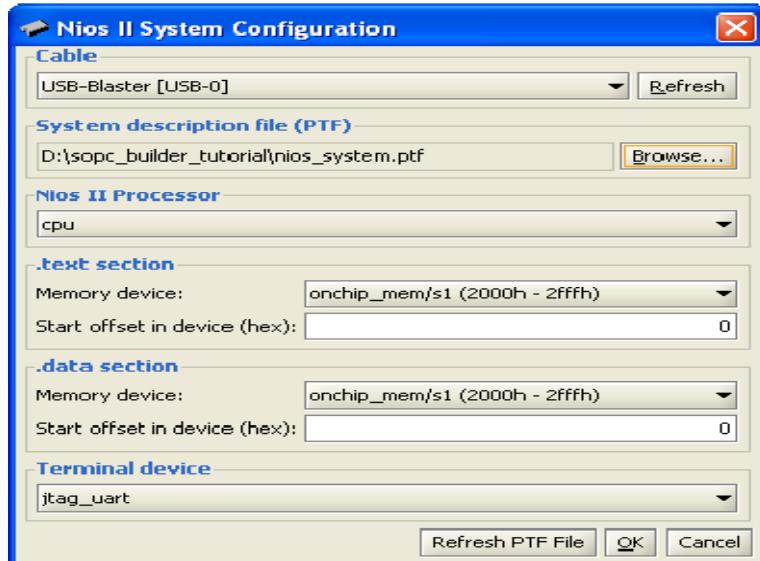
Bước 2. Sử dụng một phần mềm khác của Altera đó là Altera Monitor Program (phải cài đặt trước software này) để compiling, assembling, và downloading chương trình lên Nios system đã được thiết kế trên FPGA trên Kit DE2.

Bước 3. Mở phần mềm Altera Monitor Program :



Hình 6.29 Cửa sổ Altera Monitor Program

Bước 4. Chọn Configuration → Configure System



Hình 6.30 Thiết lập cấu hình

Bước 5. Chọn Cable : USB-Blaster[USB-0]

Bước 6. Chỉ đường dẫn của file nios_system.ptf (đã được tạo khi generate nios system bởi SOPC builder).

Bước 7. Nhấn **OK**

Bước 8. Chọn **Configuration -> Configure Program**



Hình 6.31 Chọn file chương trình

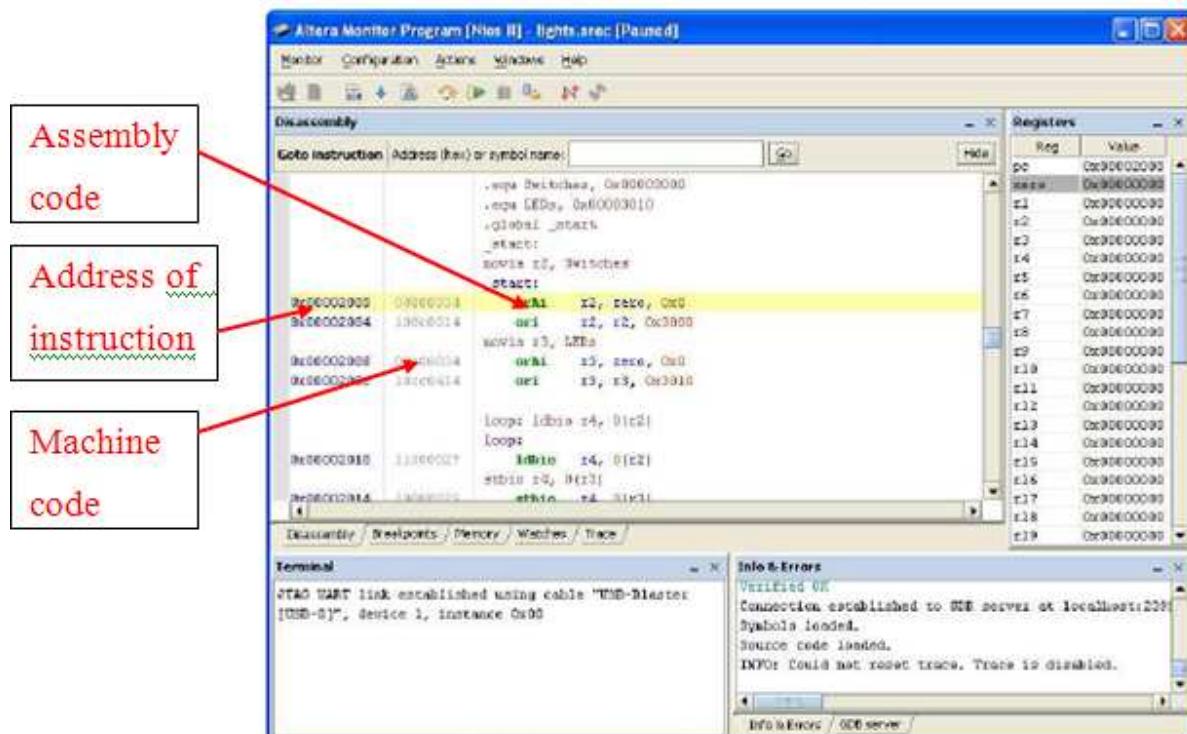
Bước 9. Chọn **Program Type : Assembly** (vì chúng ta đang muốn download một chương trình viết bằng Assembly language).

Bước 10. Nhấn **Add**, chỉ đường dẫn tới file lights.s (File chứa chương trình viết bằng Assembly language).

Bước 11. Nhấn **OK**

Bước 12. Chọn **Actions → Compile & Load**

Bước 13. Chọn **Actions → Continue** để thực thi chương trình trên Kit DE2. (chương trình sẽ được thực thi cho đến khi có một lệnh yêu cầu processor ngưng, chẳng hạn như **breakpoint** hay **Action → Stop**). Processor sẽ ngừng ở lệnh kế tiếp. Khi chương trình ngừng thực thi thì tất cả cửa sổ debugging sẽ được update giá trị.



Hình 6.32 Cửa sổ Debug

Bước 14. Điều khiển đóng mở Switch[0:7] và quan sát Leds[0:7].

Ta cũng có thể viết chương trình bằng ngôn ngữ C để nạp cho nios system.

Ta viết một file.c chứa chương trình được viết bằng C :

```

#define Switches (volatile char *) 0x0003000
#define LEDs (char *) 0x0003010
void main()
{
    while (1)
        *LEDs = *Switches;
}

```

Hình 6.33 Chương trình dùng C

Cũng thực hiện tương tự những bước trên , chỉ khác ở chỗ khi chọn Program Type thay vì chọn Assembly thi ta chọn C

Quan sát kết quả, ta thấy nó cũng cho kết quả tương tự .

- ✚ Chúng ta có thể dùng Altera Monitor Program để debug chương trình:
 - Ta có thể thực thi chương trình từng bằng việc thực thi từng lệnh một theo tuần tự.
 - Ta có thể dừng việc thực thi một chương trình tại một vị trí lệnh nào đó bằng việc tạo ra một breakpoint.
 - Ta có thể thay đổi giá trị của Register.
 - Ta có thể thay đổi giá trị của memory.
- ❖ Thực thi chương trình từng bằng việc thực thi từng lệnh một theo tuần tự:

Action ➔ Restart

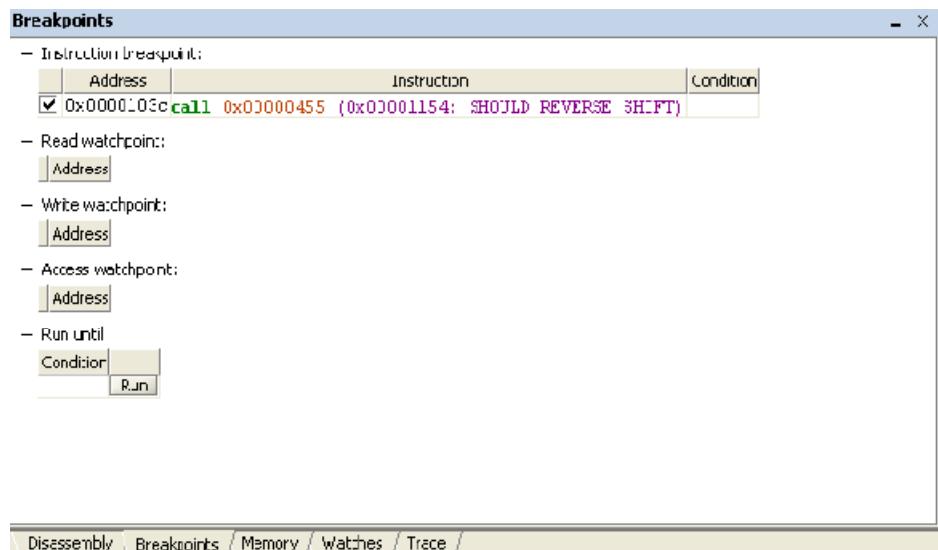
Action ➔ Single step

❖ Dừng việc thực thi một chương trình tại một vị trí lệnh nào đó:

Altera Monitor Program hỗ trợ tạo điểm breakpoint cho 4 trường hợp:

- Khi chương trình thực thi đến một địa chỉ nào đó mà ta muốn dừng.
- Khi có một thực hiện read data tại một địa chỉ nào đó mà ta muốn dừng.
- Khi có một thực hiện write data vào một địa chỉ nào đó mà ta muốn dừng.
- Vì xử lý truy cập vào một địa chỉ nào đó trong memory mà ta muốn dừng.

Để tạo điểm breakpoint cho chương trình, ta chuyển sang cửa sổ Breakpoints và thực hiện các bước sau

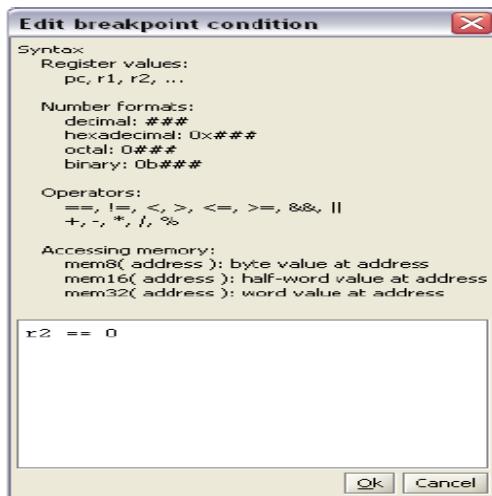


Hình 6.34 Tạo Breakpoints

Bước 1. Nhấn chuột phải vào header nào mà ta muốn tạo breakpoint, ở đây ta có 5 header (**Instruction breakpoint**, **Read watchpoint**, **Write watchpoint**, **Access watchpoint**, **Run until**)

Bước 2. Nhấn **Add** và nhập địa chỉ mà ta muốn thực hiện việc dừng chương trình.

Bước 3. Ta có thể thiết lập điều kiện để cho điểm breakpoint xảy ra bằng cách: double click vào cell bên dưới cột **Condition**.



Hình 6.35 Thiết lập điều kiện tạo Breakpoints

Bước 4. Nhập điều kiện cho điểm breakpoint, nhấn **OK**.

Bước 5. Sau khi tạo breakpoint, thực thi lại chương trình. Khi gặp đúng điều kiện mà ta đã tạo cho breakpoint thì chương trình sẽ dừng lại tại lệnh kế tiếp.

❖ Thay đổi giá trị của Register:

Bước 1. Chuyển sang cửa sổ **Disassembly**

Bước 2. Ta thấy có cửa sổ **Register** (hiển thị giá trị của từng Registers trong processor).



Hình 6.36 Thanh ghi Registers

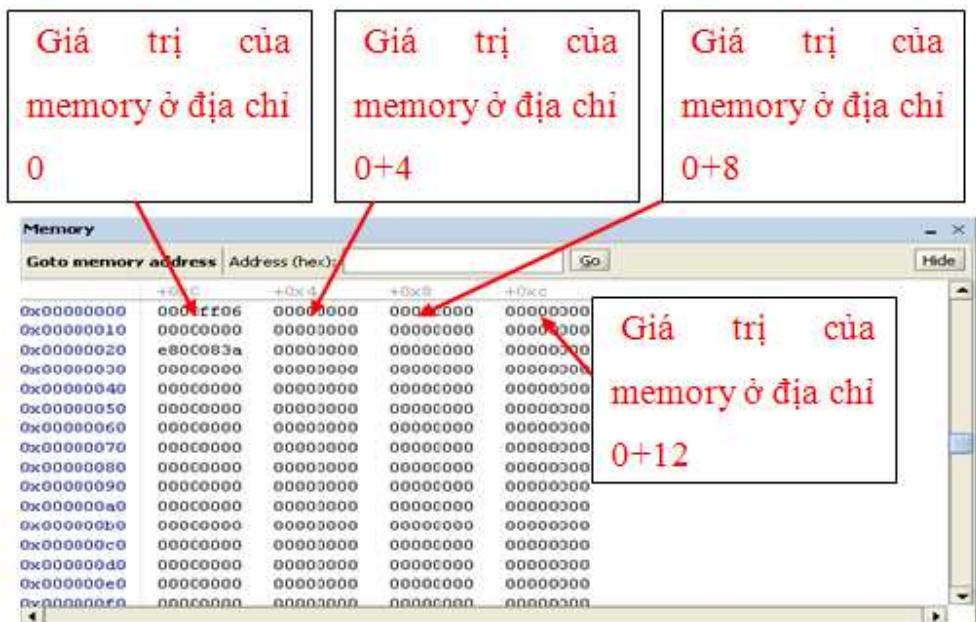
Pc register: program counter register (chứa giá trị địa chỉ của lệnh hiện hành).

Bước 3. Thay đổi giá trị register bằng cách double-click vào ô giá trị của register cần thay đổi và nhập giá trị cần đổi.

Bước 4. Thực thi lại chương trình để kiểm tra xem chương trình có thực thi theo đúng giá trị đã thay đổi trong register không.

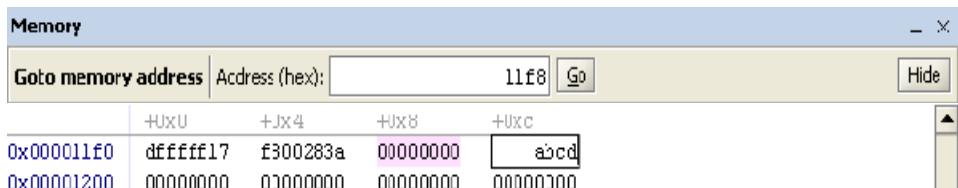
❖ Thay đổi giá trị của memory (chứa data và mã máy của chương trình)

Bước 1. Chuyển sang cửa sổ Memory



Hình 6.37 Vùng nhớ của On-Chip memory

Bước 2. Double-click vào giá trị của địa chỉ nào cần thay đổi để thay đổi.



Hình 6.38 Thay đổi nội dung ô nhớ

Bước 3. Nhập giá trị cần thay đổi.

Bước 4. Thực thi lại chương trình và quan sát xem chương trình có thực thi đúng với giá trị đã được thay đổi trong memory hay không.

6.3 Nội dung thực hành môn Kiến trúc máy tính nâng cao

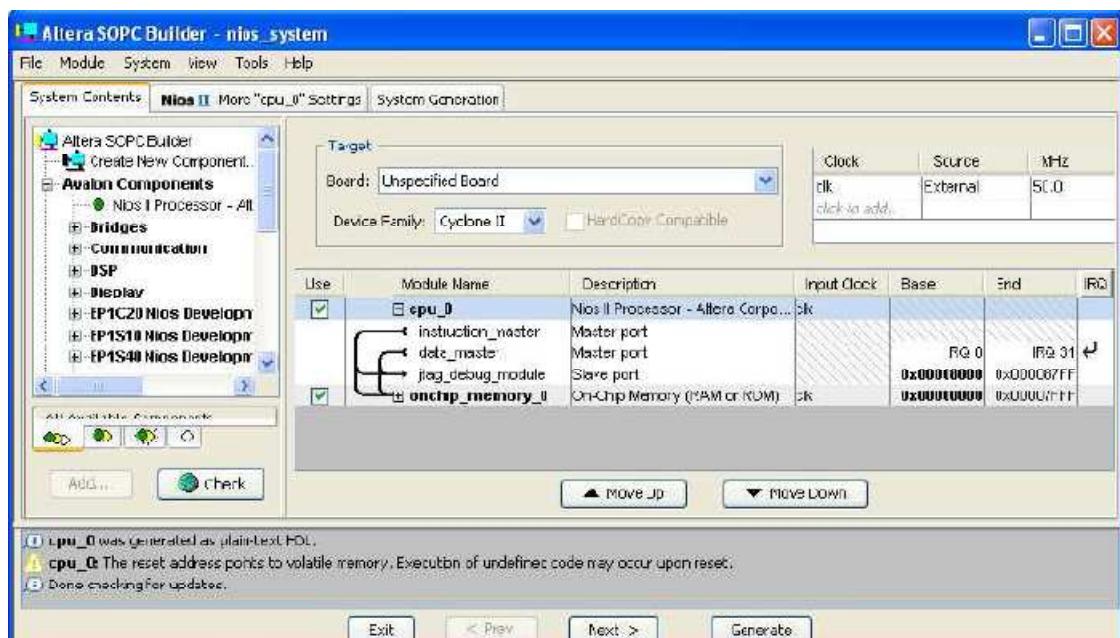
6.3.1 Bài thực hành số 1 – Thiết kế và sử dụng một hệ thống máy tính đơn giản

Mục đích:

- Xây dựng một hệ thống vi xử lý đơn giản
- Làm quen cách sử dụng công cụ debug Altera Monitor Program.

6.3.1.1 Phần 1

Dùng SOPC Builder tạo một Nios system như hình dưới :



Hình 6.39 SOPC Builder

Các bước thực hiện:

- Bước 1. Mở một project QuartusII có tên : nios_system_lab1
- Bước 2. Chọn target chip : Cyclone II EP2C35F672C6
- Bước 3. Dùng SOPC Builder tạo một Nios system có tên nios_sytem, có chứa những module sau :

- Nios II/e processor with JTAG Debug Module Level 1.

- On-Chip Memory – RAM mode với dung lượng 32 Kbytes, mỗi byte 32 bits.

Bước 4. Generate system.

Bước 5. Tạo một top design.v (verilog) để gọi tới nios_system.

Bước 6. Gán pins cho top design.v (verilog). (Tên module phải giống tên module defined khi tạo project).

- clk – PIN_N2 (50 MHz clock)
- reset_n – PIN_G26.

Bước 7. Compile project

Bước 8. Program và configure nios system lên Cyclone II EP2C35F672C6 trên FPGA.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

 Đọc và thực hiện các bước từ 1 đến 7 ở nhà.

Sau khi ta đã tạo được một system (có processor), để ứng dụng system này thì ta phải viết một chương trình thực thi bằng Assemble language hoặc C language. ➔ Part II.

6.3.1.2 Phần 2

Viết và kiểm tra một chương trình ứng dụng trên hệ thống NiosII đã được tạo ở phần 1.

Các bước thực hiện:

Bước 1. Viết một chương trình bằng Assemble như sau :

```
.include "nios_macros.s"
.text
.equ TEST_NUM, 0x90abcdef
```

```

.global _start
_start:
    movia r7, TEST_NUM
    mov r4, r7
    STRING_COUNTER:
    mov r2, r0
    STRING_COUNTER_LOOP:
    beq r4, r0, END_STRING_COUNTER
    srl r5, r4, 1
    and r4, r4, r5
    addi r2, r2, 1
    br STRING_COUNTER_LOOP
    END_STRING_COUNTER:
    mov r16, r2
    END:
    br END
.end

```

- Bước 2. Nêu ý nghĩa và giải thuật của chương trình trên.
- Bước 3. Viết mã máy (machine instruction) cho những lệnh ở trên.
- Bước 4. Mở Altera Monitor Program software.
- Bước 5. Configure nios system và chương trình Assemble trên vào Altera Monitor Program software.
- Bước 6. Compile và load data.
- Bước 7. Cho thực thi từng bước . Quan sát và ghi lại sự thay đổi giá trị của register pc, zero, r2, r4, r5, r7, r16 qua từng bước thực thi.
- Bước 8. Giải thích hiện tượng thay đổi đó.

Bước 9. Tạo breakpoint cho chương trình với điều kiện sau :

- Instruction breakpoint address : 801c (địa chỉ chứa lệnh : addi r2, r2, 0x1) và gán condition : r2 == 3
- Thực thi lại chương trình. (Nhớ “restart” lại trước khi “continue”). Quan sát và ghi lại và giải thích sự thay đổi giá trị của register pc, zero, r2, r4, r5, r7, r16 khi breakpoint xảy ra.

Bước 10. Thay đổi giá trị register :

- Pc register : 8008 (địa chỉ chứa lệnh : add r4, r7, zero).
- R7 = abcdef90
- Thực thi lại từng bước chương trình. (Không “restart” lại). Quan sát và ghi lại và giải thích sự thay đổi giá trị của register pc, zero, r2, r4, r5, r7, r16 qua từng bước thực thi.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

- ➡ Đọc và thực hiện các bước từ 1 đến 3 ở nhà.
- ➡ Trình bày các yêu cầu còn lại trong bài báo cáo.

6.3.1.3 Phần 3

Tìm hiểu về sự thay đổi giá trị trong onchip memory.

Các bước thực hiện

- Bước 1. Reload lại chương trình (Action ➔ Load)
- Bước 2. Thực thi lại chương trình (Action ➔ Continue).
- Bước 3. Dừng thực thi chương trình tại cuối chương trình (Action ➔ Stop).
- Bước 4. Viết mã máy cho hai lệnh sau:

- and r3, r7, r16
- sra r7, r7, r3

Bước 5. Thay đổi giá trị trong memory tại địa chỉ 8000 (địa chỉ chứa lệnh : orhi r7, zero, 0x90ab) và 8004 (địa chỉ chứa lệnh : ori r7, r7, 0xcdef) bởi hai giá trị mã máy trên.

Bước 6. Thay đổi giá trị của pc register thành 8000

Bước 7. Chạy thực thi từng bước. (Không “restart” lại chương trình).
Quan sát và ghi lại và giải thích sự thay đổi giá trị của register pc, zero, r2, r4, r5, r7, r16 qua từng bước thực thi.

Bước 8. Lặp lại các bước trên cho hai lệnh sau :

- srl r7, r7, r3.
- sra r7, r7, r3

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

 Viết mã máy cho các lệnh trên ở nhà.

6.3.1.4 Phần 4

Viết chương trình assemble trên dưới dạng một chương trình con (subroutine). Và chương trình con này sẽ được gọi (call) bởi chương trình chính. (Chuẩn bị trước ở nhà)

-  Chương trình con sẽ dùng r4 để nhận input data từ chương trình chính.
-  Chương trình con sẽ dùng r2 làm giá trị trả về.

6.3.2 Bài thực hành số 2 – Điều khiển nhập xuất dữ liệu từ Vi xử lí

Mục đích:

Tìm hiểu cách tạo Processor với khả năng Input/Output và cách sử dụng những SWs/KEYs/LEDs để nhập xuất dữ liệu với Processor.

6.3.2.1 Phần 1

Tạo Nios System bao gồm Nios II/s processor, onchip_memory và 3 PIOs dùng cho việc nhập và xuất dữ liệu.

Từng bước thực hiện:

Bước 1. Tạo một project Quartus mới, đặt tên:
.../lab2/nios_system_lab2

Bước 2. Mở SOPC Builder tạo một Nios System với tên nios_system theo yêu cầu:

- ❖ Nios II/s processor với JTAG Debug Module Level 1, lựa chọn những option sau:
 - Embedded Multipliers for Hardware Multiply
 - Hardware Divide
- ❖ On-chip memory –RAM mode và size là 32 Kbytes
- ❖ Một PIO input 8 bits (đặt tên cho module này là : new_number, tương ứng với SW [7:0])
- ❖ Một PIO output 8 bits (đặt tên cho module này là : green_LEDs, tương ứng với LEDG [7:0])
- ❖ Một PIO output 16 bits (đặt tên cho module này là : red_LEDs, tương ứng với LEDR [15:0])

Chú ý:

SOPC Builder sẽ tự động đặt tên 3 PIO vừa tạo ra là pio_0, pio_1, pio_2. Ta nên đổi tên cho dễ hiểu và gần với ý nghĩa của chúng, như new_number, green_LEDs và red_LEDs.

Bước 3. Tạo một top module cho nios_system_lab2.v cho system vừa tạo.(Nhớ đặt tên pin để có thể sử dụng file DE2_pin_assignments cho bước assign pin).

nios_system NiosII (CLOCK_50, KEY[0], LEDG, SW, LEDR);

Bước 4. Compile system để tạo ra file nios_system_lab2.sof và nios_system_lab2.ptf

Bước 5. Nạp xuống FPGA

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó).

■ Yêu cầu sinh viên chuẩn bị và thực hiện các bước 1 và 2 ở nhà để trên lớp có nhiều thời gian nghiên cứu trên Kit DE2. (nên đặt tên project và tên nios system và tên module theo quy ước ở trên để thống nhất).

■ Tìm hiểu và trả lời các câu hỏi sau :

- Vẽ symbol cho 4 module : RAM và 3 PIOs. (nêu ý nghĩa của từng pin trong module).
- Vẽ sơ đồ khối và phương thức hoạt động của Nios system vừa tạo ở trên.

6.3.2.2 Phần 2

Viết chương trình cộng tích lũy một số 8 bits được nhập vào bằng 8 Switches trên DE2. Dùng 8 đèn LEDG để biểu diễn số tương ứng với số trên SW nhập vào, dùng 16 LEDR để biểu diễn kết quả tổng tính được. Chương trình như sau:



Hình 6.40 Based address của các components

```
.include "nios_macros.s"

.equ      NEW_NUMBER, 0x11000
.equ      GREEN_LEDs, 0x11010
.equ      RED_LEDs , 0x11020

.text

.global _start
_start:

    add      r17, r0, r0
    movia r8, NEW_NUMBER
    movia r9, GREEN_LEDs
    movia r10, RED_LEDs

MAIN_LOOP:
    ldwio   r16, 0(r8)
    stwio   r16, 0(r9)
    add     r17, r17, r16
    stwio   r17, 0(r10)
```

```
br          MAIN_LOOP  
.end
```

Bước 1. Dùng Altera Monitor để configure, compile và load chương trình trên cho Nios system mà đã được tạo trên FPFA trong phần 1.

Bước 2. Thay đổi giá trị của SW[0] lên “1” (SW[7:1] vẫn giữ “0”), cho thực thi từng bước. Quan sát và ghi lại hiện tượng trên LEDG[7:0] và LEDR[15:0]. Cho chương trình thực thi đến khi giá trị của LEDR bằng 5 thì bật tiếp SW[1] lên “1” (không restart lại chương trình), tiếp tục cho thực thi từng bước đến khi giá trị LEDR bằng 14 thì dừng. Quan sát và ghi lại hiện tượng trên LEDG[7:0] và LEDR[15:0]. Giải thích hiện tượng.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau : (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó).

- Nêu ý nghĩa từng instruction của chương trình trên.
- Nêu ý nghĩa của chương trình trên.
- Nêu giải thuật của chương trình trên.

6.3.2.3 Phần 3

Mục đích của phần này là tạo một cờ hiệu để báo cho nios processor biết khi có sự thay đổi giá trị của input. Khi cờ này không được bật lên thì dù ta có thay đổi giá trị của input (SW[7:0]), giá trị này cũng sẽ không tác động vào chương trình, giá trị output sẽ không thay đổi. Chỉ khi nào cờ này được bật thì giá trị input dùng trong việc thực thi tính toán của chương trình mới được update và output của

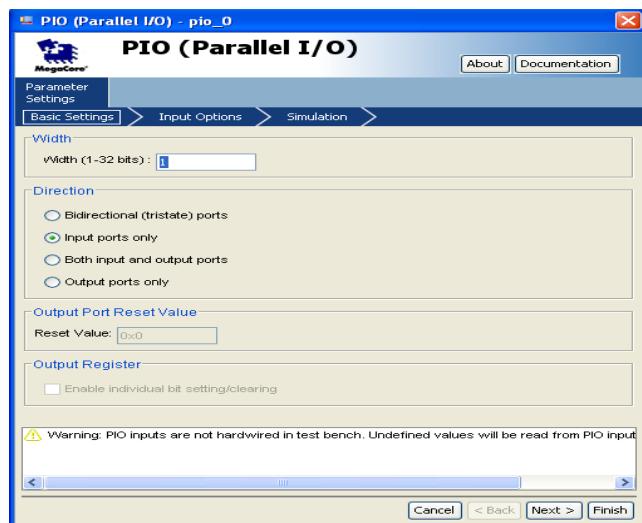
chương trình sẽ tiếp tục được cộng với input vừa được update. Và giá trị output sẽ được hiện thị vừa trên LEDR vừa trên 7-segments LEDs.

Để thực hiện được mục đích này ta cần phải tạo một cờ báo hiệu cho Nios system. Các bước thực hiện như sau :

Bước 1. Thoát khỏi Altera Monitor Program. Quay về SOPC Builder (Bước 2, phần 1), thêm vào hệ thống một component input I/O

Bước 2. Chọn Tab : System Contents → Peripherals → Microcontroller Peripherals → PIO (Parallel I/O).

Bước 3. Click **Add**.

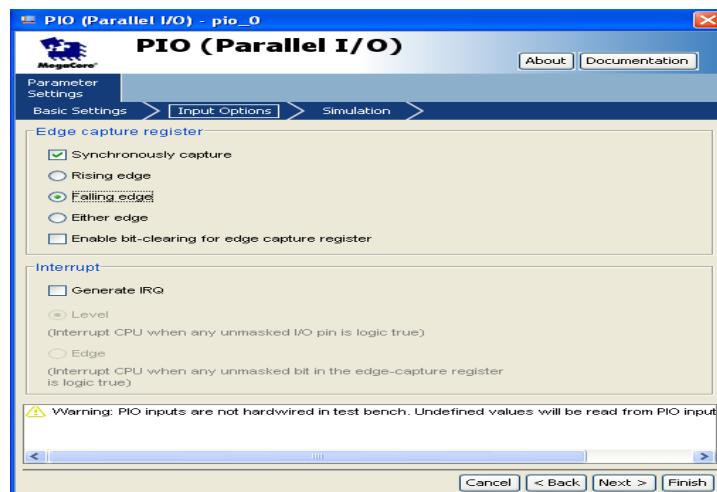


Hình 6.41 Thiết lập thông số cho PIO

Bước 4. Chọn Width : 1bits

Bước 5. Chọn : Input ports only

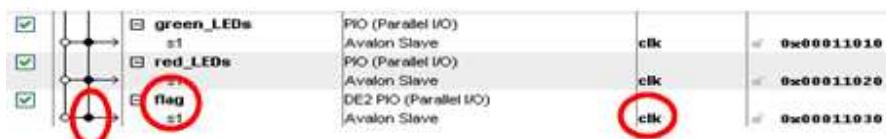
Bước 6. Chuyển sang Tab : Input Options



Hình 6.42 Thiết lập thông số đồng bộ cho Input Port

Bước 7. Chọn : Synchronously capture -> Falling Edge

Bước 8. Finish



Hình 6.43 Đổi thông số trên cửa sổ SOPC Builder

Bước 9. Đổi tên module thành “flag”

Bước 10. Connect “flag”

Bước 11. Chọn clk

Bước 12. Re-generate nios system.

Bước 13. Quay về QuartusII

Bước 14. Tạo lại một top module cho nios_system_lab2.v như sau

```
module nios_system_lab2 (
    // Inputs
```

```

CLOCK_50,
KEY,
SW,
// Outputs
LEDR,
LEDG,
HEX0,
HEX1,
HEX2,
HEX3

);

// Inputs
input CLOCK_50;
input [3:0] KEY;
input [17:0] SW;
// Outputs
output [17:0] LEDR;
output [8:0] LEDG;
output [6:0] HEX0;
output [6:0] HEX1;
output [6:0] HEX2;
output [6:0] HEX3;

```

*

```

    wire [15:0] SUM;
    // Output Assignments
    assign LEDR[15:0] = SUM;

*****  

*  

nios_system the_nios_system (
    // Inputs
    .clk      (CLOCK_50),
    .reset_n  (KEY[0]),
    .in_port_to_the_new_number      (SW[7:0]),
    .in_port_to_the_flag           (KEY[1]),
    // Outputs
    .out_port_from_the_green_LEDs (LEDG[7:0]),
    .out_port_from_the_red_LEDs   (SUM)
);

```

Hexadecimal_To_Seven_Segment_Digit0 (

```

    // Inputs
    .hex_number      (SUM[3:0]),

```

```

    // Outputs

```

```

    .seven_seg_display(HEX0)

```

```

);

```

Hexadecimal_To_Seven_Segment_Digit1 (

```

    // Inputs

```

```

    .hex_number      (SUM[7:4]),
    // Outputs
    .seven_seg_display(HEX1)
);

Hexadecimal_To_Seven_Segment Digit2 (
    // Inputs
    .hex_number      (SUM[11:8]),
    // Outputs
    .seven_seg_display(HEX2)
);

```

```

Hexadecimal_To_Seven_Segment Digit3 (
    // Inputs
    .hex_number      (SUM[15:12]),
    // Outputs
    .seven_seg_display(HEX3)
);
endmodule

```

```

***** Module:    Hexadecimal_To_Seven_Segment
module Hexadecimal_To_Seven_Segment (
    // Inputs
    hex_number,
    // Outputs
    seven_seg_display
);

```

```

// Inputs
 [3:0] hex_number;
// Outputs
output [6:0] seven_seg_display;
assign seven_seg_display =
    ({7{(hex_number == 4'h0)}} & 7'b1000000) /
    ({7{(hex_number == 4'h1)}} & 7'b1111001) /
    ({7{(hex_number == 4'h2)}} & 7'b0100100) /
    ({7{(hex_number == 4'h3)}} & 7'b0110000) /
    ({7{(hex_number == 4'h4)}} & 7'b0011001) /
    ({7{(hex_number == 4'h5)}} & 7'b0010010) /
    ({7{(hex_number == 4'h6)}} & 7'b0000010) /
    ({7{(hex_number == 4'h7)}} & 7'b1111000) /
    ({7{(hex_number == 4'h8)}} & 7'b0000000) /
    ({7{(hex_number == 4'h9)}} & 7'b0010000) /
    ({7{(hex_number == 4'hA)}} & 7'b0001000) /
    ({7{(hex_number == 4'hB)}} & 7'b0000011) /
    ({7{(hex_number == 4'hC)}} & 7'b1000110) /
    ({7{(hex_number == 4'hD)}} & 7'b0100001) /
    ({7{(hex_number == 4'hE)}} & 7'b0000110) /
    ({7{(hex_number == 4'hF)}} & 7'b0001110);

```

endmodule

Bước 15. Add lại tất cả các files.v (nios system vừa được tạo lại) cùng với top module cho nios_system_lab2.v ở trên trước khi re-compile.

Bước 16. Compile system để tạo ra file nios_system_lab2.sof và nios_system_lab2.ptf

Bước 17. Nạp xuống FPGA

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó).

- + Yêu cầu sinh viên chuẩn bị và thực hiện các bước 1 và 2 ở nhà để trên lớp có nhiều thời gian nghiên cứu trên Kit DE2. (nên đặt tên project và tên nios system và tên module theo quy ước ở trên để thống nhất).
- + Tìm hiểu và trả lời các câu hỏi sau :
 - Giải thích ý nghĩa của module nios_system_lab2.v ở trên.

6.3.2.4 Phần 4

Viết chương trình cộng tích lũy một số 8 bits được nhập vào bằng 8 Switches trên DE2. Dùng 8 đèn LEDG để biểu diễn số tương ứng với số trên SW nhập vào, dùng 16 LEDR và 4 LED 7-segments để biểu diễn kết quả tổng tính được. Chương trình như sau:



Hình 6.44 Mapping giữa Based address trên SOPC với address trong program

.include "nios_macros.s"

```
.equ NEW_NUMBER,      0x11000
.equ GREEN_LEDS,     0x11010
.equ RED_LEDS,       0x11020
.equ STATUS_FLAG,    0x11030
```

.text

```
.global _start
_start:
        add      r17, r0, r0
        movia r8, NEW_NUMBER
        movia r9, GREEN_LEDS
        movia r10, RED_LEDS
        movia r11, STATUS_FLAG
```

MAIN_LOOP:

```
ldwio  r16, 0(r8)
stwio  r16, 0(r9)
ldwio  r18, 12(r11)
beq   r18, r0, MAIN_LOOP
stwio  r0, 12(r11)
add   r17, r17, r16
stwio  r17, 0(r10)
br    MAIN_LOOP
```

.end

- Bước 1. Dùng Altera Monitor để configure, compile và load chương trình trên cho Nios system mà đã được tạo trong phần 3 lên FPFA .
- Bước 2. Thay đổi giá trị của SW[0] lên “1” (SW[7:1] vẫn giữ “0”) , cho thực thi từng bước. Quan sát và ghi lại hiện tượng trên LEDG [7:0] và LEDR[15:0] và 7 segment LEDs.
- Bước 3. Nhấn KEY[1], sau đó tiếp tục cho thực thi từng bước, quan sát và ghi lại hiện tượng trên LEDG[7:0] và LEDR[15:0] và 7 segment LEDs.
- Bước 4. Thay đổi giá trị SW[1] lên “1” (vẫn giữ SW[0] = ”1”). sau đó tiếp tục cho thực thi từng bước, quan sát và ghi lại hiện tượng trên LEDG[7:0] và LEDR[15:0] và 7 segment LEDs.
- Bước 5. Nhấn KEY[1], sau đó tiếp tục cho thực thi từng bước, quan sát và ghi lại hiện tượng trên LEDG[7:0] và LEDR[15:0] và 7 segment LEDs.
- Bước 6. Nhấn KEY[1], sau đó tiếp tục cho thực thi từng bước, quan sát và ghi lại hiện tượng trên LEDG[7:0] và LEDR[15:0] và 7 segment LEDs.

6.3.3 Bài thực hành số 3 – Tìm hiểu cách thức hoạt động và sử dụng Subroutine và Stack của Vi xử lí NiosII

Mục đích:

Tìm hiểu về cách tạo subroutines và subroutine linkage cho Nios system và những cách thức truyền param và ý nghĩa của một số thanh ghi như stack pointer (sp), ra.

6.3.3.1 Phần 1

Tạo Nios System bao gồm Nios II/s processor, onchip_memory và một JTAG UART module sử dụng để tương tác với host computer.

Từng bước thực hiện:

Bước 1. Tạo một project Quartus mới, đặt tên:
.../lab3/nios_system_lab3

Bước 2. Mở SOPC Builder tạo một Nios System với tên nios_system theo yêu cầu:

- Nios II/s processor với JTAG Debug Module Level 1.
- On-chip memory –RAM mode và size là 32 Kbytes.

Bước 3. Tạo một top module cho nios_system_lab3.v cho system vừa tạo và gọi tắt cả các module.v của nios_system được tạo ra bởi SOPC_Builder.

nios_system NiosII (CLK, RESET);

Bước 4. Assign pin :

- CLK – PIN_N2
- RESET – PIN_G26.

Bước 5. Compile system để tạo ra file nios_system_lab3.sof và nios_system_lab3.ptf

Bước 6. Nạp xuống FPGA

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó).

- Yêu cầu sinh viên chuẩn bị và thực hiện các bước 1 và 2 ở nhà để trên lớp có nhiều thời gian nghiên cứu trên Kit DE2. (nên đặt tên project và tên nios system và tên module theo quy ước ở trên để thống nhất).
- Tìm hiểu và trả lời các câu hỏi sau :
 - Vẽ sơ đồ khối và phương thức hoạt động của Nios system vừa tạo ở trên.

6.3.3.2 Phần 2

Cho một list những số dương (chứa trong một file, những số dương được cách nhau bởi dấu phẩy(,) , số đầu tiên chứa số những số dương trong file, không kể chính nó) . Những số này sẽ được đưa vào vùng nhớ trong memory on-chip. Viết chương trình Assemble dùng để sort những số dương (không kể số đầu tiên) theo chiều tăng dần, những giá trị sau khi sort sẽ store lại trong vùng nhớ mà đã chứa trước đó.

Chương trình tham khảo:

```
.include "nios_macros.s"
.text
.global _start
```

_start:

movia r8, SIZE

movia r9, LIST

BEGIN_SORT:

ldwio r20, 0(r8)

RESTART_SORT:

mov r18, r0

movi r19, 1

mov r10, r9

SORT_LOOP:

ldwio r16, 0(r10)

ldwio r17, 4(r10)

blt r16, r17, SKIP_SWAP

SWAP:

stwio r17, 0(r10)

stwio r16, 4(r10)

movi r18, 1

SKIP_SWAP:

addi r19, r19, 1

addi r10, r10, 4

bne r19, r20, SORT_LOOP

bne r18, r0, RESTART_SORT

END:

br END

.org 0x01000

LIST_FILE:

SIZE = 0x01000 - 0x08000 = 0x09000

LIST= 0x09004, 0x09008, 0x0900C,...

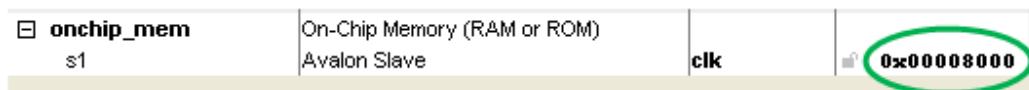
➔ Khi ghi giá trị trong file chứa những số cần sort vào trong memory on-chip thì phải chọn địa chỉ bắt đầu ghi là 0x09000

SIZE:

.word 0

LIST:

.end

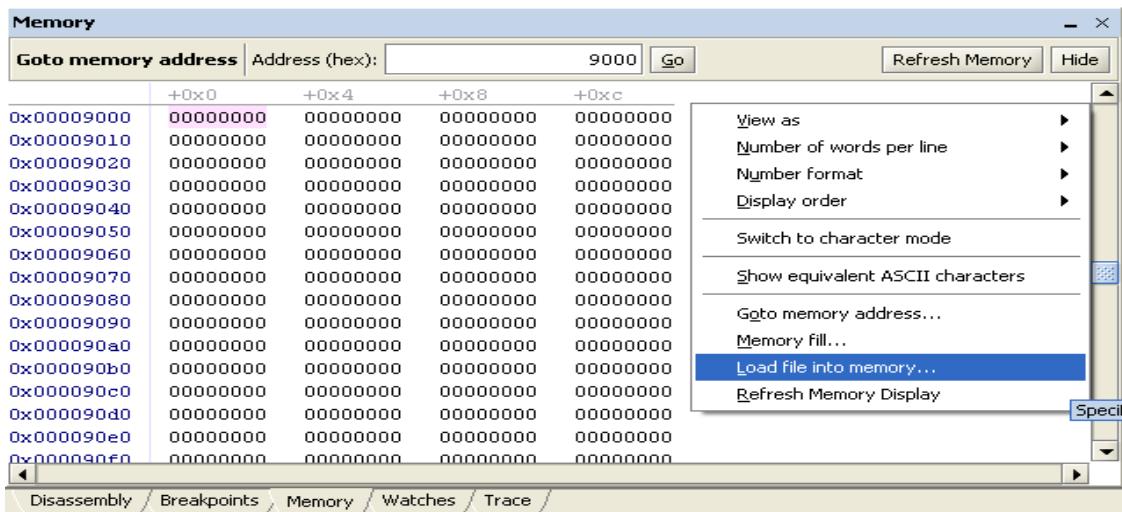


Hình 6.45 Based address

Bước 1. Dùng Altera Monitor để configure, compile và load chương trình trên cho Nios system mà đã được tạo trong Phần 1 lên FPFA.

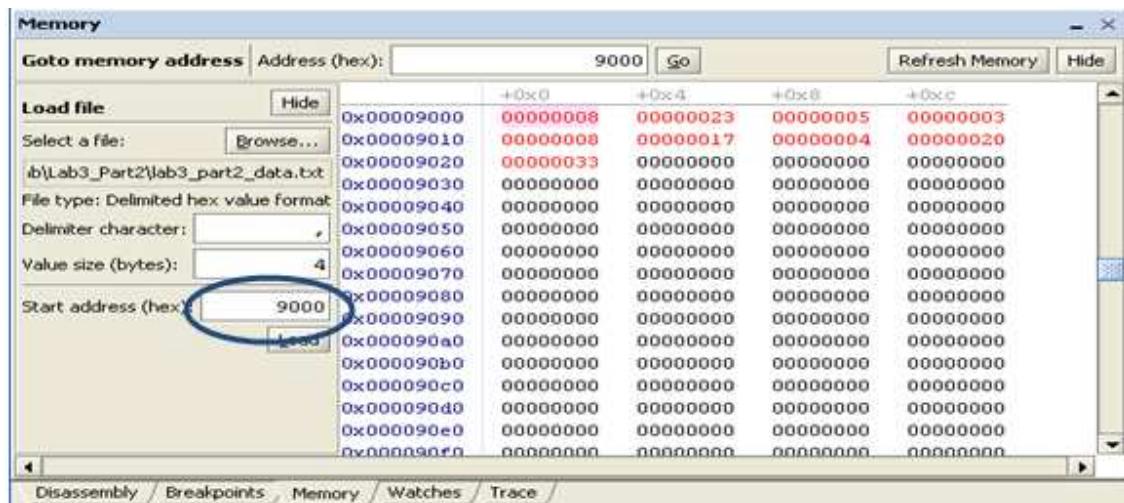
Bước 2. Load file chứa giá trị cần sort, nội dung file: 8,23,5,3,8,17,4,20,33 (phần tử đầu tiên “8” chỉ số phần tử cần sort. Load file như sau:

Bước 3. Chọn Tab memory trên Altera Monitor Program, nhấn chuột phải:



Hình 6.46 Load nội dung cho bộ nhớ

Bước 4. Chọn Load file into memory



Hình 6.47 Nội dung đã được load vào bộ nhớ

Bước 5. Select file : nhập đường dẫn của file chứa data.

Bước 6. Vì trong file data, những số được ngăn cách bởi dấu phẩy(,) ➔

Delimiter character : ,

Bước 7. Vì những số này sẽ được lưu trong memory dưới dạng số nhị phân 32 bits, nên một số sẽ cần 4 bytes để lưu trữ ➔ Value size (bytes): 4

Như ta đã đề cập ở trên số đầu tiên sẽ được lưu tại ô nhớ 0x9000.

Bước 8. Cho thực thi (Continue).

Bước 9. Stop thực thi, quan sát giá trị đã được sort trong memory. Nhận xét và giải thích hiện tượng.

Bước 10. Cho thực thi từng bước. Quan sát và rút ra giải thuật của chương trình.

Note : Sinh viên cần chuẩn bị trước ở nhà những công việc sau : (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó).

⚠️ Nêu ý nghĩa từng instruction của chương trình trên.

 Nêu giải thuật của chương trình trên.

6.3.3.3 Phần 3

Khảo sát cách tạo một sub-routine cho chương trình trên assemble, cũng như ý nghĩa của những thanh ghi stack pointer (sp), frame pointer (fp).

Cũng giống như yêu cầu trong phần Phần 2, viết một chương trình Assemble để sort data theo chiều tăng dần. Tuy nhiên trong chương trình này ta sẽ dùng một subroutine SORT để thực hiện việc sort data. Trong chương trình chính ta sẽ dùng thanh ghi r2 để chứa số phần tử cần sort, thanh ghi r3 để chứa địa chỉ của phần tử đầu tiên. Hai thanh ghi này sẽ được dùng để truyền data đến subroutine. Trong subroutine chúng ta sẽ sử dụng một số thanh ghi, nội dung của thanh ghi trước khi sử dụng cho subroutine phải được lưu trữ lại, sau khi subroutine thực hiện xong thì giá trị những thanh ghi trên phải được trả về giá trị trước khi nó được sử dụng trong subroutine.

Chương trình tham khảo:

```
.include "nios_macros.s"

.equ STACK 0xa000

.text
/* Main Program */

.global _start
_start:
    movia sp, STACK
    mov fp, sp
    movia r8, SIZE
```

<i>movia</i>	<i>r9, LIST</i>
<i>ldwio</i>	<i>r2, 0(r8)</i>
<i>mov</i>	<i>r3, r9</i>
<i>call</i>	<i>SORT</i>

END:

<i>br</i>	<i>END</i>
-----------	------------

/ SORT - Subroutine */*

SORT:

<i>subi</i>	<i>sp, sp, 28</i>
<i>stw</i>	<i>ra, 0(sp)</i>
<i>stw</i>	<i>fp, 4(sp)</i>
<i>stw</i>	<i>r8, 8(sp)</i>
<i>stw</i>	<i>r16, 12(sp)</i>
<i>stw</i>	<i>r17, 16(sp)</i>
<i>stw</i>	<i>r18, 20(sp)</i>
<i>stw</i>	<i>r19, 24(sp)</i>
<i>addi</i>	<i>fp, sp, 28</i>

BEGIN_SORT:

RESTART_SORT:

<i>mov</i>	<i>r18, r0</i>
<i>movi</i>	<i>r19, 1</i>
<i>mov</i>	<i>r8, r3</i>

SORT_LOOP:

<i>ldwio</i>	<i>r16, 0(r8)</i>
--------------	-------------------

<i>ldwio</i>	<i>r17, 4(r8)</i>
<i>blt</i>	<i>r16, r17, SKIP_SWAP</i>

SWAP:

<i>stwio</i>	<i>r17, 0(r8)</i>
<i>stwio</i>	<i>r16, 4(r8)</i>
<i>movi</i>	<i>r18, 1</i>

SKIP_SWAP:

<i>addi</i>	<i>r19, r19, 1</i>
<i>addi</i>	<i>r8, r8, 4</i>
<i>bne</i>	<i>r19, r2, SORT_LOOP</i>
<i>bne</i>	<i>r18, r0, RESTART_SORT</i>

END_SORT:

<i>ldw</i>	<i>ra, 0(sp)</i>
<i>ldw</i>	<i>fp, 4(sp)</i>
<i>ldw</i>	<i>r8, 8(sp)</i>
<i>ldw</i>	<i>r16, 12(sp)</i>
<i>ldw</i>	<i>r17, 16(sp)</i>
<i>ldw</i>	<i>r18, 20(sp)</i>
<i>ldw</i>	<i>r19, 24(sp)</i>
<i>addi</i>	<i>sp, sp, 28</i>
<i>ret</i>	

.org 0x01000

LIST_FILE:

SIZE:

.word 0

LIST:

.end

Bước 1. Dùng Altera Monitor để configure, compile và load chương trình trên cho Nios system mà đã được tạo trong Phần 1 lên FPFA

Bước 2. Load file chứa giá trị cần sort : (Giống Phần 2)

Bước 3. Cho thực thi (Continue).

Bước 4. Stop thực thi, quan sát giá trị đã được sort trong memory. Nhận xét và giải thích hiện tượng.

Bước 5. Quan sát giá trị trong vùng stack pointer và nhận xét.

Bước 6. Cho thực thi từng bước. Quan sát và rút ra giải thuật của chương trình.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau : (Không có bài chuẩn bị không được vào lớp làm thí nghiệm → Tính vắng buổi đó).

- Nêu cấu trúc của một subroutine cũng như cách gọi subroutine từ chương trình chính (Có thể tham khảo từ NiosII processor Reference Handbook).
- Nêu ý nghĩa của stack, stack pointer register, frame pointer register (Tham khảo NiosII processor Reference Handbook).
- Nêu ý nghĩa từng instruction của chương trình trên.
- Nêu giải thuật của chương trình trên.

6.3.3.4 Phần 4

Sửa lại chương trình trong phần 3, thay vì giá trị số phần tử và giá trị địa chỉ của phần tử đầu tiên được truyền vào subroutine từ chương trình chính thông qua register r2, r3. Ta sẽ sửa lại ta sẽ không dùng r2, r3 mà ta sẽ sử dụng stack để truyền data từ chương trình chính đến subroutine.

Bước 1. Sinh viên chuẩn bị trước ở nhà phần chương trình.

Bước 2. Dùng Altera Monitor để configure, compile và load chương trình trên cho Nios system mà đã được tạo trong Phần 1 lên FPFA .

Bước 3. Thực thi tương tự các bước (3 → 7) như trong Phần 3.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau : (Không có bài chuẩn bị không được vào lớp làm thí nghiệm → Tính vắng buổi đó).

#[blue] Chuẩn bị bước 1 ở nhà.

6.3.3.5 Phần 5

Khảo sát chức năng của thanh ghi ra.

Viết một chương trình tính giai thừa của một số bất kì :

- ❖ $n! = n(n-1)(n-2)\dots \times 2 \times 1$; trong đó chương trình chính sẽ gọi đến một subroutine, trong subroutine này sẽ gọi đến subroutine khác (mục đích để khảo sát chức năng của thanh ghi ra).
- ❖ Dùng SW[2:0] nhập giá trị n.
- ❖ Dùng LED 7-Segment (HEX3 → HEX0) để hiển thị kết quả n!

Các bước thực hiện:

Bước 1. Tạo Nios system như sau:

Use	Con...	Module Name	Description	Clock	Base
		instruction_master	Avalon Master		
		data_master	Avalon Master		IRQ 0
		jtag_debug_module	Avalon Slave		0x00010800
<input checked="" type="checkbox"/>		onchip_mem	On-Chip Memory (RAM or ROM)		
<input checked="" type="checkbox"/>		input_number	PIO (Parallel I/O)	clk	0x00008000
<input checked="" type="checkbox"/>		s1	Avalon Slave	clk	0x00011000
<input checked="" type="checkbox"/>		output_number	PIO (Parallel I/O)	clk	0x00011010
<input checked="" type="checkbox"/>		green_LEDs	PIO (Parallel I/O)	clk	0x00011020
		s1	Avalon Slave		

Hình 6.48 Nios System

Bước 2. Dựa vào phần 3 của bài thực hành 2, tạo một module top-level cho nios system trên để có thể nhập số vào bằng SW[2:0] và xuất số vừa nhập vào ra LEDG[2:0], và xuất kết quả ra LED 7 segments (HEX3 → HEX0]).

Chương trình tham khảo

```
.include "nios_macros.s"

.equ INPUT_NUM,      0x11000
.equ OUTPUT_NUM,     0x11010
.equ GREEN_LEDS,    0x11020
.equ STACK,          0xa000

.text

/* Main Program */

.global _start

_start:
        add      r4, r0, r0
        movia r10, INPUT_NUM
        movia r11, OUTPUT_NUM
```

moviar16, GREEN_LEDS

ldwio r2, 0(r10)

stwio r2, 0(r16)

moviasp, STACK

mov fp, sp

subi sp, sp, 4

stw r2, 0(sp)

call FACTOR

END:

br END

/ FACTOR - Subroutine */*

FACTOR:

subi sp, sp, 12

stw ra, 0(sp)

stw fp, 4(sp)

stw r17, 8(sp)

addi fp, sp, 12

ldw r17, 0(fp)

bne r17, r0, NON_ZERO

movi r4, 1

br END_FACTOR

NON_ZERO:

subi r17, r17, 1

subi sp, sp, 4

stw r17, 0(sp)

call FACTOR

```
    addi  sp, sp, 4  
    addi  r17, r17, 1  
    mul   r4, r4, r17  
    stwio r4, 0(r11)
```

END_FACTOR:

```
    ldw      ra, 0(sp)  
    ldw      fp, 4(sp)  
    ldw      r17, 8(sp)  
    addi   sp, sp, 12  
    ret
```

.end

Bước 3. Dùng Altera Monitor để configure, compile và load chương trình trên cho Nios system mà đã được tạo trên FPFA (bước 2, Part5).

Bước 4. Dùng SW[2:0] nhập một số cần tính giai thừa. Thực thi (Continue) để kiểm tra kết quả trên LED 7 Segment (HEX3 → HEX0).

Bước 5. Chạy thực thi từng bước, quan sát và giải thích giải thuật chương trình.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau : (Không có bài chuẩn bị không được vào lớp làm thí nghiệm → Tính vắng buổi đó).

- Chuẩn bị bước 2 ở nhà.
- Tìm hiểu giải thuật chương trình assemble trên.

6.3.4 Bài thực hành số 4 – Tìm hiểu cách thức hoạt động và sử dụng Polling và Interrupt của Vi xử lí NiosII

Mục đích:

Tìm hiểu cách gửi và nhận data tới/từ thiết bị xuất nhập.

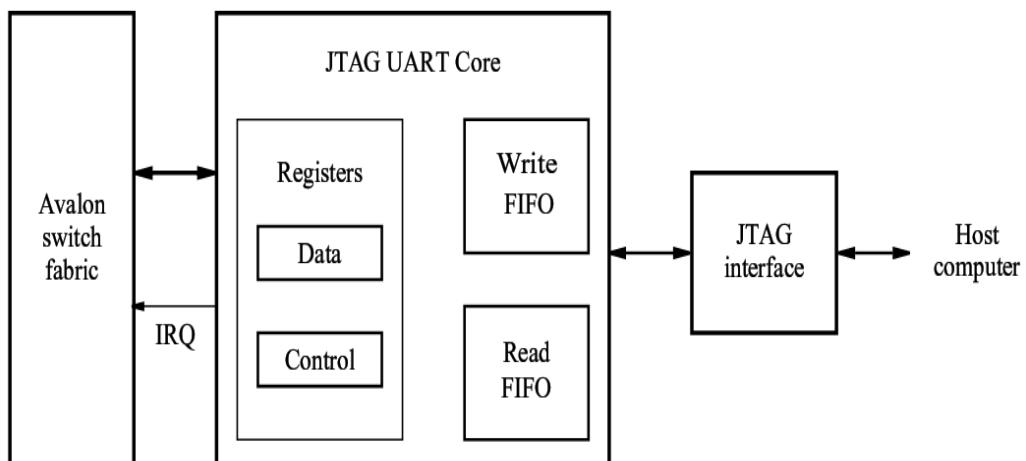
Có 2 phương thức được sử dụng để biết trạng thái data:

- Pooling: Processor truy xuất những thiết bị liên tục để xem trạng thái của data.

- Interrupts: Khi những thiết bị có thể nhận data hoặc data đã sẵn sàng để dùng, chúng sẽ tự interrupt tới processor.

Một phương pháp thông thường để chuyển data giữa processor và thiết bị xuất nhập là Universal Asynchronous Receiver Transmitter (UART). Một UART interface (circuit) được đặt giữa processor và thiết bị xuất nhập. Tại 1 thời điểm, nó có thể nhận hoặc gửi data là kí tự 8-bit. Việc chuyển dữ liệu giữa UART và processor được thực hiện song song. Tuy nhiên, việc chuyển dữ liệu giữa UART và thiết bị xuất nhập được thực hiện tuần tự từng bit.

Altera SOPC Builder có thể hiện thực 1 interface của dạng UART cho Nios II system, được gọi là JTAG UART. JTAG UART interface dùng để kết nối giữa Nios II processor và host computer.



Hình 6.49 Sơ đồ khái niệm của JTAG UART

Xem chương 5 của Altera *Embedded Peripherals Handbook* để hiểu rõ chức năng của thanh ghi Data, thanh ghi Control, Write FIFO, Read FIFO.

Chú ý: Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm → Tính vắng buổi đó).

- Tìm hiểu thanh ghi Data, Control.
- Tìm hiểu Write FIFO, Read FIFO

6.3.4.1 Phần 1

Sử dụng SOPC Builder để tạo 1 system gồm Nios II/e processor, một JTAG UART, một On-chip memory và một Interval Timer.

Từng bước thực hiện:

Bước 1. Tạo một project Quartus mới, đặt tên:
./lab4/nios_system_lab4

Bước 2. Mở SOPC Builder tạo một Nios System với tên nios_system theo yêu cầu:

- Nios II/e processor với JTAG Debug Module Level 1
- On-chip memory – RAM với size là 32 Kbytes.
- JTAG UART (sử dụng default setting): **Interface Protocols ➔ Serial ➔ JTAG UART; Nhấn Add**

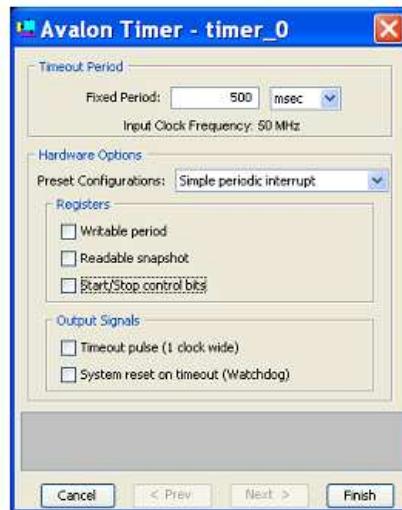


Hình 6.50 Chọn và thiết lập thông số cho JTAG UART

➤ Interval Time: **Peripherals → Microcontroller Peripherals → Interval Timer; Nhấn Add**

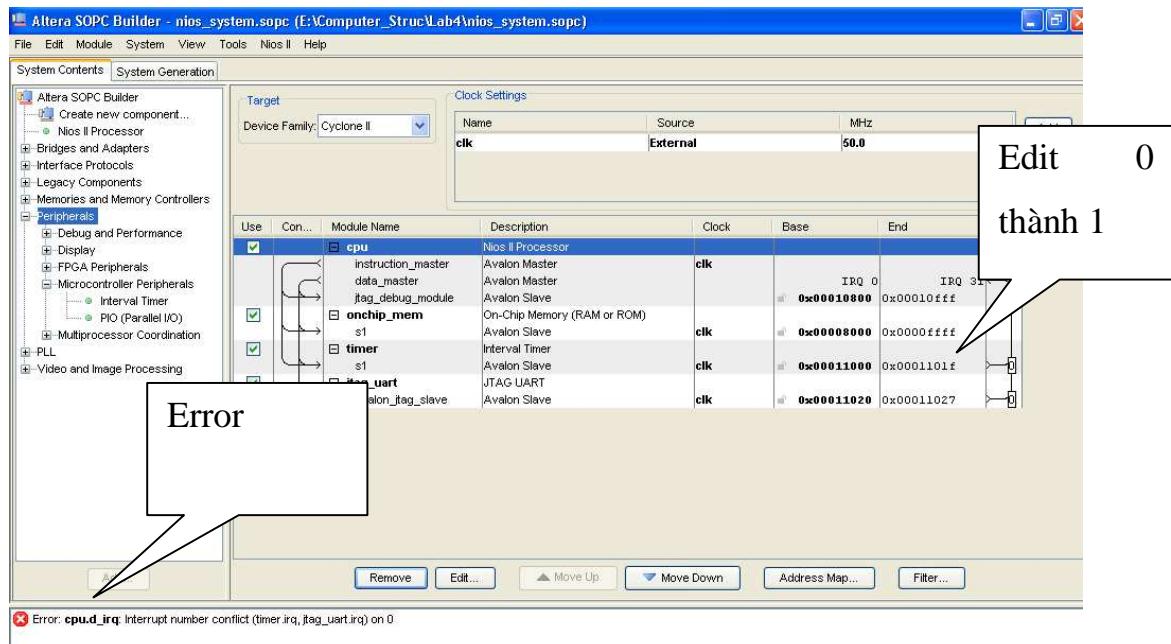
Khi add Interval Time cần chú ý những tùy chọn sau:

- ❖ Tại Hardware Options, Preset Configurations, chọn Simple periodic interrupt
- ❖ Tại Timeout Period, thiết lập Fixed Period với 500 msec



Hình 6.51 Chọn và thiết lập thông số cho timer

Lưu ý lỗi như hình sau:



Hình 6.52 Chọn interrupt request

Bước 3. Generate nios_system.

Bước 4. Tạo một top module cho nios_system_lab4.v cho system vừa tạo

Bước 5. Assign pin

- CLK – PIN_N2
- RESET – PIN_G26.

Bước 6. Compile system để tạo ra file nios_system_lab4.sof và nios_system_lab4.ptf

Bước 7. Nạp xuống FPGA

Chú ý: Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm → Tính vắng buổi đó).

■ Yêu cầu sinh viên chuẩn bị và thực hiện các bước 1 và 2 ở nhà để trên lớp có nhiều thời gian nghiên cứu trên Kit DE2. (nên đặt tên project và tên nios system và tên module theo quy ước ở trên để thống nhất).

■ Tìm hiểu và trả lời các câu hỏi sau :

- Vẽ sơ đồ khối và phương thức hoạt động của Nios system vừa tạo ở trên.

6.3.4.2 Phần 2

JTAG UART có thể gửi kí tự ASCII tới terminal window của Altera Debug Client. Khi vùng WSPACE trong thanh ghi Control của JTAG UART có giá trị khác 0, JTAG UART chấp nhận 1 kí tự mới được viết ra Altera Debug Client. Như vậy, để viết 1 kí tự ra Debug Client, ta phải đọc liên tục (Polling) thanh ghi Control cho tới khi nào vùng WSPACE khác 0.

Offset	Register Name	R/W	Bit Description															
			31	...	16	15	14	...	11	10	9	8	7	...	2	1	0	
0	data	RW	RAVAIL			RVALID	Reserved				DATA							
1	control	RW	WSPACE			Reserved				AC	WI	RI	Reserved		WE	RE		

Hình 6.53 JTAG UART Core Register Map

Bit(s)	Name	Access	Description
[7:0]	DATA	R/W	The value to transfer to/from the JTAG core. When writing, the DATA field holds a character to be written to the write FIFO. When reading, the DATA field holds a character read from the read FIFO.
[15]	RVALID	R	Indicates whether the DATA field is valid. If RVALID=1, then the DATA field is valid, otherwise DATA is undefined.
[32:16]	RAVAIL	R	The number of characters remaining in the read FIFO (after the current read).

Hình 6.54 Data Register Bits

Bit(s)	Name	Access	Description
0	RE	R/W	Interrupt-enable bit for read interrupts.
1	WE	R/W	Interrupt-enable bit for write interrupts.
8	RI	R	Indicates that the read interrupt is pending.
9	WI	R	Indicates that the write interrupt is pending.
10	AC	R/C	Indicates that there has been JTAG activity since the bit was cleared. Writing 1 to AC clears it to 0.
[32:16]	WSPACE	R	The number of spaces available in the write FIFO.

Hình 6.55 Control Register Bits

Các bước thực hiện

Bước 1. Sử dụng Nios II assembly, viết chương trình sử dụng thanh ghi Control và Data của JTAG UART để viết kí tự “Z” ra terminal window của Altera Debug Client.

Tham khảo đoạn code sau:

```
.include "nios_macros.s"  
.equ UART, 0x00009000
```

Mở SOPC Builder, địa chỉ này
phải giống Base Address của

```
.global _start  
  
_start:  
    movia r4, UART  
    movi r3, 'Z'
```

CHECK:

```
ldwio r2, 4(r4)  
srl r2, r2, 16  
beq r2, r0, CHECK  
stbio r3, 0(r4)
```

br CHECK

END:

br END

.end

Bước 2. Sử dụng Altera Debug Client compile và load chương trình trên.

Bước 3. Chạy chương trình, sử dụng single step (Nếu sử dụng chế độ Continue, kí tự sẽ xuất ra terminal window nhanh tới không thể theo dõi được).

- Bước 4. Theo dõi và giải thích ý nghĩa chương trình.
- Bước 5. Sửa lại đoạn code trên sao cho kí tự chỉ được in ra terminal sau khoảng thời gian xấp xỉ nửa giây.

Gợi ý:

- ❖ Input Clock Frequency: 50 MHz
- ❖ Lệnh của mỗi loại Nios II/e, Nios II/s hay Nios II/f thực hiện cần số chu kì khác nhau. Tham khảo:

<i>Instruction Execution Performance for Nios II/e Core</i>	
Instruction	Cycles
Normal ALU instructions (e.g., add, cmplt)	6
branch, jmp, jmpi, ret, call, callr	6
trap, break, eret, bret, flushp, wrctl, rdctl, unimplemented	6
load word	6 + Duration of Avalon-MM read transfer
load halfword	9 + Duration of Avalon-MM read transfer
load byte	10 + Duration of Avalon-MM read transfer
store	6 + Duration of Avalon-MM write transfer
Shift, rotate	7 to 38
All other instructions	6
Combinatorial custom instructions	6
Multi-cycle custom instructions	≥ 6

Hình 6.56 Số chu kì cho mỗi lệnh

Instruction Execution Performance for Nios II/s Core		
Instruction	Cycles	Penalties
Normal ALU instructions (e.g., add, cmplt)	1	
Combinatorial custom instructions	1	
Multi-cycle custom instructions	> 1	
Branch (correctly predicted taken)	2	
Branch (correctly predicted not taken)	1	
Branch (mispredicted)	4	Pipeline flush
trap, break, eret, bret, flushhp, wrctl, unimplemented	4	Pipeline flush
jmp, jmpi, ret, call, callr	4	Pipeline flush
rdctl	1	
load, store	> 1	
flushi, initi	4	
Multiply	(1)	
Divide	(1)	
Shift/rotate (with hardware multiply using embedded multipliers)	3	
Shift/rotate (with hardware multiply using LE-based multipliers)	4	
Shift/rotate (without hardware multiply present)	1 to 32	
All other instructions	1	

Hình 6.57 Số chu kì cho mỗi lệnh

Instruction Execution Performance for Nios II/I Core 4byte/line data cache		
Instruction	Cycles	Penalties
Normal ALU instructions (e.g., add, cmplt)	1	
Combinatorial custom instructions	1	
Multi-cycle custom instructions	> 1	Late result
Branch (correctly predicted, taken)	2	
Branch (correctly predicted, not taken)	1	
Branch (mis-predicted)	4	Pipeline flush
trap, break, eret, bret, flushp, wrctl; illegal and unimplemented instructions	4	Pipeline flush
call, jmpi	2	
jmp, ret, callr	3	
rdctl	1	Late result
load (without Avalon-MM transfer)	1	Late result
load (with Avalon-MM transfer)	> 1	Late result
store (without Avalon-MM transfer)	1	
store (with Avalon-MM transfer)	> 1	
flushd, flushda (without Avalon-MM transfer)	2	
flushd, flushda (with Avalon-MM transfer)	> 2	
initd, initda	2	
flushi, initi	4	
Multiply	(1)	Late result
Divide	(1)	Late result
Shift/rotate (with hardware multiply using embedded multipliers)	1	Late result
Shift/rotate (with hardware multiply using LE-based multipliers)	2	Late result
Shift/rotate (without hardware multiply present)	1 - 32	Late result
All other instructions	1	

Hình 6.58 Số chu kì cho mỗi lệnh

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó).

- Viết code cho phần 4
- Recompile, load và chạy lại chương trình.
- Giải thích ý nghĩa chương trình phần code mẫu và phần code tạo nửa giây.

6.3.4.3 Phần 3

JTAG UART không chỉ có thể viết kí tự ASCII ra terminal window của Debug Client mà còn có thể nhận kí tự từ terminal window.

Bit RVALID_bit 15 của thanh ghi Data chỉ ra khi nào vùng Data trong thanh ghi Data là 1 kí tự hợp lệ. Nếu nhiều kí tự chờ đợi để được đọc, vùng RAVAIL sẽ có giá trị khác 0.

Các bước thực hiện

Bước 1. Viết chương trình đọc mỗi kí tự mà được nhận bởi JTAG UART từ host computer và hiển thị kí tự này trong terminal window của Debug Client. Sử dụng polling (1 vòng lặp liên tục) để biết khi nào có kí tự mới từ JTAG UART

Bước 2. Code tham khảo:

```
.include "nios_macros.s"  
.equ  UART_BASE,      0x8820  
.equ  TIMER,          0x8800
```

```
.global _start  
_start:  
    movia r8, UART_BASE
```

```
GET_CHAR_LOOP:  
ldwio r17, 0(r8)
```

```
andi    r20, r17, 0x8000  
beq    r20, r0, GET_CHAR_LOOP  
andi    r16, r17, 0x00ff
```

PUT_CHAR_LOOP:

```
ldwio   r17, 4(r8)  
andhi   r17, r17, 0xffff  
beq    r17, r0, PUT_CHAR_LOOP  
stwio   r16, 0(r8)
```

br GET_CHAR_LOOP

END:

br END

.end

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó).

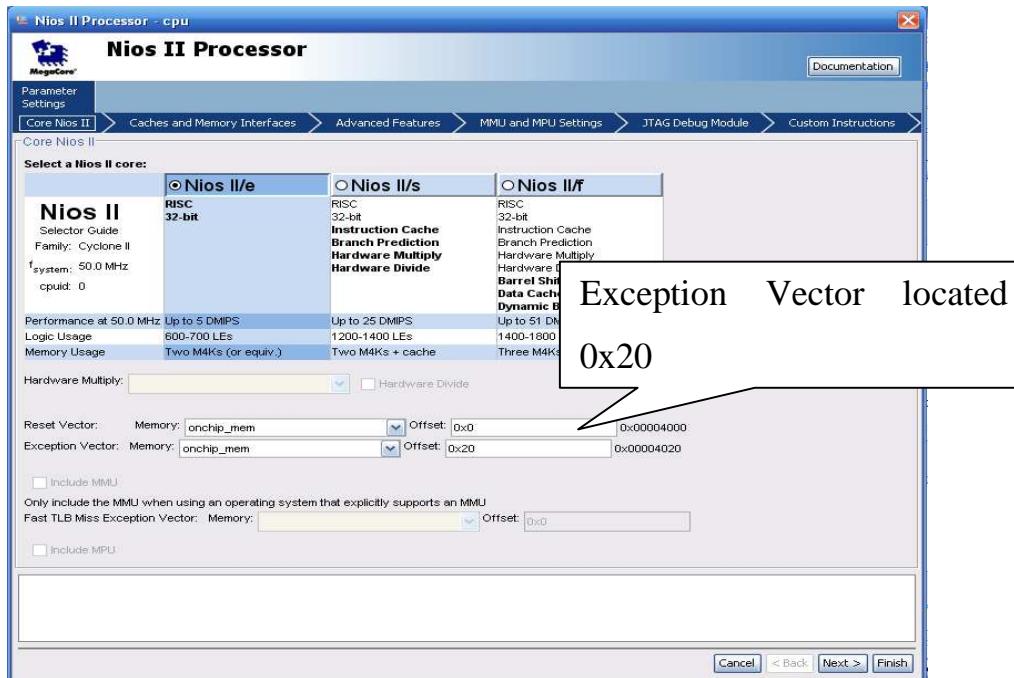
- Giải thích ý nghĩa cho đoạn code mẫu trên
- Compile, load và chạy chương trình trên.
- Quan sát và giải thích ý nghĩa giải thuật, vẽ lưu đồ giải thuật.

6.3.4.4 Phần 4

Từ Phần 1 tới Phần 3 ta đã làm quen với polling, Phần 4 hướng dẫn như thế nào sử dụng interrupts.

Khi tạo 1 interrupt-service routine để đọc kí tự được nhận bởi JTAG UART từ host computer, ta đặt interrupt-service routine tại địa chỉ 0x20 (Thật sự là Base

Address của memory + 0x20, địa chỉ này là mặc định cho exception handler do SOPC Builder chọn). Exception trả về địa chỉ trong thanh ghi ea phải được giảm 4 cho những external interrupts.



Hình 6.59 Exception Vector

Đây là mẫu chung khi viết 1 interrupt-service routine

.include "nios macros.s"

.text

```
.org 0x20          /* Place the interrupt service routine */
/* at the appropriate address */
```

ISR:

```
rdctl et, ctl4      /* Check if an external */
beq et, r0, SKIP EA DEC    /* interrupt has occurred */
```

```
    subi ea, ea, 4          /* If yes, decrement ea to
                               execute */
```

```
/* interrupted instruction */
```

SKIP EA DEC:

... the interrupt-service routine

END ISR:

```
    eret                  /* Return from exception */
```

.global start

```
._start:                /* Program start location */
```

... enable interrupts code

... the main program code

LOOP:

```
    br LOOP               /* Endless loop */
```

.end

Chú ý tới ý nghĩa những thanh ghi điều khiển: (xem Nios II Processor Reference Handbook)

- ❖ Thanh ghi ctl3: Trong Part I, JTAG UART được đặt tại interrupt level 0. Điều này có nghĩa là bit 0 của ctl3 phải được gán giá trị 1 để interrupt của JTAG UART hoạt động.
- ❖ Thanh ghi ctl0: Bit 0 của thanh ghi ctl0 được gán giá trị 1 cho phép external interrupt.

- ❖ Thanh ghi điều khiển của JTAG UART: RE = 1

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó).

- ❖ Vẽ lưu đồ giải thuật của interrupt-service routine tổng quát trên.

Yêu cầu Phần 4:

- ❖ Tạo 1 interrupt-service routine để đọc 1 kí tự từ JTAG UART.
Trong interrupt-service routine, sử dụng dạng polling để hiển thị kí tự nhận được từ host computer.
- ❖ Code tham khảo:

```

.include "nios_macros.s"

.equ UART_BASE,      0x9020
.equ TIMER,          0x9000
.equ STACK,          0x8000

.text
/*****************/
.global _start
_start:
    br      PROGRAM_START
/*****************/
.org 0x0020

ISR:
    subi  sp, sp, 24
    stw   et, 20(sp)

```

```

rdctl et, ctl4
    beq et, r0, SKIP_EA_DEC
    subi ea, ea, 4
SKIP_EA_DEC:
    stw ea, 0(sp)
    stw ra, 4(sp)
    stw fp, 8(sp)
    stw r21, 12(sp)
    stw r22, 16(sp)
    addi fp, sp, 24
rdctl et, ctl4
    bne et, r0, CHECK_LEVEL_0
NOT_EI:
    br END_ISR
CHECK_LEVEL_0:
    movi r21, 1
    and r22, et, r21
    beq r22, r0, CHECK_LEVEL_1
    call JTAG_UART_ISR
    br END_ISR
CHECK_LEVEL_1:
    br END_ISR
END_ISR:
    ldw ea, 0(sp)
    ldw ra, 4(sp)
    ldw fp, 8(sp)

```

ldw r21, 12(sp)

ldw r22, 16(sp)

ldw et, 20(sp)

addi sp, sp, 24

eret

PROGRAM_START:

movia sp, STACK

mov fp, sp

movia r8, UART_BASE

movi r23, 1

stw r23, 4(r8)

wrctl ctl3, r23

wrctl ctl0, r23

END:

br END

.end

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm → Tính vắng buổi đó).

❸ Viết 2 sub_routine: JTAG_UART_ISR và

JTAG_UART_PUT_CHAR

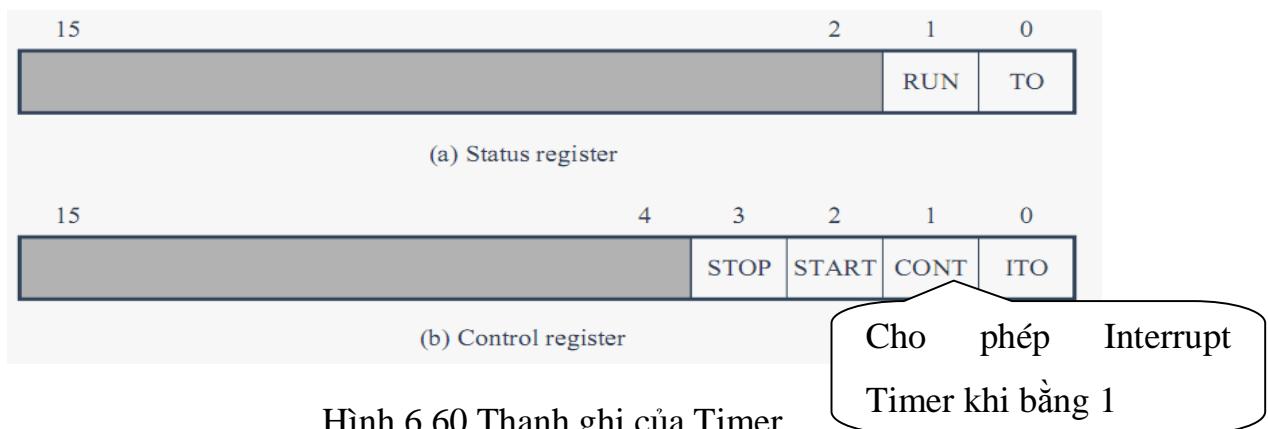
➤ **JTAG_UART_ISR** kiểm tra nếu UART có data mới và lưu data mới vào memory, sau đó gọi **JTAG_UART_PUT_CHAR**.

- **JTAG_UART_PUT_CHAR** sẽ lấy kí tự từ memory và hiển thị lên terminal window của Altera Debug Client.
- Compile, load và chạy chương trình.

6.3.4.5 Phân 5

Viết chương trình sử dụng interrupts để đọc kí tự nhận từ host computer và hiển thị kí tự nhận được sau 500 msec.

Trong Phân 2 ta đã sử dụng 1 delay loop để tạo khoảng thời gian xấp xỉ 500 msec. Phân 5, chúng ta sẽ sử dụng Interval Timer. Sau mỗi 500 msec, Interval Timer sẽ interrupt processor.



Hình 6.60 Thanh ghi của Timer

Tài liệu tham khảo Interval Timer: Chương 12 Altera Embedded Peripherals Hand-Book

Yêu cầu Phân 5:

Tham khảo đoạn code sau:

```

.include "nios_macros.s"

.equ  UART_BASE,      0x9020
.equ  TIMER,          0x9000
.equ  STACK,          0x8000

.text

.global _start
_start:
    br    PROGRAM_START

/*****************/
*/
.org  0x0020
ISR:
    subi sp, sp, 24
    stw  et, 20(sp)
    rdctl et, ctl4
    beq  et, r0, SKIP_EA_DEC
    subi ea, ea, 4

SKIP_EA_DEC:
    stw  ea, 0(sp)
    stw  ra, 4(sp)
    stw  fp, 8(sp)
    stw  r21, 12(sp)
    stw  r22, 16(sp)

```

```

addi  fp, sp, 24
rdctl et, ctl4
bne  et, r0, CHECK_LEVEL_0
NOT_EI:
br    END_ISR

```

CHECK_LEVEL_0:

```

movi  r21, 1
and   r22, et, r21
beq   r22, r0, CHECK_LEVEL_1
call  JTAG_UART_ISR
br    END_ISR

```

CHECK_LEVEL_1:

```

slli  r21, r21, 1
and   r22, et, r21
beq   r22, r0, CHECK_LEVEL_2
call  INTERVAL_TIMER_ISR
br    END_ISR

```

CHECK_LEVEL_2:

```

br      END_ISR
END_ISR:
ldw     ea, 0(sp)
ldw     ra, 4(sp)

```

```
ldw fp, 8(sp)
ldw r21, 12(sp)
ldw r22, 16(sp)
ldw et, 20(sp)
addi sp, sp, 24
```

eret

```
*****
```

```
/* INTERVAL_TIMER_ISR sub routine */
```

```
*****
```

INTERVAL_TIMER_ISR:

```
subi sp, sp, 12
stw ra, 0(sp)
stw fp, 4(sp)
stw r10, 8(sp)
addi fp, sp, 12
```

movia r10, TIMER

```
stw r0, 0(r10)
```

```
call JTAG_UART_PUT_CHAR
```

END_INTERVAL_TIMER_ISR:

```
ldw ra, 0(sp)
ldw fp, 4(sp)
ldw r10, 8(sp)
addi sp, sp, 12
```

ret

```
*****
```

PROGRAM_START:

```
    movia sp, STACK  
    mov fp, sp  
    movia r8, UART_BASE  
    movia r10, TIMER  
    movi r18, 1  
    stwio r18, 4(r8)  
    sthio r18, 4(r10)  
    movi r18, 3  
    wrctl ctl3, r18  
    movi r18, 1  
    wrctl ctl0, r18
```

END:

```
br END
```

THE_CHAR:

```
.ascii "Z"  
.end
```

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm → Tính vắng buổi đó).

- ─  Nêu ý nghĩa đoạn code trên.
- ─  Chú ý: sinh viên viết các sub_routine sau:
 - **JTAG_UART_ISR:** kiểm tra nếu UART có data mới và lưu data mới vào memory.
 - **JTAG_UART_PUT_CHAR:** lấy kí tự từ memory và hiển thị lên terminal window của Altera Debug Client.

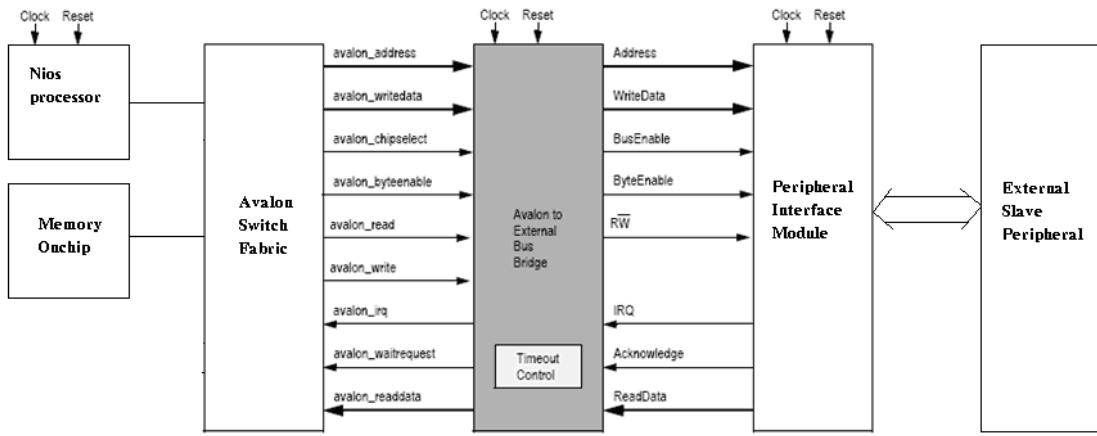
- Compile, load và chạy chương trình.
- Quan sát hiện tượng và giải thích, nêu ý nghĩa giải thuật chương trình.

6.3.5 Bài thực hành số 5 – Tìm hiểu cách thức giao tiếp Bus

Mục đích

Tìm hiểu về cách thức giao tiếp thông qua bus. Trong những thiết kế được tạo bởi SOPC-Builder, các thiết bị ngoại vi giao tiếp với Nios processor thông qua những Peripheral Interface Module (được tạo ra bởi SOPC-Builder) và Avalon Switch Fabric. Tuy nhiên, ta có thể sử dụng bus truyền/nhận dữ liệu giữa các thiết bị ngoại vi với Nios processor mà không cần dùng những Peripheral Interface Module được tạo bởi SPOC-Builder. Để thực hiện được điều này, ta sẽ sử dụng một module Avalon to External Bus Bridge. Với việc sử dụng module này, để giao tiếp giữa Nios processor với một External Slave Interface ta chỉ cần tạo một Peripheral Interface Module (by manual) đơn giản để thực hiện việc kết nối.

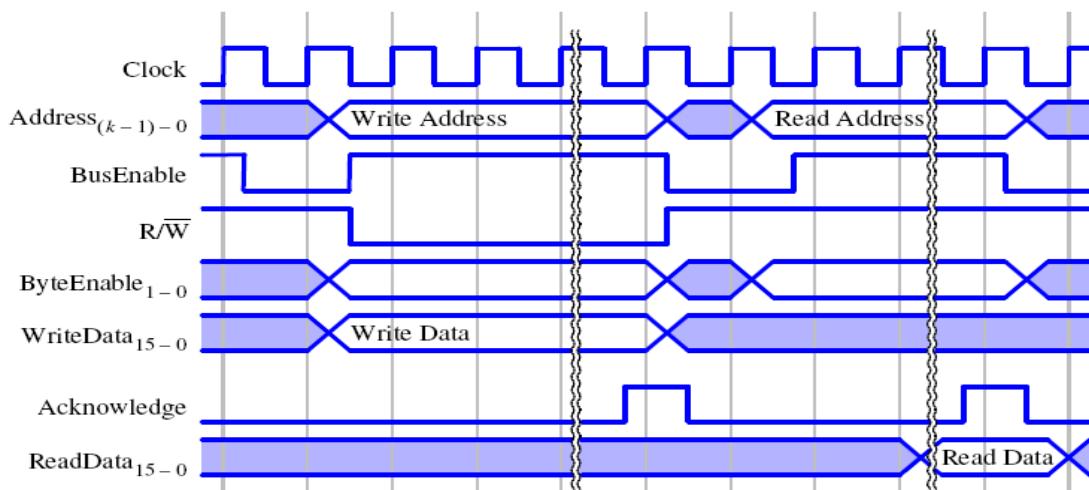
Để có thể tạo được một Peripheral Interface Module (by manual), trước hết ta phải hiểu ý nghĩa của những tín hiệu connect giữa “Avalon to External Bus Bridge” với “External Slave Peripheral”.



Hình 6.61 Mô hình Nios System dùng Avalon to External Bus Bridge

- ❖ Address : Địa chỉ của Data được đọc ra hoặc địa chỉ mà Data cần ghi vào trong “Virtual Memory” of Avalon to External Bus Bridge.
- ❖ WriteData : Data bus được ghi vào External Slave Peripheral từ “Memory of Avalon to External Bus Bridge” khi WR\ = “0”. Ta có thể chọn độ rộng của Data bus : 8bits , 16 bits, 32 bits, 64 bits, 128bits.
- ❖ BusEnable : Đây là cờ báo hiệu khi tất cả những signals khác đã valid, Data có thể được Read or Write.
- ❖ ByteEnable : Cho phép users chọn Bytes khi truy xuất Data.
- ❖ RW\ : Khi cờ này lên “1”, nó cho phép Read Data từ External Slave Peripheral vào “Virtual Memory” of Avalon to External Bus Bridge; khi cờ này có giá trị “0”, nó cho phép Write Data từ “Virtual Memory” of Avalon to External Bus Bridge ra External Slave Peripheral.
- ❖ IRQ : External Slave Peripheral phát ra tín hiệu yêu cầu ngắt đến Nios processor.

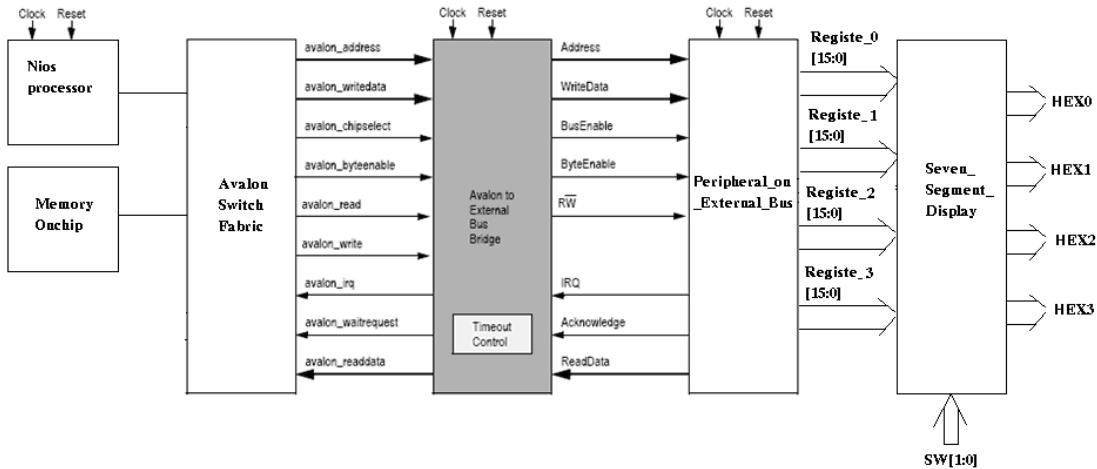
- ❖ Acknowledge : External Slave Peripheral báo cho Nios processor biết việc Read/Write Data đã được hoàn thành.
- ❖ ReadData : Data bus được đọc ra từ External Slave Peripheral để ghi vào “Virtual Memory” of Avalon to External Bus Bridge khi $RW\backslash = "1"$. Ta có thể chọn độ rộng của Data bus : 8bits , 16 bits, 32 bits, 64 bits, 128bits.



Hình 6.62 Giản đồ xung hoạt động của SRAM

6.3.5.1 Phần 1

Thiết kế một hệ thống như sau:

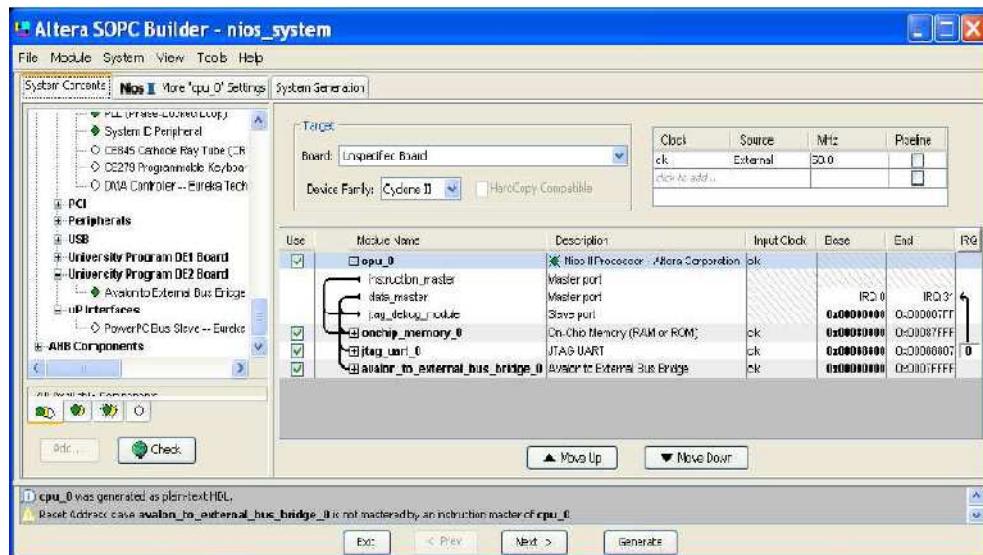


Hình 6.63 Giao tiếp LED 7 đoạn

Từng bước thực hiện:

- Bước 1. Tạo một project Quartus mới, đặt tên: .../lab5/Lab5_Part1
- Bước 2. Mở SOPC Builder tạo một Nios System với tên **nios_system** theo yêu cầu:
 - Nios II/s processor với JTAG Debug Module Level 1.
 - On-chip memory –RAM mode và size là 32 Kbytes.
 - JTAG_UART- use default setting.
 - Avalon to External Bus Bridge, để thêm module này cần thực hiện các bước sau:
 - i. Download module này từ: ftp://ftp.altera.com/up/pub/University_Program_IP_Cores/avalon_to_external_bus_bridge.zip), đặt trong thư mục chứa project.
 - ii. Trên SOPC Builder window, chọn File ➔ Refresh Component List...
 - iii. Chọn Avalon Components ➔ University Program DE2 Board ➔ Avalon to External Bus Bridge.
 - iv. Chọn Width data: 16 bits; Address range : 512 Kbytes.

- v. Connect Avalon to External Bus Bridge đến NiosII's data_master port, không connect đến NiosII's instruction_master.



Hình 6.64 Connection giữa các components

- Bước 3. Chọn System → Auto-Assign Base Addresses.
- Bước 4. Generate System.
- Bước 5. Add 3 modules Lab5_Part1, Peripheral_on_External_Bus, Seven_Segment_Display (gửi kèm theo file nội dung bài thực hành 5 này) vào project.
- Bước 6. Assign pins.
- Bước 7. Compile system.
- Bước 8. Program and Configure the Cyclone II FPGA.
- Bước 9. Viết một chương trình assemble thực hiện việc ghi 4 giá trị : 0x0123; 0x4567; 0x89ab ; 0xcdef vào “Virtual Memory” of Avalon to External Bus Bridge với địa chỉ đầu tiên là 0x0000.

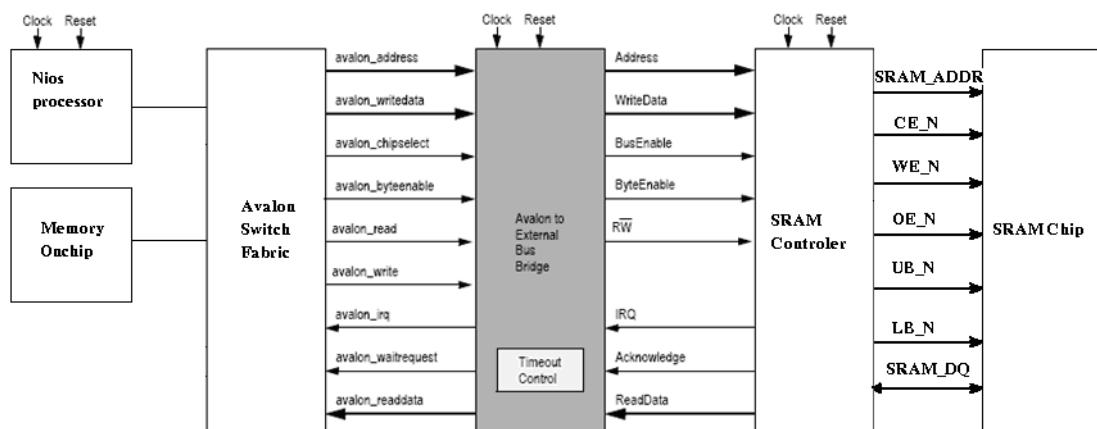
Bước 10. Dùng Altera Monitor Program download và thực thi chương trình.

Chú ý: Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

- ✚ Tìm hiểu và giải thích hoạt động của 3 modules (gửi kèm theo file nội dung bài thực hành 5 này).
- ✚ Chuẩn bị trước ở nhà chương trình assemble thực hiện việc ghi 4 giá trị : 0x0123; 0x4567; 0x89ab ; 0xcdef vào “Virtual Memory” of Avalon to External Bus Bridge với địa chỉ đầu tiên là 0x0000.

6.3.5.2 Phần 2:

Thiết kế một hệ thống như sau:



Hình 6.65 Giao tiếp với RAM

- Bước 1. Tạo một project Quartus mới, đặt tên: .../lab5/Lab5_Part2
- Bước 2. Làm giống Phần 1

Bước 3. Add 2 modules Lab5_Part2, SRAM_Controller (gửi kèm theo file nội dung bài thực hành 5 này) vào project.

Bước 4. Viết một chương trình Assemble thực hiện ghi 4 giá trị : 0x0123; 0x4567; 0x89ab ; 0xcdef vào SRAM với địa chỉ đầu tiên là 0x0000, sau đó đọc ngược trở lại 4 giá trị đó vào những thanh ghi của processor.

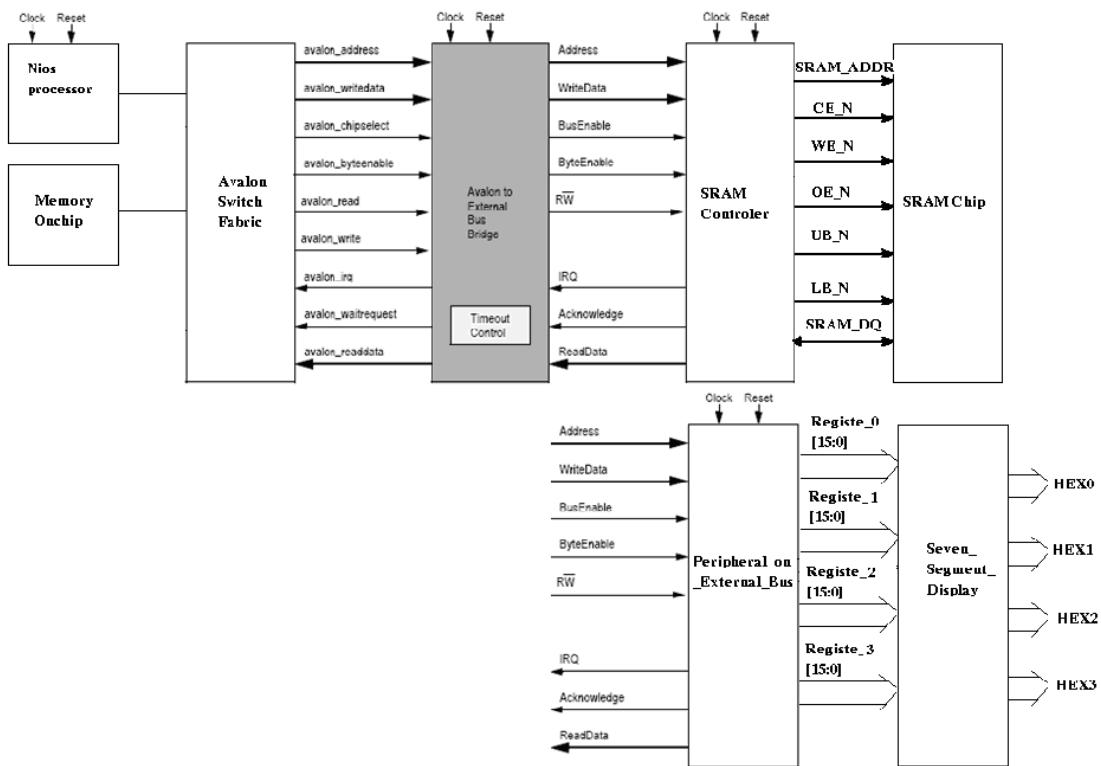
Bước 5. Dùng Altera Monitor Program download và thực thi chương trình.

Note : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

- TÌM HIỂU VÀ GIẢI THÍCH HOẠT ĐỘNG CỦA 2 MODULES Lab5_Part2, SRAM_Controller (gửi kèm theo nội dung bài thực hành 5 này).
- CHUẨN BỊ TRƯỚC Ở NHÀ CHƯƠNG TRÌNH ASSEMBLE THỰC HIỆN VIỆC GHI 4 GIÁ TRỊ : 0x0123; 0x4567; 0x89ab ; 0xcdef VÀO SRAM VỚI ĐỊA CHỈ ĐẦU TIÊN LÀ 0x0000 , SAU ĐÓ ĐỌC NGƯỢC TRỞ LẠI 4 GIÁ TRỊ ĐÓ VÀO NHỮNG THANH GHI CỦA PROCESSOR.

6.3.5.3 Phản 3:

Thiết kế một hệ thống gồm hai External Slave Peripheral nhưng chỉ sử dụng một “Avalon to External Bus Bridge” như sau:



Hình 6.66 Giao tiếp với RAM và LED 7 đoạn

- Bước 1. Tạo một project Quartus mới, đặt tên: .../lab5/Lab5_Part3
- Bước 2. Làm giống Phần 1
- Bước 3. Sửa 5 modules đã sử dụng trong Part1 và Part2 để thỏa mãn hệ thống như trên (gửi kèm theo nội dung bài thực hành 5 này) vào project.
- Bước 4. Viết một chương trình Assemble thực hiện ghi 4 giá trị : 0x0123; 0x4567; 0x89ab ; 0xcdef vào SRAM với địa chỉ đầu tiên là 0x0000, sau đó đọc 4 giá trị đó vào những Seven Segment Display.
- Bước 5. Dùng Altera Monitor Program download và thực thi chương trình.

Chú ý : Sinh viên cần chuẩn bị trước ở nhà những công việc sau (Không có bài chuẩn bị không được vào lớp làm thí nghiệm ➔ Tính vắng buổi đó)

- Sửa 5 modules đã sử dụng trong Phần 1 và Phần 2 để thỏa mãn hệ thống như trên (gửi kèm theo nội dung bài thực hành 5 này) vào project.
- Chuẩn bị trước ở nhà chương trình assemble thực hiện việc ghi 4 giá trị : 0x0123; 0x4567; 0x89ab ; 0xcdef vào SRAM với địa chỉ đầu tiên là 0x0000 , sau đó đọc 4 giá trị đó vào những Seven Segment Display.

Chương 7. Mô phỏng mô tả thiết kế bằng ModelSim

7.1 Giới thiệu

Phần này sẽ trình bày các bước để chạy mô phỏng pre-synthesis và post-synthesis cho một mô tả thiết kế Verilog sử dụng công cụ ModelSim.

Thông qua phần này, sinh viên sẽ hiểu sự khác nhau giữa hai kiểu mô phỏng trên cũng như lí do khi nào phải chạy mô phỏng pre-synthesis hoặc post-synthesis.

Điểm khác nhau cơ bản giữa chạy mô phỏng pre-synthesis và post-synthesis đó:

- Khi chạy mô phỏng pre-synthesis thì delay timing của thiết kế sẽ không được phân tích trong quá trình chạy mô phỏng và do đó delay timing giữa các node bên trong thiết kế sẽ bằng 0. Vì vậy thời gian chạy mô phỏng loại này sẽ tốn ít thời gian, hiệu quả trong việc kiểm tra và debug về chức năng (**function**) của thiết kế.
- Khi chạy mô phỏng post-synthesis thì delay timing của thiết kế sẽ được lấy sau quá trình synthesis và được dùng để phân tích trong quá trình chạy mô phỏng và do đó delay timing giữa các node bên trong thiết kế sẽ được lấy từ file SDF (Standard Delay Format). Do phải phân tích delay timing cũng như Verilog netlist sau quá trình synthesis lớn nên thời gian chạy mô phỏng loại này sẽ tốn rất nhiều thời gian, đặc biệt trong những thiết kế hệ thống lớn, thời gian chạy mô phỏng có thể lên đến hàng giờ đồng hồ nên sẽ không hiệu quả trong việc kiểm tra chức năng (function) của thiết kế, nhưng nó sẽ cung cấp thông tin khá chính xác về định thời cho thiết kế, điều này sẽ giúp người thiết kế trong quá trình kiểm tra và debug về **timing** của thiết kế.

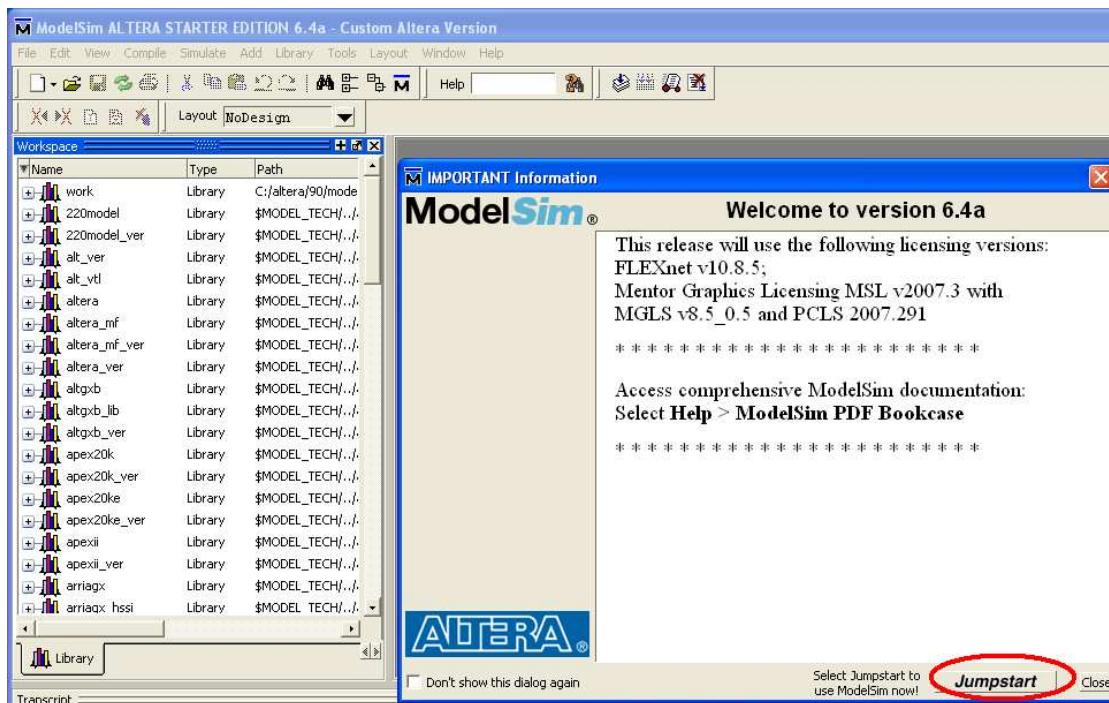
Hai ưu điểm của chạy mô phỏng dùng ModelSim so với chạy mô phỏng trên Quartus đó là:

- Không thể chạy mô phỏng pre-synthesis trên Quartus.
- Không thể chạy mô phỏng dùng testbench trên Quartus. Hạn chế này có thể không nhận ra khi tạo input cho một thiết kế đơn giản với ít inputs với dạng sóng đơn giản. Tuy nhiên, đối với một thiết kế hệ thống lớn thì mô phỏng dùng Quartus là hoàn toàn không khả thi vì ta không thể tạo dạng sóng “manually” như trên Quartus, mà ta phải viết testbench để tạo dạng sóng cho inputs.

7.2 Mô phỏng pre-synthesis

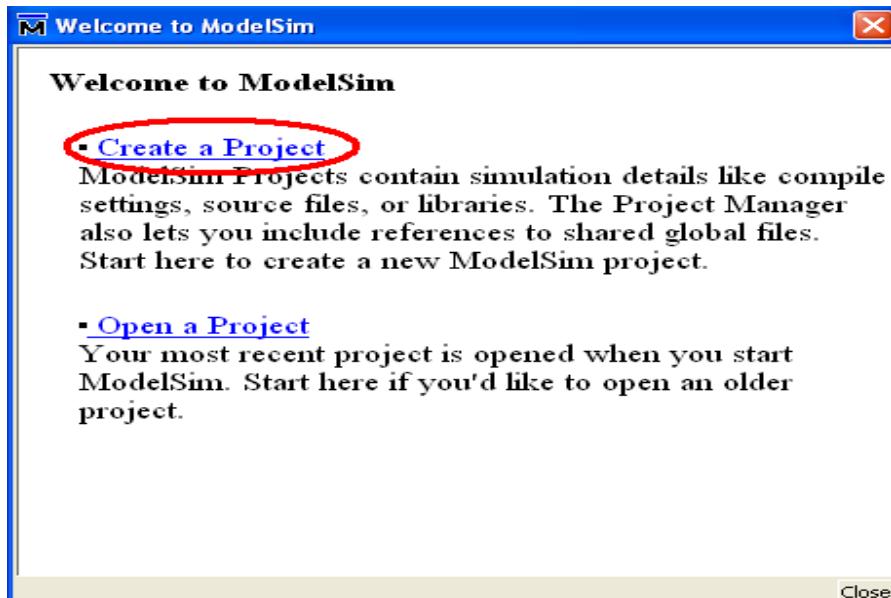
Để thực hiện mô phỏng cho một thiết kế trước khi synthesis, ta thực hiện các bước sau.

Bước 1. Mở phần mềm ModelSim



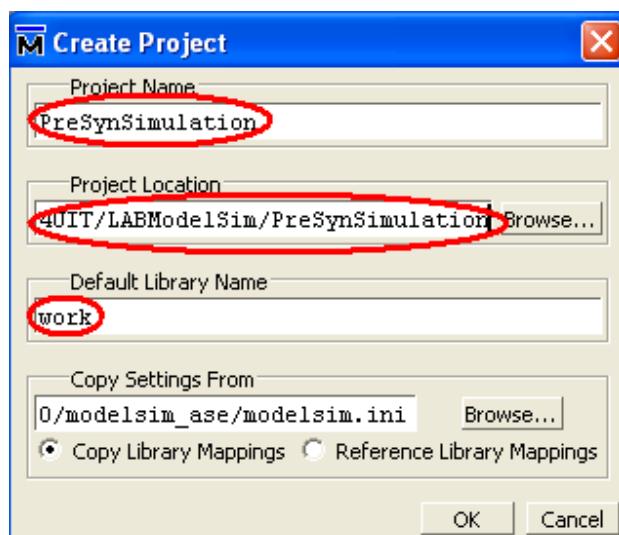
Hình 7.1 Cửa sổ ModelSim

Bước 2. Nhấn Jumpstart để bắt đầu làm việc với ModelSim



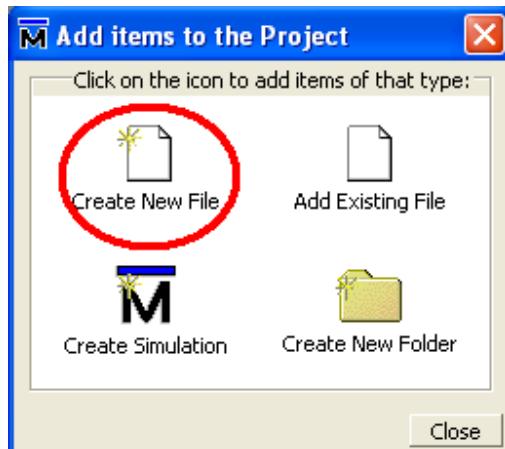
Hình 7.2 Cửa sổ tạo project mới

Bước 3. Điền thông tin về tên project, đường dẫn cũng như tên Library cho project mới. Ta có thể đặt tên cho project cũng như đường dẫn của project tùy ý.



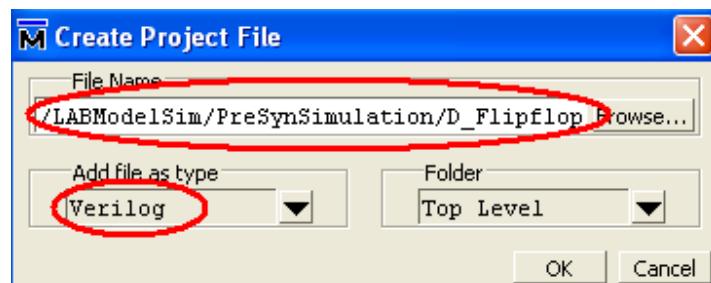
Hình 7.3 Cửa sổ điền thông tin cho project

Bước 4. Nhấn OK, một cửa sổ mới hiện ra, chọn **Create New File**



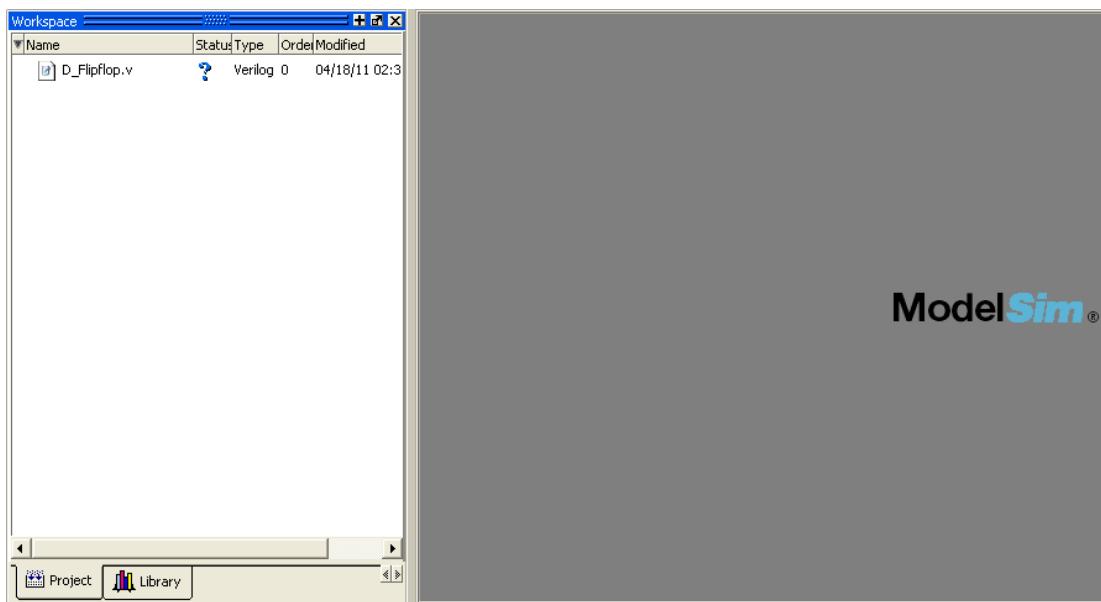
Hình 7.4 Tạo new file

Bước 5. Nhập đường dẫn và tên cho file mới cần tạo, giả sử ở đây ta đang muốn thiết kế một D-Flipflop, ta đặt tên file là **ModelSimTest.v**, và đặt file này vào trong thư mục **vừa tạo project**



Hình 7.5 Điền thông tin cho new file

Bước 6. Nhấn OK, quay trở về cửa sổ trong bước 4, nhấn Close



Hình 7.6 Cửa sổ sau khi tạo project

Bước 7. Double click vào D_Flipflop.v bên màn hình Workspace, một cửa sổ dùng cho việc mô tả thiết kế dùng Verilog xuất hiện

A screenshot of a Verilog editor window. The title bar shows the path: D:/UIT/Verilog/Verilog_LAB/DE2_labs_verilog/LAB4UIT/LABModelSim/PreSynSimulation/D_Flipflop.v. The editor area has a column labeled 'Ln#' with two lines of code: '1' and '2'. At the bottom, the status bar shows 'D_Flipflop.v'.

```
D:/UIT/Verilog/Verilog_LAB/DE2_labs_verilog/LAB4UIT/LABModelSim/PreSynSimulation/D_Flipflop.v
Ln# | 1
2
```

Hình 7.7 Cửa sổ dùng cho mô tả thiết kế

Bước 8. Dùng ngôn ngữ Verilog mô tả thiết kế, ở đây ta đang thực hiện ví dụ là mô tả thiết kế một D-Flipflop

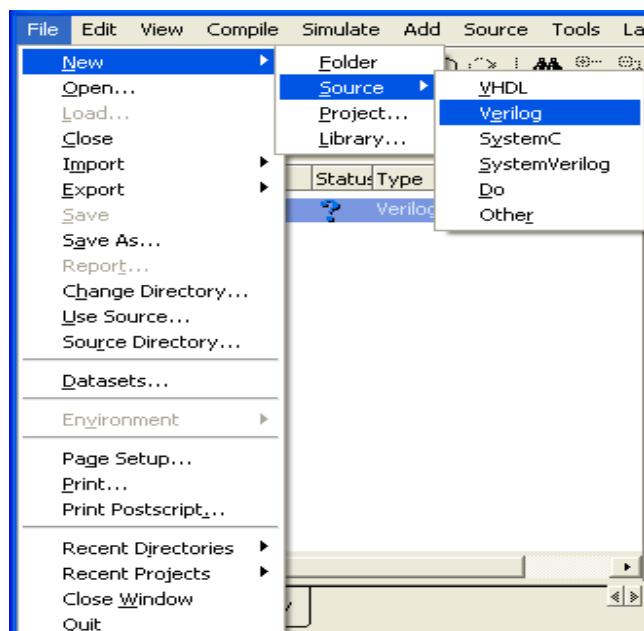
```

Ln# D:/Uit/Verilog/Verilog_LAB/DE2_labs_verilog/LAB4UIT/LABModelSim/PreSynSimulation/D_Flipflop.v
1 module ModelSimTest (Din, Clk, Resetn, Qout); // D-Flipflop
2   input Din;
3   input Clk;
4   input Resetn;
5   output Qout;
6   reg Qout;
7   always @ (posedge Clk or negedge Resetn)
8   begin
9     if (!Resetn)
10       Qout = 1'b0;
11     else
12       Qout = Din;
13   end
14 endmodule
15

```

Hình 7.8 Mô tả thiết kế D-Flipflop

Bước 9. Tạo Testbench để định nghĩa dạng sóng cho các tín hiệu input của thiết kế. Ta mở tiếp một cửa sổ khác cũng dùng Verilog để thiết kế bằng cách chọn tab File → New → Source → Verilog



Hình 7.9 Tạo new file

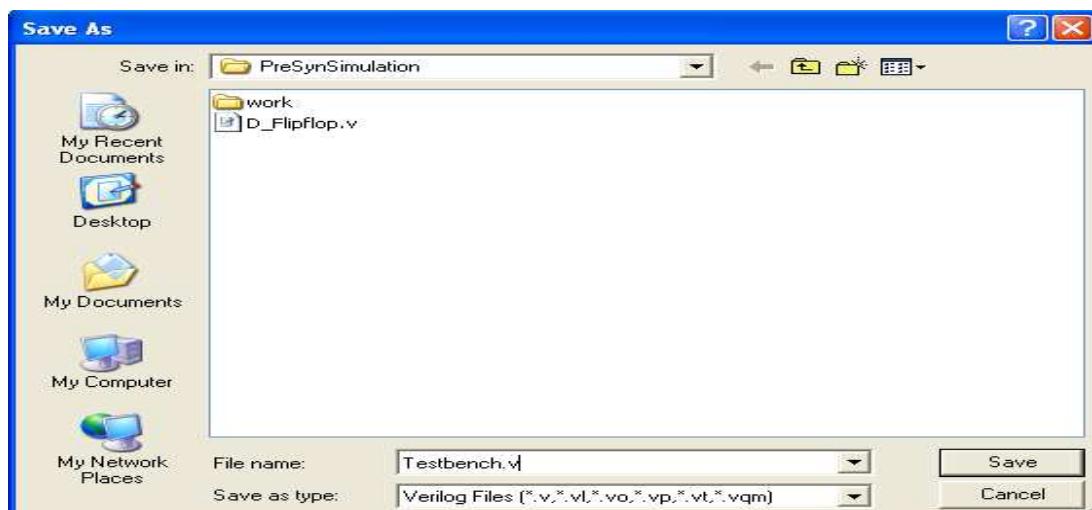
Bước 10. Một cửa sổ mới dùng Verilog để mô tả thiết kế khác xuất hiện, ta mô tả testbench cho thiết kế như sau

```
[h] D:/Ult/Verilog/Verilog_LAB/DE2_labs_verilog/LAB4UIT/LABModelSim/PreSynSimulation/Untitled-1.v *
```

Ln#	
1	'timescale 1ns/1ps
2	module Testbench();
3	reg CLK, RST_N, DATA_IN;
4	wire QOUT;
5	//Create the value for the input signals
6	initial begin
7	CLK = 0;
8	RST_N = 0;
9	DATA_IN = 0;
10	#205;
11	RST_N = 1'b1;
12	#500;
13	DATA_IN = 1'b1;
14	#500;
15	DATA_IN = 1'b0;
16	#2000 \$finish;
17	end
18	// Create the clock with a period of 10ns
19	always @ (CLK)
20	#10 CLK <= ~CLK;
21	//Instantiate the module named ModelSimTest (D-Flipflop) which described in D_Flipflop.v
22	ModelSimTest ins1 (
23	.Clk (CLK),
24	.Resetn (RST_N),
25	.Din (DATA_IN),
26	.Qout (QOUT)
27);
28	endmodule

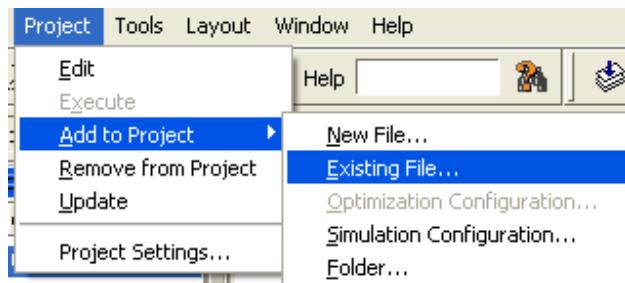
Hình 7.10 Mô tả Testbench cho thiết kế

Bước 11. Lưu testbench cho thiết kế với tên Testbench.v



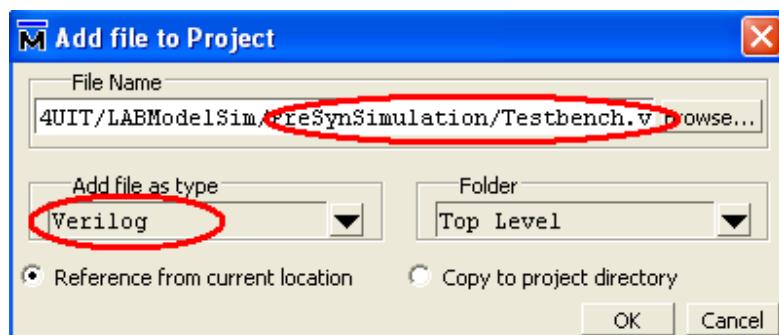
Hình 7.11 Lưu mô tả testbench

Bước 12. Add file Testbench.v vừa tạo vào project, chọn tab Project ➔ Add to Project ➔ Existing File



Hình 7.12 Add file Testbench vào project

Bước 13. Chỉ đường dẫn đến file Testbench



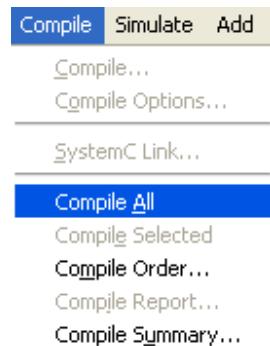
Hình 7.13 Chỉ đường dẫn đến file Testbench

Bước 14. Cửa sổ Workspace xuất hiện như sau

Name	Status	Type	Order	Modified
D_Flipflop.v	?	Verilog	0	04/18/11 02:3
Testbench.v	?	Verilog	1	04/18/11 02:5

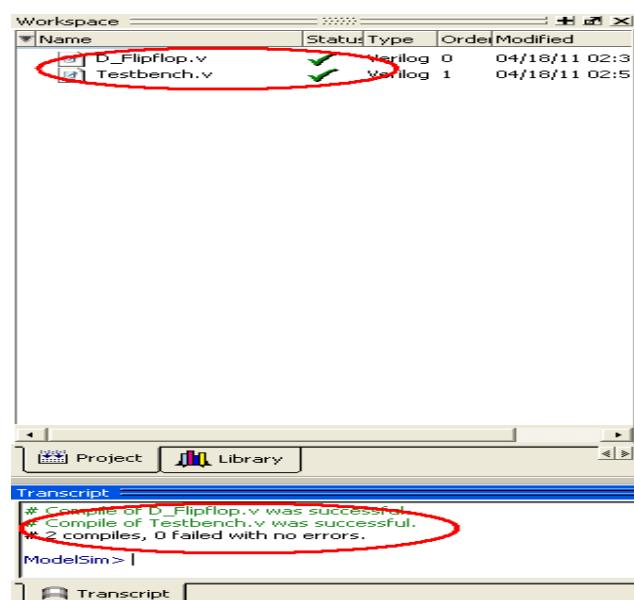
Hình 7.14 Cửa sổ Workspace sau thiết kế

Bước 15. Compile mô tả thiết kế, chọn tab Compile ➔ Compile All



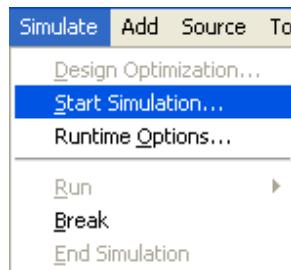
Hình 7.15 Compile thiết kế

Bước 16. Nếu mô tả thiết kế không có lỗi thì sẽ có thông báo sau



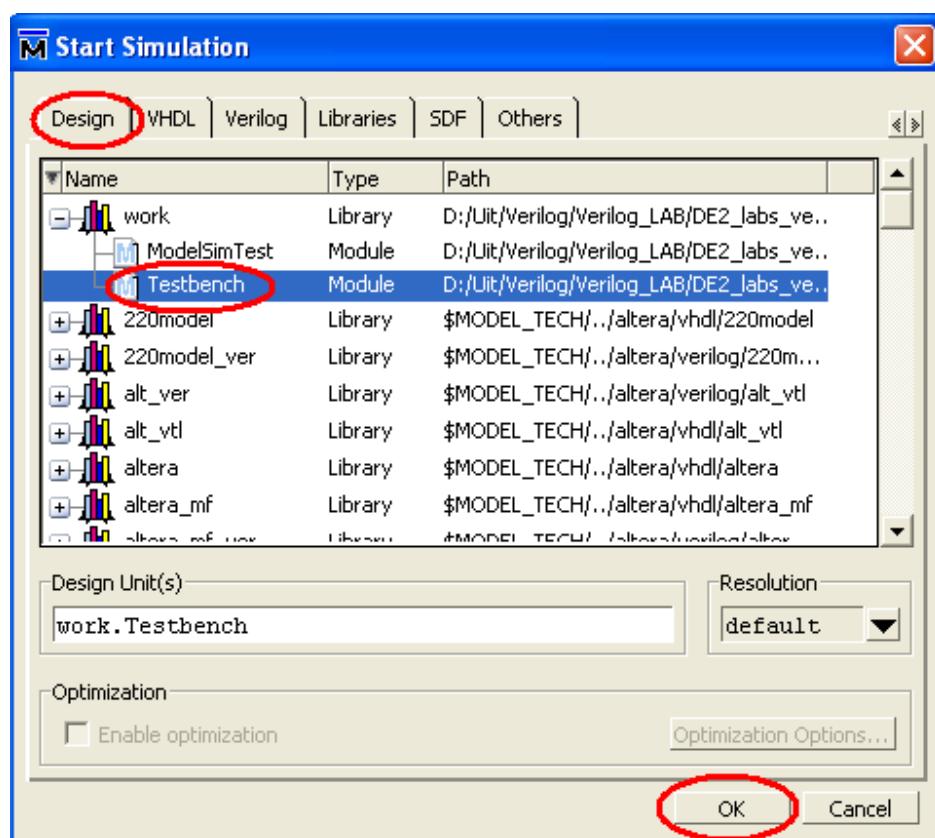
Hình 7.16 Compile thành công

Bước 17. Sau khi ta mô tả thiết kế và mô tả testbench cho thiết kế thành công ta sẽ thực hiện các bước chạy mô phỏng, chọn tab Simulate → Start Simulation



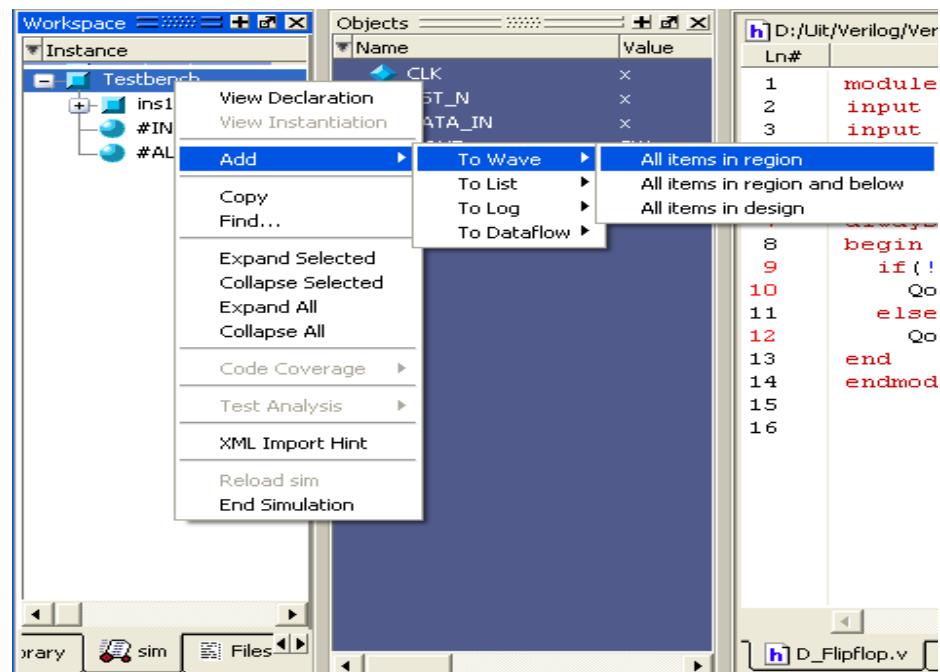
Hình 7.17 Thiết lập mô phỏng

Bước 18. Một cửa sổ như hình dưới xuất hiện, ta chọn tab Design → work → Testbench. Nhấn OK.



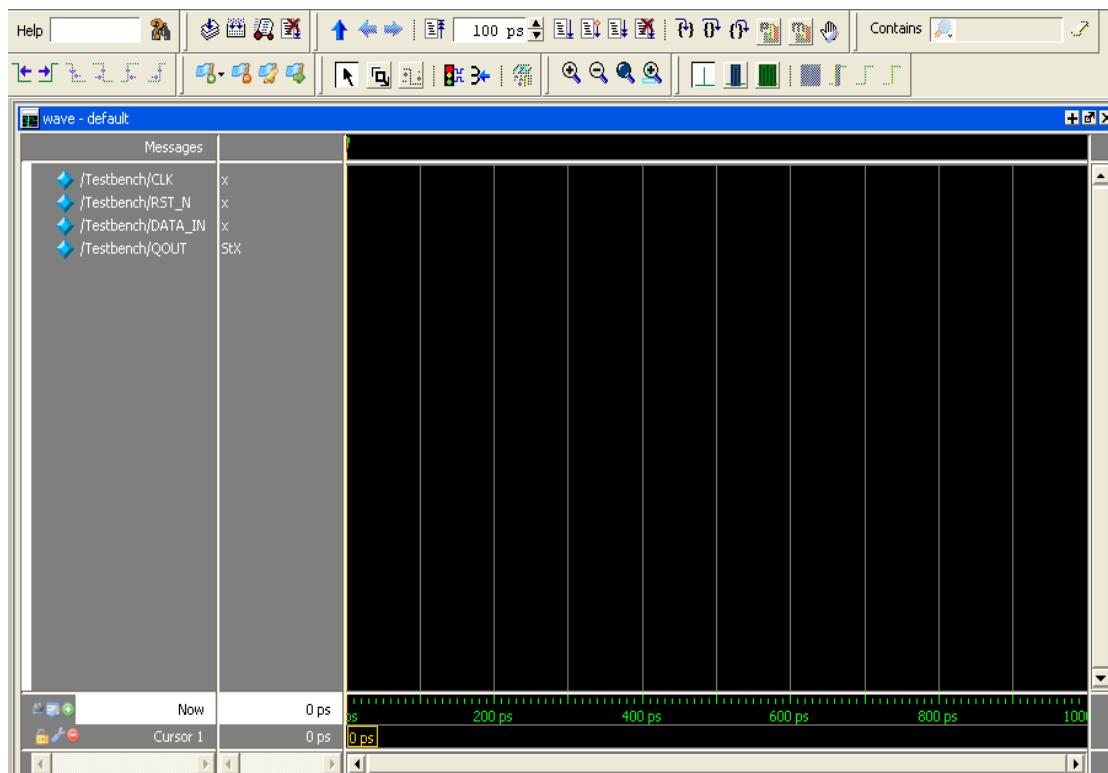
Hình 7.18 Chọn thiết kế cần mô phỏng

Bước 19. Một cửa sổ như sau xuất hiện, Click phải chuột lên Testbench → Add → To Wave → All items in region



Hình 7.19 Chọn tín hiệu dạng sóng cần quan sát

Bước 20. Cửa sổ dạng sóng được mở ra



Hình 7.20 Cửa sổ dạng sóng

Bước 21. Chọn khoảng thời gian chạy mô phỏng



Hình 7.21 Thiết lập thời gian chạy mô phỏng

Bước 22. Nhấn nút chạy mô phỏng



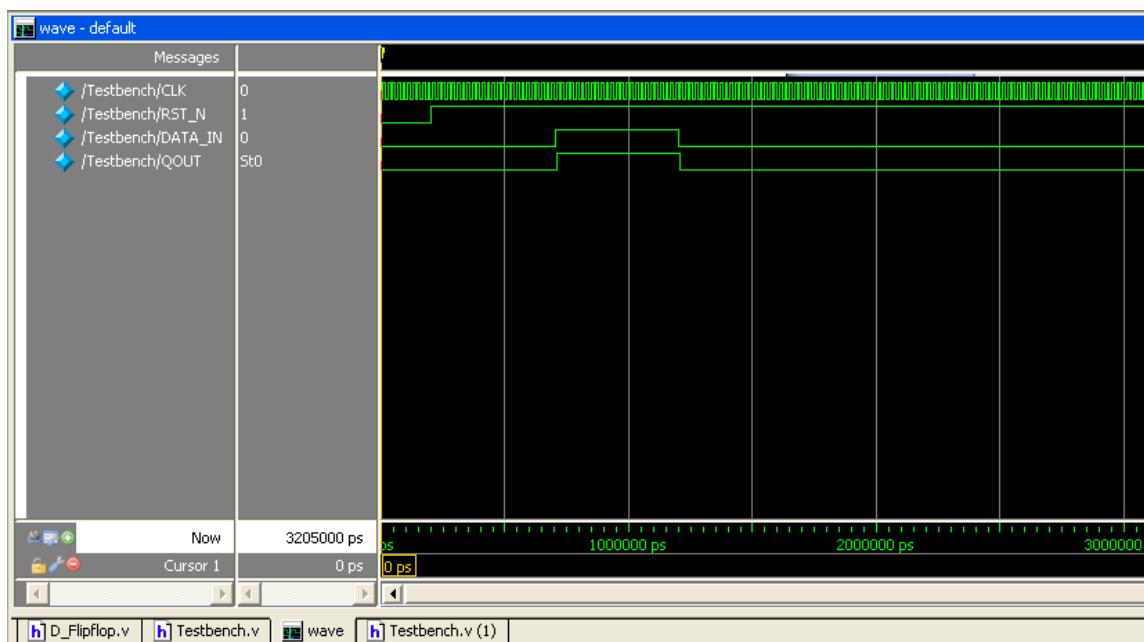
Hình 7.22 Chạy mô phỏng

Bước 23. Sau khi chạy xong, một thông báo xuất hiện



Hình 7.23 Nhấn "No"

Bước 24. Chọn “No”, dạng sóng sau khi mô phỏng xuất hiện, nhấn nút “Zoom Full” để thấy toàn bộ dạng sóng



Hình 7.24 Dạng sóng sau mô phỏng

Bước 25. Quan sát ta thấy, tín hiệu ngõ ra không có delay timing so với các tín hiệu ngõ vào. Bởi vì đây là mô phỏng pre-synthesis nên trong quá trình chạy mô phỏng, thông tin về định thời cho các node bên trong thiết kế không được cung cấp, do đó delay timing giữa các node này là bằng 0.

Bước 26. Quan sát dạng sóng và debug function nếu có sai.

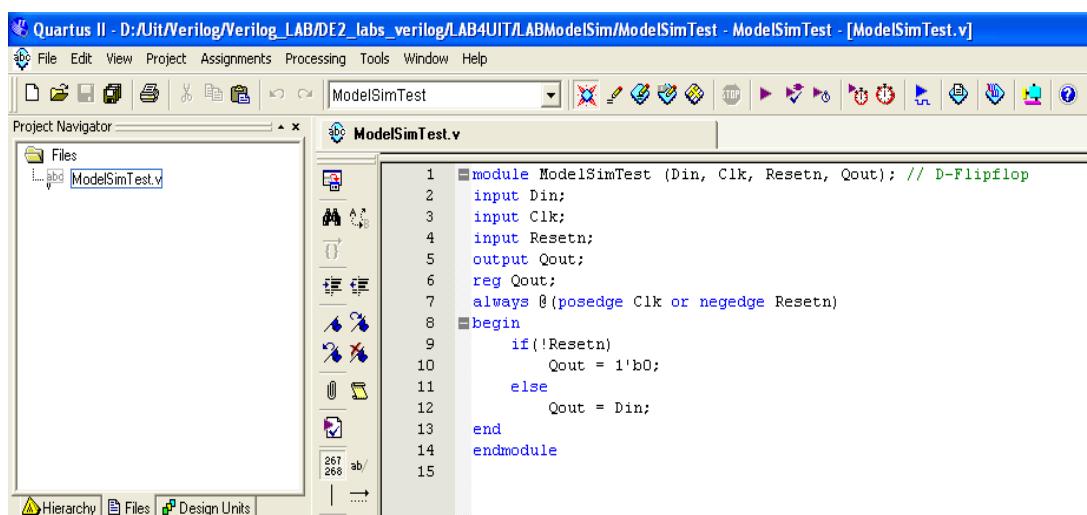
7.3 Mô phỏng post-synthesis

Để thực hiện mô phỏng post-synthesis một mô tả thiết kế dùng verilog, ta thực hiện gồm hai quá trình, để dễ hiểu ta sẽ thực hiện các bước với một mô tả thiết kế một mạch D-flipflop.

7.3.1 Dùng Quartus tạo Verilog netlist cho việc mô phỏng post-synthesis

Bước 1. Tạo Quartus project, tên project được đặt là : ModelSimTest
(phần này đã được trình bày chi tiết trong phần 5.1.1)

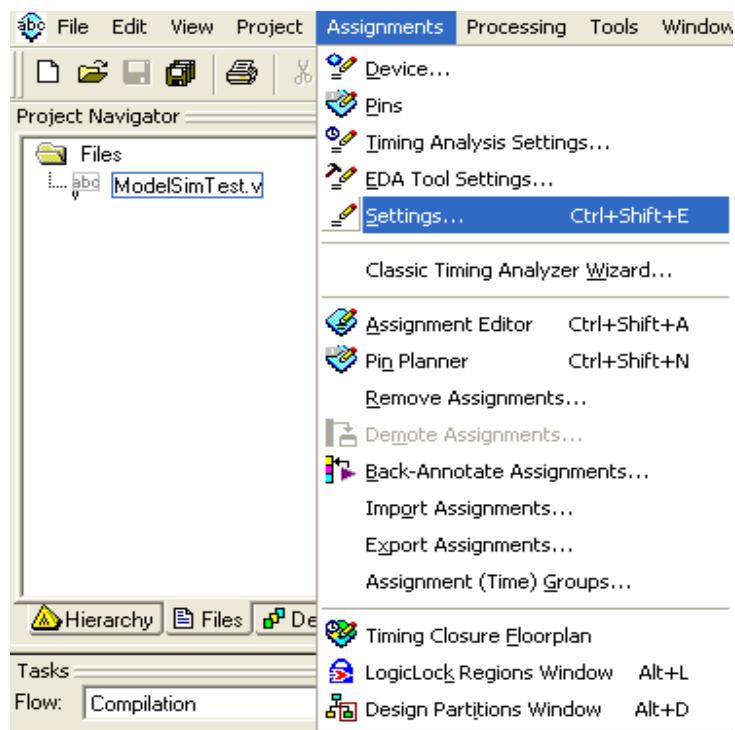
Bước 2. Sử dụng Verilog mô tả thiết kế module D-flipflop như sau:
(phần này đã được trình bày chi tiết trong phần 5.1.2)



```
1 module ModelSimTest (Din, Clk, Resetn, Qout); // D-Flipflop
2   input Din;
3   input Clk;
4   input Resetn;
5   output Qout;
6   reg Qout;
7   always @ (posedge Clk or negedge Resetn)
8     begin
9       if (!Resetn)
10         Qout = 1'b0;
11       else
12         Qout = Din;
13     end
14   endmodule
15 
```

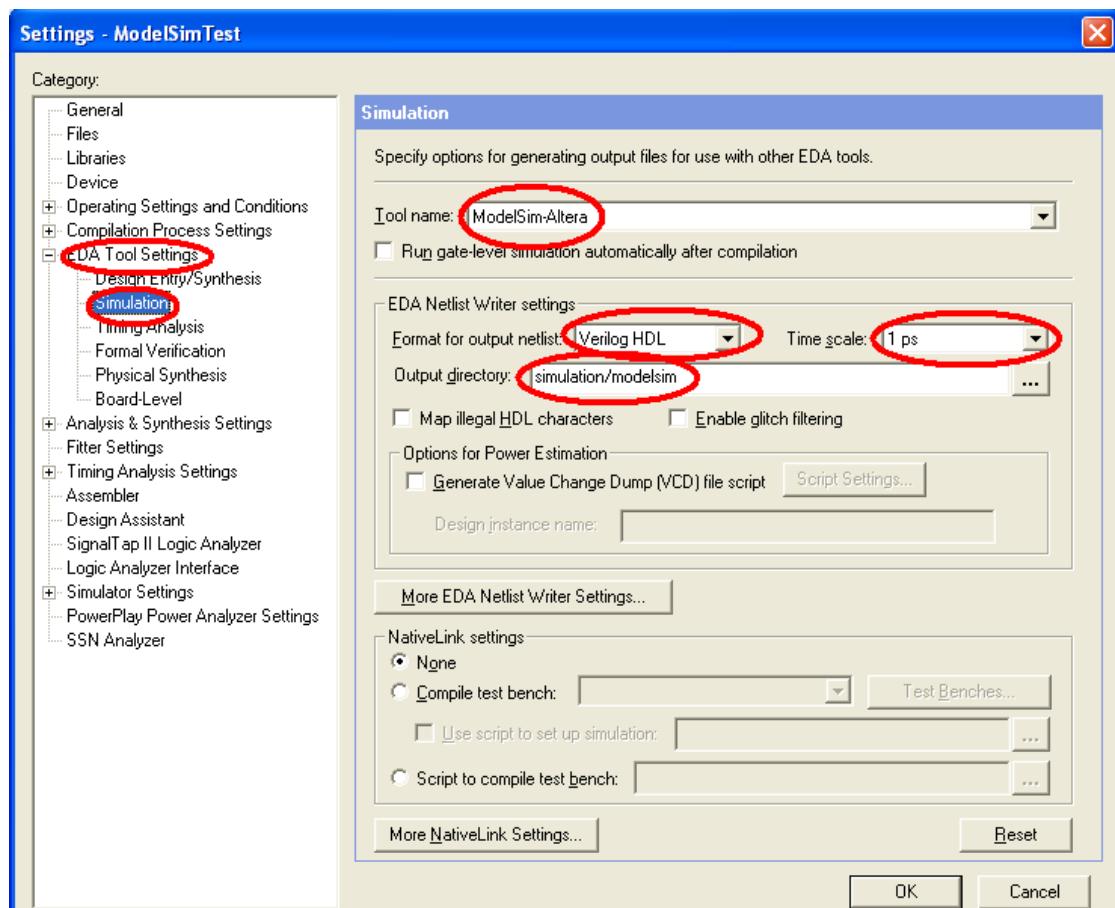
Hình 7.25 Mô tả thiết kế D-Flipflop

Bước 3. Thiết lập thông số cho quá trình synthesis và compilation



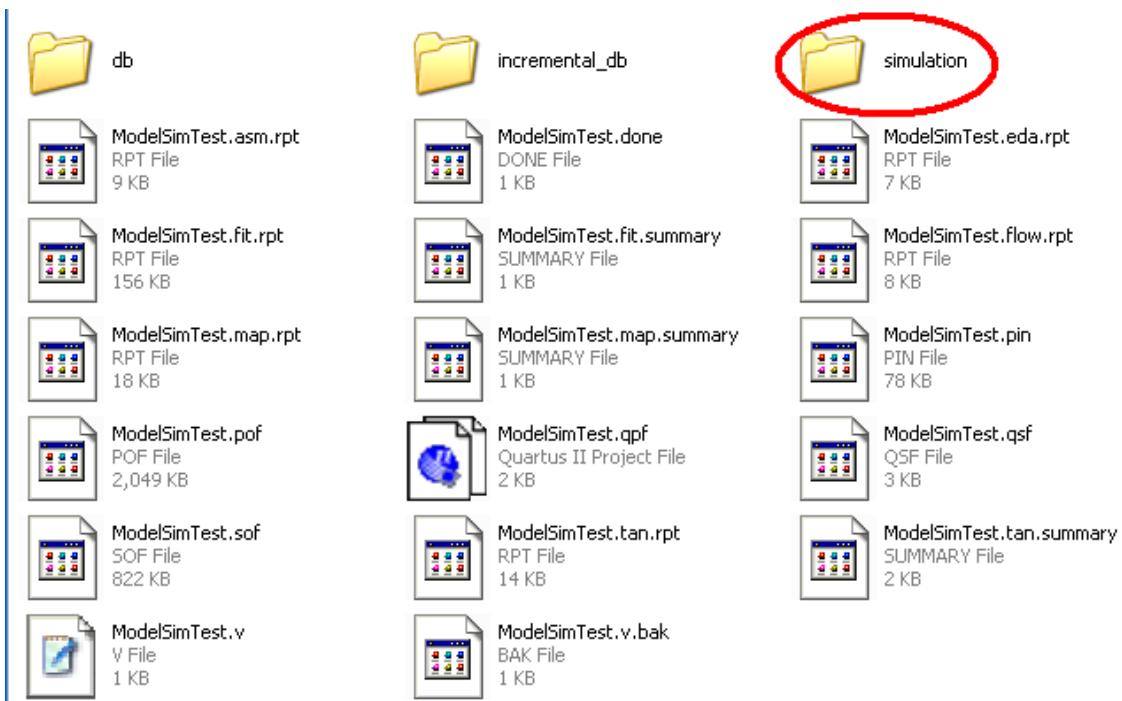
Hình 7.26 Thiết lập thông số cho quá trình synthesis

Bước 4. Chọn **EDA Tool Settings** → **Simulation**, chọn các option và điền thông tin như sau



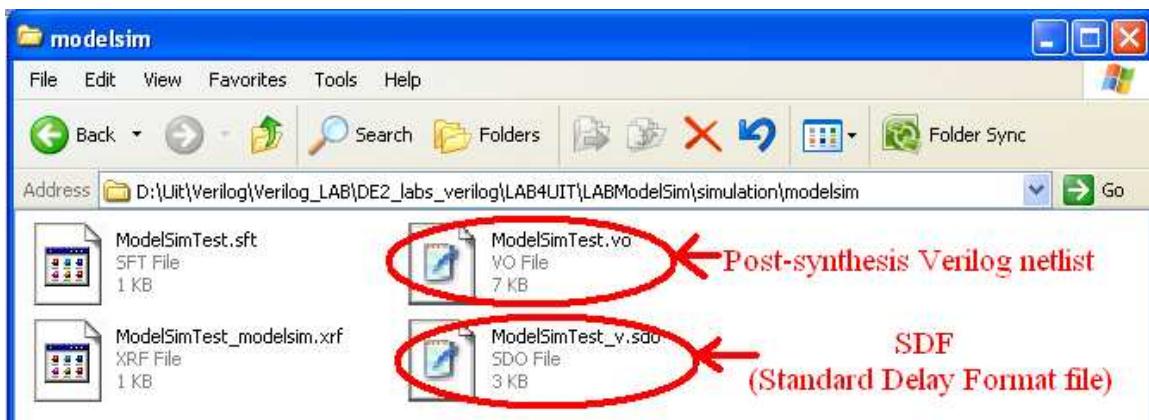
Hình 7.27 Thiết lập thông số

Bước 5. Biên dịch (compilation) thiết kế, một thư mục simulation/modelsim (vì ta đã chỉ đường dẫn cho **Output directory** trong bước 4) được tạo ra ngay tại thư mục chứa project



Hình 7.28 Thư mục sau quá trình synthesis

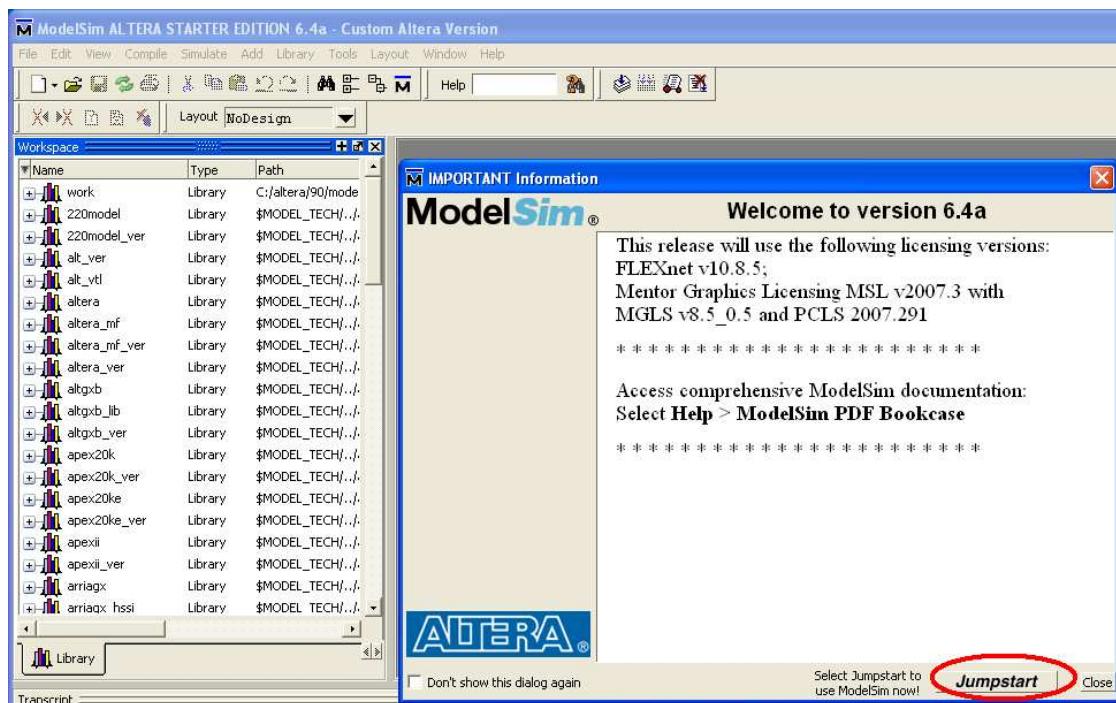
Bước 6. Trong thư mục này ta thấy hai file quan trọng sẽ sử dụng cho việc chạy mô phỏng post-synthesis được tạo ra



Hình 7.29 Hai file quan trọng được tạo ra

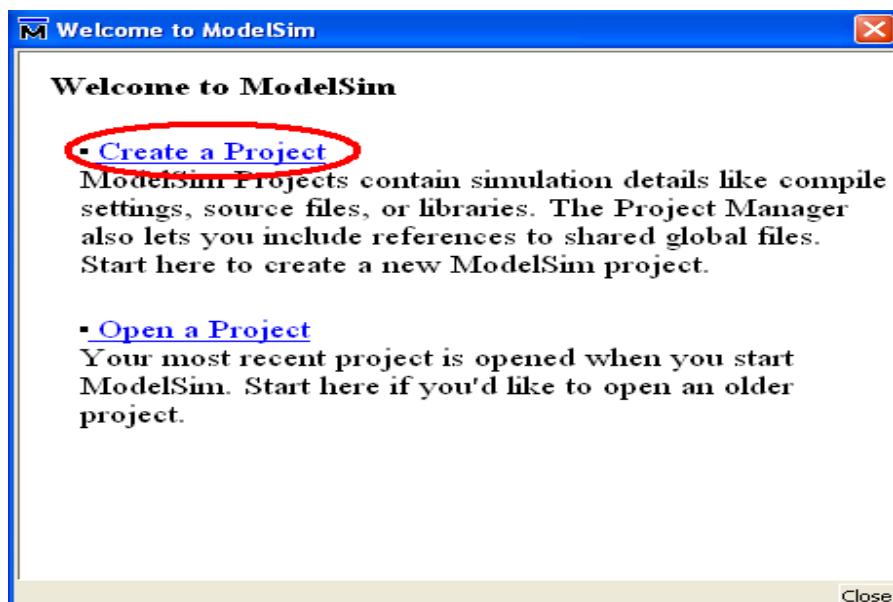
7.3.2 Dùng ModelSim để chạy mô phỏng post-synthesis

Bước 1. Mở phần mềm ModelSim



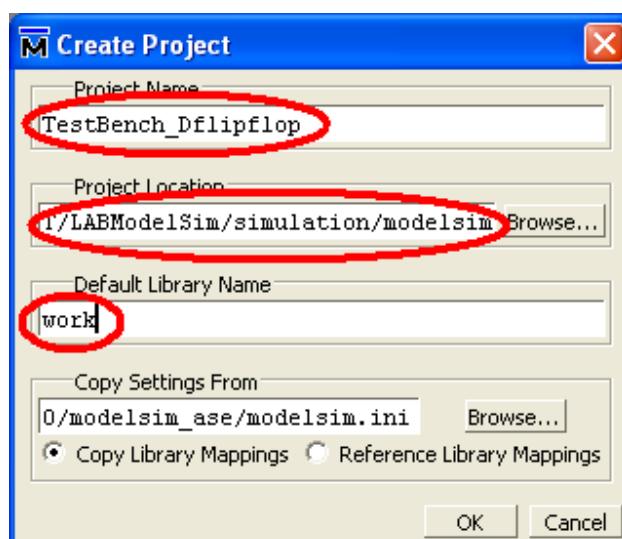
Hình 7.30 Cửa sổ ModelSim

Bước 2. Nhấn Jumpstart để bắt đầu làm việc với ModelSim



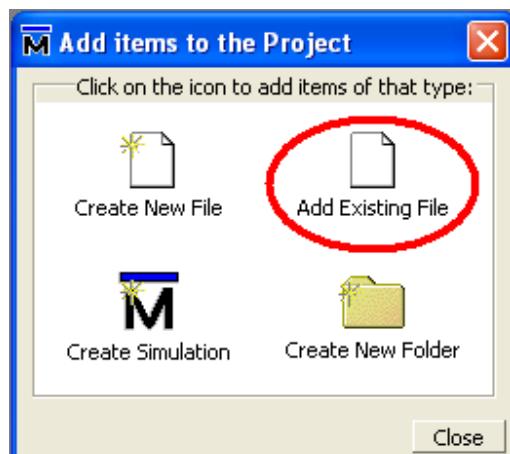
Hình 7.31 Tạo project mới

Bước 3. Điền thông tin về tên project, đường dẫn cũng như tên Library cho project mới, ở đây ta cần chú ý vì ta đã tạo file netlist.vo và file SDF .sdo trong thư mục **simulation/modelsim** như đã trình bày trong phần 7.3.1, nên để cho thuận tiện ta chỉ đường dẫn cho project ModelSim mới này nằm trong thư mục **simulation/modelsim** luôn, tuy nhiên ta có thể tạo đường dẫn bất kì cho project mới này, sau đó ta chỉ cần copy hai file netlist.vo và file SDF .sdo vào thư mục vừa tạo là được.



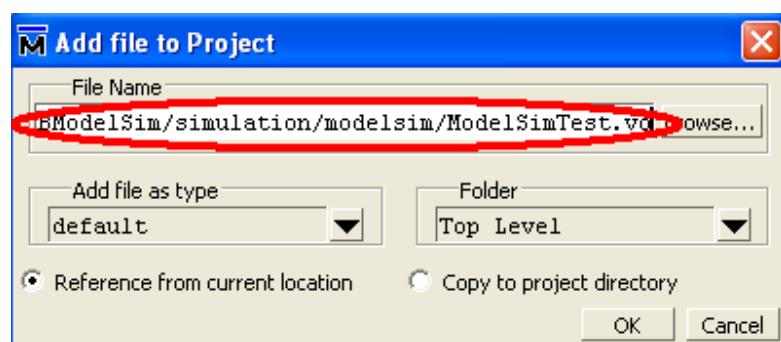
Hình 7.32 Điền thông tin cho project mới

Bước 4. Nhấn OK



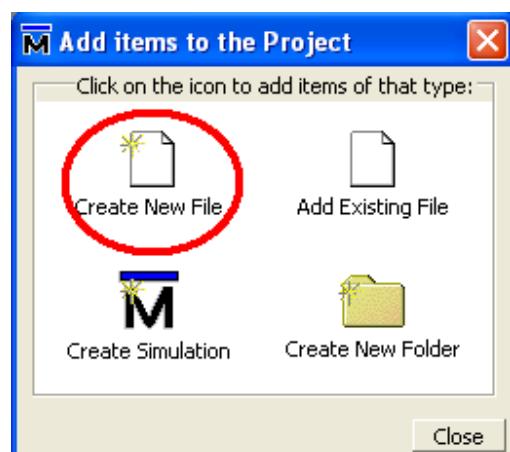
Hình 7.33 Add file thiết kế có sẵn

Bước 5. Chọn Add Existing File, chỉ đường dẫn đến file netlist.v



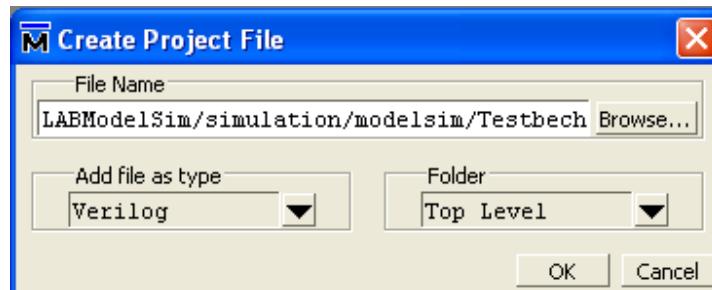
Hình 7.34 Chỉ đường dẫn đến file Verilog netlist

Bước 6. Nhấn OK, quay về cửa sổ trong bước 4, chọn Create New File



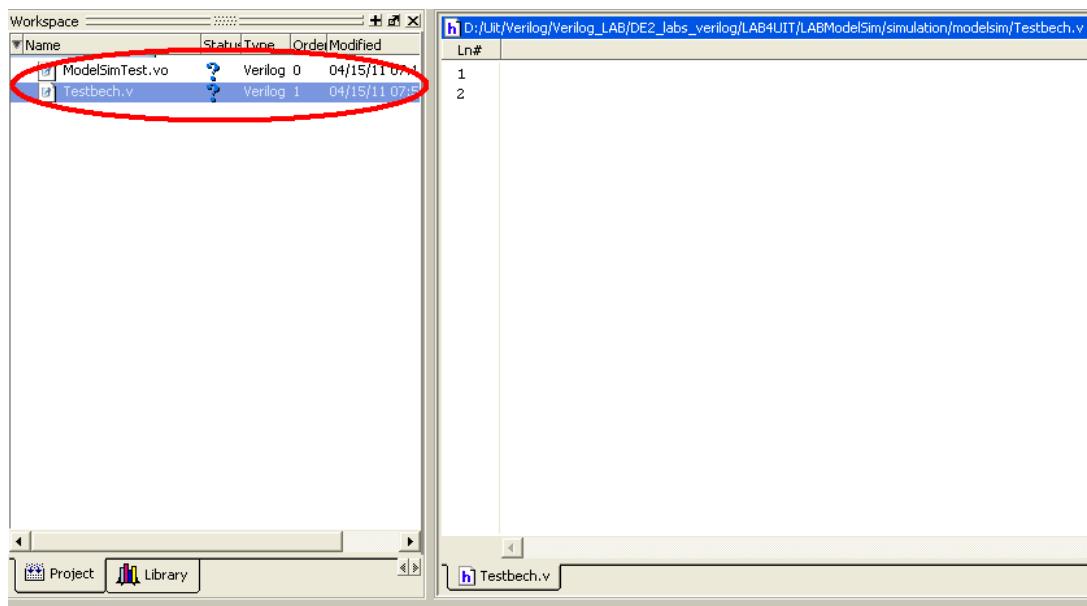
Hình 7.35 Tạo file mô tả thiết kế mới

Bước 7. Nhập đường dẫn và tên cho file mới cần tạo, ta đặt tên file là **Testbech**, và đặt file này vào trong thư mục **simulation/modelsim**



Hình 7.36 Điền thông tin cho new file

Bước 8. Nhấn OK, quay trở về cửa sổ trong bước 4, nhấn Close



Hình 7.37 Cửa sổ mô tả thiết kế được mở ra

Bước 9. Nhập double-click vào Testbench.v, một cửa sổ được xuất hiện, cho phép ta dùng ngôn ngữ Verilog để mô tả Testbench cho thiết kế. Giả sử Testbench được mô tả như sau

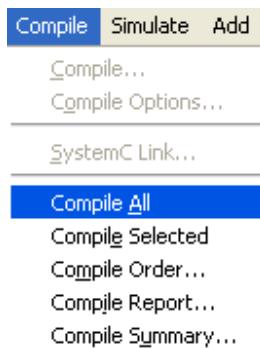
```

D:\Uit\Verilog\Verilog_LAB\DE2_Labs_verilog\LAB4UIT\LABModelSim\simulation\modelsim\Testbech.v
Ln#
1 `timescale 1ns/1ps
2 module Testbench ();
3 reg CLK, RST_N, DATA_IN;
4 wire QOUT;
5 //Create the value for the input signals
6 initial begin
7   CLK = 0;
8   RST_N = 0;
9   DATA_IN = 0;
10  #205;
11  RST_N = 1'b1;
12  #500;
13  DATA_IN = 1'b1;
14  #500;
15  DATA_IN = 1'b0;
16  #2000 $finish;
17 end
18 // Create the clock with a period of 10ns
19 always @ (CLK)
20   #10 CLK <= ~CLK;
21 //Instantiate the module named ModelSimTest (D-Flipflop) which described in file.vo
22 ModelSimTest ins1 (
23   .Clk (CLK),
24   .Resetn (RST_N),
25   .Din (DATA_IN),
26   .Qout (QOUT)
27 );
28 endmodule

```

Hình 7.38 Mô tả Testbench cho thiết kế

Bước 10. Lưu file Testbench, chạy compile cho thiết kế



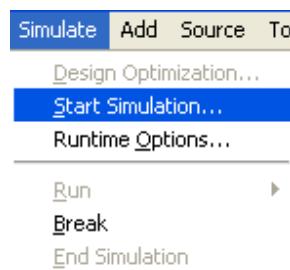
Hình 7.39 Compile thiết kế

Bước 11. Nếu không có lỗi gì, thì sẽ có thông báo sau

The screenshot shows the ModelSim transcript window. It displays the following text:
Transcript
Loading project TestBench_Dflipflop
Compile of ModelSimTest.v was successful.
Compile of Testbench.v was successful.
2 compiles, 0 failed with no errors.
ModelSim>
Below the transcript window, there is a toolbar with a 'Transcript' button, and at the bottom, a status bar showing 'Project : TestBench_Dflipflop <No Design Loaded>'.

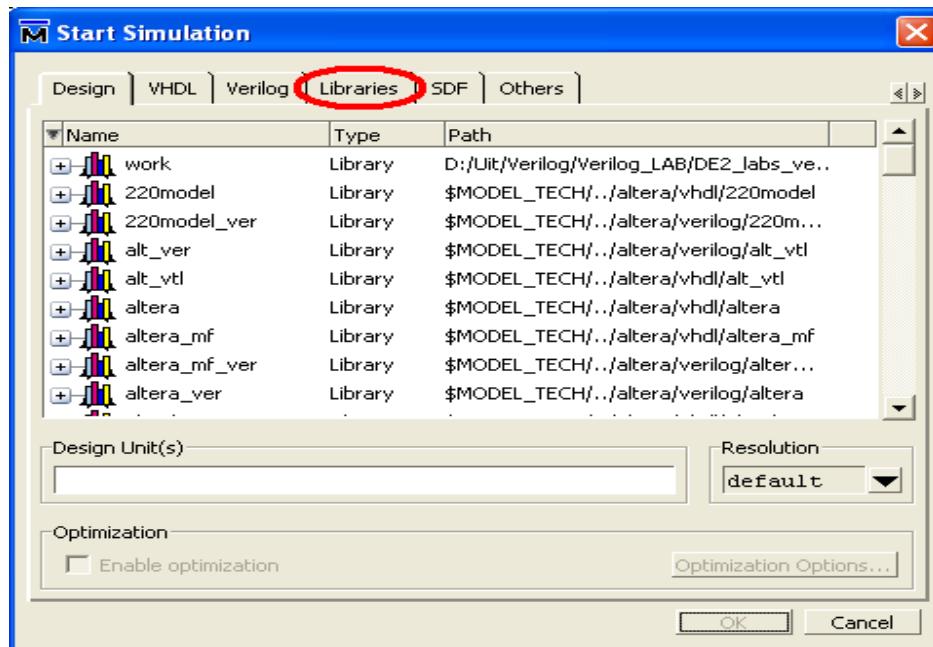
Hình 7.40 Mô tả thiết kế thành công

Bước 12. Thiết lập chạy mô phỏng



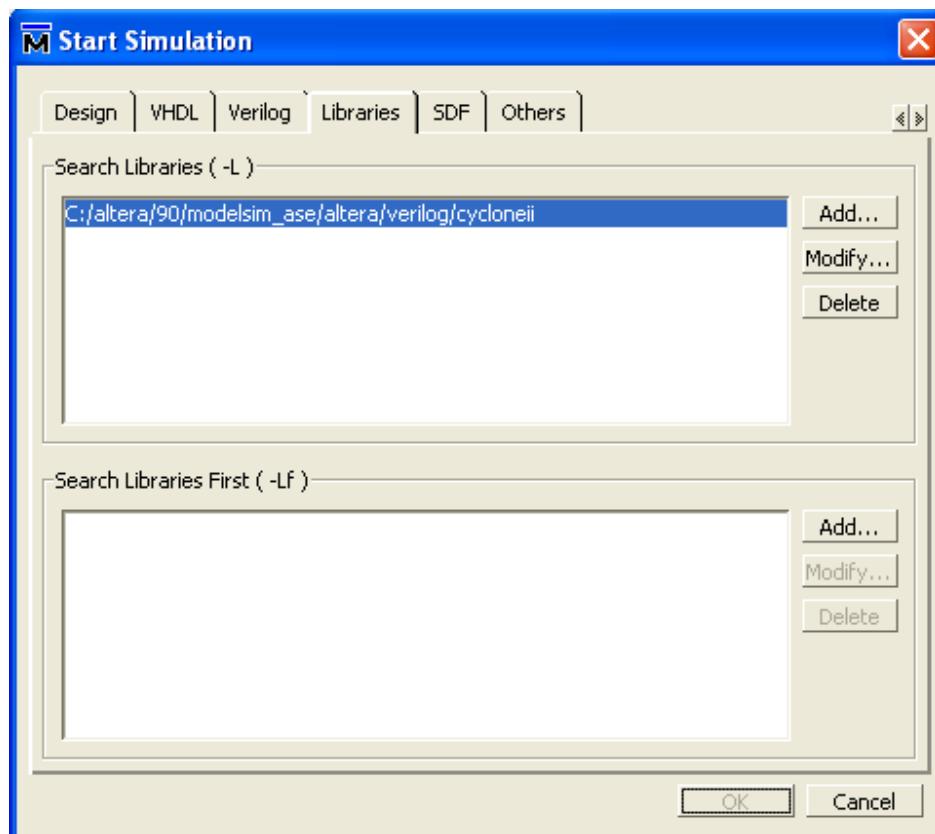
Hình 7.41 Thiết lập mô phỏng

Bước 13. Cửa sổ thiết lập được mở ra



Hình 7.42 Cửa sổ thiết lập mô phỏng

Bước 14. Chọn Tab **Libraries**, nhấn **Add**, chỉ đường dẫn cho library như hình dưới. Chú ý, do khi tạo Quartus project, ta chỉ đến chip FPGA là CycloneII, nên trong bước này ta cũng phải chỉ đến đường dẫn thư mục chứa library của CycloneII. Trong library này chứa các standard components, mà những standard component này được gọi trong file netlist.vo



Hình 7.43 Chỉ Libraries cho thiết kế

Bước 15. Ta thấy có Tab SDF, tuy nhiên ta không cần thiết lập đường dẫn chỉ đến file SDF .sdo này do file này đã được đặt trong cùng thư mục với file netlist.vo, nếu ta mở file netlist.vo lên, ta có thể thấy trong file này đã gọi đến file .sdo rồi.

```

`timescale 1 ps/ 1 ps

module ModelSimTest (
    Din,
    Clk,
    Resetn,
    Qout);
    input      Din;
    input      Clk;
    input      Resetn;
    output     Qout;

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

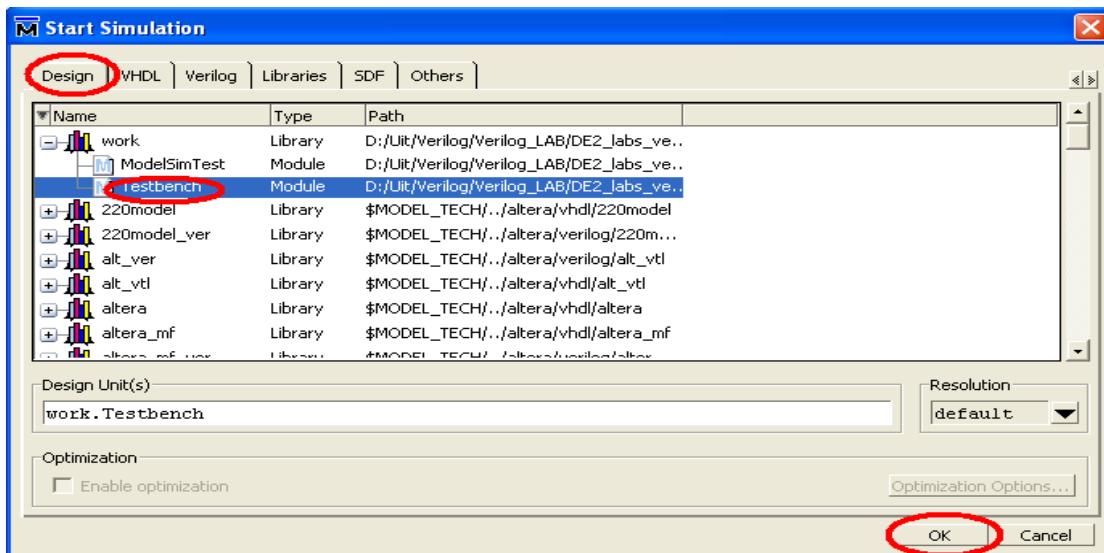
    tri1 devclr;
    tri1 devpor;
    tri1 devoe;
// synopsys translate_off
initial $sdf_annotate("ModelSimTest_v.sdo");
// synopsys translate_on

    wire \Clk~combout ;
    wire \Din~combout ;
    wire \Qout~reg0feeder_combout ;
    wire \Resetn~combout ;
    wire \Qout~reg0_regout ;

```

Hình 7.44 File SDF được gọi trong Verilog netlist

Bước 16. Chọn Tab Design → Work → Testbench → OK



Hình 7.45 Chọn thiết kế cần chạy mô phỏng

Bước 17. Cửa sổ chạy mô phỏng được tạo ra

The screenshot shows the ModelSim interface with the following components:

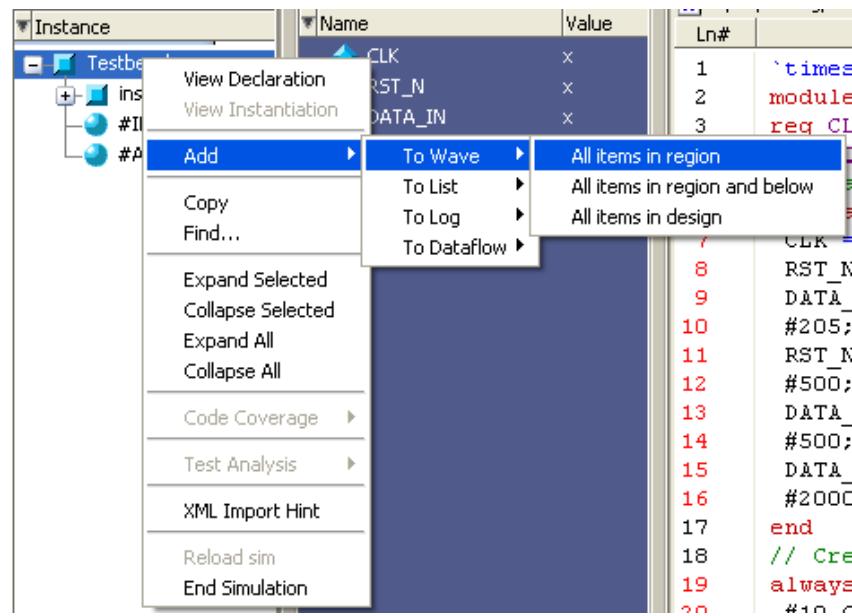
- Workspace:** Shows the project structure with "Testbench" as the current instance.
- Objects:** A table showing signal connections:

Name	Value
CLK	x
RST_N	x
DATA_IN	x
QOUT	Stx
- Script Editor:** Displays the Verilog testbench code for "Testbench.v".
- Bottom Bar:** Includes tabs for "Library", "sim", and "Files".

```
1 `timescale 1ns/1ps
2 module Testbench ();
3   reg CLK, RST_N, DATA_IN;
4   wire QOUT;
5   //Create the value for the input signals
6   initial begin
7     CLK = 0;
8     RST_N = 0;
9     DATA_IN = 0;
10    #20ns;
11    RST_N = 1'b1;
12    #500;
13    DATA_IN = 1'b1;
14    #500;
15    DATA_IN = 1'b0;
16    #2000 $finish;
17  end
18  // Create the clock with a period of 10ns
19  always @ (CLK)
20    #10 CLK <= ~CLK;
21  //Instantiate the module named ModelSimTest (D-Flipflop) which described in file.vo
22  ModelSimTest inst1 (
23    .Clk (CLK),
24    .Resetn (RST_N),
25    .Din (DATA_IN),
26    .Qout (QOUT)
27  );
28 endmodule
```

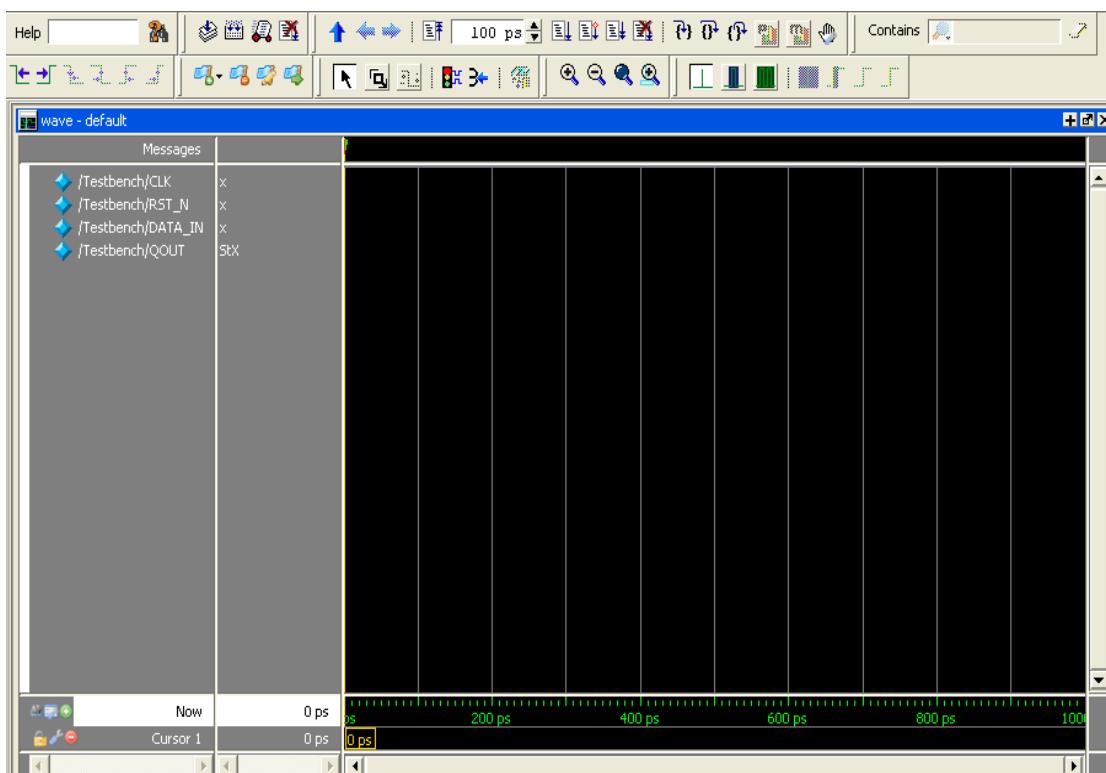
Hình 7.46 Cửa sổ waveform mở ra

Bước 18. Click chuột phải lên Testbench, chọn Add → To Wave → All items in region



Hình 7.47 Chọn tín hiệu cần xem waveform

Bước 19. Cửa sổ dạng sóng được mở ra



Hình 7.48 Cửa sổ dạng sóng mô phỏng

Bước 20. Chọn khoảng thời gian chạy mô phỏng



Hình 7.49 Thiết lập thời gian chạy mô phỏng

Bước 21. Nhấn nút chạy mô phỏng



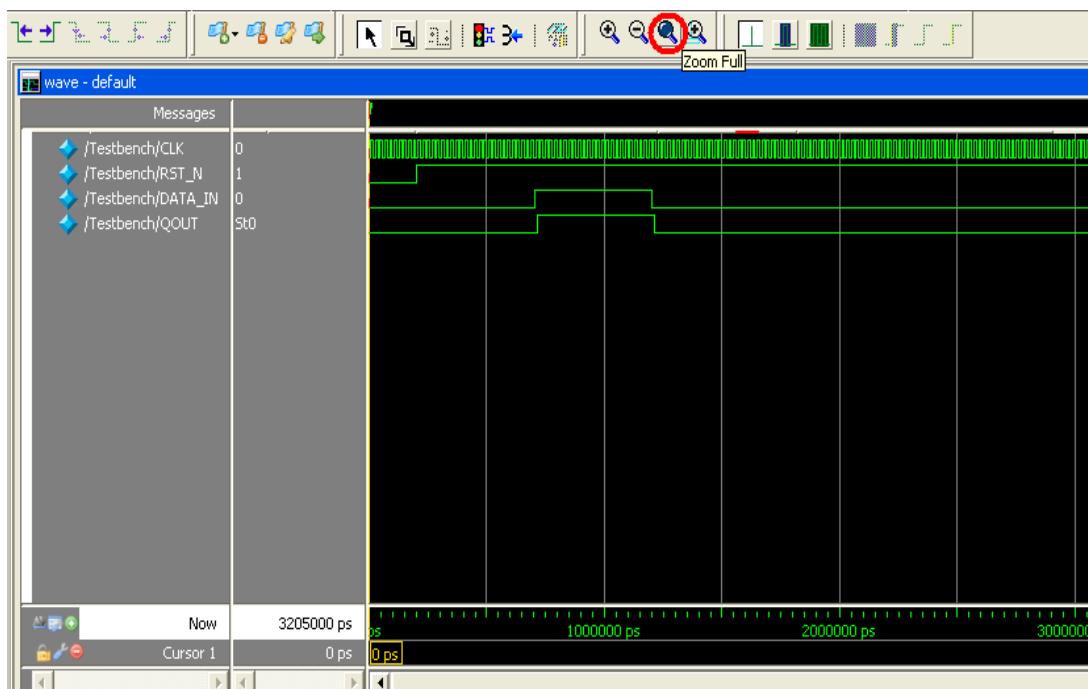
Hình 7.50 Chạy mô phỏng

Bước 22. Sau khi chạy xong, một thông báo lỗi xuất hiện



Hình 7.51 Nhấn "No"

Bước 23. Chọn “No”, dạng sóng sau khi mô phỏng xuất hiện, nhấn nút “Zoom Full” để thấy toàn bộ dạng sóng



Hình 7.52 Dạng sóng sau mô phỏng

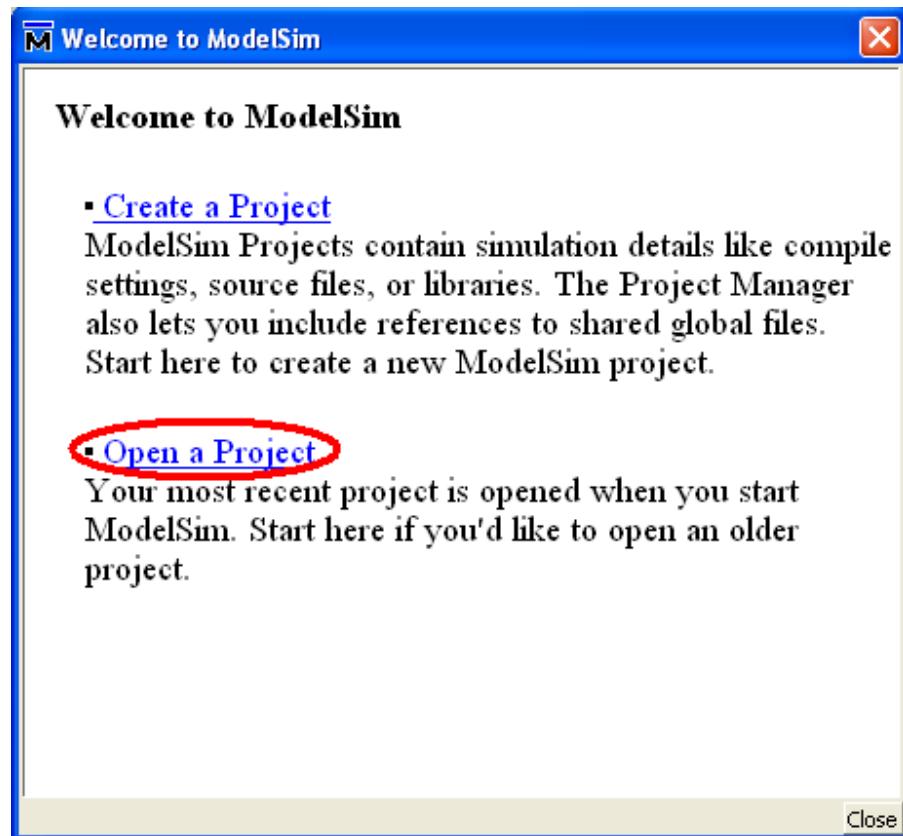
Bước 24. Quan sát ta thấy, tín hiệu ngõ ra có delay timing so với các tín hiệu ngõ vào. Delay timing này do file SDF .sdo tạo ra. Mà timing trong file SDF được tạo ra dựa trên công nghệ thiết kế của FPGA CycloneII.

Bước 25. Quan sát dạng sóng và debug function và timing nếu có sai.

7.3.3 Mở lại project và waveform đã chạy mô phỏng

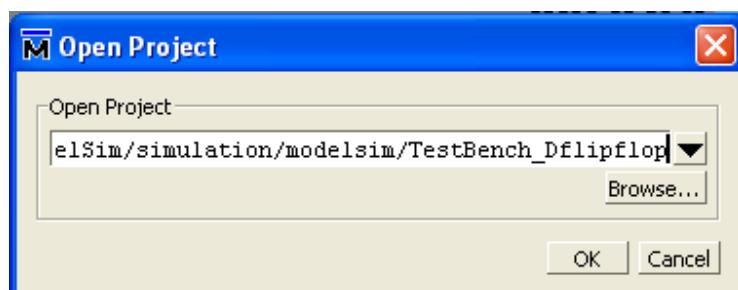
Phần này tập trung trình bày các bước để mở lại waveform đã được tạo ra khi chạy mô phỏng rồi mà không cần chạy lại mô phỏng lần nữa. Vì sao ta cần quan tâm đến điều này. Nếu một thiết kế nhỏ như D-flipflop như trên chẳng hạn thì netlist được tạo ra nhỏ và dạng sóng chạy mô phỏng cũng đơn giản, thời gian dạng sóng cũng ngắn do đó thời gian đợi quá trình chạy mô phỏng là không đáng kể. Tuy nhiên, nếu một thiết kế hệ thống lớn, netlist sẽ rất lớn, dạng sóng cũng dài và phức tạp, nên thời gian đợi cho một lần chạy mô phỏng rất lâu, có thể lên đến hàng giờ đồng hồ. Như vậy, chẳng hạn ta mất 5 giờ để chạy mô phỏng lần đầu xong, ngày hôm sau ta muốn xem lại waveform, ta không thể lại phải mất 5 giờ đồng hồ chạy lại mô phỏng lần nữa. Chính vì thế ta cần mở lại waveform đã chạy rồi theo các bước như sau:

Bước 1. Mở lại project



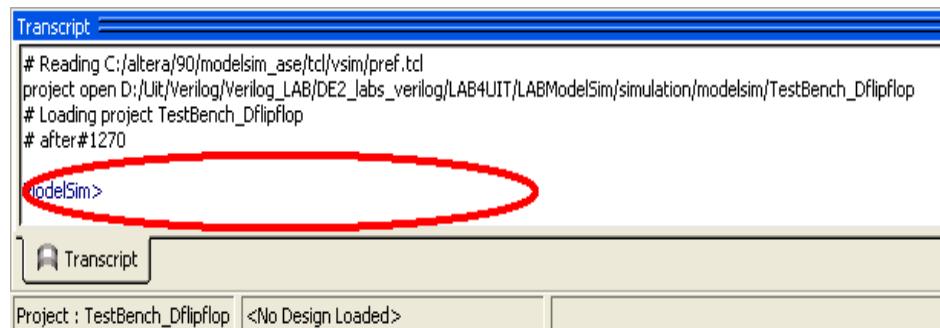
Hình 7.53 Mở lại project

Bước 2. Chỉ đường dẫn project cần mở



Hình 7.54 Chỉ đường dẫn project cần mở

Bước 3. Nhập các lệnh sau ở dấu nháy ModelSim>



```

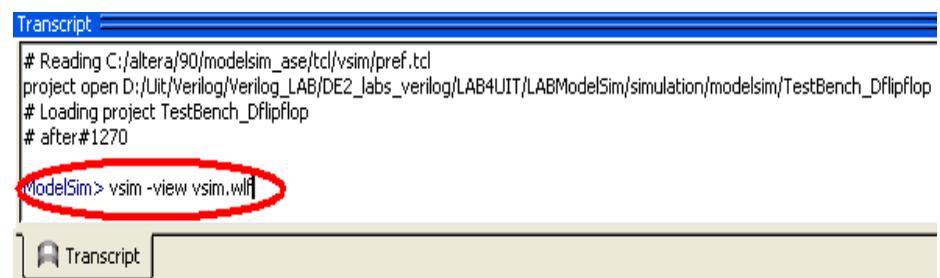
Transcript
# Reading C:/altera/90/modelsim_ase/tcl/vsim/pref.tcl
project open D:/Ulti/Verilog/Verilog_LAB/DE2_labs_verilog/LAB4UIT/LABModelSim/simulation/modelsim/TestBench_Dflipflop
# Loading project TestBench_Dflipflop
# after#1270
ModelSim>

```

Project : TestBench_Dflipflop <No Design Loaded>

Hình 7.55 Cửa sổ Transcript

➤ ModelSim> vsim –view vsim.wlf , nhấn Enter



```

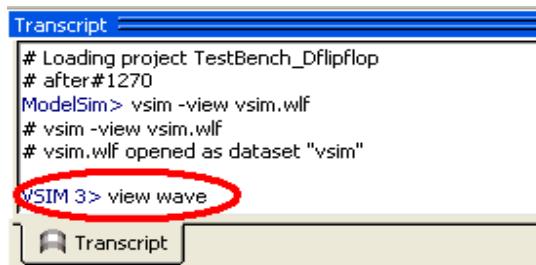
Transcript
# Reading C:/altera/90/modelsim_ase/tcl/vsim/pref.tcl
project open D:/Ulti/Verilog/Verilog_LAB/DE2_labs_verilog/LAB4UIT/LABModelSim/simulation/modelsim/TestBench_Dflipflop
# Loading project TestBench_Dflipflop
# after#1270
ModelSim> vsim -view vsim.wlf

```

Project : TestBench_Dflipflop <No Design Loaded>

Hình 7.56 Nhập dòng lệnh như trên

➤ ModelSim> view wave , nhấn Enter



```

Transcript
# Loading project TestBench_Dflipflop
# after#1270
ModelSim> vsim -view vsim.wlf
# vsim -view vsim.wlf
# vsim.wlf opened as dataset "vsim"
VSIM 3> view wave

```

Project : TestBench_Dflipflop <No Design Loaded>

Hình 7.57 Nhập dòng lệnh như trên

➤ ModelSim> add wave * , nhấn Enter

Transcript

```
ModelSim> vsim -view vsim.wlf
# vsim -view vsim.wlf
# vsim.wlf opened as dataset "vsim"
VSIM 3> view wave
# .main_pane.mdi.interior.cs.vm.paneset.cli_1.wf.clip.cs.pw.wf
VSIM 4> add wave *
```

Hình 7.58 Nhập dòng lệnh như trên

Bước 4. Dạng sóng mô phỏng xuất hiện, nhấn nút “Zoom Full” để thấy toàn bộ dạng sóng

Bước 5. Quan sát dạng sóng và debug function cũng như timing.