

Class:		CPE200L - 1002		Semester:	Spring 2020
Points		Document author:	David Nakasone, Luis Cardenas		
		Author's email:	nakasd3@unlv.nevada.edu , Cardell1@unlv.nevada.edu		
		Document topic:	Post Lab 12 Final Project		
Instructor's comments:					

1. Description:

This a Group project of 2 people, David Nakasone & Luis Cardenas, in this project two devices are built one for encryption and the other for decryption. To realize these functions Verilog was used to code both the encryption and decryption devices. Two test benches are written respectively for the encryption and decryption device. The test bench stimulates this circuit by acting as the buffer for an arbitrary plain text input stream. Both devices Input 200 words of 16-bit length and then output 200 words of 16-bit length. If the message is less than 200 words, the NULL character will be encrypted. This project is comprised of 5 program files: one for getting the ascii message, a function for encryption, its respective testbench function, a decryption function, and its respective testbench.

2. Background and Theory:

This is a 2-circuit project; for this to work the sender and receiver (both parties) should each have an encryption or/and decryption circuit so they can both transmit and receive. The first circuit is an encryption device is designed to input 200 unencrypted 16-

bit words and then output 200 16-bit words; that encrypt the original input. In each word, the input represents a 7-bit ASCII character that the sender wants to transmit, starting with the LSB of the 16-bit word. The remaining bits are 0. The 16-bit word is encrypted on each rising clock edge. For encryption, the 16-bit word just adds $\{2^{15} + 1957\} = **34725**$ to the input, producing an output. In binary, the key is 16'b 1000 0111 1010 0101; for example, 'A' input = ASCII 65 (decimal), so input is [0000 0000 0100 0001]. The input is encrypted to (decimal) $34725 + 65 = 34790$, so the output is [1000 0111 1110 0110] the process continues until all 200 words have been encrypted. If the message is less than 200 words, the NULL character will be encrypted.

The test bench stimulates this circuit by acting as the buffer for an arbitrary plain text input stream. It is what the sender used to input his message. Using these 200 unencrypted 16-bit words as stimuli, the input is changed between clock edges. As the stimuli provide input to the DUT and output is recorded by the test bench, this represents the point at which the receiver gets the encrypted message. Since the transmission was encrypted, the received data will not be coherent unless the receiver has the proper KEY.

The second circuit is a decryption device; it is customizable to virtually anything, but both parties must use the same encrypt and decrypt method. It is designed to input 200 encrypted 16-bit words and output 200 decrypted 16-bit words, revealing the message from the sender. In each word, the received input represents a character. Since the encrypted input was an unencrypted ASCII character with a key of $\{2^{15} + 1957\} = **34725**$ added to it. For example, if you input [1000 0111 1110 0110] to be able to perform the decryption the device will subtract decimal 34725 or binary 16'b 1000 0111 1010 0101.

this produces the output of [0000 0000 0100 0001], revealing the character 'A'; this process continues until all the characters have been decrypted.

The test bench simulates this circuit with 200 encrypted 16-bit words that would have come from the sending circuit. As the stimuli produce output, the receiver can easily convert the binary ASCII numbers to characters and read the message. The encryption method is customizable to virtually anything, but both parties must use the same encrypt and decrypt method.

3. Schematics, Diagrams, and Photos:

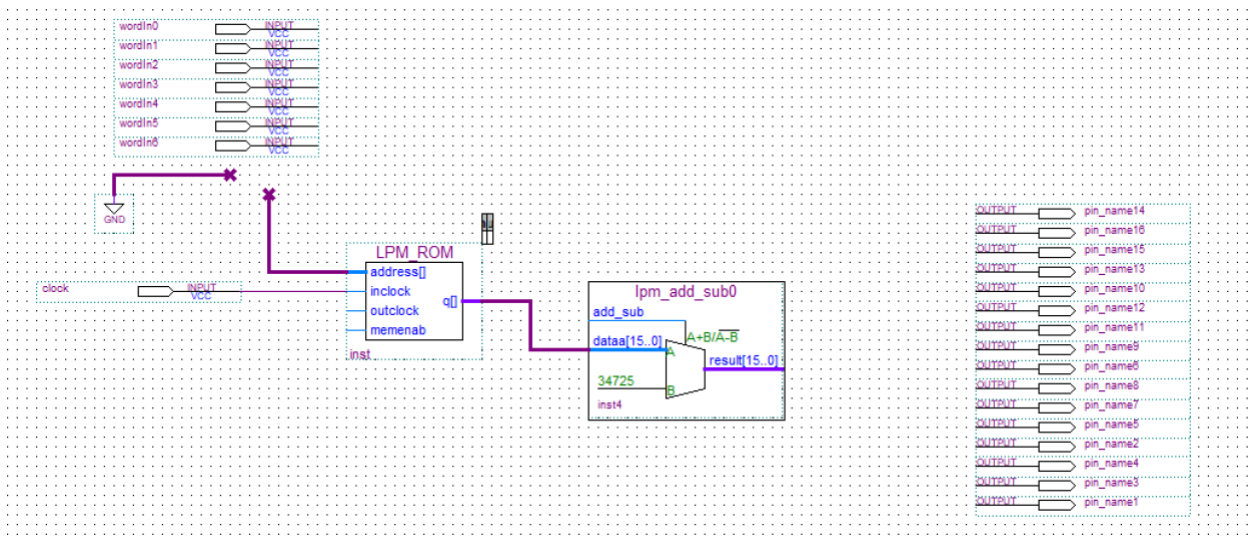


Fig (1) Encryption Device Code:

Quartus II 64-bit - C:/altera/13.0sp1/encryptor - encryptor

File Edit View Project Assignments Processing Tools Window Help

Project Navigator

Entity

Cyclone II: AUTO

encryptor

Tasks

low: Compila Customize...

13 reg [7:0] counter = 8'b0; // count to 199, reflects when all 200 lines encrypted, state control

14 //reg [15:0] carray = 16'b0; // takes input value to a register...for verification, but going to need more states

15 // reg [15:0] verify [0:199]; // holds the 200 lines of 16-bit input, can use to verify

16

17 always @ (posedge clock)

18 begin

19 if (abort)

20 begin

21 ps <= s0; // go to state 0 if abort = 1

22 wordOut <= 16'b0; // output is set to 0

23 readyTX <= 0; // not ready to TX

24 end

25 else ps <= ns; // otherwise go to next state

26 end

27

28

29 always @ (charIn, ps) // when present state or input changes

30 begin

31

32 case (ps)

33

34 s0: begin

35 wordOut = 16'b0; // output is 0

36 readyTX = 0;

37 ns = abort==0 ? s1 : s0; // if abort is 0, encryption can begin

38 end

39

40 s1: begin

41 wordOut = charIn + 16'b1000011110100101; // encrypt current character

42 counter = counter + 1; // increment counter

43 if (counter == 8'b11001000) ns = s2; // once 200 characters encrypted, proceed

44 else ns = s1; // keep encrypting if 200 characters have not been encrypted

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

856

857

858

859

860

861

862

863

864

865

866

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

981

982

983

984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

1001

1002

1003

1004

1005

1006

1007

1008

1009

1010

1011

1012

1013

1014

1015

1016

1017

1018

1019

1020

1021

1022

1023

1024

1025

1026

1027

1028

1029

1030

1031

1032

1033

1034

1035

1036

1037

1038

1039

1040

1041

1042

1043

1044

1045

1046

1047

1048

1049

1050

1051

1052

1053

1054

1055

1056

1057

1058

1059

1060

1061

1062

1063

1064

1065

1066

1067

1068

1069

1070

1071

1072

1073

1074

1075

1076

1077

1078

1079

1080

1081

1082

1083

1084

1085

1086

1087

1088

1089

1090

1091

1092

1093

1094

1095

1096

1097

1098

1099

1100

1101

1102

1103

1104

1105

1106

1107

1108

1109

1110

1111

1112

1113

1114

1115

1116

1117

1118

1119

1120

1121

1122

1123

1124

1125

1126

1127

1128

1129

1130

1131

1132

1133

1134

1135

1136

1137

1138

1139

1140

1141

1142

1143

1144

1145

1146

1147

1148

1149

1150

1151

1152

1153

1154

1155

1156

1157

1158

1159

1160

1161

1162

1163

1164

1165

1166

1167

1168

1169

1170

1171

1172

1173

1174

1175

1176

1177

1178

1179

1180

1181

1182

1183

1184

1185

1186

1187

1188

1189

1190

1191

1192

1193

1194

1195

1196

1197

1198

1199

1200

1201

1202

1203

1204

1205

1206

1207

1208

1209

1210

1211

1212

1213

1214

1215

1216

1217

1218

1219

1220

1221

1222

1223

1224

1225

1226

1227

1228

1229

1230

1231

1232

1233

1234

1235

1236

1237

1238

1239

1240

1241

1242

1243

1244

1245

1246

1247

1248

1249

1250

1251

1252

1253

1254

1255

1256

1257

1258

1259

1260

1261

1262

1263

1264

1265

1266

1267

1268

1269

1270

1271

1272

1273

1274

1275

1276

1277

1278

1279

1280

1281

1282

1283

1284

1285

1286

1287

1288

1289

1290

1291

1292

1293

1294

1295

1296

1297

1298

1299

1300

1301

1302

1303

1304

1305

1306

1307

1308

1309

1310

1311

1312

1313

1314

1315

1316

1317

1318

1319

1320

1321

1322

1323

1324

1325

1326

1327

1328

1329

1330

1331

1332

1333

1334

1335

1336

1337

1338

1339

1340

1341

1342

1343

1344

1345

1346

1347

1348

1349

1350

1351

1352

1353

1354

1355

1356

1357

1358

1359

1360

1361

1362

1363

1364

1365

1366

1367

1368

1369

1370

1371

1372

1373

1374

1375

1376

1377

1378

1379

1380

1381

1382

1383

1384

1385

1386

1387

1388

1389

1390

1391

1392

1393

1394

1395

1396

1397

1398

1399

1400

1401

1402

1403

1404

1405

1406

1407

1408

1409

1410

1411

1412

1413

1414

1415

1416

1417

1418

1419

1420

1421

1422

1423

1424

1425

1426

1427

1428

1429

1430

1431

1432

1433

1434

1435

1436

1437

1438

1439

1440

1441

1442

1443

1444

1

```

1 module encryptor (clock, abort, charIn, wordOut, readyTX); // end of
  port list
2
3   input clock;           // gets the 16-bit character every
  positive edge
4   input abort;           // cease and clear when = 1, synchronus
5   input [15:0] charIn;   // input as 16-bit ASCII character with
  only 7 bits used
6   output reg [15:0] wordOut; // output as 16-bit encrypted word
7   output reg readyTX;    // when circuit is ready to
  transmit, "readyT" = 1
8
9   parameter s0 = 0, s1 = 1, s2 = 2, s3 = 3; // states
10  reg [1:0] ps, ns;      // present state, next
  state registers
11  initial ps = 2'b00;    // start in state 0
12
13  reg [7:0] counter = 8'b0; // count to 199,
  reflects when all 200 lines encrypted, state control
14  //reg [15:0] carray = 16'b0; // takes input value to a
  register...for verification, but going to need more states
15  // reg [15:0] verify [0:199]; // holds the 200 lines of 16-
  bit input, can use to verify
16
17  always @ (posedge clock)
18  begin
19      if (abort)
20      begin
21          ps <= s0; // go to state 0 if abort = 1
22          wordOut <= 16'b0; // output is set to 0
23          readyTX <= 0; // not ready to TX
24      end
25      else ps <= ns; // otherwise go to next state
26  end
27
28

```

```

28
29 always @ (charIn, ps) // when present state or input changes
30 begin
31
32     case (ps)
33
34         s0: begin
35             wordOut = 16'b0; // output is 0
36             readyTX = 0;
37             ns = abort==0 ? s1 : s0; // if abort is 0,
encryption can begin
38             end
39
40         s1: begin
41             wordOut = charIn + 16'b1000011110100101;
42             // encrypt current character
43             counter = counter + 1; //
increment counter
44             if (counter == 8'b11001000) ns = s2; //
once 200 characters encrypted, proceed
45             else ns = s1; //
keep encrypting if 200 characters have not been encrypted
46             readyTX = 0;
47             end
48
49         s2: begin
50             readyTX = 1'b1; // ready to transmit encrypted
message
51             #100; // use counter or wait time to
reflect all bits leaving as serial TX
52             // good spot for a confirmation
53             ns = s3; // go to the "done state"
54             end
55
56         s3: begin
57             readyTX = 1'h1;
58             readyTX = 1'b1; // clear transmit buffer
59             wordOut = 16'b0; // clear input buffer
60             //charIn = 16'b0; // stand by for next
61             ns = s0;
62             end
63     endcase
64 end
endmodule

```

Fig (2) Encryption Testbench:

```
1 //`timescale 1ns / 100ps // unit in ns....#1 is a 1ns delay 10% error
  not synthesizable
2
3 module TBencrypt;
4     reg clock;           // toggled for clock pulse
5     reg abort;           // starts in 1, 0 to encrpyt, made = 1 to
  clear and abort
6     reg [15:0] charIn;   // the 16-bit representation of the input
  character, see input buffer
7     wire [15:0] wordOut; // the 16-bit representation of the encrypted
  input character, output
8     wire readyTX;        // after 200 characters encrypted and placed
  in buffer, this starts serial TX
9
10    integer counter;
11
12    encryptor DUT (clock, abort, charIn, wordOut, readyTX);
13
14    always #10 clock = ~clock; // set clock
15
16    initial              // initial block, holds serial input of characters
17    begin
18        clock = 1'b0; // start it at 0
19        abort = 1'b0; // proceed to encrypt
20        counter = 3'd0; // just for display contol
21        $display ("          TIME(ns)          index
  value(encrypted)");
22        #5 charIn = 16'b0000000000000000; // stops Mealy lag...starts
  the offset, but THROW AWAY, not part of TX
23
24        // BEGIN INPUT BUFFER (original message entered by sender)
25        #20 charIn = 16'b00000000001001001; counter = counter + 3'd001; //
  input 1
26        #20 charIn = 16'b00000000000100000; counter = counter + 3'd001; //
  input 2
27        #20 charIn = 16'b00000000001101000; counter = counter + 3'd001; //
```

```

input 200
225 // END INPUT BUFFER (original message entered by sender)
226
227
228 #20 counter = counter + 3'd001;
229 #20 counter = counter + 3'd001; abort = 1; // disables circuit,
ready to get another transmission
230
231 #100 $finish;
232 end
233
234
235
236 // BEGIN OUTPUT BUFFER (original message entered by sender, each
charater encrypted)
237
238 always @ (posedge clock) // always block reflects serial
transmission of encrypted inputs
239 begin
240 // $write ("%s" , wordOut); // was used to test, use if needed
241 if( counter > 0 && counter <= 200)
242 begin
243 $write ("%d ", $time);
244 $write ("%d", counter);
245 $display (" %b", wordOut);
246 end
247 else if (counter == 200) $display("transmission complete");
248 else if (counter == 201) $display (" { reset,
ready for next TX }");
249 else $display ("tranmission to follow:");
250 end
251
252 // END OUTPUT BUFFER (original message entered by sender, each
charater encrypted)
253
254 endmodule

```


Fig (3) Decryption Device Code:

```
1 module decryptor (clock, abort, wordIn, charOut, readyRX, SV/Verilog/Design
   port list
2
3   input clock;                // gets the 16-bit encrypted word
   every positive edge
4   input abort;                // cease and clear when = 1,
   synchronus
5   input [15:0] wordIn;        // input as 16-bit encrypted word
6   output reg [15:0] charOut;   // output as 16-bit ASCII character
   with only 7 bits used
7   output reg readyRX;         // when circuit is ready to
   transmit, "readyT" = 1
8
9   parameter s0 = 0, s1 = 1, s2 = 2, s3 = 3; // states
10  reg [1:0] ps, ns;           // present state, next
   state registers
11  initial ps = 2'b00;         // start in state 0
12
13  reg [7:0] counter = 8'b0;    // count to 199,
   reflects when all 200 lines encrypted, state control
14  //reg [15:0] carray = 16'b0; // takes input value to a
   register...for verification, but going to need more states
15  // reg [15:0] verify [0:199]; // holds the 200 lines of 16-
   bit input, can use to verify
16
17
18  always @ (posedge clock)
19  begin
20    if (abort)
21    begin
22      ps <= s0;                // go to state 0 if abort = 1
23      charOut <= 16'b0;        // output is set to 0
24      readyRX <= 0;            // not ready to RX
25    end
26  else ps <= ns;               // otherwise go to next state
27  end
```

```

29  always @ (wordIn, ps)      // when present state or input changes
30      begin
31
32          case (ps)
33
34              s0: begin
35                  charOut = 16'b0;          // output is 0
36                  readyRX = 0;
37                  ns = abort==0 ? s1 : s0;  // if abort is 0,
decryption can begin
38                  end
39
40              s1: begin
41                  charOut = wordIn - 16'b1000011110100101;  //
decrypt current word
42                  counter = counter + 1;                    //
increment counter
43                  if (counter == 8'b11001000) ns = s2;      //
once 200 characters encrypted, proceed
44                  else ns = s1;                             //
keep decrypting if 200 characters have not been decrypted
45                  readyRX = 0;
46                  end
47
48              s2: begin
49                  readyRX = 1'b1;  // ready to receive more, last
message decrypted
50                  #100;           // use counter or wait time to
reflect all bits in as serial RX
51                                 // good spot for a confirmation
signal
52                  ns = s3;        // go to the "done state"
53                  end
54
55              s3: begin
56                  readyRX = 1'b1;
57                  charOut = 16'b0; // clear receive buffer
58                  //charIn = 16'b0; // clear output buffer
59                  ns = s0;        // stand by to decrypt next RX
60                  end
61          endcase
62      end
63
64  endmodule
65

```

Fig (4) Decryption Testbench:

```
1 module TBdecrypt;
2
3     reg clock;           // toggled for clock pulse
4     reg abort;           // starts in 1, 0 to encrypt, made = 1 to
clear and abort
5     reg [15:0] wordIn;   // the 16-bit representation of the input
character, see input buffer
6     wire [15:0] charOut; // the 16-bit representation of the encrypted
input character, output
7     wire readyRX;        // after 200 characters encrypted and placed
in buffer, this starts serial TX
8
9     integer index;
10    integer counter;
11    reg [15:0] holder [0:199]; // creates that table you want for
memory
12
13
14    decryptor DUT (clock, abort, wordIn, charOut, readyRX);
15
16    always #10 clock = ~clock; // set clock
17
18    initial           // initial block, holds serial RX input of 16-bit
encrypted words
19    begin
20        $display("\n");
21        clock = 1'b0; // start it at 0
22        abort = 1'b0; // proceed to decrypt
23        counter = 3'd0; // just for display control
24        //$display ("          TIME(ns)      index
value(encrypted)");
25        #5 wordIn = 16'b0000000000000000; // stops Mealy lag...starts
the offset, but THROW AWAY, not part of RX
26
```

```

27
28 // BEGIN INPUT BUFFER (encrypted RX from sender)
29 #20 wordIn = 16'b1000011111101110; counter = counter + 3'd001; //
   encrypted input 1
30 #20 wordIn = 16'b1000011111000101; counter = counter + 3'd001; //
   encrypted input 2
31 #20 wordIn = 16'b1000100000001101; counter = counter + 3'd001; //
   encrypted input 3
32 #20 wordIn = 16'b100010000000110; counter = counter + 3'd001; //
   encrypted input 4
33 #20 wordIn = 16'b1000100000011011; counter = counter + 3'd001; //
   encrypted input 5
34 #20 wordIn = 16'b1000100000001010; counter = counter + 3'd001; //
   encrypted input 6
35 #20 wordIn = 16'b1000011111000101; counter = counter + 3'd001; //
   encrypted input 7
36 #20 wordIn = 16'b1000011111011000; counter = counter + 3'd001; //
   encrypted input 8
37 #20 wordIn = 16'b1000011111000101; counter = counter + 3'd001; //
   encrypted input 9
38 #20 wordIn = 16'b1000100000011001; counter = counter + 3'd001; //
   encrypted input 10
39 #20 wordIn = 16'b1000100000010100; counter = counter + 3'd001; //
   encrypted input 11
40 #20 wordIn = 16'b1000100000010011; counter = counter + 3'd001; //
   encrypted input 12
41 #20 wordIn = 16'b1000100000011000; counter = counter + 3'd001; //
   encrypted input 13
42 #20 wordIn = 16'b1000011111000101; counter = counter + 3'd001; //
   encrypted input 14
43 #20 wordIn = 16'b1000100000010100; counter = counter + 3'd001; //
   encrypted input 15

```

```

212 #20 wordIn = 16'b1000100000010011; counter = counter + 3'd001; //
    encrypted input 184
213 #20 wordIn = 16'b1000100000001010; counter = counter + 3'd001; //
    encrypted input 185
214 #20 wordIn = 16'b1000011111010011; counter = counter + 3'd001; //
    encrypted input 186
215 #20 wordIn = 16'b1000011110101111; counter = counter + 3'd001; //
    encrypted input 187
216 #20 wordIn = 16'b1000011111101100; counter = counter + 3'd001; //
    encrypted input 188
217 #20 wordIn = 16'b1000011111110100; counter = counter + 3'd001; //
    encrypted input 189
218 #20 wordIn = 16'b1000011111110100; counter = counter + 3'd001; //
    encrypted input 190
219 #20 wordIn = 16'b1000011111101001; counter = counter + 3'd001; //
    encrypted input 191
220 #20 wordIn = 16'b1000011111000101; counter = counter + 3'd001; //
    encrypted input 192
221 #20 wordIn = 16'b1000011111110001; counter = counter + 3'd001; //
    encrypted input 193
222 #20 wordIn = 16'b1000011111111010; counter = counter + 3'd001; //
    encrypted input 194
223 #20 wordIn = 16'b1000011111101000; counter = counter + 3'd001; //
    encrypted input 195
224 #20 wordIn = 16'b1000011111110000; counter = counter + 3'd001; //
    encrypted input 196
225 #20 wordIn = 16'b1000011110100101; counter = counter + 3'd001; //
    encrypted input 197
226 #20 wordIn = 16'b1000011110100101; counter = counter + 3'd001; //
    encrypted input 198
227 #20 wordIn = 16'b1000011110100101; counter = counter + 3'd001; //
    encrypted input 199
228 #20 wordIn = 16'b1000011110100101; counter = counter + 3'd001; //
    encrypted input 200
229 // END INPUT BUFFER (encrypted RX from sender)

```

```

232
233     #20 counter = counter + 3'd001;
234     #20 counter = counter + 3'd001; abort = 1; // disables circuit,
ready to get another transmission
235
236     $display("\n");
237     for (index = 1; index < 200; index = index + 1)
238         begin
239             $write ("%s" , holder[index]);
240         end
241     $display("\n");
242
243
244     #100 $finish;
245     end
246
247
248     // BEGIN OUTPUT BUFFER (decrypted RX displayed as ASCII)
249
250     always @ (posedge clock) // always block reflects serial decrypted
RX of encrypted inputs provided
251     begin
252
253         if( counter > 0 && counter <= 200)
254             begin
255                 //$write ("%d ", $time);
256                 //$write ("%d", counter);
257                 //$display (" %b", wordOut);
258                 $write ("%s",charOut);
259             end
260         else if (counter == 200) $display("decryption complete");
261         else if (counter == 201)
262             begin
263                 $write ("\n");
264                 $display ("                                { reset. ready to decrypt next
265                 $display("\n");
266                 $display("SECOND TRANSMISSION, MAKE SURE IT SAME AS FIRST
:");
267             end
268         else $display ("message to follow:");
269
270     end
271
272     // END OUTPUT BUFFER (decrypted RX displayed as ASCII)
273
274
275     always @ (posedge clock)
276     begin
277         if (counter >= 2 || counter <= 201)
278             begin
279                 //$write ("%s" , char);
280                 holder[counter] = charOut;
281                 //counter = counter + 1;
282             end
283     end

```

Fig (5) encryption input (raw):

TIME(ns)	index	value(encrypted)
tranmission to follow:		
30	1	1000011111101110
50	2	10000111111000101
70	3	1000100000001101
90	4	1000100000000110
110	5	1000100000011011
130	6	1000100000001010
150	7	10000111111000101
170	8	10000111111011000
190	9	10000111111000101

Fig (6) encryption output(encrypted):

3850	192	10000111111000101
3870	193	1000011111110001
3890	194	1000011111111010
3910	195	1000011111101000
3930	196	1000011111110000
3950	197	1000011110100101
3970	198	1000011110100101
3990	199	1000011110100101
4010	200	1000011110100101
{ reset, ready for next TX }		
tranmission to follow:		

Fig(7) decryption input is same as encryption output:

3850	192	1000011111000101
3870	193	1000011111110001
3890	194	1000011111111010
3910	195	1000011111101000
3930	196	1000011111110000
3950	197	1000011110100101
3970	198	1000011110100101
3990	199	1000011110100101
4010	200	1000011110100101
{ reset, ready for next TX }		
tranmission to follow:		

Fig (8) decrytpor output with verification

```
message to follow:
I have 3 tons of gold that I need you to pick up.
Be at Melli Bank 35.686 51.393 no later than 202005271300Z.
Bring an airplane, some tools, and 5lbs of potatoes.
Pay Dr. Greg when done.
GOOD LUCK
{ reset, ready to decrypt next RX }
```

SECOND TRANSMISSION, MAKE SURE IT SAME AS FIRST :

```
I have 3 tons of gold that I need you to pick up.
Be at Melli Bank 35.686 51.393 no later than 202005271300Z.
Bring an airplane, some tools, and 5lbs of potatoes.
Pay Dr. Greg when done.
GOOD LUCK
```

message to follow:

Fig(9) SM Chart Decryption Device:

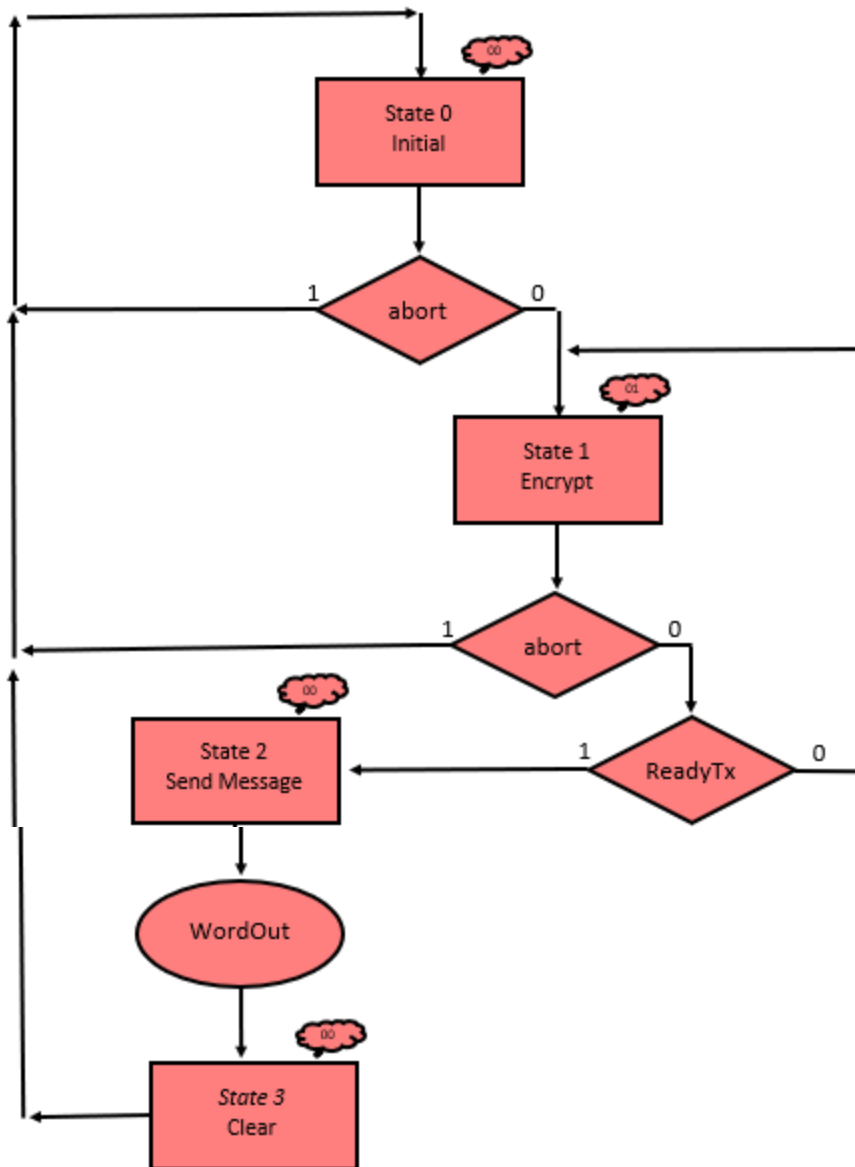


Fig (10) SM Chart Encryption Device:

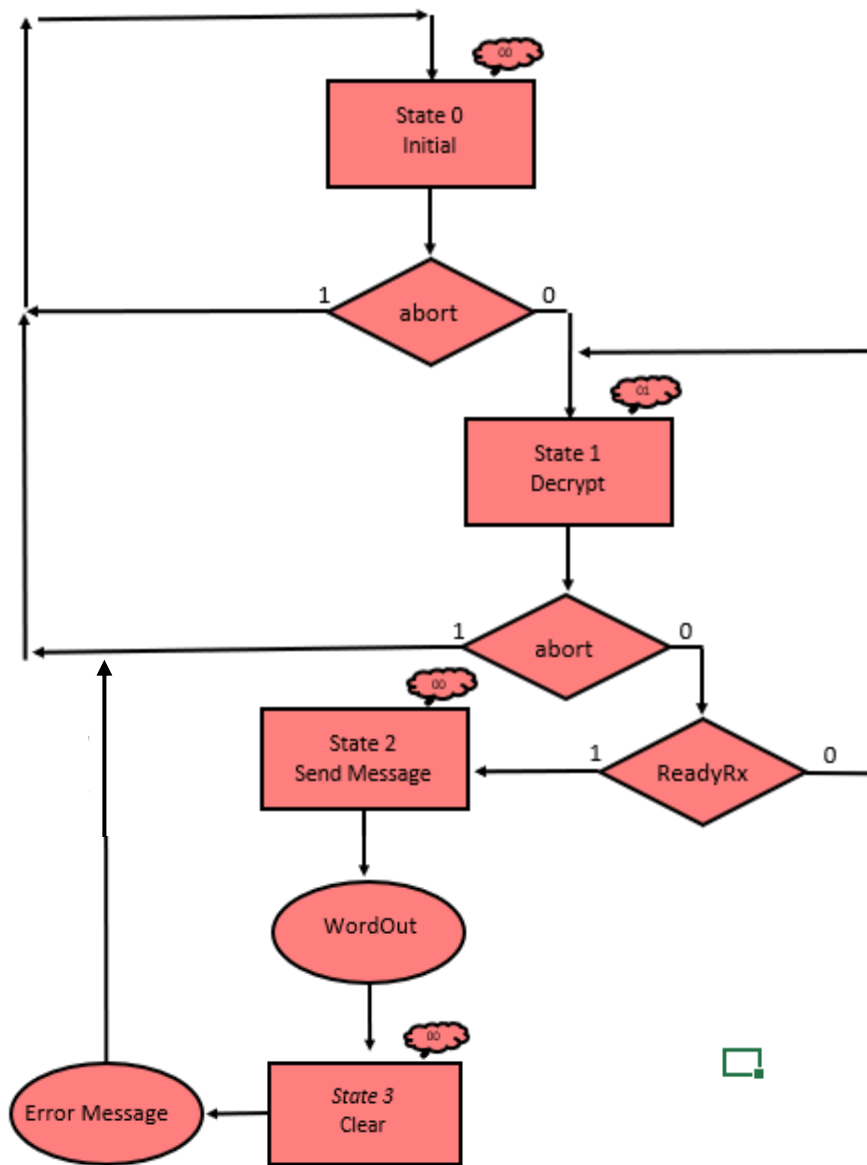
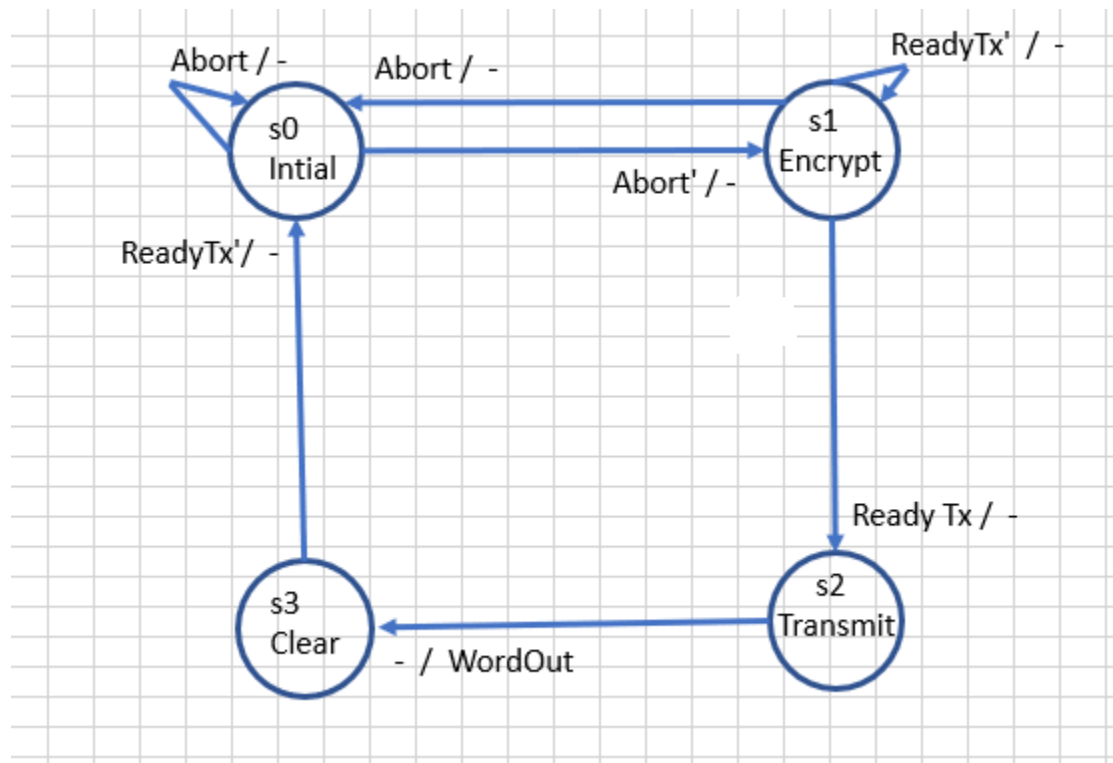
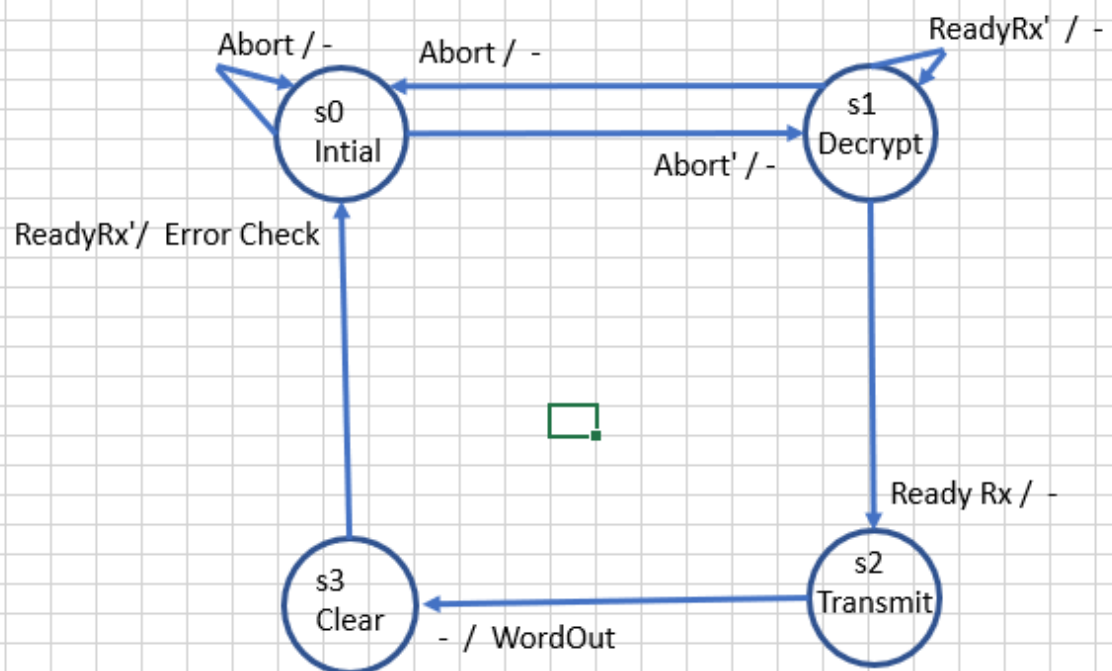


Fig (11) State graph for encryption and state table:



state table				
PS	NS		output	
	x=0	x=1		
s0	s1	s0	-	
s1	s1	s2	-	
s2	-	s3	WordOut	
s3	s0	-	-	

Fig (12) State Graph for Decryption and State Table:



	x=0	x=1	z	
s0	s1	s0	-	
s1	s1	s2	-	
s2	s3	s3	wordOUT	
s3	s0	s0	ErrorCheck	

4. Circuit operation:

The encryption circuit is constructed by using three inputs Clock, Abort, and CharIn; as well as 2 outputs WordOut and ReadyTx. The clock input will be used to get the 16 bits that comprise the word being encrypted. Abort is a synchronous input being used as a clear when set equal to one, this will cease all operations and reset the circuit to state 0. CharIn is made into a 16-bit array used to store the 16-bit ASCII character with only 7 bits being used. We have WordOut set as a 16-bit array to output the now encrypted ASCII character. The output ReadyTx is being used and initially set to 0 as a controller to tell the circuit when to transmit by changing from 0 to 1. Four states are being used in order to realize the encryption function. When abort is set to 1 or the functions begins state 0; this will set the output to 0 to initialize, and ReadyTx is set to 0 to stop loading; If abort is set to 0 then state 1 can begin. State 1 is being used to encrypt an ASCII character using $\text{WordOut} = \text{CharIn} + 16'b1000011110100101$. The counter is increased in state one to read the next variable assigned; this will sequentially continue to read each word until all 200 words are read. Once all 200 words are read the function moves onto the third state, state 2, here the function will transmit the encrypted message to the decryption device. Once state two has been completed the fourth and final state, state 3, in this state the machine will clear the transmit buffer clear the input buffer and reset back to state 0.

The decryption circuit works similar to the encryption circuit using three inputs clock abort CharIn; as well as two outputs WordOut and ReadyRx. These I/O's work in similar fashion the difference being this device serves to decrypt. The clock input will be used to get the 16 bits that comprise the word being decrypted. CharIn is made into a 16-

bit array used to store the 16-bit ASCII the additional nine bits are now being used for the encryption. The output ReadyRx is still being used and initially set to 0 as a controller to tell the circuit when to transmit by changing from 0 to 1. Four states are being used in order to realize the decryption function. When abort is set to 1 or the functions begin state 0; this will set the output to 0 to initialize, and ReadyRx is set to 0 to stop loading; If abort is set to 0 then state 1 can begin. State 1 is being used to encrypt an ASCII character using $\text{WordOut} = \text{CharIn} - 16'b1000011110100101$. The counter is increased in state one to read the next variable assigned; this will sequentially continue to read each word until all 200 words are read. Once all 200 words are read the function moves onto the third state, state 2, here the function will output the message. Once state two has been completed the fourth and final state, state 3, in this state the machine will clear the transmit buffer clear the input buffer and reset back to state 0. The encryption device will output another result with the original text to verify output.

5.Simulation Results with annotation:

Fig (5) encryption input (raw):

TIME(ns)	index	value(encrypted)
tranmission to follow:		
30	1	1000011111101110
50	2	1000011111000101
70	3	1000100000001101
90	4	1000100000000110
110	5	1000100000011011
130	6	1000100000001010
150	7	1000011111000101
170	8	1000011111011000
190	9	1000011111000101

Fig (6) encryption output(encrypted):

3850	192	1000011111000101
3870	193	1000011111110001
3890	194	1000011111111010
3910	195	1000011111101000
3930	196	1000011111110000
3950	197	1000011110100101
3970	198	1000011110100101
3990	199	1000011110100101
4010	200	1000011110100101
{ reset, ready for next TX }		
tranmission to follow:		

Fig(7) decryption input is same as encryption output:

3850	192	1000011111000101
3870	193	1000011111110001
3890	194	1000011111111010
3910	195	1000011111101000
3930	196	1000011111110000
3950	197	1000011110100101
3970	198	1000011110100101
3990	199	1000011110100101
4010	200	1000011110100101

{ reset, ready for next TX }

transmission to follow:

Fig (8) decryptor output with verification

message to follow:

I have 3 tons of gold that I need you to pick up.
 Be at Melli Bank 35.686 51.393 no later than 202005271300Z.
 Bring an airplane, some tools, and 5lbs of potatoes.
 Pay Dr. Greg when done.
 GOOD LUCK

{ reset, ready to decrypt next RX }

SECOND TRANSMISSION, MAKE SURE IT SAME AS FIRST :

I have 3 tons of gold that I need you to pick up.
 Be at Melli Bank 35.686 51.393 no later than 202005271300Z.
 Bring an airplane, some tools, and 5lbs of potatoes.
 Pay Dr. Greg when done.
 GOOD LUCK

message to follow:

6.Encountered Problems and how they were solved:

ASCII input and output are hard to realize and very tedious to obtain manually.

To overcome this obstacle and not have to manually code thousands of entries, we used simple programs to generate the segments of code that we needed. The computer program also helped guide the design, get a working model rapidly, and verify circuit output.

Working from home proved difficult when attempting to realize a full hardware implementation so this code is done with only the software in mind and written behaviorally vs structurally.

7.Any information regarding the project that might be interesting:

The code used to encrypt the algorithm is done by adding +1957 in binary to the binary ascii value. 1957 was chosen because it is the year UNLV was founded. This project was done in self-isolation. This project was done during self-isolation and was done through video messaging, Google Drive, and EDAplayground. As lab partners throughout the semester, we completed all 12 labs together. The first few labs were very basic, and we did not know much about sequential circuits. The basics are important in logic design; every topic is interrelated and adds a new level of abstraction to build upon in a later lab. Circuitry has to be properly arranged to function as desired.

It is notable that the designer needs to have the skills with both combinational and sequential circuits. With this project, we were able to apply the majority of principles that we learned throughout the semester. Working with encryption forced us to handle much larger data sets, but we found that if you can handle 2 in a row, then 200 is the same thing. The encryption scheme is very weak for simplicity, and because we are short on time and quarantined can be anything you want as long as there is an algorithm or look up table to decrypt. If decryption can't be done, then it defeats the purpose. If someone can decrypt the message, then that also defeats the purpose. But this scheme can be expanded or modified in any way. This started to get very involved as soon as work began.

8. Conclusions:

In conclusion there are some big numbers to work with and we Implemented a circuit without any I/O or ability to demonstrate, as this was done during virus outbreak of 2020, this makes it all theoretical. C++ was a good way to get the idea in writing and provides a somewhat easy transition into Verilog code. At least you can use some of the existing functions to generate some of the 1000's of inputs you will need. The array index of C++ being reversed.

To realize these functions Verilog was used to code both the encryption and decryption devices. Two test benches are written respectively for the encryption and decryption device. The test bench stimulates this circuit by acting as the buffer for an arbitrary plain text input stream. Both devices Input 200 words of 16-bit length and then

output 200 words of 16-bit length. If the message is less than 200 words, the NULL character will be encrypted. This project is comprised of 5 program files: one for getting the ascii message, a function for encryption, its respective testbench function, a decryption function, and its respective testbench.

9. Program Files:

The c++:

<https://repl.it/@davenakasone/20200424-lab12-v2>

The proof of input:

https://www.edaplayground.com/x/52_8

The encryptor:

<https://www.edaplayground.com/x/3hWW>

The decryptor:

<https://www.edaplayground.com/x/2j7c>

Google Drive:

<https://drive.google.com/open?id=1hB0f9nqWGWZmv1ZEqQpXNPdQT1TQsS0r>