```
////////////////////////////////////////////////////////////////////////////////
//
//  M65C02A soft-core microcomputer project
//
//  Copyright (C) 2012-2014  Michael A. Morris
//
//  All rights reserved. The source code contained herein is publicly released
//  under the terms and conditions of the GNU General Public License as conveyed
//  in the license provided below.
//
//  This program is free software: you can redistribute it and/or modify it
//  under the terms of the GNU General Public License as published by the Free
//  Software Foundation, either version 3 of the License, or any later version.
//
//  This program is distributed in the hope that it will be useful, but WITHOUT
//  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
//  more details.
//
//  You should have received a copy of the GNU General Public License along with
//  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
//
//  Free Software Foundation, Inc.
//  51 Franklin Street, Fifth Floor
//  Boston, MA  02110-1301 USA
//
//  Further, no use of this source code is permitted in any form or means
//  without inclusion of this banner prominently in any derived works.
//
//  Michael A. Morris <morrisma_at_mchsi_dot_com>
//  164 Raleigh Way
//  Huntsville, AL 35811
//  USA
//
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:         M. A. Morris & Assoc.
// Engineer:        Michael A. Morris
//
// Create Date:     12:49:16 11/18/2012
// Design Name:     WDC W65C02 Microprocessor Re-Implementation
// Module Name:     M65C02A.v
// Project Name:    C:\XProjects\ISE10.1i\M65C02A
// Target Devices:  Generic SRAM-based FPGA
// Tool versions:   Xilinx ISE10.1i SP3
//
// Description:
//
//  This module provides a synthesizable implementation of a 65C02 micropro-
//  cessor similar to the WDC W65C02S. The original W65C02 implemented a set of
//  enhancements to the MOS6502 microprocessor. Two new addressing modes were
//  added, several existing instructions were rounded out using the new address-
//  ing modes, and some additional instructions were added to fill in holes pre-
//  in the instruction set of the MOS6502. Rockwell second sourced the W65C02,
//  and in the process added 4 bit-oriented instructions using 32 opcodes. WDC
//  released the W65816/W65802 16-bit enhancements to the W65C02. Two of the new
//  instructions in these processors, WAI and STP, were combined with the four
//  Rockwell instructions, RMBx/SMBx and BBRx/BBSx, along with the original
//  W65C02's instruction set to realize the W65C02S.
//
//  The M65C02A core is a realization of the W65C02S instruction set. It is not
//  a cycle accurate implementation, and it does not attempt to match the idio-
//  syncratic behavior of the W65C02 or the W65C02S with respect to unused op-
//  codes. In the M65C02A core, all unused opcodes are realized as single byte,
//  single cycle NOPs.
//
//  This module demonstrates how to incorporate the M65C02_CoreV2.v logic module
//  into an application-specific implementation. The core logic incorporates
//  most of the logic required for a microprocessor implementation: ALU, regis-
//  ters, address generator, and instruction decode and sequencing. Not included
//  in the core logic are the memory interface, the interrupt handler, the clock
//  generator, and any peripherals.
//
//  This module integrates the M65C02_CoreV2.v module, a simple vectored inter-
//  rupt controller, a simple MMU, and 28kB of internal Block RAM memory. The
//  objective is a module that allows the 6502_functional_test code to be exe-
```

```
 80 //  cuted at speed from internal BRAM to verify that the new core architecture
 81 //  and microprogram implement the 65C02 instruction set architecture (ISA).
 82 //
 83 // Dependencies:    M65C02A.v
 84 //                      M65C02_CoreV2.v
 85 //                          M65C02_MPCv5.v
 86 //                              M65C02_uPgm_V4.coe      (M65C02_uPgm_V4.txt)
 87 //                              M65C02_IDecode_ROM.coe  (M65C02_IDecode_ROM.txt)
 88 //                          M65C02_AddrGenV2.v
 89 //                              M65C02_StkPtr.v
 90 //                          M65C02_ALUv2.v
 91 //                              M65C02_LST.v
 92 //                              M65C02_LU.v
 93 //                              M65C02_SU.v
 94 //                              M65C02_Add.v
 95 //                              M65C02_WrSel.v
 96 //                              M65C02_PSWv2.v
 97 //                      M65C02_IntHndlrV2.v     (Interrupt Handler)
 98 //                      M65C02_MMU.v            (Memory Management Unit)
 99 //                      M65C02_SPIxIF.v         (SPI Master I/F)
100 //                          DPSFmnCE.v          (Transmit & Receive FIFOs)
101 //                          SPIxIF.v            (Configurable SPI Master)
102 //                          fedet.v             (falling edge detector)
103 //                          redet.v             (rising edge detector)
104 //                      UART.v                  (COM0/COM1 Asynch. Serial Ports)
105 //                          DPSFmnCS.v          (Transmit & Receive FIFOs)
106 //                          UART_BRG.v          (UART Baud Rate Generator)
107 //                          UART_TXSM.v         (UART Transmit SM/Shifter)
108 //                          UART_RXSM.v         (UART Receive SM/Shifter)
109 //                          UART_INT.v          (UART Interrupt Generator)
110 //                              fedet.v         (falling edge detector)
111 //                              redet.v         (rising edge detector)
112 //
113 // Revision:
114 //
115 //  0.00    14F22   MAM     Initial File Creation
116 //
117 //  0.10    14G04   MAM     First integrated module.
118 //
119 //  1.00    14G19   MAM     Last Version before Kernel/User Mode added.
120 //
121 // Additional Comments:
122 //
123 ////////////////////////////////////////////////////////////////////////////////
124
125 module M65C02A #(
126     parameter pKernel_SP_Rst  = 8'h02,      // SP Value after Rst
127     parameter pUser_SP_Rst    = 8'hFF,      // SP Value after Rst
128
129     parameter pVec_IRQ        = 16'hFFFE,   // Ext. Maskable Interrupt
130     parameter pVec_BRK        = 16'hFFFE,   // BReaK Instruction Trap
131     parameter pVec_RST        = 16'hFFFC,   // ReSeT Trap (Highest Priority)
132     parameter pVec_NMI        = 16'hFFFA,   // Ex. Non-Maskable Interrupt
133     parameter pVec_ABRT       = 16'hFFF8,   // MMU ABoRT Trap (N/S)
134     parameter pVec_RSVD1      = 16'hFFF6,   // Reserved for Future Use (BRK)
135     parameter pVec_COP        = 16'hFFF4,   // COP Instruction Trap
136     parameter pVec_RSVD0      = 16'hFFF2,   // Reserved for Future Use
137     parameter pVec_INV        = 16'hFFF0,   // Invalid Instruction Trap
138     parameter pVec_RQST7      = 16'hFFEE,   // Int. Maskable Interrupt 7 (COM0)
139     parameter pVec_RQST6      = 16'hFFEC,   // Int. Maskable Interrupt 6 (COM1)
140     parameter pVec_RQST5      = 16'hFFEA,   // Int. Maskable Interrupt 5 (SPI0)
141     parameter pVec_RQST4      = 16'hFFE8,   // Int. Maskable Interrupt 4 (N/U)
142     parameter pVec_RQST3      = 16'hFFE6,   // Int. Maskable Interrupt 3 (N/U)
143     parameter pVec_RQST2      = 16'hFFE4,   // Int. Maskable Interrupt 2 (N/U)
144     parameter pVec_RQST1      = 16'hFFE2,   // Int. Maskable Interrupt 1 (N/U)
145     parameter pVec_RQST0      = 16'hFFE0,   // Int. Maskable Interrupt 0 (N/U)
146
147     parameter pInt_Hndlr      = 9'h021,     // Microprogram Interrupt Handler
148
149     parameter pVAL            = 3'b000,     // VALid instructions
150     parameter pINV            = 3'b001,     // INValid instructions
151     parameter pCOP            = 3'b010,     // CO-Processor instruction
152     parameter pBRK            = 3'b011,     // BRK instruction
153     parameter pPFX            = 3'b100,     // PreFiX instructions (IND/SIZ)
154     parameter pPHR            = 3'b101,     // PHR instruction
155     parameter pPHW            = 3'b110,     // PHR instruction
156     parameter pWAI            = 3'b111,     // WAI instruction
157
158     parameter pNOP            = 8'hEA,      // M65C02 Core NOP instruction
```

```
159
160     parameter pWS_Delay       = 4'b0011,    // Default Internal Wait State Delay
161
162 //    parameter pFrequency      = 14745600,   // M65C02 Development Board
163     parameter pFrequency       = 29491200,  // Chameleon Board
164
165     parameter pUART_LCR       = 8'h40,  // ~BRRE, RTSo, 232 w/o HS, 8N1
166     parameter pUART_IER       = 8'h00,  // No interrupts enabled
167     parameter pUART_BRG       = 115200,
168     parameter pUART_RTO       = 8'd3,   // RTO asserted on Rx idle 3 char times
169     parameter pUART_TF_Depth  = 8'd0,   // Tx FIFO Depth = (2**(0+4))=16
170     parameter pUART_RF_Depth  = 8'd0,   // Rx FIFO Depth = (2**(0+4))=16
171     parameter pUART_TF_Init   = "Pgms/UART_TF_16.coe",
172     parameter pUART_RF_Init   = "Pgms/UART_RF_16.coe",
173
174     parameter pSPI_CR         = 8'h30,  // Rate=1/16, Mode=0, Dir=MSB, Sel=0
175     parameter pSPI_FIFO_Depth = 8'd4,   // Depth = (1 << 4) = 16
176     parameter pSPI_FIFO_Init  = "Pgms/SPI_FIFO_Init_16.coe",
177
178     parameter pMON_AddrWidth  = 8'd12,      // Internal ROM Address Width:  4 kB
179     parameter pMON_File       = "Pgms/M65C02_Tst3.txt",
180     parameter pROM_AddrWidth  = 8'd13,      // Internal RAM Address Width:  8 kB
181     parameter pROM_File       = "Pgms/EhBASIC0.txt",
182     parameter pRAM_AddrWidth  = 8'd14,      // Internal RAM Address Width: 16 kB
183     parameter pRAM_File       = "Pgms/65C02_ft.txt",
184
185     parameter pM65C02_uPgm    = "Pgms/M65C02_uPgm_V4.coe",      // SEQ   :  2 kB
186     parameter pM65C02_IDec    = "Pgms/M65C02_IDecode_ROM.coe"   // DEC   :  2 kB
187 )(                                                            // Total : 32 kB
188     input   nRst,               // System Reset Input
189     input   Clk,                // System Clk Input (Phi2 Input)
190 //
191 //    //  65C02-compatible Interface
192 //
193 //    input   BE,                 // Bus Enable Input
194 //
195 //    output  Phi1O,              // Output Clock Phase 1
196 //    output  Phi2O,              // Output Clock Phase 2
197 //
198     input   nNMI,               // Non-Maskable Interrupt Input     (Active Low)
199     input   nIRQ,               // Maskable Interrupt Request Input (Active Low)
200     output  nVP,                // Vector Pull Output Strobe        (Active Low)
201 //
202     input   nSO,                // Set oVerflow Input               (Active Low)
203 //
204     output  Sync,               // Instruction Fetch Status Strobe Output
205     output  nML,                // Memory Lock Status Output        (Active Low)
206 //
207     input   RdyIn,              // Bus Ready Input
208 //
209 //    output  RnW,                // Read/nWrite Data Strobe Output
210     output  [15:0] AB,          // Physical Address Outputs
211 //    inout   [ 7:0] DB,          // Data Inputs/Outputs
212     output   [ 7:0] DB,         // Data Inputs/Outputs
213 //
214 //    //  M65C02A I/O Extensions
215 //
216     output  [3:0] nCE,          // Decoded Chip Select Outputs
217     output  nRD,                // Read Stobe Output                (Active Low)
218     output  nWR,                // Write Strobe Output              (Active Low)
219     output  [3:0] XA,           // Extended Physical Address Outputs
220
221     //  SPI0 - Serial Peripheral Interface Master
222
223     output  [1:0] nSSel,
224     output  SCK,
225     output  MOSI,
226     input   MISO,
227
228     //  COM0 - Universal Asynchronous Receiver/Transmitter (UART)
229
230     output  COM0_TxD,
231     input   COM0_RxD,
232     output  COM0_nRTS,
233     input   COM0_nCTS,
234
235     output  COM0_DE,
236
237     //  COM1 - Universal Asynchronous Receiver/Transmitter (UART)
```

```
238
239     output  COM1_TxD,
240     input   COM1_RxD,
241     output  COM1_nRTS,
242     input   COM1_nCTS,
243
244     output  COM1_DE
245
246 );
247
248 ////////////////////////////////////////////////////////////////////////////////
249 //
250 //  Declarations
251 //
252
253 reg     Rst;                    // Internal reset (Clk)
254
255 reg     NMI;                    // NMI latch/register to hold NMI until serviced
256 reg     IRQ;                    // External maskable interrupt request input
257
258 reg     SO;                     // External Set Overflow (falling edge active)
259 wire    Set_SO, Clr_SO;
260
261 reg     Ext_Wait;               // External Wait Request Input
262
263 wire    Int;                    // Interrupt handler interrupt signal to M65C02
264 wire    [15:0] Vector;          // Interrupt handler interrupt vector to M65C02
265
266 wire    IRQ_Msk;                // M65C02 Core interrupt mask
267 wire    LE_Int;                 // M65C02 Core Int latch enable - hold Int/Vec
268 wire    VP;                     // M65C02 Core interrupt vector pull
269 wire    Done;                   // M65C02 Core instruction complete/fetch
270 wire    [2:0] Mode;             // M65C02 Core instruction mode
271 wire    RMW;                    // M65C02 Core Read-Modify-Write indicator
272 wire    Wait;                   // M65C02 Core Microcycle Wait Request Input
273 wire    Rdy;                    // M65C02 Core Microcycle Complete Output
274 wire    [ 1:0] IO_Op;           // M65C02 Core I/O cycle type
275 wire    [15:0] VA;              // M65C02 Core Address Output
276 wor     [ 7:0] CPU_DI;          // M65C02 Core Data Input
277 wire    [ 7:0] CPU_DO;          // M65C02 Core Data Output
278
279 wire    [ 7:0] Y, P;            // M65C02 Core Processor Registers
280 wire    Kernel;                 // M65C02 Core Operating Mode
281
282 wire    WE, RE;                 // M65C02 Core Decoded IO Operations
283
284 wire    INV;                    // M65C02 Core Decoded INValid instructions
285 wire    COP;                    // M65C02 Core Decoded COP instruction
286 wire    BRK;                    // M65C02 Core Decoded BRK instruction
287 wire    PFX;                    // M65C02 Core Decoded PreFiX instructions
288 wire    PHR;                    // M65C02 Core Decoded PHR instruction
289 wire    PHW;                    // M65C02 Core Decoded PHW instructions
290 wire    WAI;                    // M65C02 Core Decoded WAI instruction
291
292 wire    IO_Page;                // IO Page Decode
293 reg     [5:0] IO_Sel;           // IO Selects: Vec, MAP, MMU, SPI0, COM0, COM1
294 wire    Sel_VEC;                // Vector Table Select; maps to Monitor ROM/RAM
295 wire    Sel_MAP, Sel_MMU;       // Mapping RAM and MMU Control/Status Registers
296 wire    Sel_SPI0;               // SPI Master Select
297 wire    Sel_COM0, Sel_COM1;     // COM0, COM1 UART Selects
298
299 wire    [ 7:0] MMU_DO;          // MMU Data Output - Used to read MMU registers
300 wire    [19:0] PA;              // MMU Physical Address Output
301 wire    [15:1] CE;              // MMU Chip Enable: Ext=CE[7:1]; Int=CE[15:8]
302 wire    Int_WS;                 // MMU Internal Wait State Request
303
304 reg     [3:0] WSGen;            // Internal Wait State Generator Counter
305 wire    Int_Wait;               // Internal Wait State Request Signal
306
307 wire    [7:0] MON_DO;           // Output Data Bus of MON
308 wire    [7:0] ROM_DO;           // Output Data Bus of ROM
309 wire    [7:0] RAM_DO;           // Output Data Bus of RAM
310
311 wire    [7:0] SPI0_DO;          // SPI0 Output Data Bus
312 wire    SPI0_IRQ;               // SPI0 Interrupt Request
313 wire    [1:0] SSel;             // SPI0 Slave Selects
314
315 wire    [7:0] COM0_DO;          // COM0 Output Data Bus
316 wire    COM0_RTS, COM0_CTS;     // COM0 Modem Control Signals
```

```
317 wire    COM0_IRQ;               // COM0 Interrupt Request
318
319 wire    [7:0] COM1_DO;          // COM1 Output Data Bus
320 wire    COM1_RTS, COM1_CTS;     // COM1 Modem Control Signals
321 wire    COM1_IRQ;               // COM1 Interrupt Request
322
323 ////////////////////////////////////////////////////////////////////////////
324 //
325 //  Implementation
326 //
327
328 always @(posedge Clk)
329 begin
330     Rst <= #1 ~nRst;
331 end
332
333 always @(negedge Clk)
334 begin
335     if(Rst)
336         Ext_Wait <= #1 1;
337     else
338         Ext_Wait <= #1 ~RdyIn;
339 end
340
341 //  Process nSO
342
343 fedet   FE1 (
344             .rst(Rst),
345             .clk(Clk),
346             .din(nSO),
347             .pls(Set_SO)
348         );
349
350 always @(posedge Clk)
351 begin
352     if(Rst | Clr_SO)
353         SO <= #1 Set_SO;
354     else if(Set_SO)
355         SO <= #1 1;
356 end
357
358 //
359 //  Process External NMI and maskable IRQ Interrupts
360 //
361
362 always @(posedge Clk)
363 begin
364     NMI <= #1 ~nNMI;
365     IRQ <= #1 ~nIRQ;
366 end
367
368 //  Instantiate M65C02 Vectored Interrupt Handler
369
370 M65C02_IntHndlrV2   IntHndlr (
371                         .Rst(Rst),              // System Reset
372                         .Clk(Clk),              // System Clock
373
374                         .Rdy(Rdy),
375
376                         .ABRT(ABRT),            // MMU Trap (Not Implemented)
377                         .INV(INV),              // INValid instruction Trap
378                         .NMI(NMI),              // NMI input
379                         .IRQ(IRQ),              // IRQ input
380                         .RQST({COM0_IRQ, COM1_IRQ, SPI0_IRQ, 5'b0}),
381                         .BRK(BRK),              // BRK Instruction
382                         .COP(COP),              // COP Instruction
383
384                         .IRQ_Msk(IRQ_Msk),      // Interrupt Mask (PSW.I)
385                         .LE_Int(LE_Int),        // Latch Int/Vector until VP
386                         .VP(VP),                // Interrupt Service Vector Pull
387
388                         .Int(Int),              // M65C02 Core Interrupt Signal
389                         .Vector(Vector)         // M65C02 Core Interrupt Vector
390                     );
391
392 //  Instantiate M65C02 Core
393
394 assign Wait = Int_Wait | Ext_Wait;
395
```

```
396 M65C02_CoreV2   #(
397                     .pStkPtr_Rst(pKernel_SP_Rst),   // M65C02A Reset Value for S
398                     .pInt_Hndlr(pInt_Hndlr),        // M65C02A Int. Microroutine
399                     .pM65C02_uPgm(pM65C02_uPgm),    // M65C02A Microprogram COE
400                     .pM65C02_IDec(pM65C02_IDec)     // M65C02A Inst. Decoder COE
401                 ) uP (
402                     .Rst(Rst),              // System Reset
403                     .Clk(Clk),              // System Clock
404 
405                     .IRQ_Msk(IRQ_Msk),      // M65C02A Core Interrupt Mask
406                     .xIRQ(IRQ),             // M65C02A Core Extrn Interrupt Flag
407                     .Int(Int),              // M65C02A Core Interrupt Request
408                     .Vector(Vector),        // M65C02A Core Interrupt Vector
409                     .LE_Int(LE_Int),        // M65C02A Core Latch Enable Int/Vec
410                     .VP(VP),                // M65C02A Core Interrupt Vec. Pull
411 
412                     .SO(SO),                // M65C02A Core Set oVerflow Input
413                     .Clr_SO(Clr_SO),        // M65C02A Core Clr SO input
414 
415                     .Done(Done),            // M65C02A Core Instruction Complete
416                     .SC(),                  // M65C02A Core Single Cycle Flag
417                     .Mode(Mode),            // M65C02A Core Instruction Mode
418                     .RMW(RMW),              // M65C02A Core Read/Modify/Write
419 
420                     .Wait(Wait),            // M65C02A Core Microcycle Wait Rqst
421 
422                     .Rdy(Rdy),              // M65C02A Core Microcycle Ready Out
423 
424                     .IO_Op(IO_Op),          // M65C02A Core Bus Cycle Type
425                     .AO(VA),                // M65C02A Core Address Output Bus
426                     .DI(CPU_DI),            // M65C02A Core Data Input Bus
427                     .DO(CPU_DO),            // M65C02A Core Data Output Bus
428 
429                     .A(),                   // M65C02A Core Accumulator Register
430                     .X(),                   // M65C02A Core X Index Register
431                     .Y(Y),                  // M65C02A Core Y Index Register
432                     .P(P),                  // M65C02A Core P Register
433 
434                     .OP1(),                 // M65C02A Core Operand Register 1
435                     .OP2(),                 // M65C02A Core Operand Register 2
436                     .IR(),                  // M65C02A Core Instruction Register
437 
438                     .IND(),                 // M65C02A Core Address Override
439                     .SIZ(),                 // M65C02A Core Size Override
440                     .OAX(),                 // M65C02A Core Op(A) <=> Op(X)
441                     .OAY()                  // M65C92A Core Op(A) <=> Op(Y)
442                 );
443 
444 //  Decode Core Control Signals
445 
446 assign WE  = IO_Op[0];
447 assign RE  = IO_Op[1];
448 
449 assign INV = (Mode == pINV);
450 assign COP = (Mode == pCOP);
451 assign BRK = (Mode == pBRK);
452 assign PFX = (Mode == pPFX);
453 assign PHR = (Mode == pPHR);
454 assign PHW = (Mode == pPHW);
455 assign WAI = (Mode == pWAI);
456 
457 //  MMU - Maps Virtual Addresses to Physical Addresses using 16 4kB pages
458 //
459 //  CE[15]  - MON   : Boot ROM - 0xF000:FFFF; ( 4 kB -  2 BRAMs)
460 //  CE[13]  - ROM   : User ROM - 0xD000:EFFF; ( 8 kB -  4 BRAMs)
461 //  CE[ 8]  - RAM   : User RAM - 0x0000:3FFF; (16 kB -  8 BRAMs)
462 //                              Total RAM/ROM: (28 kB - 14 BRAMs
463 //
464 //  Internal ROM space includes the I/O page. The I/O page is 0xFF00:FFFF;
465 //  The nWP input disable writes in the range 0xF000:FEFF. Writes in the I/O
466 //  page are not inhibited by the nWP input. The result is that the top 32
467 //  location of the ROM BRAMs contains the interrupt/trap vector table.
468 //
469 //  The following is the I/O page decode map:
470 //
471 //  Sel_VEC     - VecTbl    : BRAM(MON) - 0xFFE0:FFFF   (32 Bytes)
472 //  Sel_COM1    - COM1 (Hi) : LUTs/FFs  - 0xFFD8:FFDF   ( 8 Bytes)
473 //  Sel_COM0    - COM0 (Lo) : LUTs/FFs  - 0xFFD0:FFD7   ( 8 Bytes)
474 //  Sel_SPI0    - SPI0 (Hi) : LUTs/FFs  - 0xFFC8:FFCF   ( 8 Bytes)
```

```
475 //  Sel_MMU     - MMU  (Lo) : LUTs/FFs  - 0xFFC0:FFC7   ( 8 Bytes)
476 //  Sel_MAP     - MAP       : LUTs/FFs  - 0xFF80:FFBF   (64 Bytes)
477 //
478
479 //  Implement I/O Decode for Internal Functions
480
481 assign Kernel   = P[5];      // IO Page only mapped to Kernel mode 0xFF00:FFFF
482 assign IO_Page  = Kernel & (&VA[15:7]);                         // 0xFF80:FFFF
483
484 always @(*)
485 begin
486     case({IO_Page, VA[6:3]})
487         5'b10000 : IO_Sel <= 6'b1_0000_0;       // MAP  :   0xFF80:FFBF
488         5'b10001 : IO_Sel <= 6'b1_0000_0;       // MAP
489         5'b10010 : IO_Sel <= 6'b1_0000_0;       // MAP
490         5'b10011 : IO_Sel <= 6'b1_0000_0;       // MAP
491         5'b10100 : IO_Sel <= 6'b1_0000_0;       // MAP
492         5'b10101 : IO_Sel <= 6'b1_0000_0;       // MAP
493         5'b10110 : IO_Sel <= 6'b1_0000_0;       // MAP
494         5'b10111 : IO_Sel <= 6'b1_0000_0;       // MAP
495         5'b11000 : IO_Sel <= 6'b0_1000_0;       // MMU  :   0xFFC0:FFC7
496         5'b11001 : IO_Sel <= 6'b0_0100_0;       // SPI0 :   0xFFC8:FFCF
497         5'b11010 : IO_Sel <= 6'b0_0010_0;       // COM0 :   0xFFD0:FFD7
498         5'b11011 : IO_Sel <= 6'b0_0001_0;       // COM1 :   0xFFD8:FFDF
499         5'b11100 : IO_Sel <= 6'b0_0000_1;       // VEC  :   0xFFE0:FFFF
500         5'b11101 : IO_Sel <= 6'b0_0000_1;       // VEC
501         5'b11110 : IO_Sel <= 6'b0_0000_1;       // VEC
502         5'b11111 : IO_Sel <= 6'b0_0000_1;       // VEC
503         default  : IO_Sel <= 6'b0_0000_0;
504     endcase
505 end
506
507 assign {Sel_MAP, Sel_MMU, Sel_SPI0, Sel_COM0, Sel_COM1, Sel_VEC} = IO_Sel;
508
509 //  Instantiate Memory Management Module
510
511 M65C02_MMU  MAP (
512                 .Rst(Rst),
513                 .Clk(Clk),
514
515                 .Rdy(Rdy),
516
517                 .Mode(Kernel),
518                 .Sync(Done),
519                 .IO_Op(IO_Op),
520
521                 .VA(VA),
522
523                 .Sel_MAP(Sel_MAP),
524                 .Sel_MMU(Sel_MMU),
525                 .WE(WE),
526                 .RE(RE),
527                 .Sel(Y[4:0]),
528                 .MMU_DI(CPU_DO),
529                 .MMU_DO(MMU_DO),
530
531                 .PA(PA),
532                 .CE(CE),
533                 .Int_WS(Int_WS),
534
535                 .ABRT(ABRT)
536             );
537
538 assign CPU_DI = ((Sel_MAP | Sel_MMU) ? MMU_DO : 0);
539
540 //  Implement Internal Wait State Generator
541 //      Applies to off-chip memory cycles. MMU WS field provides a minimum delay
542 //      to allow external logic to decode the address and determine if addi-
543 //      tional wait states are required.
544
545 always @(negedge Clk)
546 begin
547     if(Rst)
548         WSGen <= #1 0;
549     else if(~|WSGen)
550         WSGen <= #1 ((Int_WS) ? pWS_Delay : (WSGen - 1));
551 end
552
553 assign Int_Wait = Int_WS & |WSGen;
```

```
554
555 //  Generate Selects for Internal BRAMs
556
557 assign Sel_MON = CE[15] & ~IO_Page | Sel_VEC;
558 assign Sel_ROM = CE[13];
559 assign Sel_RAM = CE[ 8];
560
561 //  MON - 0xF000-FF7F (Monitor), 0xFFE0-FFFF (Vector Table) ( 4 kB)
562
563 BRAM_SP_mn  #(
564                 .pBRAM_AddrWidth(pMON_AddrWidth),
565                 .pBRAM_SP_mn_Init(pMON_File)
566             ) MON (
567                 .Clk(~Clk),
568                 .WP(1'b0),
569
570                 .CE(Sel_MON),
571
572                 .WE(WE),
573                 .PA(PA[(pMON_AddrWidth - 1):0]),
574                 .DI(CPU_DO),
575                 .DO(MON_DO)
576             );
577
578 assign CPU_DI = ((Sel_MON) ? MON_DO : 0);
579
580 //  ROM  - 0xD000-EFFF ( 8 kB)
581
582 BRAM_SP_mn  #(
583                 .pBRAM_AddrWidth(pROM_AddrWidth),
584                 .pBRAM_SP_mn_Init(pROM_File)
585             ) ROM (
586                 .Clk(~Clk),
587                 .WP(1'b0),
588
589                 .CE(Sel_ROM),
590
591                 .WE(WE),
592                 .PA(PA[(pROM_AddrWidth - 1):0]),
593                 .DI(CPU_DO),
594                 .DO(ROM_DO)
595             );
596
597 assign CPU_DI = ((Sel_ROM) ? ROM_DO : 0);
598
599 //  RAM  - 0x0000-3FFF (16 kB)
600
601 BRAM_SP_mn  #(
602                 .pBRAM_AddrWidth(pRAM_AddrWidth),
603                 .pBRAM_SP_mn_Init(pRAM_File)
604             ) RAM (
605                 .Clk(~Clk),
606                 .WP(1'b0),
607
608                 .CE(Sel_RAM),
609
610                 .WE(WE),
611                 .PA(PA[(pRAM_AddrWidth - 1):0]),
612                 .DI(CPU_DO),
613                 .DO(RAM_DO)
614             );
615
616 assign CPU_DI = ((Sel_RAM) ? RAM_DO : 0);
617
618 ////////////////////////////////////////////////////////////////////////////
619 //
620 //  Internal I/O devices: 1 SPI Master (Buffered), 2 UARTs (Buffered)
621 //
622
623 //  Core 0 - SPI Master Peripheral
624
625 M65C02_SPIxIF   #(
626                     .pDefault_CR(pSPI_CR),
627                     .pSPI_FIFO_Depth(pSPI_FIFO_Depth),
628                     .pSPI_FIFO_Init(pSPI_FIFO_Init)
629                 ) SPI0 (
630                     .Rst(Rst),
631                     .Clk(Clk),
632
```

```
633                     .IRQ(SPI0_IRQ),
634
635                     .Sel(Sel_SPI0),
636                     .RE(RE),
637                     .WE(WE),
638                     .Reg(PA[1:0]),
639                     .DI(CPU_DO),
640                     .DO(SPI0_DO),
641
642                     .SSel(SSel),
643                     .SCK(SCK),
644                     .MOSI(MOSI),
645                     .MISO(MISO)
646                 );
647
648 assign CPU_DI = ((Sel_SPI0) ? SPI0_DO : 0);
649
650 assign nSSel = ~SSel;
651
652 //  Core 0 - UART Peripheral
653
654 UART    #(
655             .pFrequency(pFrequency),
656             .pDefault_LCR(pUART_LCR),
657             .pDefault_IER(pUART_IER),
658             .pBaudrate(pUART_BRG),
659             .pRTOChrDlyCnt(pUART_RTO),
660             .pTF_Depth(pUART_TF_Depth),
661             .pRF_Depth(pUART_RF_Depth),
662             .pTF_Init(pUART_TF_Init),
663             .pRF_Init(pUART_RF_Init)
664         ) COM0 (
665             .Rst(Rst),
666             .Clk(Clk),
667
668             .IRQ(COM0_IRQ),
669
670             .Sel(Sel_COM0),
671             .Reg(PA[1:0]),
672             .RE(RE),
673             .WE(RE),
674             .DI(CPU_DO),
675             .DO(COM0_DO),
676
677             .TxD(COM0_TxD),
678             .RxD(COM0_RxD),
679             .xRTS(COM0_RTS),
680             .xCTS(COM0_CTS),
681             .xDE(COM0_DE),
682
683             .Mode(),
684
685             .TxIdle(),
686             .RxIdle()
687         );
688
689 assign CPU_DI   = ((Sel_COM0) ? COM0_DO  : 0);
690
691 assign COM0_nRTS = ~COM0_RTS;
692 assign COM0_CTS  = ~COM0_nCTS;
693
694 //  Core 1 - UART Peripheral
695
696 UART    #(
697             .pFrequency(pFrequency),
698             .pDefault_LCR(pUART_LCR),
699             .pDefault_IER(pUART_IER),
700             .pBaudrate(pUART_BRG),
701             .pRTOChrDlyCnt(pUART_RTO),
702             .pTF_Depth(pUART_TF_Depth),
703             .pRF_Depth(pUART_RF_Depth),
704             .pTF_Init(pUART_TF_Init),
705             .pRF_Init(pUART_RF_Init)
706         ) COM1 (
707             .Rst(Rst),
708             .Clk(Clk),
709
710             .IRQ(COM1_IRQ),
711
```

```
712             .Sel(Sel_COM1),
713             .Reg(PA[1:0]),
714             .RE(RE),
715             .WE(WE),
716             .DI(CPU_DO),
717             .DO(COM1_DO),
718 
719             .TxD(COM1_TxD),
720             .RxD(COM1_RxD),
721             .xRTS(COM1_RTS),
722             .xCTS(COM1_CTS),
723             .xDE(COM1_DE),
724 
725             .Mode(),
726 
727             .TxIdle(),
728             .RxIdle()
729         );
730 
731 assign CPU_DI = ((Sel_COM1) ? COM1_DO  : 0);
732 
733 assign COM1_nRTS = ~COM1_RTS;
734 assign COM1_CTS  = ~COM1_nCTS;
735 
736 ////////////////////////////////////////////////////////////////////////////////
737 //
738 //  Define External Connections
739 //
740 
741 assign Sync =  Done;
742 assign nML  = ~RMW | Done;
743 assign nVP  = ~VP;
744 
745 assign nCE  = ~{CE[11], CE[15], CE[13], CE[8]};
746 assign nWR  = ~WE;
747 assign nRD  = ~RE;
748 assign XA   =  PA[19:16];
749 assign AB   =  PA[15: 0];
750 assign DB   = ((WE) ? CPU_DO : CPU_DI);
751 
752 endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
//  Interrupt Handler module for M65C02A soft-core microcomputer project.
//
//  Copyright (C) 2013-2014  Michael A. Morris
//
//  All rights reserved. The source code contained herein is publicly released
//  under the terms and conditions of the GNU General Public License as conveyed
//  in the license provided below.
//
//  This program is free software: you can redistribute it and/or modify it
//  under the terms of the GNU General Public License as published by the Free
//  Software Foundation, either version 3 of the License, or any later version.
//
//  This program is distributed in the hope that it will be useful, but WITHOUT
//  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
//  more details.
//
//  You should have received a copy of the GNU General Public License along with
//  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
//
//  Free Software Foundation, Inc.
//  51 Franklin Street, Fifth Floor
//  Boston, MA  02110-1301 USA
//
//  Further, no use of this source code is permitted in any form or means
//  without inclusion of this banner prominently in any derived works.
//
//  Michael A. Morris <morrisma_at_mchsi_dot_com>
//  164 Raleigh Way
//  Huntsville, AL 35811
//  USA
//
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:         M. A. Morris & Associates
// Engineer:        Michael A. Morris
//
// Create Date:     12:06:18 08/18/2013
// Design Name:     WDC W65C02 Microprocessor Re-Implementation
// Module Name:     M65C02_IntHndlr.v
// Project Name:    C:\XProjects\ISE10.1i\M65C02A
// Target Devices:  SRAM-based FPGAs: XC3S50A-xVQ100I, XC3S200A-xVQ100I
// Tool versions:   Xilinx ISE 10.1i SP3
//
// Description:
//
//  This module implements a simple interrupt handler for the M65C02 soft-core
//  microprocessor. It accepts external active low inputs for Non-Maskable
//  Interrupt request (nNMI) and maskable Interrupt ReQuest (nIRQ). It synchro-
//  nizes both inputs to the internal system clock (Clk), and generates internal
//  signals NMI and IRQ. NMI is falling edge sensitive, and IRQ is active low
//  level sensitive. The module also accepts the core's mode output (Mode) and
//  generates an internal BReaK software trap request (BRK).
//
//  The non-maskable interrupt request, nNMI, has priority, followed by BRK, and
//  finally nIRQ. The core, from the I bit in the processor register, provides a
//  mask that prevents the generation of the internal IRQ signal.
//
//  Vectors for each of the four interrupt/trap sources are set using para-
//  meters. The current implementation aims to maintain compatibility with the
//  WDC W65C02S processor, so IRQ and BRK share the same vector. A quick edit
//  of the parameters allows an independent vector location to be added for BRK.
//  Similarly, the vectors for any of the interrupt/trap sources can be moved
//  to any location in the memory space, if W65C02S compatibility is not desired
//  or required.
//
// Dependencies:    redet.v
//
// Revision:
//
//  0.01    13H18   MAM     File Created
//
//  1.00    13L22   MAM     Modified for the M65C02Duo CPU.
//
```

```
 80 //  2.00    14G06   MAM     Modified to support the M65C02A Mon/IO page concept.
 81 //                          Added support for additional maskable interrupt re-
 82 //                          quests. RQST[7] is the highest priority interrupt of
 83 //                          for the new maskable interrupts. Also added support
 84 //                          for the COP, INV, and SYS (XCE) traps, and an MMU
 85 //                          cycle ABoRT trap (highest priority interrupt/trap
 86 //                          after RST).
 87 //
 88 // Additional Comments:
 89 //
 90 //////////////////////////////////////////////////////////////////////////////////
 91
 92 module M65C02_IntHndlrV2 #(             // Pri
 93     parameter pVec_IRQ   = 16'hFFFE,    //  11
 94     parameter pVec_BRK   = 16'hFFFE,    //   2
 95     parameter pVec_RST   = 16'hFFFC,    //  15 - Highest Priority
 96     parameter pVec_NMI   = 16'hFFFA,    //  13
 97     parameter pVec_ABRT  = 16'hFFF8,    //  14 - Second Highest Priority
 98     parameter pVec_RSVD1 = 16'hFFF6,    //   0 - Reserved for BRK
 99     parameter pVec_COP   = 16'hFFF4,    //   1 - Lowest Practical Interrupt/Trap
100     parameter pVec_RSVD0 = 16'hFFF2,    //   0
101     parameter pVec_INV   = 16'hFFF0,    //  12
102     parameter pVec_RQST7 = 16'hFFEE,    //  10
103     parameter pVec_RQST6 = 16'hFFEC,    //   9
104     parameter pVec_RQST5 = 16'hFFEA,    //   8
105     parameter pVec_RQST4 = 16'hFFE8,    //   7
106     parameter pVec_RQST3 = 16'hFFE6,    //   6
107     parameter pVec_RQST2 = 16'hFFE4,    //   5
108     parameter pVec_RQST1 = 16'hFFE2,    //   4
109     parameter pVec_RQST0 = 16'hFFE0     //   3
110 )(
111     input   Rst,                // Highest Priority Interrupt Source
112     input   Clk,
113
114     input   Rdy,                // Microcycle Ready Input
115
116     input   ABRT,               // ABoRT MMU trap
117     input   NMI,                // Non-Maskable Interrupt Request (6502 NMI)
118     input   INV,                // Invalid Instruction Trap
119     input   IRQ,                // Maskable Interrupt Request (6502 IRQ)
120     input   [7:0] RQST,         // Maskable Interrupt Requests (RQST[7] highest)
121     input   BRK,                // BReaK instruction trap
122     input   COP,                // COProcessor instruction trap
123
124     input   IRQ_Msk,            // Interrupt Request Mask
125     input   LE_Int,             // Latch/hold Int/Vector until RE of VP
126     input   VP,                 // Vector Pull - asserted during Vector read
127
128     output  reg Int,            // Interrupt request
129     output  reg [15:0] Vector   // Vector address for interrupt/trap source
130 );
131
132 //////////////////////////////////////////////////////////////////////////////////
133 //
134 //  Local Declarations
135 //
136
137 wire    RE_NMI;                 // Rising Edge NMI (multi-stage synchronizer)
138 reg     rNMI;                   // Registered/latched NMI (held until serviced)
139
140 wire    iIRQ;                   // Masked internal IRQ
141 wire    [7:0] iRQST;            // Masked internal RQST
142
143 reg     rVP;                    // registered/delayed VP
144 wire    RE_VP;                  // Rising Edge of Vector Pull, deassert Hold
145 reg     Hold;                   // When asserted, Hold Int/Vector
146
147 //////////////////////////////////////////////////////////////////////////////////
148 //
149 //  Implementation
150 //
151
152 //  Perform rising edge detection on the external non-maskable interrupt input
153
154 redet   RE1 (
155             .rst(Rst),
156             .clk(Clk),
157             .din(NMI),
158             .pls(RE_NMI)
```

```
159         );
160
161 //  Detect the falling edge of VP to re-enable Int/Vector
162
163 always @(posedge Clk or posedge Rst)
164 begin
165     if(Rst)
166         rVP <= 0;
167     else if(Rdy)
168         rVP <= #1 VP;
169 end
170
171 assign RE_VP = ~rVP & VP;
172
173 //  Capture and hold the rising edge pulse for NMI in NMI FF until serviced by
174 //      the processor.
175
176 assign CE_rNMI = Rdy & RE_VP;
177
178 always @(posedge Clk or posedge RE_NMI)
179 begin
180     if(RE_NMI)
181         rNMI <= 1'b1;
182     else if(Rst)
183         rNMI <= #1 0;
184     else if(CE_rNMI)
185         rNMI <= #1 RE_NMI;
186 end
187
188 //  Generate Int/Vector hold (lock) signal
189
190 assign CE_Hold  = Rdy & (LE_Int | RE_VP);
191
192 always @(posedge Clk or posedge Rst)
193 begin
194     if(Rst)
195         Hold <= 1;
196     else if(CE_Hold)
197         Hold <= #1 LE_Int;
198 end
199
200 //  Resolve Highest Priority Interrupt/Trap and select vector
201
202 assign iIRQ  = ~IRQ_Msk & IRQ;
203 assign iRQST = ~IRQ_Msk & RQST;
204
205 always @(negedge Clk)
206 begin
207     if(Rst)
208         {Int, Vector} <= #1 {1'b0, pVec_RST};
209     else if(~Hold)
210         casex({ABRT, rNMI, INV, iIRQ, iRQST, BRK, COP})
211             14'b1xx_x_xxxxxxxx_xx : {Int, Vector} <= #1 {1'b1, pVec_ABRT };
212             14'b01x_x_xxxxxxxx_xx : {Int, Vector} <= #1 {1'b1, pVec_NMI  };
213             14'b001_x_xxxxxxxx_xx : {Int, Vector} <= #1 {1'b1, pVec_INV  };
214             14'b000_1_xxxxxxxx_xx : {Int, Vector} <= #1 {1'b1, pVec_IRQ  };
215             14'b000_0_1xxxxxxx_xx : {Int, Vector} <= #1 {1'b1, pVec_RQST7};
216             14'b000_0_01xxxxxx_xx : {Int, Vector} <= #1 {1'b1, pVec_RQST6};
217             14'b000_0_001xxxxx_xx : {Int, Vector} <= #1 {1'b1, pVec_RQST5};
218             14'b000_0_0001xxxx_xx : {Int, Vector} <= #1 {1'b1, pVec_RQST4};
219             14'b000_0_00001xxx_xx : {Int, Vector} <= #1 {1'b1, pVec_RQST3};
220             14'b000_0_000001xx_xx : {Int, Vector} <= #1 {1'b1, pVec_RQST2};
221             14'b000_0_0000001x_xx : {Int, Vector} <= #1 {1'b1, pVec_RQST1};
222             14'b000_0_00000001_xx : {Int, Vector} <= #1 {1'b1, pVec_RQST0};
223             14'b000_0_00000000_1x : {Int, Vector} <= #1 {1'b1, pVec_BRK  };
224             14'b000_0_00000000_01 : {Int, Vector} <= #1 {1'b1, pVec_COP  };
225             default : {Int, Vector} <= #1 {1'b0, pVec_RST  };
226         endcase
227 end
228
229 endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
//  M65C02A soft-core module for M65C02A soft-core microcomputer project.
//

//
//  Michael A. Morris <morrisma_at_mchsi_dot_com>
//  164 Raleigh Way
//  Huntsville, AL 35811
//  USA
//
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 100ps

////////////////////////////////////////////////////////////////////////////////
// Company:         M. A. Morris & Associates
// Engineer:        Michael A. Morris
//
// Create Date:     07:12:56 09/17/2013
// Design Name:     WDC W65C02 Microprocessor Re-Implementation
// Module Name:     M65C02_CoreV2.v
// Project Name:    C:\XProjects\ISE10.1i\M65C02
// Target Devices:  Generic SRAM-based FPGA
// Tool versions:   Xilinx ISE10.1i SP3
//
// Description:     (See additional comments section below)
//
// Dependencies:    M65C02_MPCv5.v
//                      M65C02_uPgm_V4.coe (M65C02_uPgm_V4.txt)
//                      M65C02_IDecode_ROM.coe (M65C02_IDecode_ROM.txt)
//                  M65C02_AddrGenV2.v
//                      M65C02_StkPtr.v
//                  M65C02_ALU_v2.v
//                      M65C02_LST.v
//                      M65C02_LU.v
//                      M65C02_SU.v
//                      M65C02_Add.v
//                      M65C02_WrSel.v
//                      M65C02_PSWv2.v
//
// Revision:
//
//  1.00    13I17   MAM     Forked from M65C02_Core.v. Changed the MPC from
//                          M65C02_MPCv4.v to M65C02_MPCv5.v. The v5 MPC does
//                          not contain the microcycle length controller. That
//                          function will either be implemented as a separate
//                          module in the core, or it will not be used. If not
//                          used, then the core will resemble the original core,
//                          M65C02_Base.v, and will require the memory interface
//                          to insert any wait states needed to access external
//                          memory.
//
//  1.00    14F28   MAM     Updated comments to reflect changes made during the
//                          integration and testing of the module: (1) updated
//                          the component list; (2) removed unused unregistered
```

```
 80 //                          input ID and adjusted the instruction decode mecha-
 81 //                          to use DI instead; (3) adjusted the NA_Op field to
 82 //                          reflect optimizations made to the Address Generator
 83 //                          module; (4) used freed NA_Op microprogram bits to
 84 //                          provide an additional control field used to prepare
 85 //                          the module to support full microprogram control of
 86 //                          the core to support virtual machines, new instruc-
 87 //                          tions, etc.; (5) separated OP1 from DI and made a
 88 //                          separate register (as in previous versions of the
 89 //                          core) and adjusted connections to ALU; (6) added
 90 //                          explicit control of address mod 256 operations using
 91 //                          the new (overloaded) microprogram control field;
 92 //                          (7) Changed the IntSvc FF to perform the Vector Pull
 93 //                          function using the microprogram ISR control and one
 94 //                          of the bits in the new microprogram control field;
 95 //                          (8) adjusted default reset value of the stack poin-
 96 //                          ter to 2 to support modification to reset behavior
 97 //                          of the core which now pushes the PC and PSW to the
 98 //                          stack on reset; and (9) adjusted the connections of
 99 //                          the Tmp and M operand ports on the ALU module.
100 //
101 //  2.00    14H09   MAM     Added multiplexer, controlled by Sel_BA (uP_Cntl==1)
102 //                          to support PHW (PEI/PEA/PHW) and PHR (PER) instruc-
103 //                          tions. Modifications allow {OP2, OP1} to be pushed
104 //                          onto the stack using PCH and PCL as the selects.
105 //
106 //  2.10    14H16   MAM     Added an additional qualifier to the CE_IR signal so
107 //                          that the IR is not loaded unless the Sel_BA is also
108 //                          asserted.
109 //
110 // Additional Comments:
111 //
112 //////////////////////////////////////////////////////////////////////////////
113
114 module M65C02_CoreV2 #(
115     parameter pAddrWidth   = 9,     // MPC Address Width
116     parameter pMPC_Stk     = 0,     // MPC Stack Select: 0 - 1 Lvl; 1 - 4 Lvl
117     parameter pRst_Addrs   = 0,     // MPC Reset Address
118
119     parameter pInt_Hndlr   = 0,     // _Int microroutine address, Reset default
120     parameter pM65C02_uPgm = "Pgms/M65C02_uPgm_V4.coe",
121     parameter pM65C02_IDec = "Pgms/M65C02_IDecode_ROM.coe",
122
123     parameter pStkPtr_Rst  = 2      // Stk Ptr Value after Reset
124 )(
125     input   Rst,            // System Reset Input
126     input   Clk,            // System Clock Input
127
128     //  Processor Core Interrupt Interface
129
130     output  IRQ_Msk,        // Interrupt mask from P to Interrupt Handler
131     output  LE_Int,         // Interrupt Latch Enable - Hold Int until serviced
132
133     input   Int,            // Interrupt input from Interrupt Handler
134     input   xIRQ,           // External Maskable Interrupt Request Input
135     input   [15:0] Vector,  // ISR Vector from Interrupt Handler
136
137     output  reg VP,         // Interrupt Vector Pull Indicator
138
139     input   SO,             // Set oVerflow Flag in PSW
140     output  Clr_SO,         // Clr SO Command - Acknowledge
141
142     //  Processor Core Status Interface
143
144     output  Done,           // Instruction Complete/Fetch Strobe
145     output  SC,             // Single Cycle Instruction
146     output  [2:0] Mode,     // Mode - Instruction Type/Mode
147     output  RMW,            // Read-Modify-Write Operation
148
149     //  Processor Core Memory Controller Interface
150
151     input   Wait,           // Wait Input
152     output  reg Rdy,        // Internal Ready
153
154     //  Processor Core Memory Interface
155
156     output  [ 1:0] IO_Op,   // Instruction Fetch Strobe
157     output  [15:0] AO,      // External Address
158     input   [ 7:0] DI,      // External Data In (Registered Data Path)
```

```
159     output  [ 7:0] DO,      // External Data Out
160
161     output  [ 7:0] A,       // Internal Processor Registers
162     output  [ 7:0] X,
163     output  [ 7:0] Y,
164     output  [ 7:0] P,
165
166     output  reg [ 7:0] OP1, // Internal Temporary/Operand Registers
167     output  reg [ 7:0] OP2,
168     output  reg [ 7:0] IR,  // Instruction Register
169
170     output  reg IND,        // Indirect Addressing Mode Override
171     output  reg SIZ,        // Size Override
172     output  reg OAX,        // Override Op(A | S) with Op(X)
173     output  reg OAY         // Override Op(A | S) with Op(Y)
174 );
175
176 ////////////////////////////////////////////////////////////////////////////
177 //
178 // Local Parameter Declarations
179 //
180
181 localparam  pROM_AddrWidth = 8'd9;
182 localparam  pROM_Width     = 8'd36;
183 localparam  pROM_Depth     = (2**pROM_AddrWidth);
184
185 localparam  pDEC_AddrWidth = 8'd8;
186 localparam  pDEC_Width     = 8'd36;
187 localparam  pDEC_Depth     = (2**pDEC_AddrWidth);
188
189 localparam  pBA_Fill = (pROM_AddrWidth - pDEC_AddrWidth);
190
191 localparam  pBRV1    = 2'b01;   // MPC Via[1:0] code for BRV1 instruction
192 localparam  pBRV2    = 2'b10;   // MPC Via[1:0] code for BRV2 instruction
193 localparam  pBRV3    = 2'b11;   // MPC Via[1:0] code for BRV3 instruction
194 localparam  pBMW     = 4'b0011; // MPC I[3:0] code for BMW instruction
195
196 localparam  pIO_WR   = 2'b01;   // Memory Write
197 localparam  pIO_RD   = 2'b10;   // Memory Read
198 localparam  pIO_IF   = 2'b11;   // Instruction Fetch
199
200 localparam  pDO_ALU  = 2'b00;   // DO <= ALU_Out
201 localparam  pDO_MDH  = 2'b01;   // DO <= ((PHW) ? OP2:((PHR) ? Hi(MAR):Hi(PC)))
202 localparam  pDO_MDL  = 2'b10;   // DO <= ((PHW) ? OP1:((PHR) ? Lo(MAR):Lo(PC)))
203 localparam  pDO_PSW  = 2'b11;   // DO <= P (also available on ALU_Out)
204 //
205 localparam  pDI_Mem  = 2'b00;   // ALU_M <= DI
206 localparam  pDI_OP2  = 2'b01;   // OP2   <= DI
207 localparam  pDI_OP1  = 2'b10;   // OP1   <= DI
208 localparam  pDI_IR   = 2'b11;   // IR    <= DI
209
210 localparam  pNOP     = 8'hEA;   // NOP opcode
211
212 localparam  pIntMsk  = 2;       // Bit number of Interrupt mask bit in P
213
214 localparam  pMd_WAI  = 7;       // Fixed Microcode Mode Field Decode
215 localparam  pMd_PHW  = 6;
216 localparam  pMd_PHR  = 5;
217 localparam  pMd_PFX  = 4;
218 localparam  pMd_BRK  = 3;
219 localparam  pMd_COP  = 2;
220 localparam  pMd_INV  = 1;
221 localparam  pMd_VAL  = 0;
222
223 ////////////////////////////////////////////////////////////////////////////
224 //
225 // Local Signal Declarations
226 //
227
228 wire    WAI;                        // Instruction Mode Decode for WAI
229 wire    PHW;                        // Instruction Mode Decode for PHW
230 wire    PHR;                        // Instruction Mode Decode for PHR
231 wire    PFX;                        // Instruction Mode Decode for PFX
232 wire    BRK;                        // Instruction Mode Decode for BRK
233 wire    COP;                        // Instruction Mode Decode for COP
234 wire    INV;                        // Instruction Mode Decode for INV
235 wire    VAL;                        // Instruction Mode Decode for VAL
236
237 wire    BRV1;                       // MPC BRV1 Instruction Decode
```

```
238 wire    BRV3;                           // MPC BRV3 Instruction Decode
239
240 reg     [(pROM_Width - 1):0] uP_ROM [(pROM_Depth - 1):0]; // Microprogram ROM
241
242 wire    [3:0] I;                        // MPC Instruction Input
243 wire    [3:0] T;                        // MPC Test Inputs
244 wire    [2:0] MW;                       // MPC Multi-way Branch Select
245 reg     [(pROM_AddrWidth - 1):0] BA;    // MPC Branch Address Input
246 wire    [(pROM_AddrWidth - 1):0] MA;    // MPC uP ROM Address Output
247 wire    [1:0] Via;                      // MPC Via Mux Control Output
248
249 reg     [(pROM_Width - 1)    :0] uPL;   // MPC uP ROM Pipeline Register
250 wire    [(pROM_AddrWidth - 1):0] uP_BA; // uP Branch Address Field
251
252 wire    [ 1:0] uPCntl;                  // Microprogram Control Field
253 wire    [10:0] NA_Op;                   // Memory Address Register Control Fld
254 wire    [ 3:0] DI_Op;                   // Memory Data Input Control Field
255 wire    [ 3:0] DO_Op;                   // Memory Data Output Control Field
256 wire    [ 2:0] Reg_WE;                  // Register Write Enable Control Field
257
258 wire    En;                             // ALU Enable Control Field
259
260 wire    CE_IR, CE_OP1, CE_OP2;          // Clock Enables: IR, OP1
261
262 //  Instruction Decoder ROM
263
264 reg     [(pDEC_Width - 1):0] ID_ROM [(pDEC_Depth - 1):0]; // Inst. Decode ROM
265
266 //  Instruction Decoder Pipeline Register (Asynchronous Distributed ROM)
267
268 reg     [(pDEC_Width - 1):0] IDEC;  // Instruction Decode ROM Pipeline Reg.
269 wire    [7:0] IDEC_A;               // Instruction Decode Address
270
271 //  Instruction Decoder (Fixed) Output
272
273 wire    [5:0] FU_Sel;               // M65C02 ALU Functional Unit Select Field
274 wire    [1:0] Op;                   // M65C02 ALU Operation Select Field
275 wire    [1:0] QSel;                 // M65C02 ALU Q Operand Select Field
276 wire    [1:0] RSel;                 // M65C02 ALU R Operand Select Field
277 wire    [1:0] CSel;                 // M65C02 ALU Adder Carry In Select Field
278 wire    [2:0] WSel;                 // M65C02 ALU Register Write Select Field
279 wire    [2:0] OSel;                 // M65C02 ALU Output Select Field
280 wire    [3:0] CCSel;                // M65C02 ALU Condition Code Control Field
281 wire    [7:0] Opcode;               // M65C02 Rockwell Instruction Mask Field
282
283 wire    [15:0] MAR;                 // M65C02 Memory Address Register (MAR)
284 wire    [15:0] PC;                  // M65C02 Program Counter (PC)
285 wire    [ 7:0] S;                   // M65C02 Stack Pointer (S)
286
287 wire    [7:0] DO_ALU;               // M65C02 ALU Data Output Bus
288 wire    Valid;                      // M65C02 ALU Output Valid Signal
289 wire    CC;                         // ALU Condition Code Output
290
291 wire    SelS;                       // Stack Pointer Select
292
293 wire    [15:0] MuxDat;              // Multiplexer Data: {OP2,OP1}, MAR, PC
294 wor     [ 7:0] OutMux;              // Data Output Multiplexer
295
296 ////////////////////////////////////////////////////////////////////////////
297 //
298 //  Start Implementation
299 //
300 ////////////////////////////////////////////////////////////////////////////
301
302 //  Define Instruction Cycle Status Signals
303
304 assign Done = (|Via);               // Instruction Complete (1)     - ~BRV0
305 assign SC   = (&Via);               // Single Cycle Instruction (1) -  BRV3
306
307 //  Generate Rdy Signal: used as a clock enable for internal components
308
309 always @(*)
310 begin
311     case({Done, (FU_Sel[5] & |Reg_WE), Valid, ~Wait})
312         4'b0000 : Rdy <= 0;
313         4'b0001 : Rdy <= 1;     // Non-ALU external cycle ready
314         4'b0010 : Rdy <= 0;
315         4'b0011 : Rdy <= 1;     // Non-ALU external cycle ready
316         4'b0100 : Rdy <= 0;
```

```
317         4'b0101 : Rdy <= 1;     // Operands not ready, external cycle ready
318         4'b0110 : Rdy <= 0;
319         4'b0111 : Rdy <= 1;     // Operands not ready, external cycle ready
320         4'b1000 : Rdy <= 0;
321         4'b1001 : Rdy <= 1;     // Non-ALU op and external fetch ready
322         4'b1010 : Rdy <= 0;
323         4'b1011 : Rdy <= 1;     // Non-ALU op and external fetch ready
324         4'b1100 : Rdy <= 0;     // ALU op and external fetch not ready
325         4'b1101 : Rdy <= 0;     // ALU op not ready and external fetch ready
326         4'b1110 : Rdy <= 0;     // ALU op ready and external fetch not ready
327         4'b1111 : Rdy <= 1;     // ALU op and external fetch cycle ready
328     endcase
329 end
330
331 ////////////////////////////////////////////////////////////////////////////
332 //
333 //  Microprogram Controller Interface
334 //
335 //  Decode MPC Instructions being used for strobes
336
337 assign BRV1 = (Via == pBRV1);
338 assign BRV2 = (Via == pBRV2);
339 assign BRV3 = (Via == pBRV3);
340
341 //  Define the Multi-Way Input Signals
342 //      Implement a 4-way branch when executing WAI, and a 2-way otherwise
343
344 assign MW = ((WAI) ? {uP_BA[2], xIRQ, Int} : {uP_BA[2:1], Int});
345
346 //  Implement the Branch Address Field Multiplexer for Instruction Decode
347
348 always @(*)
349 begin
350     case(Via)
351         pBRV1   : BA <= {{pBA_Fill{1'b1}}, DI[3:0], DI[7:4]};
352         pBRV3   : BA <= ((Int) ? pInt_Hndlr
353                                : {{pBA_Fill{1'b1}}, DI[3:0], DI[7:4]});
354         default : BA <= uP_BA;
355     endcase
356 end
357
358 //  Assign Test Input Signals
359
360 assign T = {IND, SIZ, OAY, OAX};
361
362 //  Instantiate Microprogram Controller/Sequencer - modified F9408A MPC
363
364 M65C02_MPCv5    #(
365                     .pAddrWidth(pAddrWidth),
366                     .pMPC_Stk(pMPC_Stk),
367                     .pRst_Addrs(pRst_Addrs)
368                 ) MPCv5 (
369                     .Rst(Rst),
370                     .Clk(Clk),
371
372                     .Rdy(Rdy),              // MPC Clock Enable
373
374                     .I(I),                  // Instruction
375                     .T(T),                  // Test signal input
376                     .MW(MW),                // Multi-way branch inputs
377                     .BA(BA),                // Branch address input
378                     .Via(Via),              // BRVx multiplexer control output
379
380                     .MA(MA)                 // Microprogram ROM address output
381                 );
382
383 //  Infer Microprogram ROM and initialize with file created by SMRTool
384
385 initial
386     $readmemb(pM65C02_uPgm, uP_ROM, 0, (pROM_Depth - 1));
387
388 always @(posedge Clk)
389 begin
390     if(Rdy | Rst)
391         uPL <= #1 uP_ROM[MA];
392 end
393
394 //  Assign uPL fields
395
```

```
396 assign I      = uPL[35:32];     // MPC Instruction Field (4)
397 assign uP_BA  = uPL[31:23];     // MPC Branch Address Field (9)
398 assign uPCntl = uPL[22:21];     // Microprogram Control Field (2)
399 assign NA_Op  = uPL[20:10];     // Next Address Operation (11)
400 assign IO_Op  = uPL[9:8];       // IO Operation Control (2)
401 assign DI_Op  = uPL[7:4];       // DI Demultiplexer Control (4)
402 assign DO_Op  = uPL[7:4];       // DO Multiplexer Control (4) (same as DI_Op)
403 assign Reg_WE = uPL[3:1];       // Register Write Enable Field (3)
404 assign ISR    = uPL[0];         // Set to clear D and set I on interrupts (1)
405
406 //  Decode uPCntl Field
407
408 assign Mod256 = uPCntl[1];      // Explicit control to wrap address in page
409 assign Page   = uPCntl[0];      // Explicit page select for address wrap ops.
410
411 //  Decode DI_Op Control Field
412
413 assign Ld_OP1 = IO_Op[1] &  DI_Op[2];
414 assign Ld_OP2 = IO_Op[1] &  DI_Op[1];
415
416 assign Sel_BA = (uPCntl == 2'b01);
417
418 //  Operand Register Data Input Bus
419
420 wire [7:0] OP_DI = ((Sel_BA) ? uP_BA : DI);
421
422 //  Operand Register 1
423
424 assign CE_OP1 = Ld_OP1 & Rdy & ~CE_IR;
425
426 always @(posedge Clk)
427 begin
428     if(Rst | BRV2)
429         OP1 <= #1 Vector[7:0];
430     else if(CE_OP1)
431         OP1 <= #1 OP_DI;
432 end
433
434 //  Operand Register 2
435
436 assign CE_OP2 = Ld_OP2 & Rdy & ~CE_IR;
437 assign SignDI = DI_Op[0];
438
439 always @(posedge Clk)
440 begin
441     if(Rst | BRV2)
442         OP2 <= #1 Vector[15:8];
443     else if(CE_OP2)   // OP2: {sign(OP_DI),   OP_DI}
444         OP2 <= #1 ((SignDI) ? {8{OP_DI[7]}} : OP_DI);
445 end
446
447 //  Instruction Register
448
449 assign CE_IR  = (BRV1 | (BRV3 & ~Int) | (Sel_BA & DI_Op[3])) & Rdy;
450 assign IDEC_A = ((Sel_BA) ? {uP_BA[3:0], uP_BA[7:4]} : {DI[3:0], DI[7:4]});
451
452 always @(posedge Clk)
453 begin
454     if(Rst | BRV2)
455         IR <= #1 pNOP;
456     else if(CE_IR)   // IR: {uP_BA,  DI}
457         IR <= #1 ((Sel_BA) ? uP_BA : DI);
458 end
459
460 //  Implement Prefix Registers
461 //      IND - Address mode override maps zp => (zp) and abs => (abs)
462 //      SIZ - Operation Size override remaps operation from 8 to 16 bits
463 //      OAX - Destination register override remaps instruction dest. regs
464
465 always @(posedge Clk)
466 begin
467     if(Rst)
468         {OAY, OAX, SIZ, IND} <= #1 0;
469     else if(CE_IR) begin
470         IND <= #1 ((PFX) ? ~IR[5] & ~IR[4] | IND : 0);  // IR == 8'h8B
471         SIZ <= #1 ((PFX) ? ~IR[5] &  IR[4] | SIZ : 0);  // IR == 8'h9B
472         OAX <= #1 ((PFX) ?  IR[5] & ~IR[4]       : 0);  // IR == 8'hAB
473         OAY <= #1 ((PFX) ?  IR[5] &  IR[5]       : 0);  // IR == 8'hBB
474     end
```

```
475 end
476
477 //  Infer Instruction Decode ROM and initialize with file created by SMRTool
478 //       Load IR from DI | uP_BA
479
480 initial
481     $readmemb(pM65C02_IDec, ID_ROM, 0, (pDEC_Depth - 1));
482
483 always @(posedge Clk)
484 begin
485     if(Rst)
486         IDEC <= #1 0;
487     else if(CE_IR)
488         IDEC <= #1 ID_ROM[IDEC_A];
489 end
490
491 //  Decode Fixed Microcode Word
492
493 assign  Mode   = IDEC[35:33];       // M65C02 Instruction Type/Mode
494 assign  RMW    = IDEC[32];          // M65C02 Read-Modify-Write Instruction
495 assign  FU_Sel = IDEC[31:26];       // M65C02 ALU Functional Unit Select Field
496 assign  Op     = IDEC[25:24];       // M65C02 ALU Operation Select Field
497 assign  QSel   = IDEC[23:22];       // M65C02 ALU AU Q Bus Mux Select Field
498 assign  RSel   = IDEC[21:20];       // M65C02 ALU AU/SU R Bus Mux Select Field
499 assign  CSel   = IDEC[19:18];       // M65C02 ALU AU/SU Carry Mux Select Field
500 assign  WSel   = IDEC[17:15];       // M65C02 ALU Register Write Select Field
501 assign  OSel   = IDEC[14:12];       // M65C02 ALU Register Output Select Field
502 assign  CCSel  = IDEC[11: 8];       // M65C02 ALU Condition Code Control Field
503 assign  Opcode = IDEC[ 7: 0];       // M65C02 Valid Opcode Control Field
504
505 // Decode Mode for internal signals
506
507 assign WAI = &Mode;                 // Current Instruction is WAI
508 assign PHW = (Mode == pMd_PHW);     // Current Instruction is PHW
509 assign PHR = (Mode == pMd_PHR);     // Current Instruction is PHR
510 assign PFX = (Mode == pMd_PFX);     // Current Instruction is IND/SIZ
511 assign BRK = (Mode == pMd_BRK);     // Current Instruction is BRK
512 assign COP = (Mode == pMd_COP);     // Current Instruction is COP
513 assign INV = (Mode == pMd_INV);     // Current Instruction is Invalid
514 assign VAL = (Mode == pMd_VAL);     // Current Instruction is Valid
515
516 //  Next Address Generator
517
518 M65C02_AddrGenV2    #(
519                         .pStkPtr_Rst(pStkPtr_Rst)
520                       ) AddrGen (
521                         .Rst(Rst),
522                         .Clk(Clk),
523
524                         .Vector(Vector),
525
526                         .NA_Op(NA_Op),
527
528                         .Mod256(Mod256),
529                         .Page(Page),
530
531                         .CC(CC),
532                         .BRV3(BRV3),
533                         .Int(Int),
534
535                         .Rdy(Rdy),
536                         .Valid(Valid),
537
538                         .OP1(OP1),
539                         .OP2(OP2),
540
541                         .X(X),
542                         .Y(Y),
543
544                         .AO(AO),
545
546                         .SelS(SelS),
547                         .S(S),
548
549                         .MAR(MAR),
550                         .PC(PC)
551                       );
552
553 //  Interrupt Service Flag
```

```
554
555 always @(posedge Clk)
556 begin
557     if(Rst)
558         VP <= #1 0;
559     else if(ISR)
560         VP <= #1 uPCntl[0];
561 end
562
563 //  Instantiate the M65C02 ALU Module
564
565 assign En = (|Reg_WE);
566
567 M65C02_ALUv2    ALU (
568                     .Rst(Rst),          // System Reset
569                     .Clk(Clk),          // System Clock
570
571                     .Rdy(Rdy),          // Ready
572
573                     .En(En),            // M65C02 ALU Enable Strobe Input
574                     .Reg_WE(Reg_WE),    // M65C02 ALU Register Write Enable
575                     .ISR(ISR),          // M65C02 ALU Interrupt Service Rtn Strb
576
577                     .SO(SO),            // M65C02 ALU Set oVerflow Flag in PSW
578                     .Clr_SO(Clr_SO),    // M65C02 ALU Clr SO - Acknowledge
579
580                     .SelS(SelS),        // M65C02 ALU Stack Pointer Select
581                     .S(S),              // M65C02 ALU Stack Pointer
582
583                     .FU_Sel(FU_Sel[4:0]),   // M65C02 ALU Functional Unit Select
584                     .Op(Op),            // M65C02 ALU Operation Select
585                     .QSel(QSel),        // M65C02 ALU Q Data Mux Select
586                     .RSel(RSel),        // M65C02 ALU R Data Mux Select
587                     .CSel(CSel),        // M65C02 ALU Adder Carry Select
588                     .WSel(WSel),        // M65C02 ALU Register Write Select
589                     .OSel(OSel),        // M65C02 ALU Output Register Select
590                     .CCSel(CCSel),      // M65C02 ALU Condition Code Select
591
592                     .K(Opcode),         // M65C02 ALU Rockwell Instruction Mask
593                     .Tmp(OP2),          // M65C02 ALU Temporary Holding Register
594                     .M(OP1),            // M65C02 ALU Memory Operand
595
596                     .DO(DO_ALU),        // M65C02 ALU Data Output Multiplexer
597                     .Val(Valid),        // M65C02 ALU Output Valid Strobe
598                     .CC_Out(CC),        // M65C02 ALU Condition Code Mux
599
600                     .A(A),              // M65C02 ALU Accumulator Register
601                     .X(X),              // M65C02 ALU Pre-Index Register
602                     .Y(Y),              // M65C02 ALU Post-Index Register
603
604                     .P(P)               // M65C02 Processor Status Word Register
605                 );
606
607 //  Decode P
608
609 assign IRQ_Msk = P[pIntMsk];            // Interrupt Mask Bit
610
611 //  External Bus Data Output
612
613 assign MuxDat = ((PHW) ? {OP2, OP1} : ((PHR) ? MAR : PC));
614
615 assign OutMux = ((DO_Op[0]) ? DO_ALU       : 0);
616 assign OutMux = ((DO_Op[1]) ? MuxDat[15:8] : 0);
617 assign OutMux = ((DO_Op[2]) ? MuxDat[ 7:0] : 0);
618 assign OutMux = ((DO_Op[3]) ? P            : 0);
619
620 assign DO = OutMux;
621
622 //  Assign External Interrupt Handler Control Signals
623
624 assign LE_Int = BRV2;
625
626 ////////////////////////////////////////////////////////////////////////////
627 //
628 //  End Implementation
629 //
630 ////////////////////////////////////////////////////////////////////////////
631
632 endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
//  Microprogram Sequencer module for M65C02A soft-core microcomputer project.
//
//  Copyright 2013-2014 by Michael A. Morris, dba M. A. Morris & Associates
//
//  All rights reserved. The source code contained herein is publicly released
//  under the terms and conditions of the GNU General Public License as conveyed
//  in the license provided below.
//
//  This program is free software: you can redistribute it and/or modify it
//  under the terms of the GNU General Public License as published by the Free
//  Software Foundation, either version 3 of the License, or any later version.
//
//  This program is distributed in the hope that it will be useful, but WITHOUT
//  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
//  more details.
//
//  You should have received a copy of the GNU General Public License along with
//  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
//
//  Free Software Foundation, Inc.
//  51 Franklin Street, Fifth Floor
//  Boston, MA  02110-1301 USA
//
//  Further, no use of this source code is permitted in any form or means
//  without inclusion of this banner prominently in any derived works.
//
//  Michael A. Morris <morrisma_at_mchsi_dot_com>
//  164 Raleigh Way
//  Huntsville, AL 35811
//  USA
//
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:         M. A. Morris & Associates
// Engineer:        Michael A. Morris
//
// Create Date:     12:02:40 10/28/2012
// Design Name:     WDC W65C02 Microprocessor Re-Implementation
// Module Name:     M65C02_MPCv5.v
// Project Name:    C:\XProjects\ISE10.1i\M65C02A
// Target Devices:  Generic SRAM-based FPGA
// Tool versions:   Xilinx ISE 10.1i SP3
//
// Description:
//
// This module implements a simple microprogram sequencer based on the Fair-
// child F9408. The sequencer provides:
//
//          (1) 4-bit instruction input
//          (2) four-level LIFO stack;
//          (3) program counter and incrementer;
//          (4) 4-bit registered test input;
//          (5) 8-way multi-way branch control input;
//          (6) branch address input;
//          (7) 4-way branch address select output;
//          (8) next address output.
//
// These elements provide a relatively flexible general purpose microprogram
// controller without a complex instruction set. The sixteen instructions can
// be categorized into three classes: (1) fetch, (2) unconditional branches,
// and (3) conditional branches. The fetch instruction class, a single instruc-
// tion class, simply increments the program counter and outputs the current
// value of the program counter on the next address bus. The unconditional
// branch instruction class provides instructions to select the next instruc-
// tion using the Via[1:0] outputs and output that value on the next address
// bus and simultaneously load the program counter. The unconditional branch
// instruction class also provides for 8-way multiway branching using an exter-
// nal (priority) encoder/branch selector, and microprogram subroutine call and
// return instructions.
//
// The instruction encodings of the F9408, as provided in "Principles of Firm-
// ware Engineering in Microprogram Control" by Michael Andrews. The instruc-
// tion set and operation map for the implementation is given below:
```

```
 80 //
 81 //  I[3:0] MNEM Definition       T[3:0]      MA[m:0]      Via Inh  Operation
 82 //   0000  RTS  Return            xxxx      TOS[m:0]       00  0  PC<=MA;Pop
 83 //   0001  BSR  Call Subroutine   xxxx       BA[m:0]       00  1  PC<=MA;Push
 84 //   0010  FTCH Next Instruction  xxxx        PC+1         00  0  PC<=MA[m:0]
 85 //   0011  BMW  Multi-way Branch  xxxx  {BA[m:3],MW[2:0]}  00  1  PC<=MA[m:0]
 86 //   0100  BRV0 Branch Via 0      xxxx       BA[m:0]       00  1  PC<=MA[m:0]
 87 //   0101  BRV1 Branch Via 1      xxxx       BA[m:0]       01  1  PC<=MA[m:0]
 88 //   0110  BRV2 Branch Via 2      xxxx       BA[m:0]       10  1  PC<=MA[m:0]
 89 //   0111  BRV3 Branch Via 3      xxxx       BA[m:0]       11  1  PC<=MA[m:0]
 90 //   1000  BTH0 Branch T0 High    xxx1  {T0?BA[m:0]:PC+1}  00  1  PC<=MA[m:0]
 91 //   1001  BTH1 Branch T1 High    xx1x  {T1?BA[m:0]:PC+1}  00  1  PC<=MA[m:0]
 92 //   1010  BTH2 Branch T2 High    x1xx  {T2?BA[m:0]:PC+1}  00  1  PC<=MA[m:0]
 93 //   1011  BTH3 Branch T3 High    1xxx  {T2?BA[m:0]:PC+1}  00  1  PC<=MA[m:0]
 94 //   1100  BTL0 Branch T0 Low     xxx0  {T0?PC+1:BA[m:0]}  00  1  PC<=MA[m:0]
 95 //   1101  BTL1 Branch T1 Low     xx0x  {T1?PC+1:BA[m:0]}  00  1  PC<=MA[m:0]
 96 //   1110  BTL2 Branch T2 Low     x0xx  {T2?PC+1:BA[m:0]}  00  1  PC<=MA[m:0]
 97 //   1111  BTL3 Branch T3 Low     0xxx  {T3?PC+1:BA[m:0]}  00  1  PC<=MA[m:0]
 98 //
 99 // Dependencies:    none.
100 //
101 // Revision:
102 //
103 //  1.00    13I14   MAM     Removed embedded microcycle controller from M65C02_
104 //                          MPCv3.v. Added parameter to control the implementa-
105 //                          tion of a one level or a four level return stack.
106 //
107 // Additional Comments:
108 //
109 //  The Version 5 Microprogram Controller (M65C02_MPCv5) is based on the Fair-
110 //  child F9408 MPC. It extends that microprogram controller by incorporating
111 //  an microcycle enable input which allows each microcycle to be controlled by
112 //  external logic.
113 //
114 //  The purpose of these extensions is to allow easy implementation of a varia-
115 //  ble length microprogram cycle, i.e. microcycle. In turn, this simplifies the
116 //  implementation of microprogrammed state machines which interface to synchro-
117 //  nous memories found in most FPGAs, or to external synchronous/asynchronous
118 //  memories.
119 //
120 //  When a microprogrammed state machine interfaces to a synchronous memory,
121 //  there is a one cycle delay (or more) between the presentation of the address
122 //  and the output of the data at that address. In many instances, the micro-
123 //  program is unable to perform any useful work during the first cycle. Thus,
124 //  the microprogram must perform an explicit delay operation, which generally
125 //  requires a state to be added to every read of these memories. If there are
126 //  a significant number of these read operations in the microprogram, then
127 //  there is an opportunity for the microprogram to be incorrectly programmed
128 //  when one or more of the delay cycles are not included in the microprogram.
129 //  Isolating the resulting fault in the state machine may be difficult.
130 //
131 //  To avoid errors of this type, microcycles which read from or write to
132 //  devices, such as memories, can be automatically extended explicitly by a
133 //  microprogram field or external logic. Using this type of facility reduces
134 //  the number of states required to interface a microprogrammed state machine
135 //  to these types of devices. It also makes the microprogram less tedious to
136 //  develop and improves overall productivity, which is a prime reason for
137 //  choosing a microprogrammed approach for developing complex state machines.
138 //
139 ////////////////////////////////////////////////////////////////////////////////
140
141 module M65C02_MPCv5 #(
142     parameter pAddrWidth = 10,          // Original F9408 => 10-bit Address
143     parameter pMPC_Stk   = 1'b0,        // MPC Stack Depth
144     parameter pRst_Addrs = 0            // Reset Address
145 )(
146     input   Rst,                        // Module Reset (Synchronous)
147     input   Clk,                        // Module Clock
148
149     input   Rdy,                        // Rdy - MPC Clock Enable
150
151     input   [3:0] I,                    // Instruction (see description)
152     input   [3:0] T,                    // Conditional Test Inputs
153     input   [2:0] MW,                   // Multi-way Branch Address Select
154     input   [(pAddrWidth-1):0] BA,      // Microprogram Branch Address Field
155
156     output  [1:0] Via,                  // Unconditional Branch Address Select
157     output  [(pAddrWidth-1):0] MA       // Microprogram Address
158 );
```

```
159
160 //////////////////////////////////////////////////////////////////////////////
161 //
162 //  Local Parameters
163 //
164
165 localparam pRTS  =  0;  // Return from Subroutine
166 localparam pBSR  =  1;  // Branch to Subroutine
167 localparam pFTCH =  2;  // Fetch Next Instruction
168 localparam pBMW  =  3;  // Multi-way Branch
169 localparam pBRV0 =  4;  // Branch Via External Branch Address Source #0
170 localparam pBRV1 =  5;  // Branch Via External Branch Address Source #1
171 localparam pBRV2 =  6;  // Branch Via External Branch Address Source #2
172 localparam pBRV3 =  7;  // Branch Via External Branch Address Source #3
173 localparam pBTH0 =  8;  // Branch if T[0] is Logic 1, else fetch next instr.
174 localparam pBTH1 =  9;  // Branch if T[1] is Logic 1, else fetch next instr.
175 localparam pBTH2 = 10;  // Branch if T[2] is Logic 1, else fetch next instr.
176 localparam pBTH3 = 11;  // Branch if T[3] is Logic 1, else fetch next instr.
177 localparam pBTL0 = 12;  // Branch if T[0] is Logic 0, else fetch next instr.
178 localparam pBTL1 = 13;  // Branch if T[1] is Logic 0, else fetch next instr.
179 localparam pBTL2 = 14;  // Branch if T[2] is Logic 0, else fetch next instr.
180 localparam pBMW3 = 15;  // Multi-way Branch on T[3]
181
182 //////////////////////////////////////////////////////////////////////////////
183 //
184 //  Declarations
185 //
186
187 reg     dRst;                           // Reset stretcher
188 wire    MPC_Rst;                        // Internal MPC Reset signal
189
190 wire    [(pAddrWidth - 1):0] Next;      // Output Program Counter Incrementer
191 reg     [(pAddrWidth - 1):0] PC_In;     // Input to Program Counter
192 reg     [(pAddrWidth - 1):0] PC;        // Program Counter
193
194 //////////////////////////////////////////////////////////////////////////////
195 //
196 //  Implementation
197 //
198
199 //  Implement module reset generator
200
201 always @(posedge Clk)
202 begin
203     if(Rst)
204         dRst <= #1 1;
205     else
206         dRst <= #1 0;
207 end
208
209 assign MPC_Rst = (Rst | dRst);
210
211 //  Implement Return Stack
212
213 reg [(pAddrWidth - 1):0] A;               // LIFO Stack Top-Of-Stack Reg
214
215 generate
216     if(pMPC_Stk) begin  //  Implement 4-Level LIFO Stack
217         reg [(pAddrWidth - 1):0] B, C, D;   // LIFO Stack Extra Registers
218
219         always @(posedge Clk)
220         begin
221             if(MPC_Rst)
222                 {A, B, C, D} <= #1 0;
223             else if(Rdy)
224                 if(I == pBSR)
225                     {A, B, C, D} <= #1 {Next, A, B, C};
226                 else if(I == pRTS)
227                     {A, B, C, D} <= #1 {B, C, D, {pAddrWidth{1'b0}}};
228         end
229     end else begin      //  Implement 1-Level LIFO Stack
230         always @(posedge Clk)
231         begin
232             if(MPC_Rst)
233                 A <= #1 0;
234             else if(Rdy)
235                 if(I == pBSR)
236                     A <= #1 Next;
237                 else if(I == pRTS)
```

```
238                     A <= #1 {pAddrWidth{1'b0}};
239         end
240     end
241 endgenerate
242 
243 //  Program Counter Incrementer
244 
245 assign Next = PC + 1;
246 
247 //  Generate Unconditional Branch Address Select
248 
249 assign Via = ((I[3:2] == 1) ? I[1:0] : 0);
250 
251 //  Generate Program Counter Input Signal
252 
253 always @(*)
254 begin
255     if(MPC_Rst)
256         PC_In <= pRst_Addrs;
257     else
258         case(I)
259             4'b0000 : PC_In <= A;                                 // pRTS
260             4'b0001 : PC_In <= BA;                                // pBSR
261             4'b0010 : PC_In <= Next;                              // pFTCH
262             4'b0011 : PC_In <= {BA[(pAddrWidth - 1):3], MW};      // pBMW
263             4'b0100 : PC_In <= BA;                                // pBRV0
264             4'b0101 : PC_In <= BA;                                // pBRV1
265             4'b0110 : PC_In <= BA;                                // pBRV2
266             4'b0111 : PC_In <= BA;                                // pBRV3
267             4'b1000 : PC_In <= (T[0] ? BA   : Next);              // pBTH0
268             4'b1001 : PC_In <= (T[1] ? BA   : Next);              // pBTH1
269             4'b1010 : PC_In <= (T[2] ? BA   : Next);              // pBTH2
270             4'b1011 : PC_In <= (T[3] ? BA   : Next);              // pBTH3
271             4'b1100 : PC_In <= (T[0] ? Next : BA  );              // pBTL0
272             4'b1101 : PC_In <= (T[1] ? Next : BA  );              // pBTL1
273             4'b1110 : PC_In <= (T[2] ? Next : BA  );              // pBTL2
274             4'b1111 : PC_In <= {BA[(pAddrWidth - 1):1], T[3]};    // pBMW3
275         endcase
276 end
277 
278 //  Generate Microprogram Address (Program Counter)
279 
280 always @(posedge Clk)
281 begin
282     if(MPC_Rst)
283         PC <= #1 pRst_Addrs;
284     else if(Rdy)
285         PC <= #1 PC_In;
286 end
287 
288 //  Assign Memory Address Bus
289 
290 assign MA = PC_In;
291 
292 endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
//  Address Generator module for M65C02A soft-core microcomputer project.
//

//
//  Michael A. Morris <morrisma_at_mchsi_dot_com>
//  164 Raleigh Way
//  Huntsville, AL 35811
//  USA
//
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 100ps

////////////////////////////////////////////////////////////////////////////////
// Company:         M. A. Morris & Associates
// Engineer:        Michael A. Morris
//
// Create Date:     19:59:21 09/18/2013
// Design Name:     WDC W65C02 Microprocessor Re-Implementation
// Module Name:     M65C02_AddrGenV2.v
// Project Name:    C:\XProjects\ISE10.1i\M65C02A
// Target Devices:  Generic SRAM-based FPGA
// Tool versions:   Xilinx ISE10.1i SP3
//
// Description:
//
//  This file provides the M65C02_Core module's address generator function. This
//  module is taken from the address generator originally included in the
//  M65C02_Core module. The only difference is the addition of an explicit sig-
//  nal which generates relative offset for conditional branches, Rel.
//
// Dependencies:    none
//
// Revision:
//
//  1.00    13I18   MAM     Initial creation of the file from M65C02_AddrGen.
//
//  1.10    13J08   MAM     Modifying the next address generation logic to fold
//                          NA_Op, PC_Op, Stk_Op, and ZP fields into a single
//                          one-hot control field that can replace these fields
//                          in the uPgm microcode.
//
//  2.00    14F28   MAM     Added comments to reflect changes made during inte-
//                          gration and verification of the module: (1) changed
//                          the one-hot NA_Op control field from 13 to 11 bits;
//                          (2) added explicit external inputs for address gene-
//                          mod 256 and page control; (3) added explicit control
//                          NA_Op[10] to control the loading of the PC; (4) add-
//                          ed explicit control NA_Op[9] to control the future
//                          stack relative instructions; (4) reduced the size of
//                          the left operand multiplexer; (5) made NA calcula-
//                          tion as a single 16-bit operation rather than split
//                          into two separate 8-bit operations with a control on
//                          the carry propagation between the halves; (6) imple-
```

```
 80 //                          mented the % 256 function using a multiplexer, and
 81 //                          applied to both page 0 (data) and page 1 (stack);
 82 //                          (7) changed MAR to only register NA on Rdy - removed
 83 //                          need to load Vector, etc. on Rst; (8) removed Rst
 84 //                          from PC which allows the modification to the Reset
 85 //                          behavior of the core to push the PC and PSW to the
 86 //                          stack in locations Stk[2:0] <= {PCH, PCL, PSW}; and
 87 //                          (9) adjusted the stack pointer module inputs to re-
 88 //                          flect changes to NA_Op to support stack relative
 89 //                          addressing.
 90 //
 91 //  2.01    14H09   MAM     Added comment line describing how (sp,S),Y address-
 92 //                          ing is implemented, added the {0, OP1} to AR to im-
 93 //                          plement stack relative addressing, and changed name
 94 //                          of StkRel signal to Stk_Rel.
 95 //
 96 // Additional Comments:
 97 //
 98 ////////////////////////////////////////////////////////////////////////////////
 99
100 module M65C02_AddrGenV2 #(
101     parameter pStkPtr_Rst = 2       // Init SP to push PCH, PCL, P on reset
102 )(
103     input   Rst,                    // System Reset
104     input   Clk,                    // System Clock
105
106     input   [15:0] Vector,          // Interrupt/Trap Vector
107
108     input   [10:0] NA_Op,           // Address Generator Operation Select
109
110     input   Mod256,                 // Mod 256 Control - wrap address to page
111     input   Page,                   // Address wrap page select
112
113     input   CC,                     // Conditional Branch Input Flag
114     input   BRV3,                   // Interrupt or Next Instruction Select
115     input   Int,                    // Unmasked Interrupt Request Input
116
117     input   Rdy,                    // Ready Input
118     input   Valid,                  // Data Valid
119
120     input   [7:0] OP1,              // Operand Register 1 Input
121     input   [7:0] OP2,              // Operand Register 2 Input
122
123     input   [7:0] X,                // X Index Register Input
124     input   [7:0] Y,                // Y Index Register Input
125
126     output  reg [15:0] AO,          // Address Output
127
128     input   SelS,                   // Stack Pointer Select
129     output  [7:0] S,                // Stack Pointer Register
130
131     output  reg [15:0] MAR,         // Program Counter
132     output  reg [15:0] PC           // Program Counter
133 );
134
135 ////////////////////////////////////////////////////////////////////////////////
136 //
137 //  Module Declarations
138 //
139
140 wor     [15:0] AL, AR;          // Wired-OR busses for address operands
141 wire    [15:0] NA;              // Next Address (w/o application of % 256)
142
143 wire    CE_MAR;                 // Memory Address Register Clock Enable
144
145 wire    [15:0] Rel;             // Branch Address Sign-Extended Offset
146 wire    CE_PC;                  // Program Counter Clock Enable
147
148 ////////////////////////////////////////////////////////////////////////////////
149 //
150 //  Implementation
151 //
152
153 //  Next Address Generator
154
155 //  Decode Next Address Operation Field
156
157 //           LO PSZAM XYR C
158 //           Pf CtPbA   e i
```

```
159 //           Cf  k sR   l
160 // Vec:  11'b10_00000_000_0; // NA <= {OP2,OP1}           + 0; PC <= NA
161 // Jmp:  11'b10_00010_000_0; // NA <= {OP2,OP1}           + 0; PC <= NA
162 // JmpY: 11'b10_00010_010_0; // NA <= {OP2,OP1} + {0,  Y} + 0; PC <= NA
163 // Rtn:  11'b10_00010_000_1; // NA <= {OP2,OP1}           + 1; PC <= NA
164 // PC:   11'b10_10000_000_0; // NA <= PC                  + 0; PC <= NA
165 // Inc:  11'b10_10000_000_1; // NA <= PC                  + 1; PC <= NA
166 // Bra:  11'b10_10000_001_0; // NA <= PC        + Rel     + 1; PC <= NA
167 // Rel:  11'b00_10000_001_0; // NA <= PC        + Rel     + 1;
168 // Psh:  11'b00_01000_000_0; // NA <= {  1, SP}           + 0;
169 // Pop:  11'b00_01000_000_1; // NA <= {  1, SP}           + 1;
170 // SPN:  11'b01_01000_000_0; // NA <= {  1, SP} + {0,OP1} + 1;
171 // DPN:  11'b00_00100_000_0; // NA <= {  0,OP1}           + 0;
172 // DPX:  11'b00_00100_100_0; // NA <= {  0,OP1} + {0,  X} + 0;
173 // DPY:  11'b00_00100_010_0; // NA <= {  0,OP1} + {0,  Y} + 0;
174 // LDA:  11'b00_00010_000_0; // NA <= {OP2,OP1}           + 0;
175 // LDAX: 11'b00_00010_100_0; // NA <= {OP2,OP1} + {0,  X} + 0;
176 // LDAY: 11'b00_00010_010_0; // NA <= {OP2,OP1} + {0,  Y} + 0;
177 // Nxt:  11'b00_00001_000_0; // NA <= MAR                 + 1;
178 // MAR:  11'b00_00001_000_1; // NA <= MAR                 + 0;
179 //
180
181 assign Ld_PC   = NA_Op[10];
182 assign Stk_Rel = NA_Op[ 9];
183 //
184 assign Sel_PC  = NA_Op[ 8];
185 assign Sel_SP  = NA_Op[ 7];
186 assign Sel_ZP  = NA_Op[ 6];
187 assign Sel_Abs = NA_Op[ 5];
188 assign Sel_MAR = NA_Op[ 4];
189 //
190 assign Sel_X   = NA_Op[ 3];
191 assign Sel_Y   = NA_Op[ 2];
192 assign Sel_Rel = NA_Op[ 1];
193 //
194 assign Ci      = NA_Op[ 0];
195
196 //  Generate Relative Address
197
198 assign Rel = ((CC) ? {OP2, OP1} : 0);
199
200 //  Generate Left Address Operand
201
202 assign AL = ((Sel_PC ) ? PC           : 0);
203 assign AL = ((Sel_SP ) ? {8'h01, S  } : 0);
204 assign AL = ((Sel_ZP ) ? {8'h00, OP1} : 0);
205 assign AL = ((Sel_Abs) ? {OP2  , OP1} : 0);
206 assign AL = ((Sel_MAR) ? MAR          : 0);
207
208 //  Generate Right Address Operand
209
210 assign AR = ((Sel_X  ) ? {8'h00,   X} : 0);
211 assign AR = ((Sel_Y  ) ? {8'h00,   Y} : 0);
212 assign AR = ((Sel_Rel) ? Rel          : 0);
213 assign AR = ((Stk_Rel) ? {8'h00, OP1} : 0);
214
215 //  Compute Next Address
216
217 assign NA = (AL + AR + Ci);
218
219 //  Generate Address Output - Truncate Next Address when Mod256 asserted
220 //      When Mod256 is asserted, the memory page is determined by Page
221
222 always @(*)
223 begin
224     if(Rst)
225         AO <= Vector;
226     else
227         AO <= ((Mod256) ? {{{7{1'b0}}, Page}, NA[7:0]} : NA);
228 end
229
230 //  Memory Address Register
231
232 assign CE_MAR = Rdy & ~(Sel_SP & ~Stk_Rel);
233
234 always @(posedge Clk)
235 begin
236     if(CE_MAR)
237         MAR <= #1 AO;
```

```
238 end
239
240 //  Program Counter
241
242 assign CE_PC = Rdy & ((BRV3) ? (Ld_PC & ~Int) : Ld_PC);
243
244 always @(posedge Clk)
245 begin
246     if(CE_PC)
247         PC <= #1 AO;
248 end
249
250 //  Instantiate Stack Pointer
251
252 M65C02_StkPtr   #(
253                     .pStkPtr_Rst(pStkPtr_Rst)
254                 ) Stk (
255                     .Rst(Rst),
256                     .Clk(Clk),
257
258                     .Rdy(Rdy),
259                     .Valid(Valid),
260
261                     .SelS(SelS),
262                     .Stk_Op({Sel_SP & ~Stk_Rel, Ci}),
263                     .X(X),
264
265                     .S(S)
266                 );
267
268 endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
//  Stack Pointer module for M65C02A soft-core microcomputer project.
//
//  Copyright 2013-2014 by Michael A. Morris, dba M. A. Morris & Associates
//
//  All rights reserved. The source code contained herein is publicly released
//  under the terms and conditions of the GNU General Public License as conveyed
//  in the license provided below.
//
//  This program is free software: you can redistribute it and/or modify it
//  under the terms of the GNU General Public License as published by the Free
//  Software Foundation, either version 3 of the License, or any later version.
//
//  This program is distributed in the hope that it will be useful, but WITHOUT
//  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
//  more details.
//
//  You should have received a copy of the GNU General Public License along with
//  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
//
//  Free Software Foundation, Inc.
//  51 Franklin Street, Fifth Floor
//  Boston, MA  02110-1301 USA
//
//  Further, no use of this source code is permitted in any form or means
//  without inclusion of this banner prominently in any derived works.
//
//  Michael A. Morris <morrisma_at_mchsi_dot_com>
//  164 Raleigh Way
//  Huntsville, AL 35811
//  USA
//
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:         M. A. Morris & Associates
// Engineer:        Michael A. Morris
//
// Create Date:     12:02:40 10/28/2012
// Design Name:     WDC W65C02 Microprocessor Re-Implementation
// Module Name:     M65C02_StkPtr.v
// Project Name:    C:\XProjects\ISE10.1i\M65C02A
// Target Devices:  Generic SRAM-based FPGA
// Tool versions:   Xilinx ISE 10.1i SP3
//
// Description:
//
//  This module implements the functions of the M65C02 stack pointer. The imple-
//  mentation is taken from the M65C02_ALU module so that these functions can be
//  moved to the M65C02_AddrGen module.
//
// Dependencies:    none.
//
// Revision:
//
//  1.00    13I14   MAM     Implementation pulled from M65C02_ALU.
//
//  1.10    14F28   MAM     Adjusted comments to reflect changes implemented
//                          to integration: (1) changed default for stack poin-
//                          ter to support the core's new reset behavior; and
//                          (2) corrected typos in the names of the controls.
//
// Additional Comments:
//
//  The stack pointer register is a loadable up/down counter. Valid and Rdy are
//  used to generate a local clock enable for the counter. The StkOp input from
//  the microprogram controls the functions implemented.
//
//  The stack operations supported are: hold, rsvd, ++S, and S--. The stack
//  pointer can only be loaded from the X index register. Similarly, the stack
//  pointer can only be transfered to the X index register. The stack pointer
//  points to an open location on the stack. Thus, push operations write the
//  value at the location pointed to by S, and post-decrements S, S--. Stack
//  pull operations require the value to be incremented by one, ++S, before
//  that location can be read into OP1 and subsequently written to one of four
```

```
 80 //  registers: P, A, X, or Y.
 81 //
 82 //  The stack pointer control, StkOp, field is generated by the execution
 83 //  engine:
 84 //
 85 //  00  :   Hold;
 86 //  01  :   Rsvd;
 87 //  10  :   S--;
 88 //  11  :   ++S;
 89 //
 90 //  A separate output multiplexer with a built-in incrementer is used to imple-
 91 //  ment the
 92 //
 93 //
 94 ////////////////////////////////////////////////////////////////////////////////
 95
 96 module M65C02_StkPtr #(
 97     parameter pStkPtr_Rst = 2
 98 )(
 99     input   Rst,
100     input   Clk,
101
102     input   Rdy,
103     input   Valid,
104
105     input   SelS,
106     input   [1:0] Stk_Op,
107     input   [7:0] X,
108
109     output  reg [7:0] S
110 );
111
112 ////////////////////////////////////////////////////////////////////////////////
113 //
114 //  Implementation
115 //
116
117 //  Stack Pointer
118
119 assign Ld_S = Rdy & (SelS & Valid);
120 assign CE_S = Rdy & Stk_Op[1];
121
122 always @(posedge Clk)
123 begin
124     if(Rst)
125         S <= #1 pStkPtr_Rst;
126     else if(Ld_S)
127         S <= #1 X;                          // TXS
128     else if(CE_S)
129         S <= #1 ((Stk_Op[0]) ? (S + 1)      // Pop
130                              : (S - 1));    // Push
131 end
132
133 endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
//  ALU module for M65C02A soft-core microcomputer project.
//
//  Copyright 2013-2014 by Michael A. Morris, dba M. A. Morris & Associates
//
//  All rights reserved. The source code contained herein is publicly released
//  under the terms and conditions of the GNU General Public License as conveyed
//  in the license provided below.
//
//  This program is free software: you can redistribute it and/or modify it
//  under the terms of the GNU General Public License as published by the Free
//  Software Foundation, either version 3 of the License, or any later version.
//
//  This program is distributed in the hope that it will be useful, but WITHOUT
//  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
//  more details.
//
//  You should have received a copy of the GNU General Public License along with
//  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
//
//  Free Software Foundation, Inc.
//  51 Franklin Street, Fifth Floor
//  Boston, MA  02110-1301 USA
//
//  Further, no use of this source code is permitted in any form or means
//  without inclusion of this banner prominently in any derived works.
//
//  Michael A. Morris <morrisma_at_mchsi_dot_com>
//  164 Raleigh Way
//  Huntsville, AL 35811
//  USA
//
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:         M. A. Morris & Associates
// Engineer:        Michael A. Morris
//
// Create Date:     07:00:31 11/25/2009
// Design Name:     WDC W65C02 Microprocessor Re-Implementation
// Module Name:     M65C02_ALUv2.v
// Project Name:    C:\XProjects\ISE10.1i\M65C02A
// Target Devices:  Generic SRAM-based FPGA
// Tool versions:   Xilinx ISE10.1i SP3
//
// Description:
//
//  This module implements the ALU for the M65C02 microprocessor. Included in
//  the ALU are the accumulator (A), pre/post-index registers (X and Y), and
//  processor status word (P). The microcode, composed of a fixed part from the
//  instruction decoder (IDEC) and a variable portion from the execution engine,
//  control a Load/Store/Transfer (LST) multiplexer, a Logic Unit (LU), a Shift/
//  Rotate Unit (SU), and a dual-mode (binary/BCD) Adder Unit (AU).
//
//  The Load/Store/Transfer (LST) multiplexer allows the various registers in
//  the ALU, plus the external stack pointer (S) in the address generator module
//  (from the M65C02 Address Generator), to be multiplexed onto the ALU output
//  data bus. This allows registers to be loaded from memory (data memory or the
//  external data/return stack), stored to memory, or transfered from one regis-
//  ter to another.
//
//  The LU performs bit-wise operations (OR, AND, and XOR) on two operands. The
//  LU also supports the following instructions which only use variations of the
//  LU's basic functions: BIT/TRB/TSB, RMB/SMB/BBR/BBS, CLC/SEC/CLI/SEI/CLD/SED/
//  CLV, and REP/SEP
//
//  The SU performs arithmetic and logical operations using non-circular or cir-
//  cular shifts on a register (A) or a memory (M) operand. The four basic in-
//  structions, ASL/ROL/LSR/ROR, can be implemented using just two operations:
//  left shift and right shift. Various combinations of the operand and the
//  carry logic signal paths allow the non-circular shifts and circular shifts
//  (rotates) needed to implement these four instructions. The ALU can easily
//  support arithmetic overflow detection on the ASL instruction, and support an
//  arithmetic right shift (ASR) through modifications to the microprogram ROMs.
//
```

```
//  The AU performs binary two's complement addition, or unsigned BCD addition/
//  subtraction on two operands. Subtraction is performed using 1's complement
//  logic. A 2's complement result, extended precision addition or subtraction
//  is performed if the Carry flag, C, is cleared before addition and set before
//  subtraction. The AU is also used for increments, decrements, and compares.
//  For these operations, the arithmetic mode is fully 2's complement.
//
//  The ALU requires the En input to be asserted to initiate the operation that
//  the fixed IDEC microword determines. Asserting the En input indicates that
//  the required operands for an operation are available. The addressing mode of
//  the instruction will determine when En is asserted. In the case of non-
//  program flow control instructions, En alone controls the updates to these
//  registers: A, X, Y, P. For implicit or accumulator mode instructions, En is
//  generally asserted when the Instruction Register (IR) is loaded with the op-
//  code. Otherwise, it is asserted when the required memory operand is availa-
//  ble from memory (M). The operation defined by IDEC should be completed with
//  the available operands in the ALU registers and the memory operand (M).
//
//  With the En input asserted, then the Rdy input controls the actual writing
//  of registers. To write a register, the En and Rdy inputs must be asserted,
//  and the register must be selected. A register is written if its code is set
//  in the WSel field and the Reg_WE field is set to the pWr_Reg control value,
//  or if the register's select code is set in the Reg_WE field. A second level
//  ROM in the ALU provides the decoding of this logic and generates write
//  selects for each of the ALU registers (A, X, Y, P) and for S in the address
//  generator module.
//
//  The table below presents all of the valid combinations of the FU_Sel and Op
//  fields. The FU_Sel field selects the functional unit that implements the
//  desired operation. The Op input selects the operation to be performed by LU,
//  SU, and AU functional units.
//
//      LST: LST = {M, S, P, T, Y, X, A, 0}
//      LU : LU  = {P, K, A, M}, R = {P, K, A, M}; Ci = {1, 0, Q[7], C}
//      SU : SU  = {Y, X, A, M},                   Ci = {1, 0, Q[7], C}
//      AU : Q   = {Y, X, A, M}, R = {P, K, A, M}; Ci = {1, 0, Q[7], C}
//
//   FU_Sel  Op  Mnemonic     Operation           Condition Code
//
//  11_00_00 00    XFR    ALU <= {OSel=(7..0): M, S, P, T, Y, X, A, 0}
//
//  10_10_00 01    AND    ALU <=  A & M;     N <= ALU[7]; Z <= ~|ALU;
//  10_10_00 10    ORA    ALU <=  A | M;     N <= ALU[7]; Z <= ~|ALU;
//  10_10_00 11    EOR    ALU <=  A ^ M;     N <= ALU[7]; Z <= ~|ALU;
//
//  10_10_00 01    BIT    ALU <=  A & M;     N <= M[7];   Z <= ~|(A & M);
//                                           V <= M[6];
//  10_10_00 00    TRB    ALU <= ~A & M;                  Z <= ~|(A & M);
//  10_10_00 10    TSB    ALU <=  A | M;                  Z <= ~|(A & M);
//
//  10_10_00 00    RMBx   ALU <= ~K & M;     K <= (1 << IR[6:4]);
//  10_10_00 10    SMBx   ALU <=  K | M;     K <= (1 << IR[6:4]);
//  10_10_00 01    BBRx   ALU <=  K & M;     K <= (1 << IR[6:4]); CC_Out <= EQ;
//  10_10_00 01    BBSx   ALU <=  K & M;     K <= (1 << IR[6:4]); CC_Out <= NE;
//
//  10_10_00 00    CLC    ALU <= ~K & P;     K <= 1;    P <= ALU
//  10_10_00 10    SEC    ALU <=  K | P;     K <= 1;    P <= ALU
//  10_10_00 00    CLI    ALU <= ~K & P;     K <= 4;    P <= ALU
//  10_10_00 10    SEI    ALU <=  K | P;     K <= 4;    P <= ALU
//  10_10_00 00    CLD    ALU <= ~K & P;     K <= 8;    P <= ALU
//  10_10_00 10    SED    ALU <=  K | P;     K <= 8;    P <= ALU
//  10_10_00 00    CLV    ALU <= ~K & P;     K <= 64;   P <= ALU
//
//  10_10_00 00    REP    ALU <= ~M & P;                P <= ALU
//  10_10_00 00    SEP    ALU <=  M | P;                P <= ALU
//
//  10_01_00 00    ASL    ALU <= {R[6:0],Ci} N <= ALU[7]; Z <= ~|ALU; C <= R[7];
//                                           Ci <= 0; (V <= R[7] ^ R[6];)
//  10_01_00 00    ROL    ALU <= {R[6:0],Ci} N <= ALU[7]; Z <= ~|ALU; C <= R[7];
//                                           Ci <= C; (V <= R[7] ^ R[6];)
//  10_01_00 01    LSR    ALU <= {Ci,R[7:1]} N <= ALU[7]; Z <= ~|ALU; C <= R[0];
//                                           Ci <= 0; (Ci <= R[7])
//  10_01_00 01    ROR    ALU <= {Ci,R[7:1]} N <= ALU[7]; Z <= ~|ALU; C <= R[0];
//                                           Ci <= C;
//
//  10_00_10 00    ADC    ALU <= Q +  M + C; N <= ALU[7]; Z <= ~|ALU;
//                                           V <= OV;     C <= COut;
//  10_00_10 01    SBC    ALU <= Q + ~M + C; N <= ALU[7]; Z <= ~|ALU;
//                                           V <= OV;     C <= COut;
```

```
159 //  10_00_01 00    INC    ALU <= Q +  0 + 1; N <= ALU[7]; Z <= ~|ALU;
160 //  10_00_01 01    DEC    ALU <= Q + ~0 + 0; N <= ALU[7]; Z <= ~|ALU;
161 //  10_00_01 01    CMP    ALU <= Q + ~M + 1; N <= ALU[7]; Z <= ~|ALU;
162 //                                                       C <= COut;
163 //
164 //  Although the condition codes modified by the ALU are shown above, the CCSel
165 //  input field will actually control loading condition codes into P register.
166 //  This is especially true for the N and V condition codes with respect to the
167 //  BIT instruction. N and V are modified by BIT as indicated above, except for
168 //  for the BIT #imm instruction which only modifies Z, and leaves N and V un-
169 //  changed like the TRB and TSB instructions.
170 //
171 //  The ALU is controlled using a microprogrammed architecture. Explicit control
172 //  over the functional units, functional unit operations, left operand select,
173 //  right operand select, and carry select provide the means by which the M65C02
174 //  execution unit implements the execution phase of each instruction. A func-
175 //  tional unit is selected by the FU_Sel field. A single bit in the FU_Sel is
176 //  used to select the LST data multiplexer, the logic unit (LU), and the shift
177 //  unit (SU). The AU has two associated FU_Sel bits. One is used to select the
178 //  AU only when INC, DEC, and CMP functions, i.e. IDC select. The other is used
179 //  to select the AU when the ADC and SBC functions are needed. The decimal mode
180 //  of the 6502/65C02 only applies to ADC/SBC instructions, so the FU_Sel bit
181 //  for ADC/SBC, ADD select, is qualified by the Decimal flag in the PSW. If D
182 //  is set, then a decimal mode ADC/SBC is performed. If D is not set, then the
183 //  binary mode of the adder is used to INC/DEC/CMP and ADC/SBC.
184 //
185 //  The LST multiplexer's operands are selected by the OSel field. The operands
186 //  of the LU, SU, and AU are selected by the QSel, RSel, and CSel fields. These
187 //  fields are two bit fields, which allow them to select 1 of 4 operands. The
188 //  QSel is used for the left operand of the LU and AU, and for the single
189 //  operand of the SU. There are different combinations of the left operands for
190 //  the LU and the AU, so in essence there are two left operand multiplexers
191 //  controlled a common set of control signals. The CSel selects the carry input
192 //  into both the SU and the AU.
193 //
194 //  The following table provides the mapping of the QSel, RSel, and CSel fields
195 //  to the various operands to the functional units.
196 //
197 //      QSel    RSel    QSel    CSel    QSel    RSel    CSel
198 //      LU-L    LU-R    SU-L    SU-Ci   AU-L    AU-R    AU-Ci
199 //  00    M       M       M       C       M       M       C
200 //  01    A       A       A      Q[7]     A       A      Q[7]
201 //  10    K       K       X       0       X       K       0
202 //  11    P       P       Y       1       Y       P       1
203 //
204 //  The coding for the WSel and OSel fields follows:
205 //        3     3
206 //  Sel  WSel  OSel
207 //  000   -     0
208 //  001   A     A
209 //  010   X     X
210 //  011   Y     Y
211 //  100   -     Tmp
212 //  101   S     S
213 //  110   P     P
214 //  111   M     M
215 //
216 //  The WSel field provides the register enable for writing into a register,
217 //  and the OSel field determines the value of the ALU result bus. Typically
218 //  the ALU result bus is the ALU. But on the occasion of a load, store, xfer,
219 //  push, and pull instructions, the ALU bus is driven by A, X, Y, 0, S, P, and
220 //  M. In this manner, the ALU result can be either loaded into the register
221 //  designated by the WSel field, or it can be written to memory.
222 //
223 //  The coding of these control fields and the ALU field is designed to yield a
224 //  0 at the output of the module if all the input control fields are logical
225 //  0. This means that a NOP is simply a zero for all ALU control fields.
226 //
227 //  The CC_Sel field controls the CC_Out condition code test signal and the
228 //  updating of the individual bits in P in response to specific instructions:
229 //
230 //  0_xxx: NOP/TRUE - CC_Out =  1
231 //
232 //  1_000: CC       - CC_Out = ~C;
233 //  1_001: CS       - CC_Out =  C;
234 //  1_010: NE       - CC_Out = ~Z;
235 //  1_011: EQ       - CC_Out =  Z;
236 //  1_100: VC       - CC_Out = ~V;
237 //  1_101: VS       - CC_Out =  V;
```

```
238 //  1_110: PL       - CC_OUT = ~N;
239 //  1_111: MI       - CC_Out =  N;
240 //
241 //  x_000: pPSW     - P <= ALU;
242 //  x_001: pBRK     - P.4 <= 1 on push P during BRK ISR
243 //  x_010: Z        -                          Z <= ~|(A & M);
244 //  x_011: NVZ      - N <= M[7];   V <= M[6]; Z <= ~|(A & M);
245 //  x_100: pPHP     - P.4 <= 1 on PHP
246 //  x_101: NZ       - N <= ALU[7];            Z <= ~|ALU;
247 //  x_110: NZC      - N <= ALU[7];            Z <= ~|ALU;     C <= COut
248 //  x_111: NVZC     - N <= ALU[7]; V <= OVF;  Z <= ~|ALU;     C <= COut;
249 //
250 // Dependencies:    M65C02_ALUv2.v
251 //                      M65C02_LST.v,       // Load/Store/Transfer Multiplexer
252 //                      M65C02_LU.v,        // Logic Unit
253 //                      M65C02_SU.v,        // Shift Unit
254 //                      M65C02_Add.v,       // Adder Unit
255 //                      M65C02_WrSel.v      // Write Select ROM
256 //                      M65C02_PSW.v        // Processor Status Word (Register)
257 //
258 // Revision:
259 //
260 //  1.00    13I16   MAM     Ported the M65C02_ALU.v module. Removed the Rockwell
261 //                          instructions, modified the operation code to support
262 //                          a one-hot 5-bit functional unit select and a 2-bit
263 //                          operation select; combined the Logic Unit (LU) and
264 //                          the BIT Unit (BU) into a single module; combined the
265 //                          binary mode adder module, M65C02_BIN.v, and the
266 //                          decimal mode adder module, M65C02_BCD.v, into a
267 //                          single, dual mode adder module, M65C02_Adder.v; re-
268 //                          moved the stack pointer module and added ports to
269 //                          import the stack pointer (for TSX support) and ex-
270 //                          port a stack pointer write enable, SelS; and created
271 //                          independent modules for the LU, SU, LST, and regis-
272 //                          ter write enable ROM for a more modular implementa-
273 //                          tion. Finally, implement wor busses for ALU_DO, Val,
274 //                          Co, and OV. All these changes provide an increase
275 //                          in speed from ~73 MHz to ~101 MHz in a -4 Spartan 3A
276 //                          FPGA.
277 //
278 //  1.10    13J01   MAM     Modified the integrated version to support an ALU
279 //                          that incorporates support for more instructions, and
280 //                          maintains compatibility with the capabilities of the
281 //                          M65C02 ALU, core, and microprocessor. The objective
282 //                          being an ALU and core that can be configured to pro-
283 //                          vide the instruction sets of the MOS6502, GSC65SC02,
284 //                          W65C02S, and the M65C02A simply by supplying new
285 //                          contents for the microprogram and instruction de-
286 //                          coder ROMs.
287 //
288 //  1.20    13J07   MAM     Significantly improved the description section. Made
289 //                          minor changes to the definition of the QSel, RSel,
290 //                          and CSel fields. When integrated into the
291 //                          M65C02_CoreV2 module, the result is a reported speed
292 //                          improvement of 5-6%, but also a slight reduction in
293 ///                         the number of slices and LUTs required such that the
294 //                          ALU only requires ~220 LUTs, 59 Registers, and 155
295 //                          slices.
296 //
297 //  1.30    14F21   MAM     Modified the R multiplexer to select {P, K, A, M}
298 //                          instead of {P, 0, A, M}. This modification allows
299 //                          the microprogram to support AU INC/DEC by any value
300 //                          including 1 or -1, and to support LU operations with
301 //                          its own mask values.
302 //
303 //  1.31    14G04   MAM     Exposed the Accumulator as a module port.
304 //
305 // Additional Comments:
306 //
307 //  Revision 1.10 of the module incorporates all those features required to pro-
308 //  vide the instruction set capabilities of the MOS6502, GSC65SC02, W65C02S,
309 //  and the M65C02A simply by updating the contents of the microprogram ROMs and
310 //  no logic changes. The ALU control signals provided by the microprogram and
311 //  instruction decoder ROMs have been modified to provide additional control of
312 //  new operand multiplexers. The new multiplexers will allow the implementation
313 //  of all M65C02/W65C02S instructions using only four functional units: Load/
314 //  Store/Transfer (LST) unit, Logic Unit (LU), Shift/Rotate Unit (SU), and an
315 //  Arithmetic Unit (AU). Additional multiplexers have been added, and the Op,
316 //  QSel, RSel, Sub, and CSel control fields have been modified and overloaded.
```

```
317 //
318 //  The result is an ALU substantially smaller and faster than the ALU in the
319 //  M65C02. First, the separate functional unit for implementing the Rockwell
320 //  instructions (RMBx/SMBx/BBRx/BBSx) is no longer required. Those instructions
321 //  are now provided using only the Logic Unit. Second, capabilities have been
322 //  added to the LU operand source select multiplexer so that it is possible to
323 //  perform logic operations on the PSW (P) so that the REP/SEP instructions
324 //  from the WDC '816/802 can be included simply by microprogram and instruction
325 //  decode ROM changes. Third, it will be possible to include the '816/802 PEA
326 //  and PEI instructions as well.
327 //
328 //  One of the drivers in the improved performance of this ALU is the use of
329 //  wired-OR busses for the outputs of the functional units. The output bus of
330 //  the ALU_DO is driven by all the functional units. If the unit is not
331 //  selected, ALU_DO is driven by the functional unit to 0. A similar implemen-
332 //  tation is used for the Co, OV, and data Valid signals.
333 //
334 //  Another driver in the improved performance of the ALU is the use of, pre-
335 //  decoded one-hot functional unit select signals as part of the instruction
336 //  decode ROM. These signals significantly reduce the number of logic levels
337 //  needed to implement the functional unit selects and functional unit output
338 //  multiplexers required in the ALU. The wired-OR busses are more compatible
339 //  with the FPGA internal architecture, and the synthesizer is able to better
340 //  optimize the number of logic levels needed to implement the muxes.
341 //
342 //  If there wasn't a limitation that a single 18kb Block RAM can be configured
343 //  to support, the instruction decode ROM width could be widened even more than
344 //  the 36 bits required. This would result in additional performance because
345 //  additional one-hot control bits could be included. The additional decrease
346 //  in the logic levels required to implement the ALU would result in another
347 //  significant increase in speed.
348 //
349 //  Given the target FPGA, the XC3S50A, there are no additional Block RAMs that
350 //  can be used to provide the additional width in both the instruction decode
351 //  and the microprogram ROMs. The second target FPGA, the XC3S200A in the same
352 //  VQ100 package, can provide additional Block RAMs for use in the instruction
353 //  and microprogram ROMs.
354 //
355 //  A final addition to improve the compatibility of the M65C02A to the W65C02S
356 //  has been the inclusion of an external Set_Overflow port. The nSO pin on the
357 //  W65C02 and MOS6502 sets the V flag in the PSW whenever a falling edge is
358 //  asserted on that external pin of those processors. The ALU now incorporates
359 //  the capability to include a pulsed signal, SO, that will force the V bit to
360 //  be set whenever the PSW is updated, or at the completion of any instruction
361 //  that doesn't set the PSW. In other words, the V is forced to a logic 1 on an
362 //  instruction boundary if SO is asserted.
363 //
364 ////////////////////////////////////////////////////////////////////////////////
365
366 module M65C02_ALUv2 (
367     input   Rst,            // System Reset - synchronous reset
368     input   Clk,            // System Clock
369
370     input   Rdy,            // Ready
371
372     input   En,             // Enable - ALU functions
373     input   [2:0] Reg_WE,   // Register Write Enable
374     input   ISR,            // Asserted on entry to Interrupt Service Routine
375
376     input   SO,             // Set Overflow status bit Command
377     output  Clr_SO,         // Acknowledge for SO Command
378
379     //  Address Generator (Stack Pointer) Interface
380
381     output  SelS,           // Stack Pointer Select
382     input   [7:0] S,        // Stack Pointer input from Address Generator
383
384     //  ALU Ports
385
386     input   [4:0] FU_Sel,   // ALU Functional Unit Select
387     input   [1:0] Op,       // ALU Operation Select
388     input   [1:0] QSel,     // ALU Q Bus Multiplexer Select
389     input   [1:0] RSel,     // ALU R Bus Multiplexer Select
390     input   [1:0] CSel,     // ALU Carry In Multiplexer Select
391     input   [2:0] WSel,     // ALU Register WE Select
392     input   [2:0] OSel,     // ALU Output Multiplexer Select
393     input   [3:0] CCSel,    // ALU Condition Code Operation Select
394
395     input   [7:0] K,        // ALU Bit Mask for Rockwell Instructions
```

```
396     input   [7:0] Tmp,      // ALU Temporary Operand Holding Register
397     input   [7:0] M,        // ALU Memory Operand Input
398     output  [7:0] DO,       // ALU Data Output(asynchronous)
399     output  Val,            // ALU Output Valid
400
401     output  reg CC_Out,     // Condition Code Test Output
402
403     //  Internal Processor Registers
404
405     output  reg [7:0] A,    // Accumulator Register
406     output  reg [7:0] X,    // X Index Register
407     output  reg [7:0] Y,    // Y Index Register
408
409     output  [7:0] P         // Processor Status Word
410 );
411
412 ////////////////////////////////////////////////////////////////////////////
413 //
414 //  Local Parameter Decalarations
415 //
416
417 //  {FU_Sel, Op} Encodings
418
419 //  FU_Sel  Op   Mnemonic     Operation            Condition Code
420
421 //  1_00_00 00     XFR    ALU <= {OSel: M, P, S, Tmp, Y, X, A, 0}
422 //
423 //  0_10_00 01     AND    ALU <=  A & M;     N <= ALU[7]; Z <= ~|ALU;
424 //  0_10_00 10     ORA    ALU <=  A | M;     N <= ALU[7]; Z <= ~|ALU;
425 //  0_10_00 11     EOR    ALU <=  A ^ M;     N <= ALU[7]; Z <= ~|ALU;
426 //
427 //  0_10_00 01     BIT    ALU <=  A & M;     N <= M[7];   Z <= ~|(A & M);
428 //                                           V <= M[6];
429 //  0_10_00 00     TRB    ALU <= ~A & M;                  Z <= ~|(A & M);
430 //  0_10_00 10     TSB    ALU <=  A | M;                  Z <= ~|(A & M);
431 //
432 //  0_10_00 00     RMBx   ALU <= ~K & M;
433 //  0_10_00 10     SMBx   ALU <=  K | M;
434 //  0_10_00 01     BBRx   ALU <=  K & M;
435 //  0_10_00 01     BBSx   ALU <=  K & M;
436 //
437 //  0_10_00 00     CLC    ALU <= ~K & P;     C <= 0;
438 //  0_10_00 10     SEC    ALU <=  K | P;     C <= 1;
439 //  0_10_00 00     CLI    ALU <= ~K & P;     I <= 0;
440 //  0_10_00 10     SEI    ALU <=  K | P;     I <= 1;
441 //  0_10_00 00     CLD    ALU <= ~K & P;     D <= 0;
442 //  0_10_00 10     SED    ALU <=  K | P;     D <= 1;
443 //  0_10_00 00     CLV    ALU <= ~K & P;     V <= 0;
444 //
445 //  0_10_00 00     REP    ALU <= ~M & P;     P <= ALU;
446 //  0_10_00 10     SEP    ALU <=  M | P;     P <= ALU;
447 //
448 //  0_01_00 00     ASL    ALU <= {R[6:0],Ci} N <= ALU[7]; Z <= ~|ALU; C <= R[7];
449 //  0_01_00 00     ROL    ALU <= {R[6:0],Ci} N <= ALU[7]; Z <= ~|ALU; C <= R[7];
450 //  0_01_00 01     LSR    ALU <= {Ci,R[7:1]} N <= ALU[7]; Z <= ~|ALU; C <= R[0];
451 //  0_01_00 01     ROR    ALU <= {Ci,R[7:1]} N <= ALU[7]; Z <= ~|ALU; C <= R[0];
452 //
453 //  0_00_01 00     INC    ALU <= Q +  K + 1; N <= ALU[7]; Z <= ~|ALU;
454 //  0_00_01 00     DEC    ALU <= Q + ~K + 0; N <= ALU[7]; Z <= ~|ALU;
455 //  0_00_01 00     CMP    ALU <= Q + ~M + 1; N <= ALU[7]; Z <= ~|ALU;
456 //                                                        C <= COut;
457 //  0_00_10 00     ADC    ALU <= Q +  M + C; N <= ALU[7]; Z <= ~|ALU;
458 //                                           V <= OV;     C <= COut;
459 //  0_00_10 00     SBC    ALU <= Q + ~M + C; N <= ALU[7]; Z <= ~|ALU;
460 //                                           V <= OV;     C <= COut;
461
462 //  CCSel - Condition Code Select
463
464 localparam pTRUE = 4'd0;    // Set CC_Out TRUE
465 //
466 localparam pCC   = 4'd8;    // Set CC_Out if Carry Clear        (C Clear)
467 localparam pCS   = 4'd9;    // Set CC_Out if Carry Set          (C Set)
468 localparam pNE   = 4'd10;   // Set CC_Out if Not Equal to Zero  (Z Clear)
469 localparam pEQ   = 4'd11;   // Set CC_Out if Equal to Zero      (Z Set)
470 localparam pVC   = 4'd12;   // Set CC_Out if Not Overflow       (V Clear)
471 localparam pVS   = 4'd13;   // Set CC_Out if Overflow           (V Set)
472 localparam pPL   = 4'd14;   // Set CC_Out if Plus               (N Clear)
473 localparam pMI   = 4'd15;   // Set CC_Out if Negative           (N Set)
474 //
```

```
475 localparam pPSW  = 4'd0;    // Set P from ALU Output (PLP | Set/Clr instruction)
476 localparam pBRK  = 4'd1;    // Set P[4] when P pushed using BRK instruction
477 localparam pZ    = 4'd2;    // Set Z = ~|(A & M)
478 localparam pNVZ  = 4'd3;    // Set N and V flags from M[7:6], and Z = ~|(A & M)
479 localparam pPHP  = 4'd4;    // Set P[4] when P pushed using PHP instruction
480 localparam pNZ   = 4'd5;    // Set N and Z flags from ALU Output
481 localparam pNZC  = 4'd6;    // Set N, Z, and C flags from ALU Output
482 localparam pNVZC = 4'd7;    // Set N, V, Z, and C from ALU Output
483
484 ////////////////////////////////////////////////////////////////////////////
485 //
486 //  Declarations
487 //
488
489 //  Functional Unit Select/Enables
490
491 wire    En_LST;                 // Load/Store/Transfer Enable
492 wire    En_LU;                  // Logic Unit Enable
493 wire    En_SU;                  // Shift/Rotate Unit Enable
494 wire    En_AU;                  // Adder Unit Enable
495 wire    En_DU;                  // Decimal (BCD) Unit Enable
496
497 //  ALU Busses
498
499 wire    [7:0] G, H, L;          // LU Input Busses
500 wire    [7:0] W, U, Q;          // Adder Left Operand Input Busses (SU <= W)
501 wire    [7:0] E, F, R;          // Adder Right Operand Input Busses
502
503 reg     Ci;                     // Adder Carry In signal
504
505 //  ALU Component Output Busses
506
507 wor     [8:0] Out;              // ALU Output
508 wor     OV;                     // ALU Adder Overflow Flag
509 wor     Valid;                  // ALU Output Valid
510
511 wire    COut;                   // Carry Out -> input to C
512 wire    LU_Z;                   // Zero Detector for BIT/TRB/TSB/BBR/BBS
513
514 //  ALU Registers
515
516 wire    SelA, SelX, SelY, SelP; // ALU Register Selects
517
518 wire    N, V, D, Z, C;          // ALU PSW flags
519
520 ////////////////////////////////////////////////////////////////////////////
521 //
522 //  Implementation
523 //
524
525 //  ALU Functional Unit Selection
526
527 assign En_LST = En & FU_Sel[4];     // Load/Store/Transfer
528 assign En_LU  = En & FU_Sel[3];     // AND/OR/XOR/BIT/TRB/TSB/RMB/SMB/BBR/BBS
529 assign En_SU  = En & FU_Sel[2];                     // Shift/Rotate
530 assign En_AU  = En & ((FU_Sel[1] & ~D) | FU_Sel[0]);  // Binary Adder
531 assign En_DU  = En &  (FU_Sel[1] &  D);              // Decimal Adder
532
533 //  If no functional unit selected, then assert Valid
534
535 assign Valid = ~|FU_Sel;
536
537 //  LU (Left Operand) Multiplexer
538 //      00: L = M; 01: L = A;
539 //      10: L = K; 11: L = P;
540
541 assign G = ((QSel[0]) ? A : M);     // ? A : M;
542 assign H = ((QSel[0]) ? P : K);     // ? P : K;
543 assign L = ((QSel[1]) ? H : G);
544
545 //  AU/SU (Left Operand) Multiplexer
546 //      00: Q = M; 01: Q = A;
547 //      10: Q = X; 11: Q = Y;
548
549 assign W = ((QSel[0]) ? A : M);     // ? A : M; INC/DEC/CMP/ADC/SBC
550 assign U = ((QSel[0]) ? Y : X);     // ? Y : X; INY/DEY/CPY, INX/DEX/CPX
551 assign Q = ((QSel[1]) ? U : W);
552
553 //  LU/AU (Right Operand) Multiplexer
```

```
554 //      00: R = M; 01: R = A;
555 //      10: R = 0; 11: R = P;
556
557 assign E = ((RSel[0]) ? A : M);     // ? A : M; ADC/SBC/CMP
558 assign F = ((RSel[0]) ? P : K);     // ? P : K; INC/DEC, CLC/.../CLV, REP/SEP
559 assign R = ((RSel[1]) ? F : E);
560
561 // Carry In Multiplexer
562
563 always @(*)
564 begin
565     case(CSel)
566         2'b00 : Ci <= C;            // ADC/SBC/ROL/ROR
567         2'b01 : Ci <= Q[7];         // ASR (Reserved for future use)
568         2'b10 : Ci <= 0;            // DEC/ASL/LSR
569         2'b11 : Ci <= 1;            // INC/CMP
570     endcase
571 end
572
573 //  Load/Store/Transfer Multiplexer Instantiation
574 //  0   -   : STZ
575 //  1   A   : STA/TAX/TAY/PHA/TAS
576 //  2   X   : STX/TXA/TXS/PHX/SAX/TXY
577 //  3   Y   : STY/TYA/PHY/SAY/TYX
578 //  4   Tmp : PEA/PEI (lo byte held in Tmp, Tmp pushed to stack after hi byte)
579 //  5   S   : TSX/TSA
580 //  6   P   : PHP
581 //  7   M   : LDA/PLA/LDX/PLX/LDY/PLY/PLP
582
583 M65C02_LST  LST (
584                 .En(En_LST),
585                 .OSel(OSel),
586
587                 .A(A),
588                 .X(X),
589                 .Y(Y),
590                 .Tmp(Tmp),
591                 .S(S),
592                 .P(P),
593                 .M(M),
594
595                 .Out(Out),
596                 .Val(Valid)
597             );
598
599 // Logic Unit Instantiation
600 //      AND/ORA/EOR/BIT/TRB/TSB
601 //      RMBx/SMBx/BBRx/BBSx
602 //      SEC/CLC/SEI/CLI/SED/CLD/CLV
603 //      REP/SEP
604
605 M65C02_LU   LU (
606                 .En(En_LU),
607
608                 .Op(Op),
609                 .L(L),
610                 .M(R),
611
612                 .Out(Out),
613                 .Z(LU_Z),
614                 .Val(Valid)
615             );
616
617 //  Shift Unit Instantiation
618 //      SU, Ci - ASL/ROL/LSR/ROR
619 //          support for adding ASR instruction is included
620 //          support for adding ASL overflow detection also included
621
622 M65C02_SU   SU (
623                 .En(En_SU),
624
625                 .Op(Op[0]),
626                 .SU(Q),
627                 .Ci(Ci),
628
629                 .Out(Out),
630                 .OV(OV),
631                 .Val(Valid)
632             );
```

```
633
634 //  Adder (Binary and Decimal Adders) Instantiation
635 //      Binary Adder Unit Implementation (INC/INX/INY/DEC/DEX/DEY/CMP/ADC/SBC)
636 //      Decimal (BCD) Adder Unit Implementation (ADC/SBC (Decimal-Only))
637
638 M65C02_Add  AU (
639                 .Rst(Rst),
640                 .Clk(Clk),
641
642                 .En_AU(En_AU),
643                 .En_DU(En_DU),
644                 .Op(Op[0]),
645
646                 .Q(Q),
647                 .R(R),
648                 .Ci(Ci),
649
650                 .Out(Out),
651                 .OV(OV),
652                 .Val(Valid)
653             );
654
655 //  Define ALU Output
656
657 assign DO   = Out[7:0];
658 assign COut = Out[8];
659 assign Val  = Valid;
660
661 //  Condition Code Output
662 //      Additional multiplexer to support BBRx/BBSx instructions included.
663
664 always @(posedge Clk)
665 begin
666     if(Rst)
667         CC_Out <= #1 1;
668     else
669         case(CCSel)
670             pCC     : CC_Out <= #1 ~C;
671             pCS     : CC_Out <= #1  C;
672             pNE     : CC_Out <= #1 ((En_LU) ? ~LU_Z : ~Z);
673             pEQ     : CC_Out <= #1 ((En_LU) ?  LU_Z :  Z);
674             pVC     : CC_Out <= #1 ~V;
675             pVS     : CC_Out <= #1  V;
676             pPL     : CC_Out <= #1 ~N;
677             pMI     : CC_Out <= #1  N;
678             default : CC_Out <= #1  1;
679         endcase
680 end
681
682 ////////////////////////////////////////////////////////////////////////////////
683 //
684 //  Internal ALU Registers (Distributed Dual-Port SRAM)
685 //
686
687 M65C02_WrSel    WrSel (
688                     .Rst(Rst),
689                     .Clk(Clk),
690
691                     .Reg_WE(Reg_WE),
692                     .WSel(WSel),
693
694                     .SelA(SelA),
695                     .SelX(SelX),
696                     .SelY(SelY),
697                     .SelP(SelP),
698
699                     .SelS(SelS)
700                 );
701
702 ////////////////////////////////////////////////////////////////////////////////
703 //
704 //  A - Accumulator
705 //
706
707 assign WE_A = SelA & Valid & Rdy;
708
709 always @(posedge Clk)
710 begin
711     if(Rst)
```

```
712         A <= #1 0;
713     else if(WE_A)
714         A <= #1 Out;
715 end
716
717 //////////////////////////////////////////////////////////////////////////////
718 //
719 //  X - Pre-Index Register
720 //
721
722 assign WE_X = SelX & Valid & Rdy;
723
724 always @(posedge Clk)
725 begin
726     if(Rst)
727         X <= #1 0;
728     else if(WE_X)
729         X <= #1 Out;
730 end
731
732 //////////////////////////////////////////////////////////////////////////////
733 //
734 //  Y - Post-Index Register
735 //
736
737 assign WE_Y = SelY & Valid & Rdy;
738
739 always @(posedge Clk)
740 begin
741     if(Rst)
742         Y <= #1 0;
743     else if(WE_Y)
744         Y <= #1 Out;
745 end
746
747 //////////////////////////////////////////////////////////////////////////////
748 //
749 //  P - Processor Status Word: {N, V, 1, B, D, I, Z, C}
750 //
751
752 M65C02_PSWv2    PSW (
753                     .Clk(Clk),
754
755                     .SO(SO),
756                     .Clr_SO(Clr_SO),
757
758                     .SelP(SelP),
759                     .Valid(Valid),
760                     .Rdy(Rdy),
761
762                     .ISR(ISR),
763
764                     .CCSel(CCSel[2:0]),
765
766                     .M(M[7:6]),
767                     .OV(OV),
768                     .LU_Z(LU_Z),
769                     .COut(COut),
770                     .DO(DO),
771
772                     .P(P),
773
774                     .N(N),
775                     .V(V),
776                     .D(D),
777                     .Z(Z),
778                     .C(C)
779                 );
780
781 endmodule
```

```
 1 //////////////////////////////////////////////////////////////////////////////////
 2 //
 3 //  ALU Load/Store/Transfer module for M65C02A soft-core microcomputer project.
 4 //
 5 //  Copyright 2013-2014 by Michael A. Morris, dba M. A. Morris & Associates
 6 //
 7 //  All rights reserved. The source code contained herein is publicly released
 8 //  under the terms and conditions of the GNU General Public License as conveyed
 9 //  in the license provided below.
10 //
11 //  This program is free software: you can redistribute it and/or modify it
12 //  under the terms of the GNU General Public License as published by the Free
13 //  Software Foundation, either version 3 of the License, or any later version.
14 //
15 //  This program is distributed in the hope that it will be useful, but WITHOUT
16 //  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
17 //  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
18 //  more details.
19 //
20 //  You should have received a copy of the GNU General Public License along with
21 //  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
22 //
23 //  Free Software Foundation, Inc.
24 //  51 Franklin Street, Fifth Floor
25 //  Boston, MA  02110-1301 USA
26 //
27 //  Further, no use of this source code is permitted in any form or means
28 //  without inclusion of this banner prominently in any derived works.
29 //
30 //  Michael A. Morris <morrisma_at_mchsi_dot_com>
31 //  164 Raleigh Way
32 //  Huntsville, AL 35811
33 //  USA
34 //
35 //////////////////////////////////////////////////////////////////////////////////
36
37 `timescale 1ns / 1ps
38
39 //////////////////////////////////////////////////////////////////////////////////
40 // Company:         M. A. Morris & Associates
41 // Engineer:        Michael A. Morris
42 //
43 // Create Date:     17:06:48 09/15/2013
44 // Design Name:     WDC W65C02 Microprocessor Re-Implementation
45 // Module Name:     M65C02_LST.v
46 // Project Name:    C:\XProjects\ISE10.1i\M6502A
47 // Target Devices:  Generic SRAM-based FPGA
48 // Tool versions:   Xilinx ISE10.1i SP3
49 //
50 // Description:
51 //
52 // Dependencies:    None.
53 //
54 // Revision:
55 //
56 //  0.01    13I15   MAM     Initial coding. Pulled implementation details from
57 //                          the parent module, M65C02_ALU.v, and generated a
58 //                          standalone module instantiated in the parent.
59 //
60 // Additional Comments:
61 //
62 //////////////////////////////////////////////////////////////////////////////////
63
64 module M65C02_LST(
65     input   En,
66     input   [2:0] OSel,
67
68     input   [7:0] A,
69     input   [7:0] X,
70     input   [7:0] Y,
71     input   [7:0] Tmp,
72     input   [7:0] S,
73     input   [7:0] P,
74     input   [7:0] M,
75
76     output  [8:0] Out,
77     output  Val
78 );
79
```

```
 80 //////////////////////////////////////////////////////////////////////////////////
 81 //
 82 //  Declarations
 83 //
 84
 85 wor     [8:0] Mux;
 86 reg     [7:1] Sel;
 87
 88 //////////////////////////////////////////////////////////////////////////////////
 89 //
 90 //  Implementation
 91 //
 92
 93 always @(*)
 94 begin
 95     casex({En, OSel})
 96         4'b0000 : Sel <= 7'h00;
 97         4'b0001 : Sel <= 7'h00;
 98         4'b0010 : Sel <= 7'h00;
 99         4'b0011 : Sel <= 7'h00;
100         4'b0100 : Sel <= 7'h00;
101         4'b0101 : Sel <= 7'h00;
102         4'b0110 : Sel <= 7'h00;
103         4'b0111 : Sel <= 7'h00;
104         4'b1000 : Sel <= 7'h00;
105         4'b1001 : Sel <= 7'h01;
106         4'b1010 : Sel <= 7'h02;
107         4'b1011 : Sel <= 7'h04;
108         4'b1100 : Sel <= 7'h08;
109         4'b1101 : Sel <= 7'h10;
110         4'b1110 : Sel <= 7'h20;
111         4'b1111 : Sel <= 7'h40;
112     endcase
113 end
114
115 //  Load/Store/Transfer Multiplexer
116
117 //  Generate wired-OR multiplexer
118
119 assign Mux = ((Sel[1]) ? {1'b0, A}   : 0);      // STA/TAX/TAY/TAS/PHA
120 assign Mux = ((Sel[2]) ? {1'b0, X}   : 0);      // STX/TXA/TXS/PHX
121 assign Mux = ((Sel[3]) ? {1'b0, Y}   : 0);      // STY/TYA/PHY
122 assign Mux = ((Sel[4]) ? {1'b0, Tmp} : 0);      // PHW/PHR
123 assign Mux = ((Sel[5]) ? {1'b0, S}   : 0);      // TSX/TSA
124 assign Mux = ((Sel[6]) ? {1'b0, P}   : 0);      // PHP
125 assign Mux = ((Sel[7]) ? {1'b0, M}   : 0);      // LDA/PLA/LDX/PLX/LDY/PLY/PLP
126
127 //  Assign Module Outputs
128
129 assign Out = Mux;
130 assign Val = En;
131
132 endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
//  ALU Logic Unit module for M65C02A soft-core microcomputer project.
//
//  Copyright 2013-2014 by Michael A. Morris, dba M. A. Morris & Associates
//
//  All rights reserved. The source code contained herein is publicly released
//  under the terms and conditions of the GNU General Public License as conveyed
//  in the license provided below.
//
//  This program is free software: you can redistribute it and/or modify it
//  under the terms of the GNU General Public License as published by the Free
//  Software Foundation, either version 3 of the License, or any later version.
//
//  This program is distributed in the hope that it will be useful, but WITHOUT
//  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
//  more details.
//
//  You should have received a copy of the GNU General Public License along with
//  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
//
//  Free Software Foundation, Inc.
//  51 Franklin Street, Fifth Floor
//  Boston, MA  02110-1301 USA
//
//  Further, no use of this source code is permitted in any form or means
//  without inclusion of this banner prominently in any derived works.
//
//  Michael A. Morris <morrisma_at_mchsi_dot_com>
//  164 Raleigh Way
//  Huntsville, AL 35811
//  USA
//
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:         M. A. Morris & Associates
// Engineer:        Michael A. Morris
//
// Create Date:     08:48:13 09/15/2013
// Design Name:     WDC W65C02 Microprocessor Re-Implementation
// Module Name:     M65C02_LU.v
// Project Name:    C:\XProjects\ISE10.1i\M6502A
// Target Devices:  Generic SRAM-based FPGA
// Tool versions:   Xilinx ISE10.1i SP3
//
// Description:
//
// Dependencies:    None.
//
// Revision:
//
//  0.01    13I15   MAM     Initial coding. Pulled implementation details from
//                          the parent module, M65C02_ALU.v, and generated a
//                          standalone module instantiated in the parent.
//
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module M65C02_LU(
    input   En,

    input   [1:0] Op,
    input   [7:0] L,
    input   [7:0] M,

    output  reg [8:0] Out,
    output  Z,

    output  Val
);

////////////////////////////////////////////////////////////////////////////////
//
//  Implementation
```

```
////////////////////////////////////////////////////////////////////////////////
//
//  ALU Logic Unit module for M65C02A soft-core microcomputer project.
//
//  Copyright 2013-2014 by Michael A. Morris, dba M. A. Morris & Associates
//
//  All rights reserved. The source code contained herein is publicly released
//  under the terms and conditions of the GNU General Public License as conveyed
//  in the license provided below.
//
//  This program is free software: you can redistribute it and/or modify it
//  under the terms of the GNU General Public License as published by the Free
//  Software Foundation, either version 3 of the License, or any later version.
//
//  This program is distributed in the hope that it will be useful, but WITHOUT
//  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
//  more details.
//
//  You should have received a copy of the GNU General Public License along with
//  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
//
//  Free Software Foundation, Inc.
//  51 Franklin Street, Fifth Floor
//  Boston, MA  02110-1301 USA
//
//  Further, no use of this source code is permitted in any form or means
//  without inclusion of this banner prominently in any derived works.
//
//  Michael A. Morris <morrisma_at_mchsi_dot_com>
//  164 Raleigh Way
//  Huntsville, AL 35811
//  USA
//
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:         M. A. Morris & Associates
// Engineer:        Michael A. Morris
//
// Create Date:     08:48:13 09/15/2013
// Design Name:     WDC W65C02 Microprocessor Re-Implementation
// Module Name:     M65C02_LU.v
// Project Name:    C:\XProjects\ISE10.1i\M6502A
// Target Devices:  Generic SRAM-based FPGA
// Tool versions:   Xilinx ISE10.1i SP3
//
// Description:
//
// Dependencies:    None.
//
// Revision:
//
//  0.01    13I15   MAM     Initial coding. Pulled implementation details from
//                          the parent module, M65C02_ALU.v, and generated a
//                          standalone module instantiated in the parent.
//
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module M65C02_LU(
    input   En,

    input   [1:0] Op,
    input   [7:0] L,
    input   [7:0] M,

    output  reg [8:0] Out,
    output  Z,

    output  Val
);

////////////////////////////////////////////////////////////////////////////////
//
//  Implementation
```

```
80 //
81
82 always @(*)
83 begin
84     if(En)
85         case(Op)
86             2'b00 : Out <= ~L & M;  // TRB/RMBx/SEC/SEI/SED
87             2'b01 : Out <=  L & M;  // AND/BIT/BBRx/BBSx
88             2'b10 : Out <=  L | M;  // ORA/TSB/SMBx/CLC/CLI/CLD/CLV
89             2'b11 : Out <=  L ^ M;  // EOR
90         endcase
91     else
92         Out <= 0;
93 end
94
95 assign Z   = ((En) ? ~|(L & M) : 0);
96 assign Val = En;
97
98 endmodule
```

```
 1 //////////////////////////////////////////////////////////////////////////////////
 2 //
 3 //  ALU Shift/rotate Unit module for M65C02A soft-core microcomputer project.
 4 //
 5 //  Copyright 2013-2014 by Michael A. Morris, dba M. A. Morris & Associates
 6 //
 7 //  All rights reserved. The source code contained herein is publicly released
 8 //  under the terms and conditions of the GNU General Public License as conveyed
 9 //  in the license provided below.
10 //
11 //  This program is free software: you can redistribute it and/or modify it
12 //  under the terms of the GNU General Public License as published by the Free
13 //  Software Foundation, either version 3 of the License, or any later version.
14 //
15 //  This program is distributed in the hope that it will be useful, but WITHOUT
16 //  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
17 //  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
18 //  more details.
19 //
20 //  You should have received a copy of the GNU General Public License along with
21 //  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
22 //
23 //  Free Software Foundation, Inc.
24 //  51 Franklin Street, Fifth Floor
25 //  Boston, MA  02110-1301 USA
26 //
27 //  Further, no use of this source code is permitted in any form or means
28 //  without inclusion of this banner prominently in any derived works.
29 //
30 //  Michael A. Morris <morrisma_at_mchsi_dot_com>
31 //  164 Raleigh Way
32 //  Huntsville, AL 35811
33 //  USA
34 //
35 //////////////////////////////////////////////////////////////////////////////////
36
37 `timescale 1ns / 1ps
38
39 //////////////////////////////////////////////////////////////////////////////////
40 // Company:         M. A. Morris & Associates
41 // Engineer:        Michael A. Morris
42 //
43 // Create Date:     08:15:35 09/15/2013
44 // Design Name:     WDC W65C02 Microprocessor Re-Implementation
45 // Module Name:     M65C02_SU.v
46 // Project Name:    C:\XProjects\ISE10.1i\M6502A
47 // Target Devices:  Generic SRAM-based FPGA
48 // Tool versions:   Xilinx ISE10.1i SP3
49 //
50 // Description:
51 //
52 // Dependencies:    None.
53 //
54 // Revision:
55 //
56 //  0.01    13I15   MAM     Initial coding. Pulled implementation details from
57 //                          the parent module, M65C02_ALU.v, and generated a
58 //                          standalone module instantiated in the parent.
59 //
60 //  1.00    13J23   MAM     Corrected error in the operation multiplexer. Op = 1
61 //                          is for right shift/rotate operations, and Op = 0 is
62 //                          for left shift/rotate operations.
63 //
64 // Additional Comments:
65 //
66 //////////////////////////////////////////////////////////////////////////////////
67
68 module M65C02_SU(
69     input   En,
70
71     input   Op,
72     input   [7:0] SU,
73     input   Ci,
74
75     output  reg [8:0] Out,
76     output  reg OV,
77     output  Val
78 );
79
```

```
80 always @(*)
81 begin
82     if(En)
83         {OV, Out} <= ((Op) ? {^SU[7:6], {SU[0], {Ci, SU[7:1]}}}     // LSR/ROR
84                            : {^SU[7:6], {SU[7], {SU[6:0], Ci}}});   // ASL/ROL
85     else
86         {OV, Out} <= 0;
87 end
88
89 assign Val = En;
90
91 endmodule
```

```
 1 //////////////////////////////////////////////////////////////////////////////
 2 //
 3 //  ALU dual-mode Adder Unit module for M65C02A soft-core microcomputer project.
 4 //
 5 //  Copyright 2013-2014 by Michael A. Morris, dba M. A. Morris & Associates
 6 //
 7 //  All rights reserved. The source code contained herein is publicly released
 8 //  under the terms and conditions of the GNU General Public License as conveyed
 9 //  in the license provided below.
10 //
11 //  This program is free software: you can redistribute it and/or modify it
12 //  under the terms of the GNU General Public License as published by the Free
13 //  Software Foundation, either version 3 of the License, or any later version.
14 //
15 //  This program is distributed in the hope that it will be useful, but WITHOUT
16 //  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
17 //  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
18 //  more details.
19 //
20 //  You should have received a copy of the GNU General Public License along with
21 //  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
22 //
23 //  Free Software Foundation, Inc.
24 //  51 Franklin Street, Fifth Floor
25 //  Boston, MA  02110-1301 USA
26 //
27 //  Further, no use of this source code is permitted in any form or means
28 //  without inclusion of this banner prominently in any derived works.
29 //
30 //  Michael A. Morris <morrisma_at_mchsi_dot_com>
31 //  164 Raleigh Way
32 //  Huntsville, AL 35811
33 //  USA
34 //
35 //////////////////////////////////////////////////////////////////////////////
36
37 `timescale 1ns / 1ps
38
39 //////////////////////////////////////////////////////////////////////////////
40 // Company:         M. A. Morris & Associates
41 // Engineer:        Michael A. Morris
42 //
43 // Create Date:     14:50:14 09/15/2013
44 // Design Name:     WDC W65C02 Microprocessor Re-Implementation
45 // Module Name:     M65C02_Add
46 // Project Name:    C:\XProjects\ISE10.1i\M6502A
47 // Target Devices:  Generic SRAM-based FPGA
48 // Tool versions:   Xilinx ISE10.1i SP3
49 //
50 // Description:
51 //
52 // Dependencies:    None.
53 //
54 // Revision:
55 //
56 //  0.01    13I15   MAM     Created module based on the M65C02A_Add.v module.
57 //
58 //  1.00    13K09   MAM     Reverted BCD addition/subtraction to single-cycle
59 //                          operation -- Removed pipeline registers for BCD
60 //                          results. Thus, binary addition/subtraction performed
61 //                          during first half of the cycle, and BCD adjustment
62 //                          performed during the second half of the cycle.
63 //
64 //  1.01    14F21   MAM     Put the module back to the state indicated for 1.00.
65 //
66 // Additional Comments:
67 //
68 //////////////////////////////////////////////////////////////////////////////
69
70 module M65C02_Add(
71     input   Rst,                // Module Reset
72     input   Clk,                // System Clock
73
74     input   En_AU,              // Enable Binary Adder Functional Unit
75     input   En_DU,              // Enable Decimal Adder Functional Unit
76     input   Op,                 // Adder Operation: 0 - Addition; 1 - Subtract
77
78     input   [7:0] Q,            // Adder Input Q
79     input   [7:0] R,            // Adder Input R
```

```
    input   Ci,                 // Adder Carry In

    output  [8:0] Out,
    output  OV,
    output  Val
);

//////////////////////////////////////////////////////////////////////////////
//
//  Declarations
//

wire    [7:0] M;                // Right Operand: if(Op) M = ~R, else M = R

reg     rEn_DU;                 // Pipelined/Registered Control Signals
reg     rOp;

wire    [7:0] S;                // Intermediate Binary Sum: S <= A + B + Ci
wire    C7, C6, C3;             // Sum Carry Out from Bitn

reg     [7:0] rS;               // BCD mode first stage binary sum register
reg     rC7, rC3;               // BCD mode first stage carry registers

reg     MSN_GT9, MSN_GT8, LSN_GT9;  // Digit value comparator signals

reg     [1:0] DA;               // Decimal Adjust Controls
reg     [7:0] Adj;              // Second stage BCD adjusted registers

reg     [7:0] BCD;              // Second stage adjusted Sum
reg     BCD_Co, BCD_OV, BCD_Val;

wor     [8:0] Sum;
wor     V;
wor     Valid;

//////////////////////////////////////////////////////////////////////////////
//
//  Implementation
//

//  Capture Input Control Signals

assign Rst_Add = (Rst | (~(En_AU ^ En_DU)));

always @(negedge Clk or posedge Rst_Add)
begin
    if(Rst_Add)
        {rEn_DU, rOp} <= #1 0;
    else
        {rEn_DU, rOp} <= #1 {En_DU, Op};
end

//  Adder First Stage - Combinatorial; Binary Sums and Carries

assign M = ((Op) ? ~R : R);

assign {C3, S[3:0]} = Q[3:0] + M[3:0] + Ci;
assign {C6, S[6:4]} = Q[6:4] + M[6:4] + C3;
assign {C7, S[7]  } = Q[7]   + M[7]   + C6;

//  Adder Binary Stage - Output (Binary Sum and Carry)

assign {Valid, V, Sum} = ((En_AU) ? {En_AU, (C7 ^ C6), {C7, S}} : 0);

//  Propagate binary sum and carry to Adder Second Stage - Decimal Adder

always @(negedge Clk or posedge Rst_Add)
begin
    if(Rst_Add)
        {rC7, rC3, rS} <= #1 0;
    else if(En_DU)
        {rC7, rC3, rS} <= #1 {C7, C3, S};
end

//  Generate Digit/Nibble Value Comparators

always @(*)
begin
    case(rS[7:4])
```

```
159         4'b0000 : {MSN_GT9, MSN_GT8} <= 2'b00;
160         4'b0001 : {MSN_GT9, MSN_GT8} <= 2'b00;
161         4'b0010 : {MSN_GT9, MSN_GT8} <= 2'b00;
162         4'b0011 : {MSN_GT9, MSN_GT8} <= 2'b00;
163         4'b0100 : {MSN_GT9, MSN_GT8} <= 2'b00;
164         4'b0101 : {MSN_GT9, MSN_GT8} <= 2'b00;
165         4'b0110 : {MSN_GT9, MSN_GT8} <= 2'b00;
166         4'b0111 : {MSN_GT9, MSN_GT8} <= 2'b00;
167         4'b1000 : {MSN_GT9, MSN_GT8} <= 2'b00;
168         4'b1001 : {MSN_GT9, MSN_GT8} <= 2'b01;
169         4'b1010 : {MSN_GT9, MSN_GT8} <= 2'b11;
170         4'b1011 : {MSN_GT9, MSN_GT8} <= 2'b11;
171         4'b1100 : {MSN_GT9, MSN_GT8} <= 2'b11;
172         4'b1101 : {MSN_GT9, MSN_GT8} <= 2'b11;
173         4'b1110 : {MSN_GT9, MSN_GT8} <= 2'b11;
174         4'b1111 : {MSN_GT9, MSN_GT8} <= 2'b11;
175     endcase
176 end
177
178 always @(*)
179 begin
180     case(rS[3:0])
181         4'b0000 : LSN_GT9 <= 0;
182         4'b0001 : LSN_GT9 <= 0;
183         4'b0010 : LSN_GT9 <= 0;
184         4'b0011 : LSN_GT9 <= 0;
185         4'b0100 : LSN_GT9 <= 0;
186         4'b0101 : LSN_GT9 <= 0;
187         4'b0110 : LSN_GT9 <= 0;
188         4'b0111 : LSN_GT9 <= 0;
189         4'b1000 : LSN_GT9 <= 0;
190         4'b1001 : LSN_GT9 <= 0;
191         4'b1010 : LSN_GT9 <= 1;
192         4'b1011 : LSN_GT9 <= 1;
193         4'b1100 : LSN_GT9 <= 1;
194         4'b1101 : LSN_GT9 <= 1;
195         4'b1110 : LSN_GT9 <= 1;
196         4'b1111 : LSN_GT9 <= 1;
197     endcase
198 end
199
200 //  Adder Decimal Stage - Decimal Adjustment Factor
201
202 always @(*)
203 begin
204     if(rEn_DU)  // Generate Decimal Mode Digit Adjust Signals
205         if(rOp) begin   // SBC Operations
206             DA[1] <= ~rC7 | (~rC3 & MSN_GT9);
207             DA[0] <= ~rC3;
208         end else begin  // ADC Operations
209             DA[1] <= (rC7 | MSN_GT9 | (MSN_GT8 & ~rC3 & LSN_GT9));
210             DA[0] <=  rC3 | LSN_GT9;
211         end
212     else
213         DA <= 0;
214 end
215
216 //  Adder Decimal Stage - Decimal Adjustment Factor Selection
217
218 always @(*)
219 begin
220      case({rOp, DA})
221         3'b000 : Adj <= 8'h00;  // 0
222         3'b001 : Adj <= 8'h06;  // ±06 BCD
223         3'b010 : Adj <= 8'h60;  // ±60 BCD
224         3'b011 : Adj <= 8'h66;  // ±66 BCD
225         3'b100 : Adj <= 8'h00;  // 0
226         3'b101 : Adj <= 8'hFA;  // ±06 BCD
227         3'b110 : Adj <= 8'hA0;  // ±60 BCD
228         3'b111 : Adj <= 8'h9A;  // ±66 BCD
229     endcase
230 end
231
232 //  Adder Second Stage Result Register
233
234 always @(*)
235 begin
236     {BCD, BCD_Val, BCD_OV, BCD_Co} <= {(rS[7:0] + Adj),
237                                        rEn_DU,
```

```
238                                     DA[1],
239                                     (rOp ^ DA[1])    };
240 end
241
242 //  Adder Decimal Stage - Output
243
244 assign {Valid, V, Sum} = ((BCD_Val) ? {BCD_Val, BCD_OV, {BCD_Co, BCD}} : 0);
245
246 //  Adder Output Valid and Overflow Outputs
247
248 assign Out = Sum;
249 assign OV  = V;
250 assign Val = Valid;
251
252 endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
//  ALU Register Write Cntl module for M65C02A soft-core microcomputer project.
//
//  Copyright 2013-2014 by Michael A. Morris, dba M. A. Morris & Associates
//
//  All rights reserved. The source code contained herein is publicly released
//  under the terms and conditions of the GNU General Public License as conveyed
//  in the license provided below.
//
//  This program is free software: you can redistribute it and/or modify it
//  under the terms of the GNU General Public License as published by the Free
//  Software Foundation, either version 3 of the License, or any later version.
//
//  This program is distributed in the hope that it will be useful, but WITHOUT
//  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
//  more details.
//
//  You should have received a copy of the GNU General Public License along with
//  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
//
//  Free Software Foundation, Inc.
//  51 Franklin Street, Fifth Floor
//  Boston, MA  02110-1301 USA
//
//  Further, no use of this source code is permitted in any form or means
//  without inclusion of this banner prominently in any derived works.
//
//  Michael A. Morris <morrisma_at_mchsi_dot_com>
//  164 Raleigh Way
//  Huntsville, AL 35811
//  USA
//
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:         M. A. Morris & Associates
// Engineer:        Michael A. Morris
//
// Create Date:     19:43:14 09/15/2013
// Design Name:     WDC W65C02 Microprocessor Re-Implementation
// Module Name:     M65C02_WrSel.v
// Project Name:    C:\XProjects\ISE10.1i\MAM6502
// Target Devices:  Generic SRAM-based FPGA
// Tool versions:   Xilinx ISE10.1i SP3
//
// Description:
//
//
// Dependencies:
//
// Revision:
//
//  0.01    13I15   MAM     File Created
//
//  1.00    13J07   MAM     Updated headers and modified the original implemen-
//                          tation to use two ROMs instead of one 64x5 ROM. One
//                          ROM provides a direct decode of the Reg_WE field,
//                          the other provides a decode of the WSel fiedl when
//                          (Reg_WE == 3'b100). This change provides a signifi-
//                          cant improvement to the M65C02_ALUv2 module when
//                          synthesize as a standalone module.
//
//  1.10    14F28   MAM     Adjusted comments to reflect changes made during
//                          integration: (1) corrected the uP driven write
//                          enable logic to select PSW when writing A, X, or Y.
//
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module M65C02_WrSel(
    input   Rst,
    input   Clk,

    input   [2:0] Reg_WE,
```

```
 80     input   [2:0] WSel,
 81
 82     output  reg SelA,
 83     output  reg SelX,
 84     output  reg SelY,
 85     output  reg SelP,
 86
 87     output  reg SelS
 88 );
 89
 90 //////////////////////////////////////////////////////////////////////////////
 91 //
 92 //  Declarations
 93 //
 94
 95 reg     uP_A, uP_X, uP_Y, uP_P, uP_S;
 96 reg     iD_A, iD_X, iD_Y, iD_P, iD_S;
 97 wire    WE;
 98
 99 //  Decode Register Write Enables
100
101 always @(*)
102 begin
103     case(Reg_WE)
104         3'b000 : {uP_A, uP_X, uP_Y, uP_P, uP_S} <= #1 5'b000_0_0;
105         3'b001 : {uP_A, uP_X, uP_Y, uP_P, uP_S} <= #1 5'b100_1_0;
106         3'b010 : {uP_A, uP_X, uP_Y, uP_P, uP_S} <= #1 5'b010_1_0;
107         3'b011 : {uP_A, uP_X, uP_Y, uP_P, uP_S} <= #1 5'b001_1_0;
108         3'b100 : {uP_A, uP_X, uP_Y, uP_P, uP_S} <= #1 5'b000_0_0;
109         3'b101 : {uP_A, uP_X, uP_Y, uP_P, uP_S} <= #1 5'b000_0_1;
110         3'b110 : {uP_A, uP_X, uP_Y, uP_P, uP_S} <= #1 5'b000_1_0;
111         3'b111 : {uP_A, uP_X, uP_Y, uP_P, uP_S} <= #1 5'b000_0_0;
112     endcase
113 end
114
115 assign WE = (Reg_WE == 3'b100);
116
117 always @(*)
118 begin
119     case({WE, WSel})
120         4'b0000 : {iD_A, iD_X, iD_Y, iD_P, iD_S} <= #1 5'b000_0_0;
121         4'b0001 : {iD_A, iD_X, iD_Y, iD_P, iD_S} <= #1 5'b000_0_0;
122         4'b0010 : {iD_A, iD_X, iD_Y, iD_P, iD_S} <= #1 5'b000_0_0;
123         4'b0011 : {iD_A, iD_X, iD_Y, iD_P, iD_S} <= #1 5'b000_0_0;
124         4'b0100 : {iD_A, iD_X, iD_Y, iD_P, iD_S} <= #1 5'b000_0_0;
125         4'b0101 : {iD_A, iD_X, iD_Y, iD_P, iD_S} <= #1 5'b000_0_0;
126         4'b0110 : {iD_A, iD_X, iD_Y, iD_P, iD_S} <= #1 5'b000_0_0;
127         4'b0111 : {iD_A, iD_X, iD_Y, iD_P, iD_S} <= #1 5'b000_0_0;
128         4'b1000 : {iD_A, iD_X, iD_Y, iD_P, iD_S} <= #1 5'b000_0_0;
129         4'b1001 : {iD_A, iD_X, iD_Y, iD_P, iD_S} <= #1 5'b100_1_0;
130         4'b1010 : {iD_A, iD_X, iD_Y, iD_P, iD_S} <= #1 5'b010_1_0;
131         4'b1011 : {iD_A, iD_X, iD_Y, iD_P, iD_S} <= #1 5'b001_1_0;
132         4'b1100 : {iD_A, iD_X, iD_Y, iD_P, iD_S} <= #1 5'b000_0_0;
133         4'b1101 : {iD_A, iD_X, iD_Y, iD_P, iD_S} <= #1 5'b000_0_1;
134         4'b1110 : {iD_A, iD_X, iD_Y, iD_P, iD_S} <= #1 5'b000_1_0;
135         4'b1111 : {iD_A, iD_X, iD_Y, iD_P, iD_S} <= #1 5'b000_1_0;
136     endcase
137 end
138
139 always @(negedge Clk or posedge Rst)
140 begin
141     if(Rst)
142         {SelA, SelX, SelY, SelP, SelS} <= #1 0;
143     else begin
144         SelA <= #1 (uP_A | iD_A);
145         SelX <= #1 (uP_X | iD_X);
146         SelY <= #1 (uP_Y | iD_Y);
147         //
148         SelP <= #1 (uP_P | iD_P);
149         //
150         SelS <= #1 (uP_S | iD_S);
151     end
152 end
153
154 endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
//  ALU Status Register module for M65C02A soft-core microcomputer project.
//
//  Copyright 2013-2014 by Michael A. Morris, dba M. A. Morris & Associates
//
//  All rights reserved. The source code contained herein is publicly released
//  under the terms and conditions of the GNU General Public License as conveyed
//  in the license provided below.
//
//  This program is free software: you can redistribute it and/or modify it
//  under the terms of the GNU General Public License as published by the Free
//  Software Foundation, either version 3 of the License, or any later version.
//
//  This program is distributed in the hope that it will be useful, but WITHOUT
//  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
//  more details.
//
//  You should have received a copy of the GNU General Public License along with
//  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
//
//  Free Software Foundation, Inc.
//  51 Franklin Street, Fifth Floor
//  Boston, MA  02110-1301 USA
//
//  Further, no use of this source code is permitted in any form or means
//  without inclusion of this banner prominently in any derived works.
//
//  Michael A. Morris <morrisma_at_mchsi_dot_com>
//  164 Raleigh Way
//  Huntsville, AL 35811
//  USA
//
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:         M. A. Morris & Associates
// Engineer:        Michael A. Morris
//
// Create Date:     22:14:57 10/02/2013
// Design Name:     WDC W65C02 Microprocessor Re-Implementation
// Module Name:     M65C02_PSWv2.v
// Project Name:    C:\XProjects\ISE10.1i\M6502A
// Target Devices:  Generic SRAM-based FPGA
// Tool versions:   Xilinx ISE10.1i SP3
//
// Description:
//
//  Module implements the PSW of the M65C02_ALUv2 module.
//
// Dependencies:    None
//
// Revision:
//
//  0.01    13J02   MAM     File Created
//
//  1.00    13J02   MAM     Reverted back to using a case statement for genera-
//                          ing multiplexer for a unitary 6-bit PSW register.
//                          However, optimized the encoding of the least signi-
//                          ficant 3 bits of the 4-bit CCSel field to maximize
//                          performance and logic reduction. Left implementation
//                          discussed below as comments after the released code.
//
//  1.01    14A19   MAM     Note: two case statement members, pBRK & pPHP, are
//                          included in the WE_P case statement. These CC select
//                          conditions do not assert WE_P, but are included in
//                          case statement below to allow the synthesizer to
//                          further optimize the control logic/multiplexer for
//                          each bit in the PSW register. Added comments to case
//                          statement members pBRK and pPHP.
//
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module M65C02_PSWv2(
```

```
 80     input   Clk,
 81
 82     input   SO,
 83     output  Clr_SO,
 84
 85     input   SelP,
 86     input   Valid,
 87     input   Rdy,
 88
 89     input   ISR,
 90
 91     input   [2:0] CCSel,
 92
 93     input   [7:6] M,
 94     input   OV,
 95     input   LU_Z,
 96     input   COut,
 97     input   [7:0] DO,
 98
 99     output  [7:0] P,
100
101     output  N,
102     output  V,
103     output  D,
104     output  Z,
105     output  C
106 );
107
108 ////////////////////////////////////////////////////////////////////////////
109 //
110 //  Local Parameters
111 //
112
113 localparam pPSW  = 3'b000;  // Set P from ALU
114 localparam pBRK  = 3'b001;  // Set P.4 when pushing P during interrupt handling
115 localparam pZ    = 3'b010;  // Set Z = ~|(A & M)
116 localparam pNVZ  = 3'b011;  // Set N and V flags from M[7:6], and Z = ~|(A & M)
117 localparam pPHP  = 3'b100;  // Set P.4 when executing PHP instruction
118 localparam pNZ   = 3'b101;  // Set N and Z flags from ALU
119 localparam pNZC  = 3'b110;  // Set N, Z, and C flags from ALU
120 localparam pNVZC = 3'b111;  // Set N, V, Z, and C from ALU
121
122 ////////////////////////////////////////////////////////////////////////////
123 //
124 //  Declarations
125 //
126
127 reg     [5:0] PSW;
128 wire    I;
129
130 ////////////////////////////////////////////////////////////////////////////
131 //
132 //  P - Processor Status Word: {N, V, 1, B, D, I, Z, C}
133 //
134
135 assign WE_P = (SelP | SO) & Valid & Rdy;
136
137 always @(posedge Clk)
138 begin
139     if(ISR & Rdy)
140         PSW <= #1 {1'b0, 1'b0, 1'b0, 1'b1, 1'b0, 1'b0};
141     else if(WE_P)
142         case(CCSel)
143             pPSW  : PSW <= #1 {DO[7],(DO[6] | SO), DO[3:0]};
144             pBRK  : PSW <= #1 {DO[7],(DO[6] | SO), DO[3:0]};
145             pZ    : PSW <= #1 {    N,(V     | SO),   D,   I, LU_Z,   C};
146             pNVZ  : PSW <= #1 { M[7],(M[6]  | SO),   D,   I, LU_Z,   C};
147             pPHP  : PSW <= #1 {DO[7],(V     | SO),   D,   I,~|DO ,   C};
148             pNZ   : PSW <= #1 {DO[7],(V     | SO),   D,   I,~|DO ,   C};
149             pNZC  : PSW <= #1 {DO[7],(V     | SO),   D,   I,~|DO ,COut};
150             pNVZC : PSW <= #1 {DO[7],(OV    | SO),   D,   I,~|DO ,COut};
151         endcase
152 end
153
154 //  Decode PSW bits
155
156 assign N = PSW[5];  // Negative, nominally Out[7], but M[7] if BIT/TRB/TSB
157 assign V = PSW[4];  // oVerflow, nominally OV,     but M[6] if BIT/TRB/TSB
158 assign D = PSW[3];  // Decimal, set/cleared by SED/CLD, cleared on ISR entry
```

```
159 assign I = PSW[2];  // Interrupt Mask, set/cleared by SEI/CLI, set on ISR entry
160 assign Z = PSW[1];  // Zero, nominally ~|Out, but ~|(A&M) if BIT/TRB/TSB
161 assign C = PSW[0];  // Carry, set by ADC/SBC, and ASL/ROL/LSR/ROR instructions
162
163 //  Assign PSW bits to P (PSW output port)
164
165 assign BRK = (CCSel == pBRK);
166 assign PHP = (CCSel == pPHP);
167
168 assign P = {N, V, 1'b1, (BRK | PHP), D, I, Z, C};
169
170 //  Generate for Acknowledge SO Command
171
172 assign Clr_SO = SO & Valid & Rdy;
173
174 endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
//  MMU module for M65C02A soft-core microcomputer project.
//
//  Copyright 2013-2014 by Michael A. Morris, dba M. A. Morris & Associates
//
//  All rights reserved. The source code contained herein is publicly released
//  under the terms and conditions of the GNU General Public License as conveyed
//  in the license provided below.
//
//  This program is free software: you can redistribute it and/or modify it
//  under the terms of the GNU General Public License as published by the Free
//  Software Foundation, either version 3 of the License, or any later version.
//
//  This program is distributed in the hope that it will be useful, but WITHOUT
//  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
//  more details.
//
//  You should have received a copy of the GNU General Public License along with
//  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
//
//  Free Software Foundation, Inc.
//  51 Franklin Street, Fifth Floor
//  Boston, MA  02110-1301 USA
//
//  Further, no use of this source code is permitted in any form or means
//  without inclusion of this banner prominently in any derived works.
//
//  Michael A. Morris <morrisma_at_mchsi_dot_com>
//  164 Raleigh Way
//  Huntsville, AL 35811
//  USA
//
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:         M. A. Morris & Associates
// Engineer:        Michael A. Morris
//
// Create Date:     09:39:42 12/21/2013
// Design Name:     WDC W65C02 Microprocessor Re-Implementation
// Module Name:     M65C02_MMU
// Project Name:    C:\XProjects\ISE10.1i\M65C02Duo
// Target Devices:  Generic SRAM-based FPGA
// Tool versions:   Xilinx ISE10.1i SP3
//
// Description:
//
//  This module implements a simple memory management unit for the M65C02 soft-
//  core microprocessor. It uses dual-ported distributed memory to map a virtual
//  address to a physical address and a chip enable. The uppermost 4 bits of the
//  16-bit M65C02 virtual address plus a Kernel/User mode input is used to
//  address a set of 32 memory mapping registers.
//
//  This MMU only provides simple address substitution and chip select genera-
//  tion. Address arithmetic and range checking could be added, but the addi-
//  tional logic in the critical address output data path will result in lower
//  performance. Provisions are included in the module to allow the module to be
//  expanded to support Kernel/User mode and access controls for virtual memory.
//  The present implementation only supports a simple mapping from virtual to
//  physical address, but is does allow the total address space to be expanded
//  to more that 4MB.
//
//  The 16-bit mapping register is defined by the following structure:
//
//      MMU <= {(Rsvd[2:0], WS, CS[3:0], PA[19:12]}
//
//  Rsvd[2:0]   : reserved for future use as access controls for each segment.
//  WS          : insert a number of internal wait states determined by pWS_Out.
//  CS[3:0]     : selects the chip select to use for each segment.
//  PA[19:12]   : most significant 8 bits of the physical address of a segment.
//
//  The uppermost three bits are currently not implemented. There are plans to
//  modify the M65C02 core to support Kernel/User mode and potentially virtual
//  memory. These three bits are reserved for this purpose at this time in order
//  that a future MMU module can be dropped into the design without any signifi-
```

```
 80 //  cant changes required to the implementation of the M65C02A microprocessor.
 81 //
 82 // Dependencies:    None
 83 //
 84 // Revision:
 85 //
 86 //  0.00    13L21   MAM     Initial File Creation
 87 //
 88 //  1.00    14G04   MAM     Modified to support 32 2kB pages. Reduced the PA
 89 //                          range from PA[19:0] to PA[18:0]. Changes total
 90 //                          potential external address space from 4MB to 2MB.
 91 //
 92 //  1.10    14G28   MAM     Restored the external address space to 4MB. Setup
 93 //                          the module to support Kernel/User mode and access
 94 //                          controls.
 95 //
 96 // Additional Comments:
 97 //
 98 ////////////////////////////////////////////////////////////////////////////////
 99
100 module M65C02_MMU #(
101     parameter pMAP_Init = "Pgms/M65C02_MMU32.coe"  // MAP RAM Initialization
102 )(
103     input   Rst,                        // System Reset
104     input   Clk,                        // System Clock
105
106     input   Rdy,                        // M65C02A Microcycle Ready Signal
107
108     // MMU General Inputs
109
110     input   Mode,                       // M65C02A Operating Mode
111     input   Sync,                       // M65C02A Instruction Fetch Strobe
112     input   [ 1:0] IO_Op,               // M65C02A IO Operation
113
114     input   [15:0] VA,                  // M65C02A Virtual Address
115
116     // MAP/MMU/Vec Management Interface
117
118     input   Sel_MAP,                    // MMU Mapping Registers Select
119     input   Sel_MMU,                    // MMU Status Register Select
120     input   WE,                         // MMU Write Enable
121     input   RE,                         // MMU Read Enable
122     input   [4:0] Sel,                  // MMU Register Select
123     input   [7:0] MMU_DI,               // MMU Register Data Input
124     output  [7:0] MMU_DO,               // MMU Register Data Output
125
126     // MAP Virtual-Physical Mapping Interface
127
128     output  [19:0] PA,                  // MMU Physical (Mapped) Address Output
129     output  [15:1] CE,                  // MMU Chip Enable Output
130     output  Int_WS,                     // MMU Interanal Wait State Request
131
132     // MAP Access Violation and Mapping Error Interface
133
134     output  ABRT                        // Memory Cycle ABoRT trap output
135 );
136
137 ////////////////////////////////////////////////////////////////////////////////
138 //
139 //  Declarations
140 //
141
142 reg     [15:0] MAP [31:0];              // MMU Map RAM - LUT-based Dual-Port RAM
143 wire    [15:0] MAP_DI, MAP_SPO;         // MMU Map RAM Data I/O Busses
144 wire    MAP_WE, MAP_RE;                 // MMU Map RAM Write and Read Enables
145
146 wire    [ 4:0] Page;                    // MMU Map RAM Page Select
147 wire    [15:0] MAP_DPO;                 // MMU Map RAM Dual-Port RAM Output
148 reg     [15:1] CS_Out;                  // One-Hot Decode Chip Select (CS)
149
150 ////////////////////////////////////////////////////////////////////////////////
151 //
152 //  Implementation
153 //
154
155 //  Implement Dual-Ported Distributed RAM for MMU MAP
156
157 initial
158     $readmemh(pMAP_Init, MAP, 0, 31);
```

```
159
160 assign MAP_WE = Sel_MAP & WE & Rdy;
161 assign MAP_DI = ((VA[0]) ? {MMU_DI, MAP_SPO[7:0]} : {MAP_SPO[15:8], MMU_DI});
162
163 always @(posedge Clk)
164 begin
165     if(MAP_WE)
166         MAP[Sel] <= #1 MAP_DI;
167 end
168
169 assign MAP_SPO = MAP[Sel];
170
171 //  Provide multiplexer to read out the contents of the selected MMU register
172
173 assign MAP_RE = Sel_MAP & RE;
174 assign MMU_DO = ((MAP_RE) ? ((VA[0]) ? MAP_SPO[15:8] : MAP_SPO[7:0]) : 0);
175
176 ////////////////////////////////////////////////////////////////////////////
177 //
178 //  MMU Mapping Logic
179 //
180
181 //  Map Virtual Address into Wait State, Chip Enable, and Physical Address
182
183 assign Page    = {Mode, VA[15:12]};
184 assign MAP_DPO = MAP[Page];
185
186 //  Map the 4-bit encoded CS field of the MAP RAM into One-Hot CS output
187
188 always @(*)
189 begin
190     case(MAP_DPO[11:8])
191         4'b0000  : CS_Out[15:1] <= 15'b00000000_0000000;
192         4'b0001  : CS_Out[15:1] <= 15'b00000000_0000001;
193         4'b0010  : CS_Out[15:1] <= 15'b00000000_0000010;
194         4'b0011  : CS_Out[15:1] <= 15'b00000000_0000100;
195         4'b0100  : CS_Out[15:1] <= 15'b00000000_0001000;
196         4'b0101  : CS_Out[15:1] <= 15'b00000000_0000000;
197         4'b0110  : CS_Out[15:1] <= 15'b00000000_0000000;
198         4'b0111  : CS_Out[15:1] <= 15'b00000000_0000000;
199         4'b1000  : CS_Out[15:1] <= 15'b00000001_0000000;
200         4'b1001  : CS_Out[15:1] <= 15'b00000010_0000000;
201         4'b1010  : CS_Out[15:1] <= 15'b00000100_0000000;
202         4'b1011  : CS_Out[15:1] <= 15'b00001000_0000000;
203         4'b1100  : CS_Out[15:1] <= 15'b00010000_0000000;
204         4'b1101  : CS_Out[15:1] <= 15'b00100000_0000000;
205         4'b1110  : CS_Out[15:1] <= 15'b01000000_0000000;
206         4'b1111  : CS_Out[15:1] <= 15'b10000000_0000000;
207     endcase
208 end
209
210 //  Output the mapped physical address (PA), CE, and WS signals
211
212 assign PA     = {MAP_DPO[7:0], VA[11:0]};
213 assign CE     = CS_Out;
214 assign Int_WS = MAP_DPO[12];
215
216 ////////////////////////////////////////////////////////////////////////////
217 //
218 //  MMU Access Validation and Page Modification Logic
219 //
220
221 assign ABRT = 1'b0;
222
223 endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
//  SPI Master module for M65C02A soft-core microcomputer project.
//
//  Copyright 2013-2014 by Michael A. Morris, dba M. A. Morris & Associates
//
//  All rights reserved. The source code contained herein is publicly released
//  under the terms and conditions of the GNU General Public License as conveyed
//  in the license provided below.
//
//  This program is free software: you can redistribute it and/or modify it
//  under the terms of the GNU General Public License as published by the Free
//  Software Foundation, either version 3 of the License, or any later version.
//
//  This program is distributed in the hope that it will be useful, but WITHOUT
//  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
//  more details.
//
//  You should have received a copy of the GNU General Public License along with
//  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
//
//  Free Software Foundation, Inc.
//  51 Franklin Street, Fifth Floor
//  Boston, MA  02110-1301 USA
//
//  Further, no use of this source code is permitted in any form or means
//  without inclusion of this banner prominently in any derived works.
//
//  Michael A. Morris <morrisma_at_mchsi_dot_com>
//  164 Raleigh Way
//  Huntsville, AL 35811
//  USA
//
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:         Michael A. Morris
// Engineer:        M. A. Morris & Associates
//
// Create Date:     17:27:03 12/27/2013
// Design Name:     M65C02 Dual Core
// Module Name:     M65C02_SPIxIF.v
// Project Name:    C:\XProjects\ISE10.1i\M65C02Duo
// Target Devices:  Generic SRAM-based FPGA
// Tool versions:   Xilinx ISE10.1i SP3
//
// Description:
//
//  This module provides a wrapper for an SPI Master Interface that interfaces
//  that module to the memory/IO bus of the M65C02/M65CO2Duo microprocessor.
//  The wrapper provides a control register, a status register, a transmit FIFO
//  interface, and a receive FIFO interface.
//
//  The control register is a R/W register. It controls the features of the
//  SPIxIF module, and provides a way to read the current value of the register.
//  The status register is a RO register. It allows the status of the SPI Master
//  interface to be monitored. Specifically, the Empty/Full status of the Tx/Rx
//  FIFOs can be monitored.
//
//  The SPIxIF module provides a means by which the data received simultaneously
//  with the transmitted data can be ignored or written into the receive data
//  FIFO. With SPI memory devices, the data received from the devices before the
//  command and the address have been written are undefined. Using this capabi-
//  lity of the SPIxIF, it is possible to ignore all data received from these
//  devices until after the command and address have been written.
//
//  To provide this feature, the SPIxIF module utilizes a bit written into the
//  TD FIFO. This asdditional TD bit is used to enable or disable the capturing
//  of the receive data. If the 9th TD bit is set, then the serial data captured
//  during the transmission of the TD data is saved in the RD FIFO. If the 9th
//  bit is clear, i.e. not set, then the serial data captured during the trans-
//  mission of the TD is NOT saved in the RD FIFO.
//
//  The M65C02_SPIxIF allocates two registers for this purpose. The first regis-
//  ter is set on an even address, and writes to this register clear the 9th TD
//  bit. This results in the data received being ignored. The second register is
```

```
 80 //  set on an odd address, and writes to this register set the 9th TD bit. This
 81 //  results in the data received being captured in the RD FIFO. Writes to these
 82 //  logical registers result in writing of data to a single TD FIFO with the 9th
 83 //  bit clear or set.
 84 //
 85 //  The single RD FIFO is mapped as a RO component to the same addresses as the
 86 //  single, write-only (WO) TD FIFO. Reading from the WO TD FIFO addresses reads
 87 //  from RD FIFO. Either TD FIFO address can be used for reading the RD FIFO.
 88 //
 89 // Dependencies:
 90 //
 91 // Revision:
 92 //
 93 //  0.00    13L24   MAM     Initial File Creation.
 94 //
 95 // Additional Comments:
 96 //
 97 //  The address/register map of this module is provided in the following table:
 98 //
 99 //  Reg[1:0]    Mode    Name    Description
100 //     00        RW      CR     SPIxIF Control Register
101 //     01        RO      SR     SPIxIF Status Register
102 //     10        WO      TD0    Transmit Data FIFO - Bit 9 Clr
103 //     11        WO      TD1    Transmit Data FIFO - Bit 9 Set
104 //     1x        RO      RD     Receive Data FIFO
105 //
106 //  The SPIxIF Control Register is defined in the following table:
107 //
108 //  Bit     Name    Description
109 //   0      Port    SPI Port/Slave Select: 1 - SSel[1]; 0 - SSel[0]
110 //   1      Man     SSel Operation: 0 - Auto; 1 - Manual
111 //  3:2     Mode    SPI Operating Mode: 0, 1, 2, 3
112 //  6:4     Baud    SPI Shift Clock Rate: Clk/(2**(Baud + 1))
113 //   7      IE      SPI TD FIFO Interrupt Enable: 1 - Enable; 0 - Disable
114 //
115 //  The SPIxIF Status Register is defined in the following table:
116 //
117 //  Bit     Name    Description
118 //   0      TxRdy   SPI Tx Ready: 1 - Ready (Tx FIFO not Full)
119 //   1      TF_HF   SPI Tx FIFO Half Full Flag
120 //   2      TF_EF   SPI Tx FIFO Empty Flag
121 //   3      RxErr   SPI Rx FIFO Overflow: 1 - Rx FIFO Full
122 //  6:4     Rsvd    Reseved: read as 0s
123 //   7      Active  SPI shift Active: 1 - SPI Active; 0 - Shift Completed
124 //
125 //////////////////////////////////////////////////////////////////////////////////
126
127 module M65C02_SPIxIF #(
128     parameter pDefault_CR     = 8'h30,  // Rate=1/16, Mode=0, SSel=Auto, Sel=0
129     parameter pShift_Dir      = 1'b0,   // Shift Direction: 0 - MSB, 1 - LSB
130     parameter pSPI_FIFO_Depth = 8'd4,   // Depth = (1 << 4) = 16
131     parameter pSPI_FIFO_Init  = "Src/SPI_FIFO_Init_16.coe"
132 )(
133     input   Rst,
134     input   Clk,
135
136     //  M65C02 Module Interface
137
138     input   Sel,
139     input   RE,
140     input   WE,
141
142     input   [1:0] Reg,
143     input   [7:0] DI,
144     output  wor [7:0] DO,
145
146     // M65C02 Interrupt Interface
147
148     output  IRQ,
149
150     //  SPI Interface
151
152     output  reg [1:0] SSel,
153     output  SCK,
154     output  MOSI,
155     input   MISO
156 );
157
158 //////////////////////////////////////////////////////////////////////////////////
```

```
159 //
160 //  Local Parameters
161 //
162
163 localparam pFIFO_Depth = (2**(pSPI_FIFO_Depth));
164
165 localparam pSel_CR  = 2'b00;
166 localparam pSel_SR  = 2'b01;
167 localparam pSel_TD0 = 2'b10;
168 localparam pSel_TD1 = 2'b11;
169 localparam pSel_RD0 = 2'b10;
170 localparam pSel_RD1 = 2'b11;
171
172 ////////////////////////////////////////////////////////////////////////////
173 //
174 //  Declarations
175 //
176
177 wire    WE_CR, OE_CR;
178 reg     [7:0] CR;                       // Control Register
179
180 wire    Port, Man, IE;                  // Control Register Fields
181 wire    [1:0] Mode;
182 wire    [2:0] Rate;
183
184 wire    RE_SR, OE_SR;
185 wire    [7:0] SR;                       // Status Register
186
187 wire    WE_TF, RE_TF;                   // SPI Transmit FIFO Signals
188 wire    [8:0] SPI_TD;
189 wire    TF_FF, TF_HF, TF_EF;
190
191 wire    WE_RF, RE_RF, OE_RF;            // SPI Receive FIFO Signals
192 wire    [7:0] SPI_RD, RD;
193 wire    RF_FF;
194
195 wire    SS;                             // SPI Slave Select Strobe
196
197 wire    feRE_SR, feTF_EF, reTF_EF;      // Interrupt Generator
198 wire    Rst_Int;
199 reg     Int;
200
201 ////////////////////////////////////////////////////////////////////////////
202 //
203 //  Implementation
204 //
205
206 //  Control Register
207
208 assign WE_CR = (Sel & (Reg == pSel_CR)) & WE;
209 assign OE_CR = (Sel & (Reg == pSel_CR));
210
211 always @(posedge Clk)
212 begin
213     if(Rst)
214         CR <= #1 pDefault_CR;
215     else if(WE_CR)
216         CR <= #1 DI;
217 end
218
219 //  Decode Control Register
220
221 assign Port = CR[0];                    // SPI Port/Slave Select
222 assign Man  = CR[1];                    // SPI SSel Operation: 0-Auto; 1-Man
223 assign Mode = CR[3:2];                  // SPI Mode Select
224 assign Rate = CR[6:4];                  // SPI Shift Clock Rate Select
225 assign IE   = CR[7];                    // SPI Interrupt Enable
226
227 //  Status Register
228
229 assign RE_SR = (Sel & (Reg == pSel_SR) & RE);
230 assign OE_SR = (Sel & (Reg == pSel_SR));
231
232 assign SR = {SS, 3'b000, RF_FF, TF_EF, TF_HF, ~TF_FF};
233
234 //  Instantiate Transmit and Receive Data FIFOs
235
236 assign WE_TF = Sel & Reg[1] & WE;
237
```

```
238 DPSFnmCE    #(
239                 .addr(pSPI_FIFO_Depth),
240                 .width(9),
241                 .init(pSPI_FIFO_Init)
242             ) TF (
243                 .Rst(Rst),
244                 .Clk(Clk),
245 
246                 .WE(WE_TF),
247                 .DI({Reg[0], DI}),
248 
249                 .RE(RE_TF),
250                 .DO(SPI_TD),
251 
252                 .FF(TF_FF),
253                 .HF(TF_HF),
254                 .EF(TF_EF),
255 
256                 .Cnt()
257             );
258 
259 assign RE_RF = Sel & Reg[1] & RE;
260 assign OE_RF = Sel & Reg[1];
261 
262 DPSFnmCE    #(
263                 .addr(pSPI_FIFO_Depth),
264                 .width(8),
265                 .init(pSPI_FIFO_Init)
266             ) RF (
267                 .Rst(Rst),
268                 .Clk(Clk),
269 
270                 .WE(WE_RF),
271                 .DI(SPI_RD),
272 
273                 .RE(RE_RF),
274                 .DO(RD),
275 
276                 .FF(RF_FF),
277                 .EF(),
278                 .HF(),
279 
280                 .Cnt()
281             );
282 
283 //  Instantiate SPIxIF Module
284 
285 SPIxIF  SPI (
286             .Rst(Rst),
287             .Clk(Clk),
288 
289             .LSB(pShift_Dir),
290             .Mode(Mode),
291             .Rate(Rate),
292 
293             .DAV(~TF_EF),
294 
295             .FRE(RE_TF),
296             .TD(SPI_TD),
297 
298             .FWE(WE_RF),
299             .RD(SPI_RD),
300 
301             .SS(SS),
302             .SCK(SCK),
303             .MOSI(MOSI),
304             .MISO(MISO)
305         );
306 
307 //  Assign SS to selected port
308 
309 always @(*)
310 begin
311     case({Man, Port})
312         2'b00 : SSel <= {1'b0, SS  };
313         2'b01 : SSel <= {SS  , 1'b0};
314         2'b10 : SSel <= {1'b0, 1'b1};
315         2'b11 : SSel <= {1'b1, 1'b0};
316     endcase
```

```
317 end
318
319 //  Assign Interrupt Request Output
320
321 fedet   FE1 (.rst(Rst), .clk(Clk), .din(RE_SR),  .pls(feRE_SR));
322 fedet   FE2 (.rst(Rst), .clk(Clk), .din(TF_EF),  .pls(feTF_EF));
323 redet   RE1 (.rst(Rst), .clk(Clk), .din(TF_EF),  .pls(reTF_EF));
324
325 assign Rst_Int = (Rst | feRE_SR | feTF_EF);
326
327 always @(posedge Clk or posedge Rst_Int)
328 begin
329     if(Rst_Int)
330         Int <= #1 0;
331     else if(reTF_EF)
332         Int <= #1 TF_EF;
333 end
334
335 assign IRQ = ((IE) ? Int : 0);
336
337 //  Define Data Output (DO)
338
339 assign DO = ((OE_CR) ? CR : 0);
340 assign DO = ((OE_SR) ? SR : 0);
341 assign DO = ((OE_RF) ? RD : 0);
342
343 endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
//  Parameterizable Distributed RAM Synchronous FIFO module.
//
//  Copyright 2007-2014 by Michael A. Morris, dba M. A. Morris & Associates
//
//  All rights reserved. The source code contained herein is publicly released
//  under the terms and conditions of the GNU Lesser Public License. No part of
//  this source code may be reproduced or transmitted in any form or by any
//  means, electronic or mechanical, including photocopying, recording, or any
//  information storage and retrieval system in violation of the license under
//  which the source code is released.
//
//  The source code contained herein is free; it may be redistributed and/or
//  modified in accordance with the terms of the GNU Lesser General Public
//  License as published by the Free Software Foundation; either version 2.1 of
//  the GNU Lesser General Public License, or any later version.
//
//  The source code contained herein is freely released WITHOUT ANY WARRANTY;
//  without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
//  PARTICULAR PURPOSE. (Refer to the GNU Lesser General Public License for
//  more details.)
//
//  A copy of the GNU Lesser General Public License should have been received
//  along with the source code contained herein; if not, a copy can be obtained
//  by writing to:
//
//  Free Software Foundation, Inc.
//  51 Franklin Street, Fifth Floor
//  Boston, MA  02110-1301 USA
//
//  Further, no use of this source code is permitted in any form or means
//  without inclusion of this banner prominently in any derived works.
//
//  Michael A. Morris
//  Huntsville, AL
//
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:         M. A. Morris & Associates
// Engineer:        Michael A. Morris
//
// Create Date:     12:11:30 12/22/2007
// Design Name:     HAWK Interface FPGA, 4020-0420, U35
// Module Name:     DPSFnmCE
// Project Name:    4020 HAWK ZAOM Upgrade
// Target Devices:  XC2S150-5PQ208I
// Tool versions:   ISE 8.2i
//
// Description: This module implements a parameterized version of a distributed
//              RAM synchronous FIFO. The address width, FIFO width and depth
//              are all specified by parameters. Default parameters settings
//              describe a 16x16 FIFO with Full (FF), Empty (EF), and Half
//              Full (HF) flags. The module also outputs the count words in the
//              FIFO.
//
// Dependencies:    None
//
// Revision History:
//
//  0.01    07L22   MAM     File Created
//
//  0.10    08K05   MAM     Changed depth to a localparam based on addr
//
//  1.00    13G14   MAM     Converted to Verilog 2001 standard
//
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module DPSFnmCE #(
    parameter addr  = 4,                // Sets depth of the FIFO: 2**addr
    parameter width = 16,               // Sets width of the FIFO
    parameter init  = "Src/DPSFmnRAM.coe"   // Initializes FIFO memory
)(
    input   Rst,
```

```
 80     input   Clk,
 81     input   WE,
 82     input   RE,
 83     input   [(width - 1):0] DI,
 84     output  [(width - 1):0] DO,
 85     output  FF,
 86     output  EF,
 87     output  HF,
 88     output  [addr:0] Cnt
 89 );
 90
 91 //////////////////////////////////////////////////////////////////////////////
 92 //
 93 //  Module Parameter List
 94 //
 95
 96 localparam  depth = (2**addr);
 97
 98 //////////////////////////////////////////////////////////////////////////////
 99 //
100 //  Module Level Declarations
101 //
102
103     reg     [(width - 1):0] RAM [(depth - 1):0];
104
105     reg     [ (addr - 1):0] A, DPRA;
106     reg     [ (addr - 1):0] WCnt;
107     reg     nEF, rFF;
108
109     wire    Wr, Rd, CE;
110
111 //////////////////////////////////////////////////////////////////////////////
112 //
113 //  Implementation
114 //
115
116 //
117 //  Combinatorial Control Signals
118 //
119
120 assign Wr = WE & ~FF;
121 assign Rd = RE & ~EF;
122 assign CE = Wr ^ Rd;
123
124 //
125 //  Write Address Counter
126 //
127
128 always @(posedge Clk)
129 begin
130     if(Rst)
131         A <= #1 0;
132     else if(Wr)
133         A <= #1 A + 1;
134 end
135
136 //
137 //  Read Address Counter
138 //
139
140 always @(posedge Clk)
141 begin
142     if(Rst)
143        DPRA <= #1 0;
144     else if(Rd)
145         DPRA <= #1 DPRA + 1;
146 end
147
148 //
149 //   Word Counter
150 //
151
152 always @(posedge Clk)
153 begin
154     if(Rst)
155         WCnt <= #1 0;
156     else if(Wr & ~Rd)
157         WCnt <= #1 WCnt + 1;
158     else if(Rd & ~Wr)
```

```
159         WCnt <= #1 WCnt - 1;
160 end
161
162 //
163 //  External Word Count
164 //
165
166 assign Cnt = {FF, WCnt};
167
168 //
169 //  Empty Flag Register (Active Low)
170 //
171
172 always @(posedge Clk)
173 begin
174     if(Rst)
175         nEF <= #1 0;
176     else if(CE)
177         nEF <= #1 ~(RE & (Cnt == 1));
178 end
179
180 assign EF = ~nEF;
181
182 //
183 //  Full Flag Register
184 //
185
186 always @(posedge Clk)
187 begin
188     if(Rst)
189         rFF <= #1 0;
190     else if(CE)
191         rFF <= #1 (WE & (&WCnt));
192 end
193
194 assign FF = rFF;
195
196 //
197 //  Half-Full Flag
198 //
199
200 assign HF = Cnt[addr] | Cnt[(addr - 1)];
201
202 //
203 //  Dual-Port Synchronous RAM
204 //
205
206 initial
207   $readmemh(init, RAM, 0, (depth - 1));
208
209 always @(posedge Clk)
210 begin
211     if(Wr)
212         RAM[A] <= #1 DI;    // Synchronous Write
213 end
214
215 assign DO = RAM[DPRA];      // Asynchronous Read
216
217 endmodule
218
```

```
////////////////////////////////////////////////////////////////////////////////
//
//  SPI Master Interface module.
//
//  Copyright 2008-2014 by Michael A. Morris, dba M. A. Morris & Associates
//
//  All rights reserved. The source code contained herein is publicly released
//  under the terms and conditions of the GNU General Public License as conveyed
//  in the license provided below.
//
//  This program is free software: you can redistribute it and/or modify it
//  under the terms of the GNU General Public License as published by the Free
//  Software Foundation, either version 3 of the License, or any later version.
//
//  This program is distributed in the hope that it will be useful, but WITHOUT
//  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
//  more details.
//
//  You should have received a copy of the GNU General Public License along with
//  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
//
//  Free Software Foundation, Inc.
//  51 Franklin Street, Fifth Floor
//  Boston, MA  02110-1301 USA
//
//  Further, no use of this source code is permitted in any form or means
//  without inclusion of this banner prominently in any derived works.
//
//  Michael A. Morris <morrisma_at_mchsi_dot_com>
//  164 Raleigh Way
//  Huntsville, AL 35811
//  USA
//
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:         M. A. Morris & Associates
// Engineer:        Michael A. Morris
//
// Create Date:     20:31:58 03/17/2008
// Design Name:     Serial Peripheral Interconnect (SPI) Master Interface
// Module Name:     SPIxIF.v
// Project Name:    VerilogComponentsLib\SPI and SSP Components\SPI Master
// Target Devices:  FPGA
// Tool versions:   ISE 10.1i SP3
//
// Description:
//
//  This module implements a full-duplex (Master) SPI interface. It is a major
//  revision of the previous implementation which basically implemented the SPI
//  operating modes within a Synchronous Serial Peripheral (SSP) as used in NXP
//  ARM LPC21xx microcontrollers. In an NXP SSP, an SPI-like peripheral is used
//  that has a programmable length shift register which performs serial I/O
//  data transfers in a single cycle: Slave Select (SS) is asserted and de-
//  asserted for each data transfer cycle. The only SPI-like feature of the pre-
//  vious implementation was the ability to program the clock idle state and the
//  data sampling edge, i.e. the SPI operating modes. Because a transfer cycle
//  always deasserted SS, the original implementation would require substan-
//  tial modification in order to be useful with standard SPI-compatible devices
//  such as Serial EEPROMs, serial FRAMs/MRAMs, ADCs/DACs, UARTs, I/O expanders,
//  etc.
//
//  This module will implement an SPI Master interface that avoids the limita-
//  tions of the previous implementation. It will use a fixed 8-bit interface,
//  but it will support variable length cycles. To operate in this manner, the
//  basic control interface will support an interface that can be easily attach-
//  ed to FIFOs. (The transmit FIFO will use a 9-bit interface, and the receive
//  FIFO will use an 8-bit interface.) An SPI data transfer cycle will start
//  when the transmit FIFO EF indicates that there is data to transmit, and the
//  SPI transfer cycle will terminate when the last bit is shifted out and the
//  transmit FIFO EF indicates that it is empty.
//
//  The 9th bit of the transmit data will be used to enable the writing of the
//  receive data into the receive FIFO. If the bit is not set, the data shifted
//  in during an 8-bit SPI shift cycle is not captured into the receive FIFO.
//  If the 9th bit is set, then the data shifted into the SPI shift register is
```

```
 80 //  captured into the receive FIFO. The 9th bit is expected to be part of the
 81 //  transmit FIFO, so it can be set or cleared for each 8-bit SPI transfer cy-
 82 //  cle. (Note: the 9th bit can be implemented separate from the transmit FIFO,
 83 //  but it would need to be merged into the transmit data bus by the external
 84 //  logic.)
 85 //
 86 //  The explicit read capability provided by the 9th bit is useful when working
 87 //  with SPI devices such as SPI memory devices which do not return any data
 88 //  until a command code and an address have been written. Generally, these
 89 //  devices require several command codes and address bytes to be sent to it
 90 //  before it enables its serial output signal driver and returns the requested
 91 //  data. Therefore, for these devices, the 9th bit is cleared when the command
 92 //  and address bytes are being sent, and the 9th bit set while dummy data is
 93 //  written to and the data received from the device is written into the receive
 94 //  FIFO.
 95 //
 96 //  On the other hand, there are many devices where the data on MISO is consi-
 97 //  dered valid from the onset of a transfer cycle. Devices such as ADCs provide
 98 //  data on MISO that is valid while the conversion command for the next sample
 99 //  is being simultaneously shifted out to the ADC on MOSI. For these devices,
100 //  the 9th bit is set in the transmit FIFO for each output byte. This causes
101 //  each received byte to be written to the receive FIFO.
102 //
103 //  The SPI interface operates in four modes: Mode 0, 1, 2, and 3. Generally,
104 //  the mode is selected by two control signals, CPOL and CPHA. CPHA determines
105 //  the idle state of the SPI clock signal, SCK. The interface shifts data at
106 //  the beginning of each bit cell. The four operating SPI modes are tabulated
107 //  in the following table:
108 //
109 //  Mode    CPOL    CPHA    :   SCK Idle Level    Sample Edge
110 //    0       0       0     :         0             Rising
111 //    1       0       1     :         1             Falling
112 //    2       1       0     :         0             Falling
113 //    3       1       1     :         1             Rising
114 //
115 //  Examining the table, the sampling edge is set to rising when CPOL and CPHA
116 //  are the same logic level, and it is set to falling when CPOL and CPHA are
117 //  complementary logic levels. In other words, the rising edge of SCK occurs in
118 //  the middle of the bit cell when (CPOL XNOR CPHA) == 1, and the falling edge
119 //  occurs in the middle of the bit cell when (CPOL XOR CPHA) == 1.
120 //
121 //  Two internal signals derived from Mode[1:0] determine the idle state of
122 //  SCK, SCK_Lvl, and the polarity of SCK used for data sampling, SCK_Inv. A
123 //  change-of-state (COS) detector is used to dynamically detect changes in the
124 //  value of SCK_Lvl, and to load the SCK register with the appropriate value
125 //  when the SPI interface idle, i.e. SS == 0. SCK_Lvl is taken from Mode[0],
126 //  and SCK_Inv is the XOR of Mode[1] and Mode[0], as indicated in the table
127 //  above.
128 //
129 //  The implementation shifts at one half of the CE frequency. The shift direc-
130 //  tion is programmable, but MSB first is the default.
131 //
132 //  The module contains the SCK generator. A three bit rate select input deter-
133 //  mines the rate of the SPI clock signal. A 50% duty cycle clock is produced,
134 //  and two separate clock enables are generated internally for loading and
135 //  shifting (propagating) the transmit data, and for shifting and writing the
136 //  receive data. The basic frequency is set by the equation:
137 //
138 //      F(SCK) = Clk / (2**(Rate + 1))
139 //
140 //  If rate is set to 0, then the frequency of SCK is one half that of the
141 //  module's input clock frequency. With Rate set to 7, the SCK frequecy is
142 //  Clk/256.
143 //
144 //  The output shift enable always asserts at the trailing edge of the bit cell,
145 //  and the input shift enable always asserts in the middle of the bit cell.
146 //  These shift enables are independent of the sampling and propagating edges
147 //  of SCK. The leading edge of the initial output is generated by the output
148 //  shift register load signal, which is itself generated by a rising edge
149 //  detector monitoring the DAV signal while slave select is not asserted. Once
150 //  SS is asserted, the output shift register is synchronously loaded on the TC
151 //  of the bit counter coincident with output shift register enable signal,
152 //  CE_OSR. This event also extends SS and reloads the receive enable signal,
153 //  RdEn. (RdEn was discussed above, and is determined by the 9th of the trans-
154 //  mit data.) If there is no more data to transmit, SS and RdEn are both de-
155 //  asserted.
156 //
157 //  All of the module control signals are resampled while SS is not asserted.
158 //  The control signals are held for the duration of a transfer cycle, which is
```

```
159 //  determined by the number of bytes loaded into the external transmit FIFO.
160 //  To limit the logic complexity, the module does not make prevent the control
161 //  signals from being changed as the initial transmit data is written. It is
162 //  necessary for the client of the module to ensure that the control signals
163 //  signals are stable at least one clock cycle before transmit data is availa-
164 //  ble.
165 //
166 //  With that limitation in mind, the control signals can be changed at any time
167 //  during a transfer cycle. They will be processed by the module with a one cy-
168 //  cle delay when SS is not asserted. In this manner, the client logic can dy-
169 //  namically change the shift direction, SCK operating mode, and SCK operating
170 //  frequency. This allows the module to be used in situations where the SPI
171 //  slave devices change modes, rates, and shift directions. For example, a sin-
172 //  gle SPI interface can be used to support both SPI memory devices (mode 0/3)
173 //  and SPI ADCs (mode 2) devices.
174 //
175 // Dependencies: none
176 //
177 // Revision History:
178 //
179 //  0.01    08C17   MAM     File Created
180 //
181 //  0.02    08C18   MAM     Modified to incorporate SCK_Lvl and a COS detector
182 //                          on SCK_Lvl to allow the Idle State SCK state to be
183 //                          dynamically changed.
184 //
185 //  0.03    08E09   MAM     Changed comment to reflect that this module is an
186 //                          SPI Master Interface.
187 //
188 //  1.00    12I07   MAM     Modified to bring into compliance with Verilog 2001.
189 //                          Modified the interface to use standard control sig-
190 //                          nals {CPOL, CPHA} and to map the SCK_Lvl and SCK_Inv
191 //                          signals to the standard SPI modes. To do this, the
192 //                          standard mode control signals, {CPOL, CPHA}, are run
193 //                          through a mapping function at the beginning of the
194 //                          module.
195 //
196 //  1.10    12I09   MAM     Restored use of separate transmit and receive shift
197 //                          registers. Single, combined shift register can't be
198 //                          used because input data is shifted on the opposite
199 //                          edge from that used to shift the output data.
200 //
201 //  1.20    12I10   MAM     Converted FRE and FWE output signals to FFs. FRE now
202 //                          pulsed one cycle after the OSR is loaded. FWE now
203 //                          pulsed one cycle after the last bit is loaded into
204 //                          the ISR. Converted CE_Cntr from up counter to a down
205 //                          counter. CE remains a combinatorial signal, but the
206 //                          CE counter is loaded from a ROM when CE asserts on
207 //                          basis of the rate captured before SS asserts. CE now
208 //                          asserts on a fixed value, 0, and the counter is re-
209 //                          loaded from a ROM. This contrasts with up-counter
210 //                          implementation which reloaded a fixed value, 0, and
211 //                          terminated on a variable value. The variable decode
212 //                          of the counter for CE apparently caused an issue in
213 //                          simulation that the new down counter resolves.
214 //
215 //  1.30    13G06   MAM     Removed code previously commented out. Corrected the
216 //                          SCK equation. A Rst_SCK was being generated at the
217 //                          end of every 8 bits. This introduced a discontinuity
218 //                          in SCK which does not allow frames greater than 8
219 //                          bits in length to be transmitted. With the removal
220 //                          the incorrect conditional logic in Rst_SCK, the SSP
221 //                          Slave and UART modules operate in either SPI Mode 0
222 //                          or Mode 3.
223 //
224 // Additional Comments:
225 //
226 //  The module control signals, LSB, Mode, and Rate, must be set at least one
227 //  clock cycle before DAV is asserted. Changing these control signals at the
228 //  same time that DAV is asserted will result in incorrect operation.
229 //
230 //////////////////////////////////////////////////////////////////////////////
231
232 module SPIxIF (
233     input   Rst,                // System Reset (synchronous)
234     input   Clk,                // System Clk
235 //
236     input   LSB,                // SPI LSB First Shift Direction
237     input   [1:0] Mode,         // SPI Operating Mode
```

```
238     input   [2:0] Rate,         // SPI Shift Rate Select: SCK = Clk/2**(Rate+1)
239 //
240     input   DAV,                // SPI Transmit Data Available
241     output  reg FRE,            // SPI Transmit FIFO Read Enable
242     input   [8:0] TD,           // SPI Transmit Data
243 //
244     output  reg FWE,            // SPI Receive FIFO Write Enable
245     output  [7:0] RD,           // SPI Receive Data
246 //
247     output  reg SS,             // SPI Slave Select
248     output  reg SCK,            // SPI Shift Clock
249     output  MOSI,               // SPI Master Out, Slave In: Serial Data Output
250     input   MISO                // SPI Master In, Slave Out: Serial Data In
251 );
252
253 ////////////////////////////////////////////////////////////////////////////
254 //
255 //  Module Parameters
256 //
257
258 ////////////////////////////////////////////////////////////////////////////
259 //
260 //  Module Declarations
261 //
262
263 reg     Dir;                                // Shift Register Shift Direction
264
265 reg     SCK_Lvl, SCK_Inv, COS_SCK_Lvl;      // SCK level and edge control
266
267 reg     [2:0] rRate;                        // SPI SCK Rate Select Register
268 reg     [6:0] CE_Cntr;                      // SPI CE Counter (2x SCK)
269 wire    CE;                                 // SPI Clock Enable (TC CE_Cntr)
270
271 wire    CE_SCK, Rst_SCK;                    // SCK generator control signals
272
273 reg     Ld;                                 // SPI Transfer Cycle Start Pulse
274
275 wire    CE_OSR, CE_ISR;                     // SPI Shift Register Clock Enables
276 reg     [7:0] OSR, ISR;                     // SPI Output/Input Shift Registers
277 reg     RdEn;                               // SPI Read Enable (9th bit in TD)
278
279 reg     [2:0] BitCnt;                       // SPI Transfer Cycle Length Cntr
280 wire    TC_BitCnt;                          // SPI Bit Counter Terminal Count
281
282 ////////////////////////////////////////////////////////////////////////////
283 //
284 //  Implementation
285 //
286
287 //  Capture the shift direction and hold until end of transfer cycle
288
289 always @(posedge Clk)
290 begin
291     if(Rst)
292         Dir <= #1 0;            // Default to MSB first
293     else if(~SS)
294         Dir <= #1 LSB;          // Set shift direction for transfer cycle
295 end
296
297 //  Assign SCK idle level and invert control signals based on Mode
298
299 always @(posedge Clk)
300 begin
301     if(Rst) begin
302         SCK_Inv <= #1 0;
303         SCK_Lvl <= #1 0;
304     end else if(~SS) begin
305         SCK_Inv <= #1 ^Mode;    // Invert SCK if Mode == 1 or Mode == 2
306         SCK_Lvl <= #1 Mode[0];  // Set SCK idle level from LSB of Mode
307     end
308 end
309
310 //  Generate change of state pulse when SPI clock idle level changes
311 //      while Slave Select not asserted
312
313 always @(posedge Clk)
314 begin
315     if(Rst)
316         COS_SCK_Lvl <= #1 0;
```

```
317     else
318         COS_SCK_Lvl <= #1 ((~SS) ? (SCK_Lvl ^ Mode[0]) : 0);
319 end
320 
321 //  Capture SCK rate and hold until the transfer cycle is complete
322 
323 always @(posedge Clk)
324 begin
325     if(Rst)
326         rRate <= #1 ~0;             // Default to slowest rate
327     else if(~SS)
328         rRate <= #1 Rate;
329 end
330 
331 //
332 //  SPI Transfer Cycle Load Pulse Generator
333 //
334 
335 always @(posedge Clk)
336 begin
337     if(Rst)
338         Ld <= #1 0;
339     else if(~SS)
340         Ld <= #1 DAV & ~Ld;
341     else if(Ld)
342         Ld <= #1 0;
343 end
344 
345 //
346 //  Serial SPI Clock Generator
347 //
348 
349 always @(posedge Clk)
350 begin
351     if(Rst)
352         CE_Cntr <= #1 ~0;
353     else if(CE)
354         case(rRate)
355             3'b000  : CE_Cntr <= #1 0;
356             3'b001  : CE_Cntr <= #1 1;
357             3'b010  : CE_Cntr <= #1 3;
358             3'b011  : CE_Cntr <= #1 7;
359             3'b100  : CE_Cntr <= #1 15;
360             3'b101  : CE_Cntr <= #1 31;
361             3'b110  : CE_Cntr <= #1 63;
362             3'b111  : CE_Cntr <= #1 127;
363         endcase
364     else if(SS)
365         CE_Cntr <= #1 (CE_Cntr - 1);
366 end
367 
368 assign CE = (Ld | (~|CE_Cntr));
369 
370 assign CE_SCK  = CE & SS; // Clock starts with Slave Select Strobe
371 assign Rst_SCK = Rst | Ld | (COS_SCK_Lvl & ~SS) | (TC_BitCnt & CE_OSR & ~DAV);
372 
373 always @(posedge Clk)
374 begin
375     if(Rst_SCK)
376         #1 SCK <= (Ld ? SCK_Inv : SCK_Lvl);
377     else if(CE_SCK)
378         #1 SCK <= ~SCK;
379 end
380 
381 //
382 //  SPI Output Shift Register
383 //
384 
385 assign CE_OSR = CE_SCK & (SCK_Inv ^ SCK);
386 assign Ld_OSR = Ld | (TC_BitCnt & CE_OSR);
387 
388 always @(posedge Clk)
389 begin
390     if(Rst)
391         OSR <= #1 0;
392     else if(Ld_OSR)
393         OSR <= #1 TD;
394     else if(CE_OSR)
395         OSR <= #1 ((Dir) ? {SCK_Lvl, OSR[7:1]} : {OSR[6:0], SCK_Lvl});
```

```
396 end
397
398 assign MOSI = SS & ((Dir) ? OSR[0] : OSR[7]);
399
400 //
401 //  SPI Input Shift Register
402 //
403
404 assign CE_ISR = CE_SCK & (SCK_Inv ^ ~SCK);
405
406 always @(posedge Clk)
407 begin
408     if(Rst)
409         ISR <= #1 0;
410     else if(Ld)
411         ISR <= #1 0;
412     else if(CE_ISR)
413         ISR <= #1 ((Dir) ? {MISO, ISR[7:1]} : {ISR[6:0], MISO});
414 end
415
416 //
417 //  SPI SR Bit Counter
418 //
419
420 assign CE_BitCnt  = CE_OSR & SS;
421 assign Rst_BitCnt = Rst | Ld | (TC_BitCnt & CE_OSR);
422
423 always @(posedge Clk)
424 begin
425     if(Rst_BitCnt)
426         BitCnt <= #1 7;
427     else if(CE_BitCnt)
428         BitCnt <= #1 (BitCnt - 1);
429 end
430
431 assign TC_BitCnt = ~|BitCnt;
432
433 //
434 //  SPI Slave Select Generator
435 //
436
437 always @(posedge Clk)
438 begin
439     if(Rst)
440         SS <= #1 0;
441     else if(Ld_OSR)
442         SS <= #1 DAV;
443 end
444
445 //
446 //  SPI MISO Read Enable Register
447 //
448
449 always @(posedge Clk)
450 begin
451     if(Rst)
452         RdEn <= #1 0;
453     else if(Ld_OSR)
454         RdEn <= #1 ((DAV) ? TD[8] : 0);
455 end
456
457 //
458 //  SPI Transmit FIFO Read Pulse Generator
459 //
460
461 always @(posedge Clk)
462 begin
463     if(Rst)
464         FRE <= #1 0;
465     else
466         FRE <= #1 (Ld | (DAV & (TC_BitCnt & CE_OSR)));
467 end
468
469 //
470 //  SPI Receive FIFO Write Pulse Generator
471 //
472
473 always @(posedge Clk)
474 begin
```

```
475     if(Rst)
476         FWE <= #1 0;
477     else
478         FWE <= #1 (RdEn & (TC_BitCnt & CE_ISR));
479 end
480
481 assign RD = ISR;
482
483 endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
//  Parallel Interface Universal Asynchronous Receiver/Transmiter
//
//  Copyright 2013-2014 by Michael A. Morris, dba M. A. Morris & Associates
//
//  All rights reserved. The source code contained herein is publicly released
//  under the terms and conditions of the GNU General Public License as conveyed
//  in the license provided below.
//
//  This program is free software: you can redistribute it and/or modify it
//  under the terms of the GNU General Public License as published by the Free
//  Software Foundation, either version 3 of the License, or any later version.
//
//  This program is distributed in the hope that it will be useful, but WITHOUT
//  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
//  more details.
//
//  You should have received a copy of the GNU General Public License along with
//  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
//
//  Free Software Foundation, Inc.
//  51 Franklin Street, Fifth Floor
//  Boston, MA  02110-1301 USA
//
//  Further, no use of this source code is permitted in any form or means
//  without inclusion of this banner prominently in any derived works.
//
//  Michael A. Morris <morrisma_at_mchsi_dot_com>
//  164 Raleigh Way
//  Huntsville, AL 35811
//  USA
//
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:         M. A. Morris & Associates
// Engineer:        Michael A. Morris
//
// Create Date:     16:12:34 12/28/2013
// Design Name:     M65C02 Dual Core
// Module Name:     M65C02_UART.v
// Project Name:    C:\XProjects\ISE10.1i\M65C02Duo
// Target Devices:  Generic SRAM-based FPGA, XC3S50A-4VQG100I, XC3S200A-4VQG100I
// Tool versions:   ISE 10.1i SP3
//
// Description:
//
//  This module integrates the various elements of a simplified UART to create
//  an efficient, programmable UART. The M65C02_UART provides an M65C02-compati-
//  ble interface to a simplified 16C550-style UART module. The simplified UART
//  module implements those features most often used and deletes those features
//  which are seldom used.
//
//  The UART module supports a reduced number of line formats. For example, it
//  does not support the 5-bit or 6-bit frame formats supported by 16C550-compa-
//  tible UARTs. It also does not support 1.5 stop bits, or space or mark parity
//  settings. Instead, the UART module supports four operating modes that allows
//  easier support of RS-232/RS-485 industrial automation protocols.
//
//  These four operating modes allow both 2-wire and 4-wire RS-232 protocols to
//  be easily implemented. The UART incorporates HW flow control for the 4-wire
//  RS-232 protocols. In contrast, a 16C550-compatible UART requires RTS/CTS or
//  DTR/DSR flow control to be implement in the firmware/software driver.
//
//  In addition, the two RS-485 operating modes provide automatic support for
//  half-duplex protocols w/ or w/o collision detection. Automatic support for
//  half-duplex protocols is based on the feature of the UART to automatically
//  assert and deassert the RS-485 Drive Enable (DE) signal of standard 75176-
//  compatible RS-485 transceivers. In addition, these RS-485 modes support a
//  programmable line idle time such as that used to frame Modbus-RTU and Profi-
//  bus.
//
// Dependencies:
//
//      UART.v
```

```
 80 //          re1ce.v
 81 //          BRSFmnCE.v
 82 //          UART_BRG.v
 83 //          UART_TXSM.v
 84 //          UART_RXSM.v
 85 //          UART_RTO.v
 86 //          UART_INT.v
 87 //              redet.v
 88 //
 89 // Revision History:
 90 //
 91 //  0.01    13L28   MAM     File Created
 92 //
 93 // Additional Comments:
 94 //
 95 ////////////////////////////////////////////////////////////////////////////////
 96
 97 module UART #(
 98     // Default UART Settings
 99
100     parameter pDefault_LCR = 8'h80,     // RTSo, Tx Enable, 232 w/o HS, 8N1
101     parameter pDefault_IER = 8'h00,     // All interrupts disabled
102
103     //  Default UART Baud Rate Settings
104
105     parameter pFrequency = 29491200,    // 29.4912 MHz
106     parameter pBaudrate  = 115200,      // 115200 bps
107
108     // Default Receive Time Out Character Delay Count
109
110     parameter pRTOChrDlyCnt = 3,        // Default Number of Characters for RTO
111
112     // FIFO Configuration Parameters
113
114     parameter pTF_Depth = 0,            // Tx FIFO Depth: 2**(TF_Depth + 4)
115     parameter pRF_Depth = 0,            // Rx FIFO Depth: 2**(RF_Depth + 4)
116     parameter pTF_Init  = "Src/UART_TF_16.coe", // Tx FIFO Memory Initialization
117     parameter pRF_Init  = "Src/UART_RF_16.coe"  // Rx FIFO Memory Initialization
118 )(
119     input   Rst,                        // System Reset
120     input   Clk,                        // System Clock
121
122     //  External Interrupt Request
123
124     output  reg IRQ,                    // Interrupt Request
125
126     //  Parallel Interface
127
128     input   Sel,                        // UART Select
129     input   [1:0] Reg,                  // UART Register Select
130     input   RE,                         // UART Read Strobe
131     input   WE,                         // UART Write Strobe
132     input   [7:0] DI,                   // UART Data Input Bus
133     output  wor [7:0] DO,               // UART Data Output Bus
134
135     //  External UART Interface
136
137     output  TxD,                        // TxD
138     input   RxD,                        // RxD
139     output  reg xRTS,                   // RS-232 Mode RTS (Ready-To-Receive)
140     input   xCTS,                       // RS-232 Mode CTS (Okay-To-Send)
141
142     output  xDE,                        // RS-485 Mode Transceiver Drive Enable
143
144     output  Mode,                       // Mode: 0 - RS232; 1 - RS485
145
146     //  TxSM/RxSM Status
147
148     output  TxIdle,                     // UART TxSM Idle Status Output
149     output  RxIdle                      // UART RxSM Idle Status Output
150 );
151
152 ////////////////////////////////////////////////////////////////////////////////
153 //
154 //  Module Parameters
155 //
156
157 localparam pDiv_Default = ((pFrequency / (16 * pBaudrate)) - 1);
158
```

```
159 //  Register Addresses
160
161 localparam pTHR = 0;            // Tx Data Holding Register (FIFO) (BRL)
162 localparam pRHR = 0;            // Rx Data Holding Register (FIFO)
163 localparam pLSR = 1;            // UART Line Status Register       (BRH)
164 localparam pLCR = 2;            // UART Line Control Register
165 localparam pIER = 3;            // UART Interrupt Enable Register
166
167 //  FIFO Configuration Parameters
168
169 localparam pWidth = 8;          // Maximum Character width
170 localparam pxFThr = 8;          // TF/RF Half-Full Flag Theshold (%, 4 bits)
171
172 ////////////////////////////////////////////////////////////////////////////
173 //
174 //  Declarations
175 //
176
177 reg     [7:0] LSR;              // UART Line Status Register (LCR)
178 wire    Sel_LSR;                // LSR: Register Select
179 wire    RE_LSR;                 // LSR: Read Enable Line Status Register
180
181 wire    CTSi;                   // LSR: CTS Input
182 wire    RTSi;                   // LSR: RTS Input
183 wire    iTHE;                   // LSR: Transmit Half Empty Interrupt
184 wire    iTEF;                   // LSR: Transmit FIFO Empty Interrupt
185 wire    iTEM;                   // LSR: Transmit SR Empty Interrupt
186 wire    TxRdy;                  // LSR: Transmitter Ready
187 wire    iRFE;                   // LSR: Receive Framing Error Interrupt
188 wire    iRPE;                   // LSR: Receive Parity Error Interrupt
189 wire    iRHF;                   // LSR: Receive Half Full Interrupt
190 wire    iRTO;                   // LSR: Receive Timeout Interrupt
191 wire    RxRdy;                  // LSR: Receiver Ready
192
193 reg     Clr_Int;                // Clear Interrupt Flags - read of LSR
194
195 reg     [7:0] LCR;              // UART Line Control Register (LCR)
196 wire    Sel_LCR;                // LCR: Register Select
197 wire    WE_LCR;                 // LCR: Write Enable Line Control Register
198
199 wire    BRRE, RTSo;             // LCR: BRR Access Enable, RTS Outputs
200 wire    [1:0] MD;               // LCR: Serial Interface Operating Mode
201 wire    [3:0] FMT;              // LCR: Serial Frame Format
202
203 reg     Len, NumStop, ParEn;    // Char Length, # Stop Bits, Parity Enable
204 reg     [1:0] Par;              // Parity Selector
205
206 wire    Hold;                   // Tx Hold - RS485 Send held off if asserted
207
208 reg     [7:0] IER;              // UART Interrupt Enable Register (IER)
209 wire    Sel_IER;                // IER: Register Select
210 wire    WE_IER;                 // IER: Write Enable Interrupt Control Register
211
212 wire    IE;                     // IER: Master Interrupt Enable
213 wire    IE_THE;                 // IER: Enable Tx FIFO Half Empty Interrupt
214 wire    IE_TEF;                 // IER: Enable Tx FIFO Empty Interrupt
215 wire    IE_TEM;                 // IER: Enable Tx SR Empty Interrupt
216 wire    IE_RFE;                 // IER: Enable Receive Framing Error Interrupt
217 wire    IE_RPE;                 // IER: Enable Receive Parity Error Interrupt
218 wire    IE_RHF;                 // IER: Enable Rx FIFO Half Full Interrupt
219 wire    IE_RTO;                 // IER: Enable Receive Time Out Interrupt
220
221 reg     [15:0] Div;             // UART Baud Rate Divider Register (DIV)
222 wire    WE_BRR;                 // DIV: Write Enable Divider Register
223
224 wire    [7:0] THR;              // Transmit Holding Register (Tx FIFO Output)
225 wire    WE_THR;                 // Write Enable - Transmit Holding Register(THR)
226 wire    TF_FF, TF_EF, TF_HF;    // Transmit FIFO Flags - Full, Empty, Half
227
228 wire    [9:0] RD, RHR;          // Receive Data (In), Receive Holding Register
229 wire    WE_RHR;                 // Write Enable - Receive Holding Register (RHR)
230 wire    RE_RHR;                 // Read Enable - RHR
231 wire    RF_EF, RF_HF;           // Receive FIFO Flags - Empty, Half
232
233 reg     RxDi;                   // Mode Dependent RxD input
234
235 reg     [ 3:0] CCntVal;         // RTO Character Length: {10 | 11 | 12} - 1
236 wire    [ 3:0] RTOVal;          // RTO Character Delay Value: (N - 1)
237 wire    RTO;                    // Receive Timeout Flag
```

```
238
239 ////////////////////////////////////////////////////////////////////////////
240 //
241 //  Implementation
242 //
243
244 //  Break out Register Select Address
245
246 assign Sel_THR = Sel & (Reg == pTHR);   // Offset 0
247 assign Sel_RHR = Sel & (Reg == pRHR);   // Offset 0
248 assign Sel_LSR = Sel & (Reg == pLSR);   // Offset 1
249 assign Sel_LCR = Sel & (Reg == pLCR);   // Offset 2
250 assign Sel_IER = Sel & (Reg == pIER);   // Offset 3
251
252 //  Drive UART Data Output Bus
253
254 assign DO = ((Sel_RHR) ? ((BRRE) ? Div[ 7:0] : RHR) : 0);   // Offset 0
255 assign DO = ((Sel_LSR) ? ((BRRE) ? Div[15:8] : LSR) : 0);   // Offset 1
256 assign DO = ((Sel_LCR) ? LCR                       : 0);   // Offset 2
257 assign DO = ((Sel_IER) ? IER                       : 0);   // Offset 3
258
259 ////////////////////////////////////////////////////////////////////////////
260 //
261 //  UART Line Stature Register (LSR) - Read-only
262 //
263
264 assign RE_LSR = RE & Sel_LSR;
265
266 assign RxRdy = ((IE_RTO) ? iRTO : ~RF_EF);
267 assign TxRdy = ~TF_FF;
268
269 always @(posedge Clk)
270 begin
271     if(Rst)
272         LSR <= #1 0;
273     else
274         LSR <= #1 {CTSi,                    // CTS Level
275                    RTSi,                    // RTS level
276                    (iTHE | iTEF | iTEM),    // Tx Interrupt
277                    TxRdy,                   // Tx Ready - Tx FIFO not Full
278                    iRFE,                    // Rx Framing Error
279                    iRPE,                    // Rx Parity Error
280                    iRHF,                    // Rx FIFO Half Full or more
281                    RxRdy};                  // Rx Ready - Rx FIFO not Empty
282 end
283
284 //  Generate Clear Interrupt LSR Read
285
286 always @(posedge Clk)
287 begin
288     if(Rst)
289         Clr_Int <= #1 0;
290     else
291         Clr_Int <= #1 RE_LSR;
292 end
293
294 ////////////////////////////////////////////////////////////////////////////
295 //
296 //  Write UART Line Control Register (LCR) - Read/Write
297 //
298
299 assign WE_LCR = WE & Sel_LCR;
300
301 always @(posedge Clk)
302 begin
303     if(Rst)
304         LCR <= #1 0;
305     else if(WE_LCR)
306         LCR <= #1 DI;
307 end
308
309 //  Assign LCR Fields
310
311 assign BRRE = LCR[7];               // Select Baud Rate Register
312 assign RTSo = LCR[6];               // RTS Output in 2-Wire RS-232 Mode
313 assign MD   = LCR[5:4];             // Operating Mode Select
314 assign FMT  = LCR[3:0];             // Frame Format Select
315
316 //  Format Decode
```

```
317
318 always @(FMT)
319 case(FMT)
320     4'b0000 : {Len, NumStop, ParEn, Par} <= {1'b0, 1'b0, 1'b0, 2'b00};   // 8N1
321     4'b0001 : {Len, NumStop, ParEn, Par} <= {1'b1, 1'b0, 1'b1, 2'b00};   // 7O1
322     4'b0010 : {Len, NumStop, ParEn, Par} <= {1'b1, 1'b0, 1'b1, 2'b01};   // 7E1
323     4'b0011 : {Len, NumStop, ParEn, Par} <= {1'b0, 1'b0, 1'b1, 2'b00};   // 8O1
324     4'b0100 : {Len, NumStop, ParEn, Par} <= {1'b0, 1'b0, 1'b1, 2'b01};   // 8E1
325     4'b0101 : {Len, NumStop, ParEn, Par} <= {1'b0, 1'b0, 1'b1, 2'b10};   // 8S1
326     4'b0110 : {Len, NumStop, ParEn, Par} <= {1'b0, 1'b0, 1'b1, 2'b11};   // 8M1
327     4'b0111 : {Len, NumStop, ParEn, Par} <= {1'b1, 1'b1, 1'b1, 2'b00};   // 7O2
328     4'b1000 : {Len, NumStop, ParEn, Par} <= {1'b1, 1'b1, 1'b1, 2'b01};   // 7E2
329     4'b1001 : {Len, NumStop, ParEn, Par} <= {1'b0, 1'b1, 1'b0, 2'b00};   // 8N2
330     4'b1010 : {Len, NumStop, ParEn, Par} <= {1'b0, 1'b1, 1'b1, 2'b00};   // 8O2
331     4'b1011 : {Len, NumStop, ParEn, Par} <= {1'b0, 1'b1, 1'b1, 2'b01};   // 8E2
332     4'b1100 : {Len, NumStop, ParEn, Par} <= {1'b0, 1'b1, 1'b1, 2'b10};   // 8S2
333     4'b1101 : {Len, NumStop, ParEn, Par} <= {1'b0, 1'b1, 1'b1, 2'b11};   // 8M2
334     4'b1110 : {Len, NumStop, ParEn, Par} <= 0;  // 8N1; Unused
335     4'b1111 : {Len, NumStop, ParEn, Par} <= 0;  // 8N1; Unused
336 endcase
337
338 //  Receive Timeout Character Frame Length
339
340 always @(FMT)
341 case(FMT)
342     4'b0000 : CCntVal <= 4'h9;   // 8N1,  9 <= 10 - 1
343     4'b0001 : CCntVal <= 4'h9;   // 7O1,  9 <= 10 - 1
344     4'b0010 : CCntVal <= 4'h9;   // 7E1,  9 <= 10 - 1
345     4'b0011 : CCntVal <= 4'hA;   // 8O1, 10 <= 11 - 1
346     4'b0100 : CCntVal <= 4'hA;   // 8E1, 10 <= 11 - 1
347     4'b0101 : CCntVal <= 4'hA;   // 8S1, 10 <= 11 - 1
348     4'b0110 : CCntVal <= 4'hA;   // 8M1, 10 <= 11 - 1
349     4'b0111 : CCntVal <= 4'h9;   // 7O2,  9 <= 10 - 1
350     4'b1000 : CCntVal <= 4'h9;   // 7E2,  9 <= 10 - 1
351     4'b1001 : CCntVal <= 4'hA;   // 8N2, 10 <= 11 - 1
352     4'b1010 : CCntVal <= 4'hB;   // 8O2, 11 <= 12 - 1
353     4'b1011 : CCntVal <= 4'hB;   // 8E2, 11 <= 12 - 1
354     4'b1100 : CCntVal <= 4'hB;   // 8S2, 11 <= 12 - 1
355     4'b1101 : CCntVal <= 4'hB;   // 8M2, 11 <= 12 - 1
356     4'b1110 : CCntVal <= 4'h9;   // 8N1,  9 <= 10 - 1
357     4'b1111 : CCntVal <= 4'h9;   // 8N1,  9 <= 10 - 1
358 endcase
359
360 assign RTOVal = (pRTOChrDlyCnt - 1);    // Set RTO Character Delay Count
361
362 ////////////////////////////////////////////////////////////////////////////
363 //
364 //  Write UART Interrupt Enable Register
365 //
366
367 assign WE_IER = WE & Sel_IER;
368
369 always @(posedge Clk)
370 begin
371     if(Rst)
372         IER <= #1 0;
373     else if(WE_IER)
374         IER <= #1 DI;
375 end
376
377 //  Assign LCR Fields
378
379 assign IE     = IER[7];                // Master Interrupt Enable
380 assign IE_THE = IER[6];                // Interrupt Enable Tx FIFO Half Empty
381 assign IE_TEF = IER[5];                // Interrupt Enable Tx FIFO Empty
382 assign IE_TEM = IER[4];                // Interrupt Enable Tx SR Empty
383 assign IE_RFE = IER[3];                // Interrupt Enable Rx Framing Error
384 assign IE_RPE = IER[2];                // Interrupt Enable Rx Parity Error
385 assign IE_RHF = IER[1];                // Interrupt Enable Rx FIFO Half Full
386 assign IE_RTO = IER[0];                // Interrupt Enable Rx Time Out
387
388 //  Assert IRQ when IE is set
389
390 assign Rst_IRQ = Rst | Clr_Int | ~IE;
391
392 always @(posedge Clk)
393 begin
394     if(Rst_IRQ)
395         IRQ <= #1 0;
```

```
396     else if(~IRQ)
397         IRQ <= #1 IE & (  iTHE & IE_THE
398                         | iTEF & IE_TEF
399                         | iTEM & IE_TEM
400                         | iRFE & IE_RFE
401                         | iRPE & IE_RPE
402                         | iRHF & IE_RHF
403                         | iRTO & IE_RTO );
404 end
405
406 ////////////////////////////////////////////////////////////////////////////
407 //
408 //  BRR - Baud Rate Register
409 //
410
411 assign WE_BRR = WE & (Sel_THR | Sel_LSR) & BRRE;
412
413 always @(posedge Clk)
414 begin
415     if(Rst)
416         Div <= #1 pDiv_Default;        // Default: 9600 bps
417     else if(WE_BRR) begin
418         Div[ 7:0] <= #1 ((Sel_THR) ? DI : Div[ 7:0]);
419         Div[15:8] <= #1 ((Sel_LSR) ? DI : Div[15:8]);
420     end
421 end
422
423 ////////////////////////////////////////////////////////////////////////////
424 //
425 //  Xmt/Rcv Holding Register Instantiations - Dual-Port Synchronous FIFOs
426 //
427 //  THR FIFO - 2**(pTFLen + 4) x pWidth FIFO
428
429 assign WE_THR = WE & Sel_THR & (~BRRE);
430
431 DPSFnmCE    #(
432                 .addr((pTF_Depth + 4)),
433                 .width(pWidth),
434                 .init(pTF_Init)
435             ) TF1 (
436                 .Rst(Rst),
437                 .Clk(Clk),
438
439                 .WE(WE_THR),
440                 .DI(DI),
441
442                 .RE(RE_THR),
443                 .DO(THR),
444
445                 .FF(TF_FF),
446                 .HF(TF_HF),
447                 .EF(TF_EF),
448                 .Cnt()
449             );
450
451 //  RHR FIFO - 2**(pRFLen + 4) x (pWidth + 1) FIFO
452
453 assign RE_RHR = Sel_RHR & RE;
454
455 DPSFnmCE    #(
456                 .addr((pRF_Depth + 4)),
457                 .width((pWidth + 2)),
458                 .init(pRF_Init)
459             ) RF1 (
460                 .Rst(Rst),
461                 .Clk(Clk),
462
463                 .WE(WE_RHR),
464                 .DI(RD),
465
466                 .RE(RE_RHR),
467                 .DO(RHR),
468
469                 .FF(),
470                 .HF(RF_HF),
471                 .EF(RF_EF),
472                 .Cnt()
473             );
474
```

```
475 ////////////////////////////////////////////////////////////////////////////////
476 //
477 //  Configure external/internal serial port signals according to MD[1:0]
478 //      MD[1:0] = 0,1 - RS-232; 2,3 - RS-485
479
480 assign RS485 =  MD[1];
481 assign RS232 = ~RS485;
482 assign Mode  =  RS485;
483
484 //  Assert DE in the RS-485 modes whenever the TxSM is not idle, and deassert
485 //      whenever the RS-485 modes are not selected
486
487 assign xDE = (RS485 ? ~TxIdle : 0);
488
489 //  Connect the UART's RxD serial input to the appropriate external RxD input
490 //      Hold RxD to logic 1 when in the RS-485 w/o Loopback mode and the TxSM
491 //      is transmitting data. In this manner, the external xOE signal to the
492 //      RS-485 transceiver can always be asserted.
493
494 always @(posedge Clk or posedge Rst)
495 begin
496     if(Rst)
497         RxDi <= #1 1;
498     else
499         case(MD)
500             2'b00 : RxDi <= #1 RxD;
501             2'b01 : RxDi <= #1 RxD;
502             2'b10 : RxDi <= #1 (TxIdle ? RxD : 1);
503             2'b11 : RxDi <= #1 RxD;
504         endcase
505 end
506
507 // RS-232 auto-Handshaking is implemented as Ready-To-Receive (RTR) based on
508 //      the Rcv FIFO flag settings. xRTS, which should connect to the receiving
509 //      side's xCTS, is asserted whenever the local receive FIFO is less than
510 //      half full. If a similar UART with hardware handshaking is connected,
511 //      then that transmitter should stop sending until the local FIFO is read
512 //      so that it is below the HF mark. Since local reads of the receive FIFO
513 //      are expected to be much faster than the RS-232 baud rate, it is not
514 //      expected that hysteresis will be required to prevent rapid assertion
515 //      and deassertion of RTS.
516 //
517 //      This handshaking mechanism was selected for the automatic handshaking
518 //      mode because it prevents (or attempts to prevent) receive FIFO over-
519 //      flow in the receiver. Furthermore, it reduces the software workload in
520 //      the transmitter's send routines.
521 //
522 //      For all other modes, the CTSi control signal to the UART_TXSM is held
523 //      at logic one. This effectively disables the TxSM's handshaking logic,
524 //      and allows the transmitter to send data as soon as data is written to
525 //      Xmt FIFO.
526
527 always @(*)
528 begin
529     case(MD)
530         2'b00 : xRTS <= RTSo;
531         2'b01 : xRTS <= ~RF_HF;
532         2'b10 : xRTS <= 0;
533         2'b11 : xRTS <= 0;
534     endcase
535 end
536
537 assign RTSi = ((RS232  ) ? xRTS : xDE);
538 assign CTSi = ((MD == 1) ? xCTS : 1);
539
540 ////////////////////////////////////////////////////////////////////////////////
541 //
542 //  UART Baud Rate Generator Instantiation
543 //
544
545 assign Ld_BRG = BRRE;
546
547 UART_BRG    #(
548                 .pDiv_Default(pDiv_Default)
549             )BRG (
550                 .Rst(Rst),
551                 .Clk(Clk),
552
553                 .Ld(Ld_BRG),
```

```
554                 .Div(Div),
555
556                 .CE_16x(CE_16x)
557             );
558
559 ////////////////////////////////////////////////////////////////////////////
560 //
561 //  UART Transmitter State Machine & Shift Register Instantiation
562 //
563
564 assign Hold = MD[1] & ~RTSo;    // Hold Tx when RS485 mode and RTSo not asserted
565
566 UART_TXSM   XMT (
567                 .Rst(Rst),
568                 .Clk(Clk),
569
570                 .CE_16x(CE_16x),
571
572                 .Len(Len),
573                 .NumStop(NumStop),
574                 .ParEn(ParEn),
575                 .Par(Par),
576
577                 .TF_RE(RE_THR),
578                 .THR(THR),
579                 .TF_EF(TF_EF | Hold),
580
581                 .TxD(TxD),
582                 .CTSi(CTSi),
583
584                 .TxIdle(TxIdle),
585                 .TxStart(),
586                 .TxShift(),
587                 .TxStop()
588             );
589
590 ////////////////////////////////////////////////////////////////////////////
591 //
592 //  UART Receiver State Machine & Shift Register Instantiation
593 //
594
595 UART_RXSM   RCV (
596                 .Rst(Rst),
597                 .Clk(Clk),
598
599                 .CE_16x(CE_16x),
600
601                 .Len(Len),
602                 .NumStop(NumStop),
603                 .ParEn(ParEn),
604                 .Par(Par),
605
606                 .RxD(RxDi),
607
608                 .RD(RD),
609                 .WE_RHR(WE_RHR),
610
611                 .RxWait(),
612                 .RxIdle(RxIdle),
613                 .RxStart(),
614                 .RxShift(),
615                 .RxParity(),
616                 .RxStop(),
617                 .RxError()
618             );
619
620 ////////////////////////////////////////////////////////////////////////////
621 //
622 //  UART Receive Timeout Module Instantiation
623 //
624
625 UART_RTO    TMR (
626                 .Rst(Rst),
627                 .Clk(Clk),
628
629                 .CE_16x(CE_16x),
630
631                 .WE_RHR(WE_RHR),
632                 .RE_RHR(RE_RHR),
```

```
633
634                 .CCntVal(CCntVal),
635                 .RTOVal(RTOVal),
636
637                 .RcvTimeout(RTO)
638             );
639
640 ////////////////////////////////////////////////////////////////////////////
641 //
642 //  UART Interrupt Generator Instantiation
643 //
644
645 UART_INT    INT (
646                 .Rst(Rst),
647                 .Clk(Clk),
648
649                 .TF_HF(TF_HF),
650                 .TF_EF(TF_EF),
651                 .TxIdle(TxIdle),
652
653                 .RF_FE(RHR[9]),
654                 .RF_PE(RHR[8]),
655                 .RF_HF(RF_HF),
656                 .RF_EF(RF_EF),
657
658                 .RTO(RTO),
659
660                 .Clr_Int(Clr_Int),
661
662                 .iTHE(iTHE),
663                 .iTEF(iTEF),
664                 .iTEM(iTEM),
665
666                 .iRPE(iRPE),
667                 .iRFE(iRFE),
668                 .iRHF(iRHF),
669                 .iRTO(iRTO)
670             );
671
672 endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
//  Baud Rate Generator for UART module.
//
//  Copyright 2008-2014 by Michael A. Morris, dba M. A. Morris & Associates
//
//  All rights reserved. The source code contained herein is publicly released
//  under the terms and conditions of the GNU General Public License as conveyed
//  in the license provided below.
//
//  This program is free software: you can redistribute it and/or modify it
//  under the terms of the GNU General Public License as published by the Free
//  Software Foundation, either version 3 of the License, or any later version.
//
//  This program is distributed in the hope that it will be useful, but WITHOUT
//  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
//  more details.
//
//  You should have received a copy of the GNU General Public License along with
//  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
//
//  Free Software Foundation, Inc.
//  51 Franklin Street, Fifth Floor
//  Boston, MA  02110-1301 USA
//
//  Further, no use of this source code is permitted in any form or means
//  without inclusion of this banner prominently in any derived works.
//
//  Michael A. Morris <morrisma_at_mchsi_dot_com>
//  164 Raleigh Way
//  Huntsville, AL 35811
//  USA
//
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:         M. A. Morris & Associates
// Engineer:        Michael A. Morris
//
// Create Date:     13:28:23 05/10/2008
// Design Name:     Parallel Interface UART
// Module Name:     ../VerilogCoponentsLib/UART/UART_BRG.v
// Project Name:    Verilog Components Library
// Target Devices:  XC3S50A-4VQG100I, XC3S20A-4VQG100I, XC3S700AN-4FFG484I
// Tool versions:   ISE 10.1i SP3
//
// Description:
//
//  This module implements a simple 16-bit Baud Rate Generator for the simple
//  parallel interface UART.
//
// Dependencies:
//
// Revision History:
//
//  0.01    08E10   MAM     File Created
//
//  1.00    08E10   MAM     Initial Release
//
//  1.10    08E13   MAM     Changed interface so Prescaler and Divider values
//                          passed directly in by removing Baud Rate ROM.
//
//  1.11    08E14   MAM     Reduced width of divider from 10 to 8 bits.
//
//  1.20    08E15   MAM     Changed the structure of the PSCntr and Divider
//                          to use a multiplxer on the input to load or count
//                          which results in a more efficient implementation.
//                          Added a registered TC on the PSCntr which functions
//                          to break the combinatorial logic chains and speed
//                          counter implementations.
//
//  1.30    08G26   MAM     Corrected initial condition of the PSCntr, which
//                          caused the prescaler to always divide by two.
//                          Removed FF in PSCntr TC path to remove the divide
//                          by two issue. CE_16x output remains as registered.
//
```

```
 80 //  2.00    11B06   MAM     Converted to Verilog 2001.
 81 //
 82 //  2.10    13L28   MAM     Module now part of the Parallel UART project. Modi-
 83 //                          fied to support a single 16-bit divider than the
 84 //                          4-bit pre-scaler and 8-bit divider used in the SSP
 85 //                          UART version of the module. Added parameter to set
 86 //                          default divider value so baud rate set on reset. A
 87 //                          load input, Ld, added to load baud rate divider.
 88 //
 89 // Additional Comments:
 90 //
 91 ////////////////////////////////////////////////////////////////////////////////
 92
 93 module UART_BRG #(
 94     parameter pDiv_Default = 16'h01DF   // Divider Default: 9600@73.7280MHz
 95 )(
 96     input   Rst,
 97     input   Clk,
 98
 99     input   Ld,             // Baud Rate Generator Load
100     input   [15:0] Div,     // Baud Rate Generator Divider Load Value
101
102     output  reg CE_16x      // Clock Enable Output - 16x Baud Rate Output
103 );
104
105 ////////////////////////////////////////////////////////////////////////////////
106 //
107 //  Local Signal Declarations
108 //
109
110 reg     [15:0] Divider;                 // BRG Divider
111 wire    TC_Divider;                     // BRG Divider TC
112
113 ////////////////////////////////////////////////////////////////////////////////
114 //
115 //  Implementation
116 //
117
118 // BRG Divider
119
120 always @(posedge Clk)
121 begin
122     if(Rst)
123         Divider <= #1 pDiv_Default;
124     else if(Ld | TC_Divider)
125         Divider <= #1 Div;
126     else
127         Divider <= #1 (Divider - 1);
128 end
129
130 assign TC_Divider = ~|Divider;
131
132 // Output 16x Bit Clock/CE
133
134 always @(posedge Clk)
135 begin
136     if(Rst)
137         CE_16x <= #1 1'b0;
138     else
139         CE_16x <= #1 TC_Divider;
140 end
141
142 endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
//  Transmitter SM and Shift Register module for UART.
//
//  Copyright 2008-2014 by Michael A. Morris, dba M. A. Morris & Associates
//
//  All rights reserved. The source code contained herein is publicly released
//  under the terms and conditions of the GNU General Public License as conveyed
//  in the license provided below.
//
//  This program is free software: you can redistribute it and/or modify it
//  under the terms of the GNU General Public License as published by the Free
//  Software Foundation, either version 3 of the License, or any later version.
//
//  This program is distributed in the hope that it will be useful, but WITHOUT
//  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
//  more details.
//
//  You should have received a copy of the GNU General Public License along with
//  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
//
//  Free Software Foundation, Inc.
//  51 Franklin Street, Fifth Floor
//  Boston, MA  02110-1301 USA
//
//  Further, no use of this source code is permitted in any form or means
//  without inclusion of this banner prominently in any derived works.
//
//  Michael A. Morris <morrisma_at_mchsi_dot_com>
//  164 Raleigh Way
//  Huntsville, AL 35811
//  USA
//
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:         M. A. Morris & Associates
// Engineer:        Michael A. Morris
//
// Create Date:     21:22:38 05/10/2008
// Design Name:     Parallel Interface UART
// Module Name:     ../VerilogCoponentsLib/UART/UART_TXSM.v
// Project Name:    Verilog Components Library
// Target Devices:  XC3S50A-4VQG100I, XC3S20A-4VQG100I, XC3S700AN-4FFG484I
// Tool versions:   ISE 10.1i SP3
//
// Description:
//
//  This module implements the Transmit State Machine for the Parallel Interface
//  UART.
//
//
// Dependencies:    None
//
// Revision History:
//
//  0.01    08E10   MAM     File Created
//
//  1.00    08E24   MAM     Modified and tested TxSM to allow for several addi-
//                          tional conditions regarding Mode 1 - RS232 w/ Hand-
//                          shaking. In particular, TxSM goes to pStart from
//                          pShift if CTS is not asserted when the current word
//                          has been shifted.
//
//  1.10    08E28   MAM     Modified to remove I/O signals best processed above
//                          this module: RTSi, RTSo, and DE. Removed from port
//                          list and moved the decoded state ouput bits to the
//                          end of the port list. Modified the TSRI and the TSR
//                          length. TSRI now only processes the two MSBs of the
//                          TSR load value. The length of TSR modified from 12
//                          to 10 bits because the logic 1 fill value and the
//                          bit counter allow the stop bits to be implicit. The
//                          result is a faster signal path.
//
//  1.20    08F08   MAM     Modified TxSM to use a ROM to decode the [3:0]FMT
//                          input like the RxSM. Changed the SR implementation
```

```
 80 //                          of the shift register into a registered multiplexer
 81 //                          implementation. Removed the bit counter and added
 82 //                          states to the SM to multiplex the data. Result is a
 83 //                          SM and SR that synthesizes to a speed that matches
 84 //                          the reported speed of the RxSM: 110+ MHz. This may
 85 //                          indicated that this UART may be useful as a high-
 86 //                          speed UART with an 8x over-sample instead of a 4x
 87 //                          oversample using a 48MHz clock input and a 2x DLL.
 88 //
 89 //  1.21    08F12   MAM     Pulled Format Decoder ROM and moved to upper level
 90 //                          module. Added the outputs of the ROM to the port
 91 //                          list of the module. Module now supports more format
 92 //                          directly with the new direct inputs including com-
 93 //                          binations not normally used. The upper level module
 94 //                          must restrict the format inputs to those that are
 95 //                          proper. 7-bit formats require parity, and if not
 96 //                          set on input, then the parity generated will be
 97 //                          that specified by the Par bits.
 98 //
 99 //  2.00    11B06   MAM     Converted to Verilog 2001.
100 //
101 //  2.01    13G06   MAM     Corrected placement of #1 delay statements. Changed
102 //                          combinatorial always to use @(*) instead of explicit
103 //                          listing of signals in sensitivity list.
104 //
105 //  2.10    13L29   MAM     Modified header and description to indicate module
106 //                          now part of the Parallel Interface UART project.
107 //
108 // Additional Comments:
109 //
110 ////////////////////////////////////////////////////////////////////////////////
111
112 module UART_TXSM(
113     input   Rst,
114     input   Clk,
115
116     input   CE_16x,         // 16x Clock Enable - Baud Rate x16
117
118     input   Len,            // Word length: 0 - 8-bits; 1 - 7 bits
119     input   NumStop,        // Number Stop Bits: 0 - 1 Stop; 1 - 2 Stop
120     input   ParEn,          // Parity Enable
121     input   [1:0] Par,      // 0 - Odd; 1 - Even; 2 - Space (0); 3 - Mark (1)
122
123     input   TF_EF,          // Transmit THR Empty Flag
124
125     input   [7:0] THR,      // Transmit Holding Register
126     output  reg TF_RE,      // Transmit THR Read Enable Strobe
127
128     input   CTSi,           // RS232 Mode CTS input
129
130     output  reg TxD,        // Serial Data Out, LSB First, Start bit = 0
131
132     output  TxIdle,         // Transmit State Machine - Idle State
133     output  TxStart,        // Transmit State Machine - Start State - CTS wait
134     output  TxShift,        // Transmit State Machine - Shift State
135     output  TxStop          // Transmit State Machine - Stop State - RTS clear
136 );
137
138 ////////////////////////////////////////////////////////////////////////////////
139 //
140 //  Module Parameters
141 //
142
143 localparam  pIdle       =  0;   // Idle  - wait for data
144 localparam  pStopDelay  =  1;   // Stop  - deassert DE/RTS after 1 bit delay
145 localparam  pStartDelay =  2;   // Start - assert DE/RTS, delay 1 bit & CTSi
146 localparam  pUnused     =  3;   // Unused State
147 localparam  pStartBit   = 10;   // Shift - transmit Start Bit
148 localparam  pShift0     = 11;   // Shift - transmit TSR contents (LSB)
149 localparam  pShift1     =  9;   // Shift - transmit TSR contents
150 localparam  pShift2     =  8;   // Shift - transmit TSR contents
151 localparam  pShift3     = 12;   // Shift - transmit TSR contents
152 localparam  pShift4     = 13;   // Shift - transmit TSR contents
153 localparam  pShift5     = 15;   // Shift - transmit TSR contents
154 localparam  pShift6     = 14;   // Shift - transmit TSR contents
155 localparam  pShift7     =  6;   // Shift - transmit TSR contents (MSB)
156 localparam  pParityBit  =  4;   // Shift - transmit Parity Bit
157 localparam  pStopBit2   =  5;   // Shift - transmit Stop Bit 1
158 localparam  pStopBit1   =  7;   // Shift - transmit Stop Bit 2
```

```
159
160 ////////////////////////////////////////////////////////////////////////////
161 //
162 //  Local Signal Declarations
163 //
164
165 reg     [3:0] Bit;      // Bit Rate Divider
166 reg     CE_BCnt;        // CEO Bit Rate Divider
167
168 wire    Odd, Evn;       // Odd/Even parity signals
169 reg     ParBit;         // Computed/assigned Parity Bit
170
171 reg     [8:0] TSR;      // Transmit Shift Register
172
173 (* FSM_ENCODING="SEQUENTIAL",
174    SAFE_IMPLEMENTATION="YES",
175    SAFE_RECOVERY_STATE="4'b0" *) reg [3:0] TxSM = pIdle; // Xmt State Machine
176
177 ////////////////////////////////////////////////////////////////////////////
178 //
179 //  Implementation
180 //
181
182 //  Set Transmit Idle Status Bit
183
184 assign TxIdle  = (TxSM == pIdle);       // Idle state
185 assign TxStop  = (TxSM == pStopDelay);  // Wait 1 bit to release line
186 assign TxStart = (TxSM == pStartDelay); // Take line and settle 1 bit
187 assign TxShift = (TxSM[3] | TxSM[2]);   // Xmt Shift States
188
189 //  Transmit State Machine Clock Enable
190
191 assign CE_TxSM = (TxIdle ? CE_16x : CE_BCnt);
192
193 //  Shift Register Load Signal
194
195 assign Ld_TSR = CE_TxSM & ~TF_EF & CTSi
196                 & (  (TxSM == pStartDelay)
197                    | (TxSM == pStopBit1)
198                    | (TxSM == pStopDelay) );
199
200 //  Generate Transmit FIFO Read Enable
201
202 always @(posedge Clk)
203 begin
204     if(TxIdle)
205         TF_RE <= #1 1'b0;
206     else
207         TF_RE <= #1 Ld_TSR;
208 end
209
210 //  Determine Load Value for Transmit Shift Register
211
212 assign Evn = ((Len) ? ^{1'b0, THR[6:0]} : ^THR);
213 assign Odd = ~Evn;
214
215 always @(*)
216 begin
217     case(Par)
218         2'b00 : ParBit <= Odd;  // Odd
219         2'b01 : ParBit <= Evn;  // Even
220         2'b10 : ParBit <= 0;    // Space
221         2'b11 : ParBit <= 1;    // Mark
222     endcase
223 end
224
225 //  Transmit Shift Register
226
227 always @(posedge Clk)
228 begin
229     if(TxIdle)
230         TSR <= #1 9'b1_1111_1111;
231     else if(Ld_TSR)
232         TSR <= #1 {ParBit, THR[7:0]};
233 end
234
235 always @(posedge Clk)
236 begin
237     if(Rst)
```

```
238         TxD <= #1 1;
239     else case(TxSM)
240         pStartBit   : TxD <= #1 0;
241         pShift0     : TxD <= #1 TSR[0];
242         pShift1     : TxD <= #1 TSR[1];
243         pShift2     : TxD <= #1 TSR[2];
244         pShift3     : TxD <= #1 TSR[3];
245         pShift4     : TxD <= #1 TSR[4];
246         pShift5     : TxD <= #1 TSR[5];
247         pShift6     : TxD <= #1 TSR[6];
248         pShift7     : TxD <= #1 TSR[7];
249         pParityBit  : TxD <= #1 TSR[8];
250         default     : TxD <= #1 1;
251     endcase
252 end
253
254 //  Bit Rate Divider
255
256 always @(posedge Clk)
257 begin
258     if(TxIdle)
259         Bit <= #1 0;
260     else if(CE_16x)
261         Bit <= #1 Bit + 1;
262 end
263
264 always @(posedge Clk)
265 begin
266     if(TxIdle)
267         CE_BCnt <= #1 0;
268     else
269         CE_BCnt <= #1 CE_16x & (&Bit);
270 end
271
272 //  Transmit State Machine
273
274 always @(posedge Clk)
275 begin
276     if(Rst)
277         TxSM <= #1 pIdle;
278     else if(CE_TxSM)
279         case(TxSM)
280             pIdle       : TxSM <= #1 ((TF_EF) ? pIdle
281                                       : pStartDelay);
282
283             pStartDelay : TxSM <= #1 ((TF_EF) ? pIdle
284                                       : ((CTSi) ? pStartBit
285                                                 : pStartDelay));
286
287             pStartBit   : TxSM <= #1 pShift0;
288             pShift0     : TxSM <= #1 pShift1;
289             pShift1     : TxSM <= #1 pShift2;
290             pShift2     : TxSM <= #1 pShift3;
291             pShift3     : TxSM <= #1 pShift4;
292             pShift4     : TxSM <= #1 pShift5;
293             pShift5     : TxSM <= #1 pShift6;
294             pShift6     : TxSM <= #1 ((Len)   ? pParityBit
295                                       : pShift7   );
296 //            pShift7     : TxSM <= #1 ((ParEn) ? pParityBit
297 //                                      : ((NumStop) ? pStopBit2
298 //                                                   : pStopBit1));
299 //
300 //            pParityBit  : TxSM <= #1 ((NumStop) ? pStopBit2
301 //                                                : pStopBit1);
302             pShift7     : TxSM <= #1 ((ParEn) ? pParityBit
303                                       : pStopBit2);
304
305             pParityBit  : TxSM <= #1 pStopBit2;
306
307             pStopBit2   : TxSM <= #1 pStopBit1;
308             pStopBit1   : TxSM <= #1 ((TF_EF) ? pStopDelay
309                                       : pStartBit );
310
311             pStopDelay  : TxSM <= #1 ((TF_EF) ? pIdle
312                                       : ((CTSi) ? pStartBit
313                                                 : pStartDelay));
314
315             pUnused     : TxSM <= #1 pIdle;
316         endcase
```

```
317 end
318
319 endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
//  Receiver SM and Shift Register module for UART.
//
//  Copyright 2008-2014 by Michael A. Morris, dba M. A. Morris & Associates
//
//  All rights reserved. The source code contained herein is publicly released
//  under the terms and conditions of the GNU General Public License as conveyed
//  in the license provided below.
//
//  This program is free software: you can redistribute it and/or modify it
//  under the terms of the GNU General Public License as published by the Free
//  Software Foundation, either version 3 of the License, or any later version.
//
//  This program is distributed in the hope that it will be useful, but WITHOUT
//  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
//  more details.
//
//  You should have received a copy of the GNU General Public License along with
//  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
//
//  Free Software Foundation, Inc.
//  51 Franklin Street, Fifth Floor
//  Boston, MA  02110-1301 USA
//
//  Further, no use of this source code is permitted in any form or means
//  without inclusion of this banner prominently in any derived works.
//
//  Michael A. Morris <morrisma_at_mchsi_dot_com>
//  164 Raleigh Way
//  Huntsville, AL 35811
//  USA
//
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:         M. A. Morris & Associates
// Engineer:        Michael A. Morris
//
// Create Date:     08:44:44 05/31/2008
// Design Name:     Parallel Interface UART
// Module Name:     ../VerilogCoponentsLib/UART/UART_RXSM.v
// Project Name:    Verilog Components Library
// Target Devices:  XC3S50A-4VQG100I, XC3S20A-4VQG100I, XC3S700AN-4FFG484I
// Tool versions:   ISE 10.1i SP3
//
// Description:
//
//  This module implements the Receive State Machine for the Parallel Interface
//  UART.
//
// Dependencies:    None
//
// Revision History:
//
//  0.01    08E31   MAM     File Created
//
//  1.00    08F08   MAM     Module Released
//
//  1.01    08F12   MAM     Pulled Format Decoder ROM and added its outputs to
//                          the port list. Allows greater format specification
//                          flexibility. 7-bit formats require parity. If ParEn
//                          is not set, the parity will be tested according to
//                          the settings of the Par bits, which may result in
//                          anomalous behavior.
//
//  2.00    11B06   MAM     Converted to Verilog 2001.
//
//  2.10    13L28   MAM     Modified to headers and description to indicate
//                          module is now part of the Parallel Interface UART
//                          project. Modified RxSM to support reporting parity
//                          and framing errors. Changed pChkParity state to
//                          always go through ChkStop1 and/or ChkStop2 states
//                          to ensure that framing errors are checked even after
//                          a parity error is detected or not. Previously, a
//                          detecting a parity error skipped the framing error
```

```
 80 //                          check, and required finding a mark level before con-
 81 //                          tinuing. Now, a parity error is recorded and a fram-
 82 //                          ing error is performed. If a framing error is de-
 83 //                          tected, the RxSM records the error and requires a
 84 //                          mark level before continuing. Modified bit counter
 85 //                          from an up counter to a down counter.
 86 //
 87 // Additional Comments:
 88 //
 89 //////////////////////////////////////////////////////////////////////////////////
 90
 91 module UART_RXSM(
 92     input   Rst,
 93     input   Clk,
 94
 95     input   CE_16x,         // 16x Clock Enable - Baud Rate x16
 96
 97     input   Len,            // Word length: 0 - 8-bits; 1 - 7 bits
 98     input   NumStop,        // Number Stop Bits: 0 - 1 Stop; 1 - 2 Stop
 99     input   ParEn,          // Parity Enable
100     input   [1:0] Par,      // 0 - Odd; 1 - Even; 2 - Space (0); 3 - Mark (1)
101
102     input   RxD,            // Input Asynchronous Serial Receive Data
103
104     output  reg [9:0] RD,   // Receive Data Output - bit 9 = FE, bit 8 = PE
105     output  reg WE_RHR,     // Write Enable - Receive Holding Register
106
107     output  RxWait,         // RxSM - Wait State
108     output  RxIdle,         // RxSM - Idle State
109     output  RxStart,        // RxSM - Start Bit Check State
110     output  RxShift,        // RxSM - RxD Shift State
111     output  RxParity,       // RxSM - RxD Parity Shift State
112     output  RxStop,         // RxSM - RxD Stop Bit Check States
113     output  RxError         // RxSM - Error State
114 );
115
116 //////////////////////////////////////////////////////////////////////////////////
117 //
118 //  Module Parameters
119 //
120
121 //  Receive State Machine Declarations (Binary)
122
123 localparam pWaitMark  =  0;
124 localparam pChkMark   =  1;
125 localparam pIdle      =  3;
126 localparam pChkStart  =  2;
127 localparam pShift0    = 10;
128 localparam pShift1    = 11;
129 localparam pShift2    =  9;
130 localparam pShift3    =  8;
131 localparam pShift4    = 12;
132 localparam pShift5    = 13;
133 localparam pShift6    = 15;
134 localparam pShift7    = 14;
135 localparam pChkStop1  =  6;
136 localparam pChkStop2  =  7;
137 localparam pChkParity =  5;
138 localparam pRcvError  =  4;
139
140 //////////////////////////////////////////////////////////////////////////////////
141 //
142 //  Local Signal Declarations
143 //
144
145 (* FSM_ENCODING="SEQUENTIAL",
146    SAFE_IMPLEMENTATION="YES",
147    SAFE_RECOVERY_STATE="4'b0" *) reg [3:0] RxSM = pWaitMark;
148
149 reg     [3:0] BCnt;
150 reg     [9:0] RSR;
151
152 reg     ParErr;
153
154 //////////////////////////////////////////////////////////////////////////////////
155 //
156 //  Implementation
157 //
158
```

```
159 ////////////////////////////////////////////////////////////////////////////////
160 //
161 //  RxSM Decode
162 //
163
164 assign RxSM_Wait = (  (RxSM == pWaitMark)
165                     | (RxSM == pChkMark)
166                     | (RxSM == pIdle)
167                     | (RxSM == pRcvError) );
168
169 assign RxWait   = RxSM_Wait;
170 assign RxIdle   = (RxSM == pIdle);
171 assign RxStart  = (RxSM == pChkStart);
172 assign RxShift  = (  (RxSM == pShift0)
173                    | (RxSM == pShift1)
174                    | (RxSM == pShift2)
175                    | (RxSM == pShift3)
176                    | (RxSM == pShift4)
177                    | (RxSM == pShift5)
178                    | (RxSM == pShift6)
179                    | (RxSM == pShift7) );
180 assign RxParity = (RxSM == pChkParity);
181 assign RxStop   = (RxSM == pChkStop1) | (RxSM == pChkStop2);
182 assign RxError  = (RxSM == pRcvError);
183
184 ////////////////////////////////////////////////////////////////////////////////
185 //
186 //  Bit Rate Prescaler
187 //
188 //      Prescaler is held in the half-bit load state during reset and when the
189 //      RXSM is in the RxSM_Wait states. As a consequence, the first overflow
190 //      is only half a bit period, and only occurs in the pChkStart state.
191 //      Subsequently, overflows occur once per bit period in the middle of each
192 //      of the remaining bits.
193 //
194
195 assign Rst_BCnt = Rst | RxSM_Wait;
196
197 always @(posedge Clk)
198 begin
199     if(Rst_BCnt)
200          BCnt <= #1 4'b0111;
201     else if(CE_16x)
202          BCnt <= #1 BCnt - 1;
203 end
204
205 assign TC_BCnt = CE_16x & ~|BCnt;
206
207 ////////////////////////////////////////////////////////////////////////////////
208
209 assign CE_RxSM = (RxSM_Wait ? CE_16x : TC_BCnt);
210
211 ////////////////////////////////////////////////////////////////////////////////
212 //
213 //  Receive Shift Register
214 //
215
216 always @(posedge Clk)
217 begin
218     if(Rst)
219          RSR <= #1 0;
220     else if(CE_RxSM)
221         case(RxSM)
222             pChkStart  : RSR    <= #1 8'b0;
223             pShift0    : RSR[0] <= #1 RxD;
224             pShift1    : RSR[1] <= #1 RxD;
225             pShift2    : RSR[2] <= #1 RxD;
226             pShift3    : RSR[3] <= #1 RxD;
227             pShift4    : RSR[4] <= #1 RxD;
228             pShift5    : RSR[5] <= #1 RxD;
229             pShift6    : RSR[6] <= #1 RxD;
230             pShift7    : RSR[7] <= #1 RxD;
231             pChkParity : RSR[8] <= #1 ParErr;
232             pChkStop2  : RSR[9] <= #1 ~RxD;
233             pChkStop1  : RSR[9] <= #1 ~RxD;
234             default    : RSR    <= #1 RSR;
235         endcase
236 end
237
```

```
238 ////////////////////////////////////////////////////////////////////////////////
239 //
240 //  Parity Checker
241 //      if not Check Parity State, then ParErr = 0
242 //
243
244 assign OddPar = ^{RxD, RSR};
245 assign EvnPar = ~OddPar;
246
247 always @(*)
248 begin
249     case(Par)
250         2'b00 : ParErr <= ~OddPar;
251         2'b01 : ParErr <= ~EvnPar;
252         2'b10 : ParErr <=  RxD;
253         2'b11 : ParErr <= ~RxD;
254     endcase
255 end
256
257 ////////////////////////////////////////////////////////////////////////////////
258 //
259 //  Receive Holding Register Data
260 //
261
262 assign CE_RD = CE_RxSM & (((RxSM == pChkStop1) & RxD) | (RxSM == pRcvError));
263
264 always @(posedge Clk)
265 begin
266     if(Rst)
267          RD <= #1 0;
268     else if(CE_RD)
269          RD <= #1 RSR;
270 end
271
272 always @(posedge Clk)
273 begin
274     if(Rst)
275          WE_RHR <= #1 1'b0;
276     else
277          WE_RHR <= #1 CE_RD;
278 end
279
280 ////////////////////////////////////////////////////////////////////////////////
281 //
282 //  RxSM - Receive State Machine
283 //
284 //      The Receive State Machine starts in the WaitMark state to insure that
285 //      the receive line is in the marking state before continuing. The ChkMark
286 //      state validates that the receive line is in the marking state. If it
287 //      is, then the RxSM is adanced to the Idle state to wait for the start
288 //      bit. Otherwise, RxSM returns to the WaitMark state to wait for the line
289 //      to return to the marking (idle) state.
290 //
291 //      The format is processed through the receive sequence. The length,
292 //      parity options, and the number of stop bits determine the state tran-
293 //      sitions that the RxSM makes. A parity or a framing error is simply
294 //      recorded as errors in the 9th and 10th bits of the receive shift regis-
295 //      ter. Invalid stop bits, line break conditions, etc. are handled through
296 //      the pRcvError and the pWaitMark states. Thus, line break conditions are
297 //      not expected to cause the RxSM to receive more than one character, and
298 //      the pWaitMark state will hold the RxSM in a wait state until RxD returns
299 //      to the marking (idle) level.
300 //
301
302 always @(posedge Clk)
303 begin
304     if(Rst)
305          RxSM <= #1 pWaitMark;
306     else if(CE_RxSM)
307         case(RxSM)
308             pWaitMark  :  RxSM <= #1 ( RxD ? pChkMark
309                                        : pWaitMark);
310
311             pChkMark   :  RxSM <= #1 ( RxD ? pIdle
312                                        : pWaitMark);
313
314             pIdle      :  RxSM <= #1 (~RxD ? pChkStart
315                                        : pIdle);
316
```

```
317             pChkStart  :  RxSM <= #1 ( RxD ? pIdle
318                                         : pShift0);
319
320             pShift0    :  RxSM <= #1 pShift1;
321             pShift1    :  RxSM <= #1 pShift2;
322             pShift2    :  RxSM <= #1 pShift3;
323             pShift3    :  RxSM <= #1 pShift4;
324             pShift4    :  RxSM <= #1 pShift5;
325             pShift5    :  RxSM <= #1 pShift6;
326
327             pShift6    :  RxSM <= #1 (Len    ? pChkParity
328                                         : pShift7);
329
330             pShift7    :  RxSM <= #1 (ParEn  ? pChkParity
331                                         : (NumStop ? pChkStop2
332                                                    : pChkStop1));
333
334 //            pChkParity :  RxSM <= #1 (ParErr ? pRcvError
335 //                                          : (NumStop ? pChkStop2
336 //                                                     : pChkStop1));
337
338             pChkParity :  RxSM <= #1 (NumStop ? pChkStop2
339                                         : pChkStop1);
340
341             pChkStop2  :  RxSM <= #1 (RxD ? pChkStop1
342                                         : pRcvError);
343
344             pChkStop1  :  RxSM <= #1 (RxD ? pIdle
345                                         : pRcvError);
346
347             pRcvError  :  RxSM <= #1 pWaitMark;
348
349             default    :  RxSM <= #1 pWaitMark;
350         endcase
351 end
352
353 endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
//  Receive Timeout module for UART.
//
//  Copyright 2008-2014 by Michael A. Morris, dba M. A. Morris & Associates
//
//  All rights reserved. The source code contained herein is publicly released
//  under the terms and conditions of the GNU General Public License as conveyed
//  in the license provided below.
//
//  This program is free software: you can redistribute it and/or modify it
//  under the terms of the GNU General Public License as published by the Free
//  Software Foundation, either version 3 of the License, or any later version.
//
//  This program is distributed in the hope that it will be useful, but WITHOUT
//  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
//  more details.
//
//  You should have received a copy of the GNU General Public License along with
//  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
//
//  Free Software Foundation, Inc.
//  51 Franklin Street, Fifth Floor
//  Boston, MA  02110-1301 USA
//
//  Further, no use of this source code is permitted in any form or means
//  without inclusion of this banner prominently in any derived works.
//
//  Michael A. Morris <morrisma_at_mchsi_dot_com>
//  164 Raleigh Way
//  Huntsville, AL 35811
//  USA
//
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:         M. A. Morris & Associates
// Engineer:        Michael A. Morris
//
// Create Date:     06:50:40 06/14/2008
// Design Name:     Synchronous Serial Peripheral (SSP) Interface UART
// Module Name:     ../VerilogCoponentsLib/SSP_UART/UART_RTO.v
// Project Name:    Verilog Components Library
// Target Devices:  XC3S50A-4VQG100I, XC3S20A-4VQG100I, XC3S700AN-4FFG484I
// Tool versions:   ISE 10.1i SP3
//
// Description: This module implements a receive timeout timer which sets a
//              flag to indicate that the timeout occured. The flag is cleared
//              by reset or by the reading of the Receive Holding Register.
//
//              The Receive Timeout Val should be set by the client module to a
//              value which corresponds to the desired timeout value in terms
//              of the number of bit times. This module scales the 16x bit rate
//              clock enable by 16 to set the internal bit rate clock enable.
//              The client module provide the character frame length and the
//              number of characters to delay. The character frame length
//              varies as a function of the frame format, e.g. 8N1, 8E2, or
//              7O1. The number of characters to delay for the timeout is a
//              fixed number of character periods, e.g. 3, in most cases. The
//              module allows these values to be independently specified.
//
//
// Dependencies:    none
//
// Revision History:
//
//  0.01    08E14   MAM     File Created
//
//  1.00    08E14   MAM     Initial Release
//
//  1.10    08E14   MAM     Changed the input parameters to have two delay
//                          parameters: CCntVal - character length; and
//                                      RTOVal  - # characters to delay.
//                          The result is a faster implementation, and one
//                          the RTOVal can usually be set as a constant.
//
```

```
 80 //  1.11    08E15   MAM     Changed Module Name
 81 //
 82 //  2.00    11B06   MAM     Converted to Verilog 2001.
 83 //
 84 // Additional Comments:
 85 //
 86 //////////////////////////////////////////////////////////////////////////////////
 87
 88 module UART_RTO(
 89     input   Rst,
 90     input   Clk,
 91
 92     input   CE_16x,
 93
 94     input   WE_RHR,
 95     input   RE_RHR,
 96
 97     input   [3:0] CCntVal,
 98     input   [3:0] RTOVal,
 99
100     output  reg RcvTimeout
101 );
102
103 //////////////////////////////////////////////////////////////////////////////////
104 //
105 //  Local Signal Declarations
106 //
107
108     wire    Clr_RTOArm;
109     reg     RTOArm;
110
111     wire    Clr_BCnt;
112     reg     [3:0] BCnt;
113     reg     TC_BCnt;
114
115     wire    Clr_CCnt;
116     reg     [3:0] CCnt;
117     reg     TC_CCnt;
118
119     wire    Clr_RTOCnt, CE_RTOCnt;
120     reg     [3:0] RTOCnt;
121     reg     TC_RTOCnt;
122
123     wire    Clr_RTO, CE_RTO;
124
125 //////////////////////////////////////////////////////////////////////////////////
126 //
127 //  Implementation
128 //
129 //  RTO Arm FF
130 //      Armed with each received character, cleared by Rst, RE_RHR, or timeout
131 //
132
133 assign Clr_RTOArm = RE_RHR | CE_RTO;
134
135 always @(posedge Clk)
136 begin
137     if(Rst)
138         RTOArm <= #1 0;
139     else if(Clr_RTOArm)
140         RTOArm <= #1 0;
141     else if(WE_RHR)
142         RTOArm <= #1 1;
143 end
144
145 //////////////////////////////////////////////////////////////////////////////////
146 //
147 //  Bit Rate Divider
148 //      Held in reset until RTO Armed or if Rst asserted
149 //
150
151 assign Clr_BCnt = Rst | WE_RHR | Clr_RTOArm | ~RTOArm;
152
153 always @(posedge Clk)
154 begin
155     if(Clr_BCnt)
156         BCnt <= #1 0;
157     else if(CE_16x)
158         BCnt <= #1 BCnt + 1;
```

```
159 end
160
161 always @(posedge Clk)
162 begin
163     if(Clr_BCnt)
164         TC_BCnt <= #1 0;
165     else if(CE_16x)
166         TC_BCnt <= #1 (BCnt == 4'b1110);
167 end
168
169 //////////////////////////////////////////////////////////////////////////////
170 //
171 //  Character Frame Divider
172 //      Held in reset until RTO Armed or if Rst asserted
173 //
174
175 assign Clr_CCnt = Clr_BCnt;
176 assign CE_CCnt = CE_16x & TC_BCnt;
177
178 always @(posedge Clk)
179 begin
180     if(Clr_CCnt)
181         CCnt <= #1 CCntVal;
182     else if(CE_CCnt)
183         CCnt <= #1 ((TC_CCnt) ? CCntVal : CCnt - 1);
184 end
185
186 always @(posedge Clk)
187 begin
188     if(Clr_CCnt)
189         TC_CCnt <= #1 0;
190     else if(CE_16x)
191         TC_CCnt <= #1 (CCnt == 0);
192 end
193
194 //////////////////////////////////////////////////////////////////////////////
195 //
196 //  Receive Timeout Counter
197 //      Held in reset until RTO Armed or if Rst asserted
198 //      Counts bit periods when RTO Armed
199
200 assign Clr_RTOCnt = Clr_BCnt;
201 assign CE_RTOCnt  = CE_16x & TC_BCnt & TC_CCnt;
202
203 always @(posedge Clk)
204 begin
205     if(Clr_RTOCnt)
206         RTOCnt <= #1 RTOVal;
207     else if(CE_RTOCnt)
208         RTOCnt <= #1 ((TC_RTOCnt) ? RTOVal : RTOCnt - 1);
209 end
210
211 always @(posedge Clk)
212 begin
213     if(Clr_RTOCnt)
214         TC_RTOCnt <= #1 0;
215     else if(CE_16x)
216         TC_RTOCnt <= #1 (RTOCnt == 0);
217 end
218
219 //////////////////////////////////////////////////////////////////////////////
220 //
221 //  Receive Timeout Latch
222 //      Cleared by Rst or any read of the RHR
223 //      Set by RTOCnt terminal count
224
225 assign Clr_RTO = RE_RHR;
226 assign CE_RTO  = CE_16x & TC_BCnt & TC_CCnt & TC_RTOCnt;
227
228 always @(posedge Clk)
229 begin
230     if(Rst)
231         RcvTimeout <= #1 0;
232     else if(Clr_RTO)
233         RcvTimeout <= #1 0;
234     else if(CE_RTO)
235         RcvTimeout <= #1 1;
236 end
237
```

```
238 endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//
//  Interrupt Generator module for UART.
//
//  Copyright 2008-2014 by Michael A. Morris, dba M. A. Morris & Associates
//
//  All rights reserved. The source code contained herein is publicly released
//  under the terms and conditions of the GNU General Public License as conveyed
//  in the license provided below.
//
//  This program is free software: you can redistribute it and/or modify it
//  under the terms of the GNU General Public License as published by the Free
//  Software Foundation, either version 3 of the License, or any later version.
//
//  This program is distributed in the hope that it will be useful, but WITHOUT
//  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
//  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
//  more details.
//
//  You should have received a copy of the GNU General Public License along with
//  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
//
//  Free Software Foundation, Inc.
//  51 Franklin Street, Fifth Floor
//  Boston, MA  02110-1301 USA
//
//  Further, no use of this source code is permitted in any form or means
//  without inclusion of this banner prominently in any derived works.
//
//  Michael A. Morris <morrisma_at_mchsi_dot_com>
//  164 Raleigh Way
//  Huntsville, AL 35811
//  USA
//
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:         M. A. Morris & Associates
// Engineer:        Michael A. Morris
//
// Create Date:     09:37:34 06/15/2008
// Design Name:     Parallel Interface UART
// Module Name:     ../VerilogCoponentsLib/UART/UART_INT.v
// Project Name:    Verilog Components Library
// Target Devices:  XC3S50A-4VQG100I, XC3S20A-4VQG100I, XC3S700AN-4FFG484I
// Tool versions:   ISE 10.1i SP3
//
// Description:
//
//  This module implements the interrupt request logic for the parallel inter-
//  face UART. Interrupt requests are generated for four conditions:
//
//  (1) Transmit FIFO Empty;
//  (2) Transmit FIFO Half Empty;
//  (3) Receive FIFO Half Full;
//  (4) and Receive Timeouts.
//
//  The interrupt flags will be reset/cleared as indicated below except when the
//  rst/clr pulse is coincident with the pulse which would set the flag, the
//  flag remains set so that the new assertion pulse is not lost.
//
//  There remains a small probability that the second pulse may be lost. This
//  can be remedied by stretching the setting pulse to a width equal to the un-
//  certainty between the setting and resetting pulses: approximately 4 clock
//  cycles when re1ce modules are used to generate the Clr_Int pulse.
//
// Dependencies: redet.v, fedet.v
//
// Revision History:
//
//  0.01    13L28   MAM     File Created. Implementation copied from that used
//                          for the SSP UART
//
// Additional Comments:
//
//      The interrupt flags are set and reset under a variety of conditions.
//
```

```
 80 //      iTHE -  Set on the falling edge of Transmit FIFO Half Full (TF_HF)
 81 //              Rst on Clr_Int.
 82 //
 83 //      iTFE -  Set on the rising edge of the Transmit FIFO Empty Flag (TF_EF)
 84 //              Rst on Clr_Int.
 85 //
 86 //      iRFE -  Set on the rising edge of Receive Error at output of Rx FIFO
 87 //              Rst of Clr_Int.
 88 //
 89 //      iRHF -  Set on the rising edge of Receive FIFO Half Full (RF_HF)
 90 //              Rst on Clr_Int or on the falling edge of RF_HF.
 91 //
 92 //      iRTO -  Set on the rising edge of RTO
 93 //              Rst on Clr_Int.
 94 //
 95 ////////////////////////////////////////////////////////////////////////////
 96
 97 module UART_INT(
 98     input   Rst,                    // Reset
 99     input   Clk,                    // Clock
100
101     input   TF_HF,                  // Transmit FIFO Half-Full Flag
102     input   TF_EF,                  // Transmit FIFO Empty Flag
103     input   TxIdle,                 // Transmit Idle Flag
104
105     input   RF_FE,                  // Receive FIFO Framing Error Flag
106     input   RF_PE,                  // Receive FIFO Parity Error Flag
107     input   RF_HF,                  // Receive FIFO Half-Full Flag
108     input   RF_EF,                  // Receive FIFO Empty Flag
109     input   RTO,                    // Receive Timeout Flag
110
111     input   Clr_Int,                // Clear interrupt pulse
112
113     output  reg iTHE,               // Interrupt - Tx FIFO Half Empty
114     output  reg iTEF,               // Interrupt - Tx FIFO Empty
115     output  reg iTEM,               // Interrupt - Tx SR Empty
116
117     output  reg iRFE,               // Interrupt - Rx FIFO Framing Error
118     output  reg iRPE,               // Interrupt - Rx FIFO Parity Error
119     output  reg iRHF,               // Interrupt - Rx FIFO Half Full
120     output  reg iRTO                // Interrupt - Receive Time Out Detected
121 );
122
123 ////////////////////////////////////////////////////////////////////////////
124 //
125 //  Local Signal Declarations
126 //
127
128     wire    reTF_EF, feTF_HF, reTxIdle;
129     wire    reRF_FE, reRF_PE;
130     wire    reRF_HF, feRF_HF, reRF_EF;
131     wire    reRTO;
132
133 ////////////////////////////////////////////////////////////////////////////
134 //
135 //  Implementation
136 //
137
138 redet   RE1 (.rst(Rst), .clk(Clk), .din(TF_EF),  .pls(reTF_EF));
139 fedet   FE1 (.rst(Rst), .clk(Clk), .din(TF_HF),  .pls(feTF_HF));
140 redet   RE2 (.rst(Rst), .clk(Clk), .din(TxIdle), .pls(reTxIdle));
141
142 redet   RE3 (.rst(Rst), .clk(Clk), .din(RF_FE),  .pls(reRF_FE));
143 redet   RE4 (.rst(Rst), .clk(Clk), .din(RF_PE),  .pls(reRF_PE));
144 redet   RE5 (.rst(Rst), .clk(Clk), .din(RF_HF),  .pls(reRF_HF));
145 fedet   FE2 (.rst(Rst), .clk(Clk), .din(RF_HF),  .pls(feRF_HF));
146 redet   RE6 (.rst(Rst), .clk(Clk), .din(RF_EF),  .pls(reRF_EF));
147
148 redet   RE7 (.rst(Rst), .clk(Clk), .din(RTO),    .pls(reRTO));
149
150 ////////////////////////////////////////////////////////////////////////////
151 //
152 //  Transmit FIFO Half Empty Interrupt Flag
153 //
154
155 assign Rst_iTHE = Rst | (iTHE & Clr_Int);
156
157 always @(posedge Clk or posedge feTF_HF)
158 begin
```

```
159     if(feTF_HF)
160         iTHE <= #1 1;
161     else if(Rst_iTHE)
162         iTHE <= #1 feTF_HF;
163 end
164 
165 ////////////////////////////////////////////////////////////////////////////
166 //
167 //  Transmit FIFO Empty Interrupt Flag
168 //
169 
170 assign Rst_iTEF = Rst | (iTEF & Clr_Int);
171 
172 always @(posedge Clk or posedge reTF_EF)
173 begin
174     if(reTF_EF)
175         iTEF <= #1 1;
176     else if(Rst_iTEF)
177         iTEF <= #1 reTF_EF;
178 end
179 
180 ////////////////////////////////////////////////////////////////////////////
181 //
182 //  Transmit SR Empty Interrupt Flag
183 //
184 
185 assign Rst_iTEM = Rst | (iTEM & Clr_Int);
186 
187 always @(posedge Clk or posedge reTxIdle)
188 begin
189     if(reTxIdle)
190         iTEM <= #1 1;
191     else if(Rst_iTEM)
192         iTEM <= #1 reTxIdle;
193 end
194 
195 ////////////////////////////////////////////////////////////////////////////
196 //
197 //  Receive FIFO Framing Error Interrupt Flag
198 //
199 
200 assign Rst_iRFE = Rst | (iRFE & Clr_Int);
201 
202 always @(posedge Clk or posedge reRF_FE)
203 begin
204     if(reRF_FE)
205         iRFE <= #1 1;
206     else if(Rst_iRFE)
207         iRFE <= #1 (RF_FE | reRF_FE);
208 end
209 
210 ////////////////////////////////////////////////////////////////////////////
211 //
212 //  Receive FIFO Parity Error Interrupt Flag
213 //
214 
215 assign Rst_iRPE = Rst | (iRPE & Clr_Int);
216 
217 always @(posedge Clk or posedge reRF_PE)
218 begin
219     if(reRF_PE)
220         iRPE <= #1 1;
221     else if(Rst_iRPE)
222         iRPE <= #1 (RF_PE | reRF_PE);
223 end
224 
225 ////////////////////////////////////////////////////////////////////////////
226 //
227 //  Receive Data Available Interrupt Flag
228 //
229 
230 assign Rst_iRHF = Rst | (iRHF & (Clr_Int | feRF_HF | reRF_EF));
231 
232 always @(posedge Clk or posedge reRF_HF)
233 begin
234     if(reRF_HF)
235         iRHF <= #1 1;
236     else if(Rst_iRHF)
237         iRHF <= #1 reRF_HF;
```

```
238 end
239
240 ////////////////////////////////////////////////////////////////////////////
241 //
242 //  Receive Timeout Interrupt Flag
243 //
244
245 assign Rst_iRTO = Rst | (iRTO & Clr_Int);
246
247 always @(posedge Clk or posedge reRTO)
248 begin
249     if(reRTO)
250         iRTO <= #1 1;
251     else if(Rst_iRTO)
252         iRTO <= #1 reRTO;
253 end
254
255 endmodule
```

```
header
Project: M65C02A_uP_ROM
File Revision: 0007
Author(s): Michael A. Morris
Description: M65C02A Microprogram
endh

--------------------------------------------------------------------------------
--
--  Copyright 2013-2014 by Michael A. Morris, dba M. A. Morris & Associates
--
--  All rights reserved. The source code contained herein is publicly released
--  under the terms and conditions of the GNU General Public License as conveyed
--  in the license provided below.
--
--  This program is free software: you can redistribute it and/or modify it
--  under the terms of the GNU General Public License as published by the Free
--  Software Foundation, either version 3 of the License, or any later version.
--
--  This program is distributed in the hope that it will be useful, but WITHOUT
--  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
--  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
--  more details.
--
--  You should have received a copy of the GNU General Public License along with
--  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
--
--  Free Software Foundation, Inc.
--  51 Franklin Street, Fifth Floor
--  Boston, MA  02110-1301 USA
--
--  Further, no use of this source code is permitted in any form or means
--  without inclusion of this banner prominently in any derived works.
--
--  Michael A. Morris <morrisma_at_mchsi_dot_com>
--  164 Raleigh Way
--  Huntsville, AL 35811
--  USA
--
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
-- Revision History:
--------------------------------------------------------------------------------
--
--  0001    13I16   mam     Initial conversion of M65C02_uPgm_ROM_V3a used with
--                          the M65C02_ALU.v module. Removed Rockwell instruc-
--                          tions. Instruction set supported now only supports
--                          original 65C02, otherwise known as the 65SC02, plus
--                          the WAI and STP instructions from W65C02S.
--
--  0002    14F21   mam     Completed change of microprogram to support the im-
--                          plementation of M65C02A Address Generator. In addi-
--                          tion, allocated saved two (2) NA control signals for
--                          future use as additional microprogram control sig-
--                          nals. These signals already set to allow the micro-
--                          program to use the BA field as data sources for the
--                          two temporary registers, {OP2, OP1}, and the decode
--                          ROM. It is expected that with this change and the
--                          change to the R multiplexer of the ALU, which allows
--                          the "opcode" field to determine the value by which
--                          INC/DEC, the microprogram will be able to implement
--                          a virtual FORTH machine or the SWEET16 virtual
--                          machine using registers in the zero/data page of
--                          memory.
--
--  0003    14G04   mam     Changed definition of the IO_Op so that it is one-
--                          hot encoded as IO_Op[1] is (RE | IF), and IO_Op[0]
--                          is WE.
--
--  0004    14G25   mam     Corrected an error in the DI_Op field of the STY dp
--                          and STY dp,X instructions. Rather than IF, the field
--                          was coded as IR which defines the wrong operation
--                          for reading the zero page operand address after the
--                          opcode. The result was an incorrect address output.
--
--  0005    14G27   mam     Changed all WE_M to WE_R. WE_M does properly allow
--                          the PSW to be updated during RMW instructions. WE_R
--                          was the means by which memory was written in the
```

```
 80 --                          previous microprogram version: M65C02_uPgm_V3.txt.
 81 --
 82 --  0006    14H01   mam     Modified to support relative addressing as 8-bit w/
 83 --                          sign extension or as 16-bit. If OP2 loaded simulta-
 84 --                          neously with OP1, then DI_Op[0] will load OP2 with
 85 --                          the sign extension of the operand register data in-
 86 --                          put. This operation supports 8-bit relative address-
 87 --                          ing. The address generator requires a simple modifi-
 88 --                          cation to remove the Rel internal data bus and to
 89 --                          substitute {OP2, OP1} in its stead. The microprogram
 90 --                          must control whether OP2 is loaded with the sign
 91 --                          of the relative offset loaded into OP1 or not in
 92 --                          order to implement the 8-bit relative branches.
 93 --
 94 --  0007    14H23   mam     Updating uP_Cntl field with mnemonics and updating
 95 --                          comment fields.
 96 --
 97 ------------------------------------------------------------------------------
 98 --  Comments
 99 ------------------------------------------------------------------------------
100 --
101 --  The microprogram controller being targeted is the F9408A MPC. That control-
102 --  ler provides for sequential execution (FTCH), microroutine subroutines (RTS,
103 --  BSR), multi0way branching (BMW), unconditional externally controlled branch-
104 --  ing (BRV0, BRV1, BRV2, BRV3), and conditional branching using external test
105 --  inputs (BTL0, BTH0, BTL1, BTH1, BTL2, BTH2, BTL3, BTH3). Past use of this
106 --  controller has been focused on using the FTCH, BRV0, BSR, RTS, and BTL0/BTH0
107 --  as the basis of control. An external multiplexer controlled by the microword
108 --  and tied to the T0 test pin has been used for most tests. BRV0 has been used
109 --  in a conventional manner, and as such, it has only been used as an uncondi-
110 --  tional branch to a microprogram address supplied in the microword. No extra-
111 --  ordinary use of these basic control structures has been attempted.
112 --
113 --  With the program and data memory of the target, the M65C02 synthesizable
114 --  microprocessor/microcomputer, external to the core, there is a need to
115 --  implement a memory interface. From an implementation perspective, the exter-
116 --  nal memory interface would need to supply a ready signal so that the M65C02
117 --  logic can capture any input data into the instruction register, IR, one of
118 --  the two internal temporary operand registers, OP1 and OP2, or one of the
119 --  programmer-visible registers of the processor core: A, X, Y, P, or S.
120 --
121 --  If the direct approach to using the F9408A MPC is maintained, a number of
122 --  additional clock cycles will be added to each operation. A discrete logic
123 --  FSM approach for the processor core controller would branch to any number of
124 --  multiple states as needed to minimize the total number of cycles needed to
125 --  implement any instruction of the processor core. A microprogrammed approach
126 --  for implementing the processor core is the objective because it provides a
127 --  more flexible approach to the implementation, and provides an easier path to
128 --  upgrading the instruction set with additional instructions from the Rockwell
129 --  and the Western Design Center versions of the basic processor core. With an
130 --  F9408A MPC, a simple straight forward approach can be taken to the develop-
131 --  ment of the microprogrammed state machine. Without using external logic to
132 --  augment the operation of the F9408A, the resulting micoprogram can be limit-
133 --  ed to simple, single variable tests, which will result in the additional of
134 --  clock cycles to most operations/algorithms. This is not a limitation of the
135 --  F9408A MPC itself, or microprogramming in general, but of the application in
136 --  which the F9408A is included.
137 --
138 --  Mult-way branching in standard FSMs is natural, but is also one of the most
139 --  difficult design aspects of FSMs. Depending on the type of FSM being develp-
140 --  ed, implementing (area and speed) efficient state transition equations for
141 --  FSMs with many branches in many states is difficult and can be very diffi-
142 --  cult to test and debug. The same statement applies to microprogrammed state
143 --  machines, and is one reason why most MPCs have only a limited number of
144 --  instructions which support multi-way branching. However, microprogrammed
145 --  state machines are not limited by the architectural limitation imposed by
146 --  standard MPCs.
147 --
148 --  The M65C02 supports the reset trap, a non-maskable interrupt (NMI), a mask-
149 --  able interrupt, and the break instruction trap. In most cases, the two
150 --  interrupts are evaluated at the completion of each instruction. However, the
151 --  first instruction after one of these traps/interrupts is always executed. To
152 --  allow for this behavior to be easily implemented, BRV1 is used to initiate
153 --  the execution of a instruction regardless of the state of the NMI and/or IRQ
154 --  signals. BMW is used to test up to three signals and select the appropriate
155 --  action. Thus, most instructions will initiate the fetch of the next instruc-
156 --  tion by terminating their execution with BMW. Using a 2-way table, the BMW
157 --  will either complete the execution of the current instruction and the fetch
158 --  of the next instruction's opcode, or it will branch into the interrupt/traps
```

```
159 --  handler microroutine. Single cycle instructions will use BRV3 for the same
160 --  purpose, but implemented in a different manner. The next state for BMW is
161 --  either BRV1 or BRV2. BRV1 is used to complete the execution of the current
162 --  instruction and fetch the next instruction's opcode. BRV2 is used to capture
163 --  the interrupt vector and complete the execution of the current instruction.
164 --  Both the IRQ/NMI trap handler and the BRK handler start with a BRV2 instruc-
165 --  tion, which completes the current instruction and captures the PC. The BRV3
166 --  MPC operation performs the same function as BRV1, but the next state is
167 --  either the first state of the next instruction or the IRQ/NMI trap micro-
168 --  routine, which capture's the current instruction PC with BRV2.
169 --
170 --  The microprogram behavior is based on two assumptions: (1) external memory
171 --  is of a type in which the read data available is related to the address pre-
172 --  sented during the cycle, i.e. asynchronous, no-wait state RAM; and (2) the
173 --  PC control field causes the modification of the PC for the next clock cycle.
174 --  With these two assumptions, the microprogram starts with an unconditional
175 --  jump to a 2-way jump table which initiates the fetch of an instruction op-
176 --  code from memory, or vectors to the microprogram's interrupt handler. The
177 --  BRV1 instruction is used to capture and decode using ROM/RAM the fetched op-
178 --  code. The opcode is used directly as it is being read from memory to provide
179 --  to address a 256-way branch table in the microprogram ROM, and simultaneous-
180 --  ly a second decode ROM/RAM that provides the fixed portion of each instruc-
181 --  tions operation. That is, the 256-way instruction decoder built into
182 --  the microprogram ROM is the decoder for the variable microprogram, and the
183 --  second ROM is the decoder for the fixed microword. The variable micropro-
184 --  gram implements the control sequences necessary for an instruction from the
185 --  perspective of the addressing mode of the instruction, and the fixed micro-
186 --  program word defines the ALU operation to be performed when all operands are
187 --  available. The fixed microword is applied to the ALU under control of the
188 --  microprogram. Altogether, the number of bits required is 32 for the variable
189 --  microwords, and 24 for the fixed microword, or 56 bits total. (An additional
190 --  8 bits are included in the fixed microword, but they are simply reserved for
191 --  future use should that be required.) For debugging purposes, the opcode is
192 --  also loaded into the Instruction Register (IR), but it's not required.
193 --
194 --  Following the initial word at address 0, there are 31 microwords reserved
195 --  for future use. The intended use of these 31 locations is as a microprogram
196 --  bootloader for the remainder of the microprogram microstore, and for the
197 --  fixed instruction decoder ROM. Thus, at some future date, it may be possible
198 --  to update the microprogram ROMs dynamically from external memory or a serial
199 --  port.
200 --
201 --  The 2-way jump table, which is the target of the first unconditional branch,
202 --  has two locations which are expected to be accessed by a BMW instruction.
203 --  The first location is labeled as _Nxt to signify that it is the fetch cycle
204 --  for the next opcode. The second location is used to initiate the interrupt
205 --  handler in the event that the external INT signal is asserted. An external
206 --  interrupt handler is expected to determine if an NMI or unmasked IRQ inter-
207 --  rupt should be taken. If INT is asserted, then the BRV2 instruction in the
208 --  second location of the jump table will capture the interrupt vector and jump
209 --  to the microprogram's interrupt handler.
210 --
211 --  Following the jump table are microroutines for handling specific instruc-
212 --  tions, or for handling specific addressing modes. The most significant 256
213 --  locations in the microprogram ROM/RAM constitute the initial microstate for
214 --  each of the 256 possible instruction opcodes. In the present implementation
215 --  only 177/178 of these instructions represent valid instructions. The remain-
216 --  der are executed as NOPs, and are reserved for future use. (The Rockwell
217 --  extensions use an additonal 32 of the opcodes, leaving 46 opcodes undefined.
218 --  Western Design Center uses all 256 of the opcodes to implement the 16-bit
219 --  W65C816 processor, which also provides an emulation of the 8-bit W65C02.)
220 --
221 --  A second 2-way jump table is included specifically for the RMW instructions.
222 --  The purpose of the microword with the BRV1 instruction is to complete the
223 --  execution of the current instruction, and simultaneously to fetch and decode
224 --  the next instruction. For most instructions, the ALU operation is performed
225 --  during in a terminal microstate with a BRV1 instruction which has Done and
226 --  Reg_WE asserted. For RMW instructions, the ALU operation initiated by the
227 --  Reg_WE control field occurs before the write back to memory of the computed
228 --  result. To use a BMW instruction to jump to the same 2-way jump table used
229 --  for RO or WO multicycle instructions may result in the M65C02 registers,
230 --  including the PSW, being written twice during a cycle. To avoid this issue,
231 --  a second BRV1, BRV2 2-way jump accessed by a BMW instruction is used for the
232 --  RMW instructions. The BRV1 microstate in the RMW jump table does not assert
233 --  Reg_WE. This allows a RMW instruction to complete in the same manner as
234 --  other multicycle instructions, and prevents any of the registers from being
235 --  written more than once per instruction cycle.
236 --
237 --  In implementing the microroutines, no attempt was made to combine the micro-
```

```
238 --  routines for various addressing modes such as Pre-Indexed and Post-Indexed
239 --  Data Page or Absolute that yield the same result. Therefore, the bit in the
240 --  fixed microword which previously identified the index register used by a
241 --  specific opcode has been reused for other purposes. The result of this opti-
242 --  mization is that all of the indexed addressing modes require separate micro-
243 --  routines for correct implementation. All that being the case, the implemen-
244 --  tation of the M65C02 microprogram uses only 256 microwords for instruction
245 --  decode, and an additional 79 microwords to implement the complete micropro-
246 --  gram. Therefore, there remain 177 microwords with which to implement addi-
247 --  tional instructions or capabilities such as bootloading the microprogram
248 --  memory. The current implementation uses approximately 1.88 microwords per
249 --  instruction, and this includes the 78 unimplemented/unused opcodes which the
250 --  M65C02 implements as NOPs. If those opcodes are not included, then a total
251 --  of 257 states, 178 + 79, are used to implement the M65C02 microprogram, or
252 --  1.444 microwords per instruction. The number of microstates per instruction
253 --  are a good measure of the efficiency of the implementation. For virtually
254 --  all instructions, the M65C02, due to its pipelined implementation, saves at
255 --  least one cycle per instruction when compared to the W65C02, R65C02, or the
256 --  original MOS6502 implementations.
257 --
258 --  The test program used for diagnostics and proofing of the implementation is
259 --  averaging 1.88 clock cycles per instruction. This is an improvement of more
260 --  than 40% over a standard implementation of the 6502 instruction set.
261 --
262 ----------------------------------------------------------------------------
263 ----------------------------------------------------------------------------
264 --  Forked from M65C02_uPgm_V3
265 ----------------------------------------------------------------------------
266 ----------------------------------------------------------------------------
267 --
268 --  No changes were implemented to the structure of the microwords in the basic
269 --  microprogram to accomodated the new M65C02_MPCv3 microprogram controller.
270 --  Instead, external changes to the M65C02_Core module allowed several of the
271 --  original fields to be redefined. In redefining the fields, the widths were
272 --  not changed and neither were their location in the microword. The single bit
273 --  Wait field, bit 18, was redefined as the ZP field. The single bit SC and
274 --  Done fields, bit 17 and 16, respectively, were redefined as the uLen[1:0]
275 --  field. In the previous implementation, SC and Done were used as a two bit
276 --  field, so replacing them with the uLen field is direct drop-in replacement.
277 --
278 --  Version three of the MPC uses dynamic microcycle length control to stretch
279 --  the microcycle of the M65C02 core. Microcycle length extension is required
280 --  when a BCD mode ADC/SBC is performed. Microcycle length extension may also
281 --  be required to access various types of memory.
282 --
283 --  The new MemTyp[1:0] field is expected to be used to let external logic know
284 --  the type of memory cycle that is being performed: program memory fetch, zero
285 --  page access, stack page access, or data memory access. The intent is for the
286 --  microprogram to let a memory controller external to the core know what the
287 --  next memory cycle will so that the memory controller can select between LUT
288 --  RAM (zero page), block RAM (stack page and internal program/data memory), or
289 --  external memory. Alternatively, this field can be reconfigured to directly
290 --  control the length of the next microcycle.
291 --
292 --  In general, it is expected that the MemTyp[1:0] field will directly connect
293 --  to a memory controller external to the MPC. That memory controller will set
294 --  the uLen[1:0] input of the core to control the microcycle controller of the
295 --  MPC. The current architecture of the address generator in the core makes it
296 --  difficult to use AO to determine whether the memory is internal or external.
297 --  However, the microprogram does "know" what memory type is being addressed
298 --  next. Thus, the microprogram can inform an external memory controller
299 --  whether page 0, page 1, or program/data memory is to be accessed next. Since
300 --  this data is only available to the microprogram, the MemTyp field can be
301 --  used by an memory controller outside of the core to implement address detec-
302 --  tion in a later state of the memory access cycle to dynamically change from
303 --  1 cycle distributed RAM (page 0), to 2 cycle block RAM memory with 0 wait
304 --  states (page 1 and internal program/data memory), or 4 cycle memory with
305 --  external wait state insertion (external program/data memory).
306 --
307 ----------------------------------------------------------------------------
308 ----------------------------------------------------------------------------
309 --
310 ----------------------------------------------------------------------------
311 -- F9408A Instruction definitions
312 ----------------------------------------------------------------------------
313
314 RTS         .asm    0       -- Return from Subroutine
315 BSR         .asm    1       -- Branch to subroutine
316 FTCH        .asm    2       -- Fetch next instruction
```

```
317 BMW         .asm    3       -- Branch multi-way
318 BRV0        .asm    4       -- Branch via 0
319 BRV1        .asm    5       -- Branch via 1
320 BRV2        .asm    6       -- Branch via 2
321 BRV3        .asm    7       -- Branch via 3
322 BTH0        .asm    8       -- Branch if T0 is high
323 BTH1        .asm    9       -- Branch if T1 is high
324 BTH2        .asm    10      -- Branch if T2 is high
325 BTH3        .asm    11      -- Branch if T3 is high
326 BTL0        .asm    12      -- Branch if T0 is low
327 BTL1        .asm    13      -- Branch if T1 is low
328 BTL2        .asm    14      -- Branch if T2 is low
329 BMW3        .asm    15      -- Branch multi-way on T[3]
330
331 ---------------------------------------------------------------------------
332 -- ROM ( output ) Field definitions
333 ---------------------------------------------------------------------------
334
335 Inst        .def    4       -- Instruction
336 BA          .def    9       -- Branch Address
337 uP_Cntl     .def    2       -- Microprogram Control
338 NA_Cntl     .def    11      -- Next Address Control Field
339 IO_Cntl     .def    2       -- I/O Cycle Control Field
340 DIO_Cntl    .def    4       -- Data Input/Output Demux/Mux Control Field
341 RegWE_Cntl  .def    3       -- Register Write Enable (A, X, Y, S, P)
342 PSW_Cntl    .def    1       -- Asserted to Clear D and Set I in PSW
343
344 ---------------------------------------------------------------------------
345 -- Constant definitions
346 ---------------------------------------------------------------------------
347
348 --  Next Address Control Definitions
349
350 --                          LO PSZAM XYR C
351 --                          Pf CtPbA   e i
352 --                          Cf  k sR   l
353 ---------------------------------------------------------------------------
354 Vec     .equ    1056    -- 10_00010_000_0; Vec:  NA <= {OP2,OP1}             + 0
355 Jmp     .equ    1056    -- 10_00010_000_0; Jmp:  NA <= {OP2,OP1}             + 0
356 JmpY    .equ    1060    -- 10_00010_010_0; JmpY: NA <= {OP1,OP1} + {  0,  Y} + 0
357 Rtn     .equ    1057    -- 10_00010_000_1; Rtn:  NA <= {OP2,OP1}             + 1
358 ---------------------------------------------------------------------------
359 PC      .equ    1280    -- 10_10000_000_0; PC:   NA <= PC                    + 0
360 Inc     .equ    1281    -- 10_10000_000_1; Inc:  NA <= PC                    + 1
361 ---------------------------------------------------------------------------
362 Bra     .equ    1283    -- 10_10000_001_1; Bra:  NA <= PC        + {OP2,OP1} + 1
363 Rel     .equ     259    -- 00_10000_001_1; Rel:  NA <= PC        + {OP2,OP1} + 1
364 ---------------------------------------------------------------------------
365 Psh     .equ     128    -- 00_01000_000_0; Psh:  NA <= {  1, SP}             + 0
366 Pop     .equ     129    -- 00_01000_000_1; Pop:  NA <= {  1, SP}             + 1
367 ---------------------------------------------------------------------------
368 SPN     .equ     641    -- 01_01000_000_1; Stk:  NA <= {  1, SP} + {  0,OP1} + 1
369 ---------------------------------------------------------------------------
370 DPN     .equ      64    -- 00_00100_000_0; DPN:  NA <= {  0,OP1}             + 0
371 DPX     .equ      72    -- 00_00100_100_0; DPX:  NA <= {  0,OP1} + {  0,  X} + 0
372 DPY     .equ      68    -- 00_00100_010_0; DPY:  NA <= {  0,OP1} + {  0,  Y} + 0
373 ---------------------------------------------------------------------------
374 LDA     .equ      32    -- 00_00010_000_0; LDA:  NA <= {OP2,OP1}             + 0
375 LDAX    .equ      40    -- 00_00010_100_0; LDAX: NA <= {OP2,OP1} + {  0,  X} + 0
376 LDAY    .equ      36    -- 00_00010_010_0; LDAY: NA <= {OP2,OP1} + {  0,  Y} + 0
377 ---------------------------------------------------------------------------
378 MAR     .equ      16    -- 00_00001_000_0; MAR:  NA <= MAR                   + 0
379 Nxt     .equ      17    -- 00_00001_000_1; Nxt:  NA <= MAR                   + 1
380 ---------------------------------------------------------------------------
381
382 --  uP_Cntl Definitions
383
384 BA          .equ    1       -- uPgm control of ALU, etc. using BA field constant
385 ZP          .equ    2       -- Force % 256 addressing in Page 0
386 SP          .equ    3       -- Force % 256 addressing in Page 1
387
388 --  Bus Interface Unit Definitions
389
390 WR          .equ    1       -- Bus Operand Write
391 RD          .equ    2       -- Bus Operand Read
392 IF          .equ    2       -- Bus Insruction Fetch
393
394 --  Memory Data Input Demultiplexer Definitions
395
```

```
396 SGN         .equ    7       -- OP1 <= DI; OP2 <= sign(DI)
397 OP2         .equ    2       -- OP2 <= DI
398 OP1         .equ    4       -- OP1 <= DI
399 IR          .equ    8       -- IR  <= DI
400
401 --  Memory Output Data Multiplexer Definitions
402 --      Output Data Multiplexer provides a means by which 16-bit operands may be
403 --      pushed/written to memory. Currently, three 16-bit registers are tied to
404 --      the Output Data Multiplexer: PC, MAR, and {OP2, OP1}. {PCH, PCL} are
405 --      the default 8-bit components that are output through this multiplexer.
406 --      Whenever the instruction mode is set to PHW, the {OP2, OP1} register
407 --      pair are tied to the multiplexer. When the instruction mode is PHR, the
408 --      MAR is tied to the multiplexer.
409
410 ALU         .equ    1       -- DO  <= Out
411 DMH         .equ    2       -- DO  <= {OP2 | MAR[15:8] | PC[15:8]}
412 DML         .equ    4       -- DO  <= {OP1 | MAR[ 7:0] | PC[ 7:0]}
413 PSW         .equ    8       -- DO  <= PSW (P)
414
415 --  Register Write Enable Control Field Definitions
416
417 WE_A        .equ    1       -- Write Enable A
418 WE_X        .equ    2       -- Write Enable X
419 WE_Y        .equ    3       -- Write Enable Y
420 WE_R        .equ    4       -- Write Enable Register - write selected register
421 WE_S        .equ    5       -- Write Enable S
422 WE_P        .equ    6       -- Write Enable P
423 WE_M        .equ    7       -- Write Enable M(emory)
424
425 --  Miscellaneous Control Field Definitions
426
427 ISR         .equ    1       -- Assert ISR: Clear D, Set I
428
429 ------------------------------------------------------------------------------
430 --
431 --  Microprogram Controller Resources
432 --
433 --  T[0]    -   Valid - ALU Operation Complete/Done
434 --  T[1]    -   Unused
435 --  T[2]    -   Unused
436 --  T[3]    -   Unused
437 --
438 --  Via[0]  -   BA, but also waits for the completion of a memory or ALU cycle
439 --  Via[1]  -   Instruction Decoder, effectively functions as a 256 way branch
440 --  Via[2]  -   Samples Vector and loads it into {OP2, OP1}
441 --  Via[3]  -   Instruction Decoder with branch to Interrupt Handler, _Int
442 --
443 --  MW[2:0] -   MW[2] - uP_BA[2]; MW[1] - uP_BA[1]; MW[0] - Int;
444 --
445 --   xx0    -   Instruction Fetch
446 --   xx1    -   Interrupt Trap
447 --
448 ------------------------------------------------------------------------------
449 -- M65C02A Microprogram Start
450 ------------------------------------------------------------------------------
451
452 --  I   BA, NA, IO, DI, Reg_WE, ISR
453
454 _Start: .org    0
455     BRV2    _Rst                                -- {OP2, OP1}<={RST}
456 _Rst:
457     FTCH    $,SP, Psh, WR, DMH                   -- NA<={1,SP}; DO<=PCH; SP--
458     FTCH    $,SP, Psh, WR, DML                   -- NA<={1,SP}; DO<=PCL; SP--
459     FTCH    $,SP, Psh, WR, PSW,, ISR             -- NA<={1,SP}; DO<=PSW; SP--
460 --
461     FTCH    $,, LDA, RD, OP1                    -- NA<={OP2,OP1}; OP1<=Lo(RST)
462     FTCH    $,, Nxt, RD, OP2,, ISR              -- NA<=MAR+1;     OP2<=Hi(RST)
463 --
464     BRV1    $,, Jmp, IF, IR                     -- PC<={OP2,OP1}; NA<={OP2,OP1}
465
466 ------------------------------------------------------------------------------
467
468 ------------------------------------------------------------------------------
469 --
470 --  this space reserved for future use - boot loader for the microprogram ROMs
471 --
472 ------------------------------------------------------------------------------
473
474 ------------------------------------------------------------------------------
```

```
475 --  WAI - Wait for Interrupt
476 -------------------------------------------------------------------------------
477
478 _WAI:       .org    28          -- Set up 4-way table for WAI instruction
479     BMW     _WAI,, PC           -- No external interrupts asserted
480     BRV0    _Int,, PC           -- Int asserted by NMI, do NMI interrupt
481     BRV0    _Nxt,, PC           -- xIRQ asserted with IRQ_Msk asserted, continue
482     BRV0    _Int,, PC           -- Int asserted by xIRQ, do IRQ interrupt
483
484 -------------------------------------------------------------------------------
485
486         .org    32
487
488 -------------------------------------------------------------------------------
489 -- 2-Way Jump Table: _Nxt and _Int
490 -------------------------------------------------------------------------------
491 -------------------------------------------------------------------------------
492 -- Instruction Fetch and Execute Microstate
493 -------------------------------------------------------------------------------
494
495 _Nxt:
496 _Psh:
497 _Pop:
498 _Imm:
499     BRV1    _Nxt,, Inc, IF, IR, WE_R                -- Instruction Fetch/Execute
500
501 -------------------------------------------------------------------------------
502 --  Interrupt Entry - NMI, (unmasked) IRQ (falls through to second state of BRK)
503 -------------------------------------------------------------------------------
504
505 _Int:
506     BRV2    _PshPCL,SP, Psh, WR, DMH, WE_R          -- {OP2,OP1}<=Vector(INT)
507
508 -------------------------------------------------------------------------------
509 --  BRK Entry - BRK #imm (_Int falls through to _PshPCL, see comment above)
510 -------------------------------------------------------------------------------
511
512 _Brk:
513     BRV2    _PshPCL,SP, Psh, WR, DMH                -- Push PCH
514 _PshPCL:
515     FTCH    $,SP, Psh, WR, DML                      -- Push PCL
516     FTCH    $,SP, Psh, WR, PSW,, ISR                -- Push PSW; Clr D, Set I;
517                                                     -- {OP2,OP1}<=Vector(BRK)
518 --
519     FTCH    $,, Vec, RD, OP1                        -- Read Indirect Dst Ptr Lo
520     FTCH    $,, Nxt, RD, OP2,, ISR                  -- Read Indirect Dst Ptr Hi
521 --
522     BRV1    $,, Jmp, IF, IR                         -- Instruction Fetch
523
524 -------------------------------------------------------------------------------
525 --  Branch if CC - Bcc Rel                          (Not interruptable)
526 -------------------------------------------------------------------------------
527
528 _Rel:
529     BRV1    $,, Bra, IF, IR                         -- Instruction Fetch
530
531 -------------------------------------------------------------------------------
532 --  Jump To Subroutine - JSR Abs                    (Not interruptable)
533 -------------------------------------------------------------------------------
534
535 _JSR:
536     FTCH    $,, Inc, RD, OP2                        -- Load Dst Ptr Hi
537     FTCH    $,SP, Psh, WR, DMH                      -- Push PC Hi
538     BMW3    _JSR_End,SP, Psh, WR, DML               -- Push PC Lo
539 _JSR_End:
540     BRV1    $,, Jmp, IF, IR                         -- Instruction Fetch
541
542 -------------------------------------------------------------------------------
543 --  Jump To Subroutine - JSR (Abs)                  (Not interruptable)
544 -------------------------------------------------------------------------------
545
546     FTCH    $,, LDA, RD, OP1                        -- Load Dst Ptr Lo
547     BRV0    _JSR_End,, Nxt, RD, OP2                 -- Load Dst Ptr Hi
548
549 -------------------------------------------------------------------------------
550 --  Return from Interrupt - RTI                     (Not interruptable)
551 -------------------------------------------------------------------------------
552
553 _RTI:
```

```
554     FTCH    $,SP, Pop, RD, OP1, WE_P                -- Pop PCL, PSW <= OP1
555
556 ------------------------------------------------------------------------------
557 --  Return From Subroutine - RTS                     (Not interruptable)
558 ------------------------------------------------------------------------------
559
560 _RTS:
561     FTCH    $,SP, Pop, RD, OP2                      -- Pop PCH
562     BRV1    $,, Rtn, IF, IR                         -- Execute RTS
563
564 ------------------------------------------------------------------------------
565 --  Jump Pre-Indexed Indirect - JMP (Abs, X)        (Not interruptable)
566 ------------------------------------------------------------------------------
567
568 _JmpXI:
569     FTCH    $,, Inc, IF, OP2                        -- Load Dst Ptr Hi
570     BRV0    _Jmp,, LDAX, RD, OP1                    -- Read Dst Ptr Lo
571
572 ------------------------------------------------------------------------------
573 --  Jump Indirect - JMP (Abs)                       (Not interruptable)
574 ------------------------------------------------------------------------------
575
576 _JmpI:
577     FTCH    $,, Inc, IF, OP2                        -- Load Dst Ptr Hi
578     FTCH    $,, LDA, RD, OP1                        -- Read Dst Ptr Lo
579
580 ------------------------------------------------------------------------------
581 --  Jump Absolute - JMP  Abs                        (Not interruptable)
582 ------------------------------------------------------------------------------
583
584 _Jmp:
585     FTCH    $,, Nxt, RD, OP2                        -- Read Dst Ptr Hi
586 --
587     BRV1    $,, Jmp, IF, IR                         -- Next, no Reg_WE, P okay
588
589 ------------------------------------------------------------------------------
590 --  Memory Read-Only Data Page Direct - xxx DP
591 ------------------------------------------------------------------------------
592
593 _RO_DP:
594     BMW     _Nxt,ZP, DPN, RD, OP1                   -- Read DP Mem
595
596 ------------------------------------------------------------------------------
597 --  Memory Read-Only Data Page Indirect - xxx (DP)
598 ------------------------------------------------------------------------------
599
600 _RO_DPI:
601     FTCH    $,ZP, DPN, RD, OP1                      -- Read DP Mem Ptr Lo
602     FTCH    $,ZP, Nxt, RD, OP2                      -- Read DP Mem Ptr Hi
603     BMW     _Nxt,, LDA, RD, OP1                     -- Read Operand
604
605 ------------------------------------------------------------------------------
606 --  Memory Read-Only Pre-Indexed Data Page Direct - xxx DP, X
607 ------------------------------------------------------------------------------
608
609 _RO_DPX:
610     BMW     _Nxt,ZP, DPX, RD, OP1                   -- Read DP Mem
611
612 ------------------------------------------------------------------------------
613 --  Memory Read-Only Pre-Indexed Data Page Indirect - xxx (DP, X)
614 ------------------------------------------------------------------------------
615
616 _RO_DPXI:
617     FTCH    $,ZP, DPX, RD, OP1                      -- Read DP Mem Ptr Lo (DP,X)
618     FTCH    $,ZP, Nxt, RD, OP2                      -- Read DP Mem Ptr Hi
619     BMW     _Nxt,, LDA, RD, OP1                     -- Read Operand
620
621 ------------------------------------------------------------------------------
622 --  Memory Read-Only Post-Indexed Data Page Direct - xxx DP, Y
623 ------------------------------------------------------------------------------
624
625 _RO_DPY:
626     BMW     _Nxt,ZP, DPY, RD, OP1                   -- Read DP Mem
627
628 ------------------------------------------------------------------------------
629 --  Memory Read-Only Post-Indexed Data Page Indirect - xxx (DP), Y
630 ------------------------------------------------------------------------------
631
632 _RO_DPIY:
```

```
633     FTCH    $,ZP, DPN, RD, OP1                      -- Read DP Mem Ptr Lo
634     FTCH    $,ZP, Nxt, RD, OP2                      -- Read DP Mem Ptr Hi
635     BMW     _Nxt,, LDAY, RD, OP1                    -- Read Operand (DP),Y
636
637 -------------------------------------------------------------------------------
638 --  Memory Write-Only Data Page Direct - xxx DP
639 -------------------------------------------------------------------------------
640
641 _WO_DP:
642     BMW     _Nxt,ZP, DPN, WR, ALU, WE_R             -- Write to DP
643
644 -------------------------------------------------------------------------------
645 --  Memory Write-Only Data Page Indirect - xxx (DP)
646 -------------------------------------------------------------------------------
647
648 _WO_DPI:
649     FTCH    $,ZP, DPN, RD, OP1                      -- Read DP Mem Ptr Lo
650     FTCH    $,ZP, Nxt, RD, OP2                      -- Read DP Mem Ptr Hi
651     BMW     _Nxt,, LDA, WR, ALU, WE_R               -- Write to (DP)
652
653 -------------------------------------------------------------------------------
654 --  Memory Write-Only Pre-Indexed Data Page Direct - xxx DP, X
655 -------------------------------------------------------------------------------
656
657 _WO_DPX:
658     BMW    _Nxt,ZP, DPX, WR, ALU, WE_R              -- Write to DP,X
659
660 -------------------------------------------------------------------------------
661 --  Memory Write-Only Data Page Indirect - xxx (DP, X)
662 -------------------------------------------------------------------------------
663
664 _WO_DPXI:
665     FTCH    $,ZP, DPX, RD, OP1                      -- Read DP Mem Ptr Lo
666     FTCH    $,ZP, Nxt, RD, OP2                      -- Read DP Mem Ptr Hi
667     BMW     _Nxt,, LDA, WR, ALU, WE_R               -- Write to (DP)
668
669 -------------------------------------------------------------------------------
670 --  Memory Write-Only Post-Indexed Data Page Direct - xxx DP, Y
671 -------------------------------------------------------------------------------
672
673 _WO_DPY:
674     BMW    _Nxt,ZP, DPY, WR, ALU, WE_R              -- Write to DP,Y
675
676 -------------------------------------------------------------------------------
677 --  Memory Write-Only Post-Indexed Data Page Indirect - xxx (DP), Y
678 -------------------------------------------------------------------------------
679
680 _WO_DPIY:
681     FTCH    $,ZP, DPN, RD, OP1                      -- Read DP Mem Ptr Lo
682     FTCH    $,ZP, Nxt, RD, OP2                      -- Read DP Mem Ptr Hi
683     BMW     _Nxt,, LDAY, WR, ALU, WE_R              -- Write to (DP), Y
684
685 -------------------------------------------------------------------------------
686 --  Memory Read-Only Absolute - xxx Abs
687 -------------------------------------------------------------------------------
688
689 _RO_Abs:
690     FTCH    $,, Inc, IF, OP2                        -- Read Mem Ptr Hi
691     BMW     _Nxt,, LDA, RD, OP1                     -- Read Operand
692
693 -------------------------------------------------------------------------------
694 --  Memory Read-Only Pre-Indexed Absolute - xxx Abs, X
695 -------------------------------------------------------------------------------
696
697 _RO_AbsX:
698     FTCH    $,, Inc, IF, OP2                        -- Read Mem Ptr Hi
699     BMW     _Nxt,, LDAX, RD, OP1                    -- Read Operand Abs,X
700
701 -------------------------------------------------------------------------------
702 --  Memory Read-Only Post-Indexed Absolute - xxx Abs, Y
703 -------------------------------------------------------------------------------
704
705 _RO_AbsY:
706     FTCH    $,, Inc, IF, OP2                        -- Read Mem Ptr Hi
707     BMW     _Nxt,, LDAY, RD, OP1                    -- Read Operand Abs,Y
708
709 -------------------------------------------------------------------------------
710 --  Memory Write-Only Absolute - xxx Abs
711 -------------------------------------------------------------------------------
```

```
712
713 _WO_Abs:
714     FTCH    $,, Inc, IF, OP2                        -- Read Mem Ptr Hi
715     BMW     _Nxt,, LDA, WR, ALU, WE_R               -- Write to Abs
716
717 ------------------------------------------------------------------------------
718 --  Memory Write-Only Pre-Indexed Absolute - xxx Abs, X
719 ------------------------------------------------------------------------------
720
721 _WO_AbsX:
722     FTCH    $,, Inc, IF, OP2                        -- Read Mem Ptr Hi
723     BMW     _Nxt,, LDAX, WR, ALU, WE_R              -- Write to Abs,X
724
725 ------------------------------------------------------------------------------
726 --  Memory Write-Only Post-Indexed Absolute - xxx Abs, Y
727 ------------------------------------------------------------------------------
728
729 _WO_AbsY:
730     FTCH    $,, Inc, IF, OP2                        -- Read Mem Ptr Hi
731     BMW     _Nxt,, LDAY, WR, ALU, WE_R              -- Write to Abs,Y
732
733 ------------------------------------------------------------------------------
734 --  2-way Read-Modify-Write Instruction/Interrupt Jump Table
735 ------------------------------------------------------------------------------
736
737 _RMW:       .org    92
738     BRV1    _RMW,, Inc, IF, IR                      -- Instruction Fetch/Execute
739     BRV2    _Brk,SP, Psh, WR, DMH                   -- Push PCH, capture Vector
740
741 ------------------------------------------------------------------------------
742 --  Memory Read-Modify-Write Data Page Direct - xxx DP
743 ------------------------------------------------------------------------------
744
745 _RMW_DP:
746     FTCH    $,ZP, DPN, RD, OP1                      -- Read from DP
747     BMW     _RMW,, MAR, WR, ALU, WE_R               -- Write to DP
748
749 ------------------------------------------------------------------------------
750 --  Memory Read-Modify-Write Pre-Indexed Data Page Direct - xxx DP, X
751 ------------------------------------------------------------------------------
752
753 _RMW_DPX:
754     FTCH    $,ZP, DPX, RD, OP1                      -- Read from DP,X
755     BMW     _RMW,, MAR, WR, ALU, WE_R               -- Write to DP,X
756
757 ------------------------------------------------------------------------------
758 --  Memory Read-Modify-Write Post-Indexed Data Page Direct - xxx DP, Y
759 ------------------------------------------------------------------------------
760
761 _RMW_DPY:
762     FTCH    $,ZP, DPY, RD, OP1                      -- Read from DP,Y
763     BMW     _RMW,, MAR, WR, ALU, WE_R               -- Write to DP,Y
764
765 ------------------------------------------------------------------------------
766 --  Memory Read-Modify-Write Absolute - xxx Abs
767 ------------------------------------------------------------------------------
768
769 _RMW_Abs:
770     FTCH    $,, Inc, IF, OP2                        -- Read Mem Ptr Hi
771     FTCH    $,, LDA, RD, OP1                        -- Read from Abs
772     BMW     _RMW,, MAR, WR, ALU, WE_R               -- Write to Abs
773
774 ------------------------------------------------------------------------------
775 --  Memory Read-Modify-Write Pre-Indexed Absolute - xxx Abs, X
776 ------------------------------------------------------------------------------
777
778 _RMW_AbsX:
779     FTCH    $,, Inc, IF, OP2                        -- Read Mem Ptr Hi
780     FTCH    $,, LDAX, RD, OP1                       -- Read from Abs,X
781     BMW     _RMW,, MAR, WR, ALU, WE_R               -- Write to Abs,X
782
783 ------------------------------------------------------------------------------
784 --  Memory Read-Modify-Write Post-Indexed Absolute - xxx Abs, Y
785 ------------------------------------------------------------------------------
786
787 _RMW_AbsY:
788     FTCH    $,, Inc, IF, OP2                        -- Read Mem Ptr Hi
789     FTCH    $,, LDAY, RD, OP1                       -- Read from Abs,Y
790     BMW     _RMW,, MAR, WR, ALU, WE_R               -- Write to Abs,Y
```

```
791
792 ------------------------------------------------------------------------------
793 --  Rockwell BBRx/BBSx dp,rel instructions
794 ------------------------------------------------------------------------------
795
796 _BByx_dp_rel:
797     FTCH    $,ZP, DPN, RD, OP1                      -- Read from DP
798     FTCH    $,, Inc, IF, SGN, WE_R                  -- Read rel value
799 --
800     BRV1    $,, Bra, IF, IR                         -- Execute BByx
801
802 ------------------------------------------------------------------------------
803 --  M65C02A Specific instructions
804 ------------------------------------------------------------------------------
805
806 ------------------------------------------------------------------------------
807 --  COP dp Entry - Fetch Signature and fall through _Int (interrupt handler)
808 ------------------------------------------------------------------------------
809
810 _COP_DP:
811     BRV0    _Int,ZP, DPN, RD, OP1                   -- Fetch COP Signature
812
813 ------------------------------------------------------------------------------
814 --  Branch To Subroutine - BSR rel16                (Not interruptable)
815 ------------------------------------------------------------------------------
816
817 _BSR_Rel16:
818     FTCH    $,, Inc, RD, OP2                        -- Load Dst Ptr Hi
819     FTCH    $,SP, Psh, WR, DMH                      -- Push PC Hi
820     FTCH    $,SP, Psh, WR, DML                      -- Push PC Lo
821 --
822     BRV1    $,, Bra, IF, IR                         -- Instruction Fetch
823
824 ------------------------------------------------------------------------------
825 --  Unconditional Branch - BRA rel16                (Not interruptable)
826 ------------------------------------------------------------------------------
827
828 _BRA_Rel16:
829     FTCH    $,, Inc, RD, OP2                        -- Load Dst Ptr Hi
830 --
831     BRV1    $,, Bra, IF, IR                         -- Instruction Fetch
832
833 ------------------------------------------------------------------------------
834 --  Jump To Subroutine - JSR (sp,S),Y               (Not interruptable)
835 ------------------------------------------------------------------------------
836
837 _JSR_SPIY:
838     FTCH    $,SP, SPN, RD, OP1                      -- Load Indirect Dst Ptr Lo
839     FTCH    $,SP, Nxt, RD, OP2                      -- Load Indirect Dst Ptr Hi
840     FTCH    $,SP, Psh, WR, DMH                      -- Push PC Hi
841     FTCH    $,SP, Psh, WR, DML                      -- Push PC Lo
842 --
843     BRV1    $,, JmpY, IF, IR                        -- Instruction Fetch
844
845 ------------------------------------------------------------------------------
846 --  Jump Post-Indexed Stack-Relative Indirect - JMP (sp,S),Y (Not interruptable)
847 ------------------------------------------------------------------------------
848
849 _JMP_SPIY:
850     FTCH    $,SP, SPN, RD, OP1                      -- Load Indirect Dst Ptr Lo
851     FTCH    $,SP, Nxt, RD, OP2                      -- Load Indirect Dst Ptr Hi
852 --
853     BRV1    $,, JmpY, IF, IR                        -- Instruction Fetch
854
855 ------------------------------------------------------------------------------
856 --  Memory Read-Only Stack-relative Direct - xxx sp,S
857 ------------------------------------------------------------------------------
858
859 _RO_SP:
860     BMW     _Nxt,SP, SPN, RD, OP1                   -- Read sp,S Mem
861
862 ------------------------------------------------------------------------------
863 --  Memory Read-Only Post-Indexed Stack-relative Indirect - xxx (sp,S), Y
864 ------------------------------------------------------------------------------
865
866 _RO_SPIY:
867     FTCH    $,SP, SPN, RD, OP1                      -- Read DP Mem Ptr Lo
868     FTCH    $,SP, Nxt, RD, OP2                      -- Read DP Mem Ptr Hi
869     BMW     _Nxt,, LDAY, RD, OP1                    -- Read Operand (sp,S),Y
```

```
870
871 ------------------------------------------------------------------------------
872 --  Memory Write-Only Stack-relative Direct - xxx sp,S
873 ------------------------------------------------------------------------------
874
875 _WO_SP:
876     BMW     _Nxt,SP, SPN, WR, ALU, WE_R             -- Write to sp,S
877
878 ------------------------------------------------------------------------------
879 --  Memory Write-Only Post-Indexed Stack-relative Indirect - xxx (sp,S), Y
880 ------------------------------------------------------------------------------
881
882 _WO_SPIY:
883     FTCH    $,SP, SPN, RD, OP1                      -- Read Mem Ptr Lo
884     FTCH    $,SP, Nxt, RD, OP2                      -- Read Mem Ptr Hi
885     BMW     _Nxt,, LDAY, WR, ALU, WE_R              -- Write to (sp,S), Y
886
887     BRV0    0,                                      -- Fill: Reset
888
889 ------------------------------------------------------------------------------
890 --  Push 16-bit Value - PHW #Imm16; PHW dp; PHW (dp); PHW abs; PHW (abs)
891 ------------------------------------------------------------------------------
892
893 _PHW_Imm:
894     FTCH    _$,, Inc, IF, OP2                       -- Read Hi(imm16)
895
896 _PHW_Exit:
897     FTCH    $,SP, Psh, WR, DMH                      -- Push Hi(imm16) (OP2)
898     BMW     _Nxt,SP, Psh, WR, DML                   -- Push Lo(imm16) (OP1)
899
900 _PHW_DP:
901     BMW3    $+1,ZP, DPN, RD, OP1                    -- Read Operand Lo
902     BRV0    _PHW_Exit,ZP, Nxt, RD, OP2              -- Read Operand Hi
903
904 _PHW_DPI:
905     FTCH    $,ZP, Nxt, RD, OP2                      -- Read Operand Hi
906     FTCH    $,, LDA, RD, OP1                        -- Read Operand Lo Indirect
907     BRV0    _PHW_Exit,, Nxt, RD, OP2                -- Read Operand Hi Indirect
908
909 _PHW_Abs:
910     FTCH    $,, Inc, IF, OP2                        -- Read Data Ptr Hi
911     BMW3    $+1,, LDA, RD, OP1                      -- Read Operand Hi
912     BRV0    _PHW_Exit,, Nxt, RD, OP2                -- Read Operand Lo
913
914 _PHW_AbsI:
915     FTCH    $,, Nxt, RD, OP2                        -- Read Operand Hi
916     FTCH    $,, LDA, RD, OP1                        -- Read Operand Lo Indirect
917     BRV0    _PHW_Exit,, Nxt, RD, OP2                -- Read Operand Hi Indirect
918
919
920 ------------------------------------------------------------------------------
921 --  Push 16-bit PC-Relative Value - PHR Rel16
922 ------------------------------------------------------------------------------
923
924 _PHR_Rel16:
925     FTCH    $,, Inc, IF, OP2                        -- Read Hi(Rel16)
926     FTCH    $,, Rel                                 -- MAR = PC + {OP2, OP1} + 1
927     FTCH    $,SP, Psh, WR, DMH                      -- Push Hi(MAR) (MAR[15:8])
928     BMW     _Nxt,SP, Psh, WR, DML                   -- Push Lo(MAR) (MAR[ 7:0])
929
930
931 ------------------------------------------------------------------------------
932 --  Pull 16-bit Value - PLW dp; PLW (dp)
933 ------------------------------------------------------------------------------
934
935 _PLW_DP:
936     BRV0    _PLW_Exit,ZP, DPN                       -- MAR = {0, OP1}
937
938 _PLW_DPI:
939     FTCH    $,ZP, DPN, RD, OP1
940     BRV0    $+2,ZP, Nxt, RD, OP2
941
942 ------------------------------------------------------------------------------
943 --  Pull 16-bit Value - PLW abs; PLW (abs)
944 ------------------------------------------------------------------------------
945
946 _PLW_Abs:
947     BMW3    $+1,, Inc, IF, OP2                      -- Read Data Ptr Hi
948     BRV0    _PLW_Exit,, LDA                         -- MAR = {OP2, OP1}
```

```
949
950 _PLW_AbsI:
951     FTCH    $,, LDA, RD, OP1
952     BRV0    $-2,, Nxt, RD, OP2
953
954 _PLW_Exit:
955     FTCH    $,SP, Pop, RD, OP1                      -- Pop Operand Lo
956     FTCH    $,ZP, MAR, WR, ALU, WE_R                -- Write Operand Lo
957     FTCH    $,SP, Pop, RD, OP1                      -- Pop Operand Hi
958     BMW     _Nxt,ZP, Nxt, WR, ALU, WE_R             -- Write Operand Hi
959
960 ------------------------------------------------------------------------------
961 --  End of Microprogram Routines for Normal Instructions
962 ------------------------------------------------------------------------------
963
964 _End_uPgm:
965
```

```
 966 _IDEC_Start:    .org    256
 967
 968 ------------------------------------------------------------------------------
 969 --  Start of Instruction Decode Table (Entry for each Opcode)
 970 --
 971 --  Instead of being organized in numerical order, the table is organized by
 972 --  rows: the least significant nibble and the most significant nibble of the
 973 --  opcode are swapped. This organization more clearly shows the arrangement of
 974 --  the addressing modes of the WDC W65C02 microprocessor being emulated. It al-
 975 --  so more clearly shows the regularity of the ALU instructions that are imple-
 976 --  mented. The implementation of the microprogram is first based on the addres-
 977 --  sing mode, and then on the ALU function. Single cycle instructions will be
 978 --  easily recognized in the following table because their table entry use the
 979 --  BRV3 instruction. Multi-cycle instructions use the BRV0 instruction to vec-
 980 --  tor a microroutine in the lower 256 words of the microprogram ROM/RAM.
 981 --
 982 --  Single byte instructions such as BRK, RTS, RTI, and register push/pull in-
 983 --  structions (PHA, PLA, PHP, PLP, PHX, PLX, PHY, PLY), and multi-byte instruc-
 984 --  tions like JSR abs are implemented with special microroutines that perform
 985 --  the necessary stack accesses. The remainder of the microroutines are orga-
 986 --  nized by addressing mode, and whether the addressing mode is used in a RO,
 987 --  WO, or RMW manner.
 988 --
 989 --  Microprogram Word Format:
 990 --
 991 --  I   BA, uP, NA, IO, DI, Reg_WE, ISR
 992 --
 993 ------------------------------------------------------------------------------
 994
 995 ------------------------------------------------------------------------------
 996 --  Row 0 : 0x00-0xF0 (All Bcc/JMP/JSR/RTS/RTI implemented as uninterruptable)
 997 --  I   BA, uP, NA, IO, DI, Reg_WE, ISR
 998 ------------------------------------------------------------------------------
 999 _BRK_imm:
1000     BRV0    _Brk,, Inc, IF, OP1                     -- Start Break Handler
1001 _BPL_rel:
1002     BRV0    _Rel,, Inc, IF, SGN                     -- Read rel Value
1003 _JSR_abs:
1004     BRV0    _JSR,, Inc, IF, OP1                     -- Read Dst Ptr Lo
1005 _BMI_rel:
1006     BRV0    _Rel,, Inc, IF, SGN                     -- Read rel Value
1007 _RTI_imp:
1008     BRV0    _RTI,SP, Pop, RD, OP1                   -- Read PSW from Stack
1009 _BVC_rel:
1010     BRV0    _Rel,, Inc, IF, SGN                     -- Read rel Value
1011 _RTS_imp:
1012     BRV0    _RTS,SP, Pop, RD, OP1                   -- Read PCL from Stack
1013 _BVS_rel:
1014     BRV0    _Rel,, Inc, IF, SGN                     -- Read rel Value
1015 _BRA_rel:
1016     BRV0    _Rel,, Inc, IF, SGN                     -- Read rel Value
1017 _BCC_rel
1018     BRV0    _Rel,, Inc, IF, SGN                     -- Read rel Value
1019 _LDY_imm:
1020     BMW     _Imm,, Inc, IF, OP1                     -- Read #imm Value
1021 _BCS_rel:
1022     BRV0    _Rel,, Inc, IF, SGN                     -- Read rel Value
1023 _CPY_imm:
1024     BMW     _Imm,, Inc, IF, OP1                     -- Read #imm Value
1025 _BNE_rel:
1026     BRV0    _Rel,, Inc, IF, SGN                     -- Read rel Value
1027 _CPX_imm:
1028     BMW     _Imm,, Inc, IF, OP1                     -- Read #imm Value
1029 _BEQ_rel:
1030     BRV0    _Rel,, Inc, IF, SGN                     -- Read rel Value
1031 ------------------------------------------------------------------------------
1032 -- Row 1 : 0x01-0xF1
1033 --  I   BA, uP, NA, IO, DI, Reg_WE, ISR
1034 ------------------------------------------------------------------------------
1035 _ORA_dpXi:
1036     BRV0    _RO_DPXI,, Inc, IF, OP1                 -- Read DP Ptr
1037 _ORA_dpiY:
1038     BRV0    _RO_DPIY,, Inc, IF, OP1                 -- Read DP Ptr
1039 _AND_dpXi:
1040     BRV0    _RO_DPXI,, Inc, IF, OP1                 -- Read DP Ptr
1041 _AND_dpiY:
1042     BRV0    _RO_DPIY,, Inc, IF, OP1                 -- Read DP Ptr
1043 _EOR_dpXi:
1044     BRV0    _RO_DPXI,, Inc, IF, OP1                 -- Read DP Ptr
```

```
1045 _EOR_dpiY:
1046     BRV0    _RO_DPIY,, Inc, IF, OP1                -- Read DP Ptr
1047 _ADC_dpXi:
1048     BRV0    _RO_DPXI,, Inc, IF, OP1                -- Read DP Ptr
1049 _ADC_dpiY:
1050     BRV0    _RO_DPIY,, Inc, IF, OP1                -- Read DP Ptr
1051 _STA_dpXi:
1052     BRV0    _WO_DPXI,, Inc, IF, OP1                -- Read DP Ptr
1053 _STA_dpiY:
1054     BRV0    _WO_DPIY,, Inc, IF, OP1                -- Read DP Ptr
1055 _LDA_dpXi:
1056     BRV0    _RO_DPXI,, Inc, IF, OP1                -- Read DP Ptr
1057 _LDA_dpiY:
1058     BRV0    _RO_DPIY,, Inc, IF, OP1                -- Read DP Ptr
1059 _CMP_dpXi:
1060     BRV0    _RO_DPXI,, Inc, IF, OP1                -- Read DP Ptr
1061 _CMP_dpiY:
1062     BRV0    _RO_DPIY,, Inc, IF, OP1                -- Read DP Ptr
1063 _SBC_dpXi:
1064     BRV0    _RO_DPXI,, Inc, IF, OP1                -- Read DP Ptr
1065 _SBC_dpiY:
1066     BRV0    _RO_DPIY,, Inc, IF, OP1                -- Read DP Ptr
1067 ------------------------------------------------------------------------------
1068 -- Row 2 : 0x02-0xF2
1069 --  I   BA, uP, NA, IO, DI, Reg_WE, ISR
1070 ------------------------------------------------------------------------------
1071 _COP_dp:
1072     BRV0    _COP_DP,, Inc, IF, OP1                 -- Read DP
1073 _ORA_dpi:
1074     BRV0    _RO_DPI,, Inc, IF, OP1                 -- Read DP
1075 _JSR_22:
1076     BRV0    _JSR_SPIY,, Inc, IF, OP1               -- Read Stk Offset
1077 _AND_dpi:
1078     BRV0    _RO_DPI,, Inc, IF, OP1                 -- Read DP
1079 _COP_imm:
1080     BRV0    _Int,, Inc, IF, OP1                    -- Read #imm
1081 _EOR_dpi:
1082     BRV0    _RO_DPI,, Inc, IF, OP1                 -- Read DP
1083 _PHR_rel16:
1084     BRV0    _PHR_Rel16,, Inc, IF, OP1              -- Read low(rel16)
1085 _ADC_dpi:
1086     BRV0    _RO_DPI,, Inc, IF, OP1                 -- Read DP
1087 _JMP_spiY:
1088     BRV0    _JMP_SPIY,, Inc, IF, OP1               -- Read Stk Offset
1089 _STA_dpi:
1090     BRV0    _WO_DPI,, Inc, IF, OP1                 -- Read DP
1091 _LDX_imm:
1092     BMW     _Imm,, Inc, IF, OP1                    -- Read #imm Value
1093 _LDA_dpi:
1094     BRV0    _RO_DPI,, Inc, IF, OP1                 -- Read DP
1095 _PLW_dp:
1096     BMW3    _PLW_DP,, Inc, IF, OP1                 -- Read DP
1097 _CMP_dpi:
1098     BRV0    _RO_DPI,, Inc, IF, OP1                 -- Read DP
1099 _PLW_abs:
1100     BRV0    _PLW_Abs,, Inc, IF, OP1                -- Read low(abs)
1101 _SBC_dpi:
1102     BRV0    _RO_DPI,, Inc, IF, OP1                 -- Read DP
1103 ------------------------------------------------------------------------------
1104 -- Row 3 : 0x03-0xF3
1105 --  I   BA, uP, NA, IO, DI, Reg_WE, ISR
1106 ------------------------------------------------------------------------------
1107 _ORA_sp:
1108     BRV0    _RO_SP,, Inc, IF, OP1                  -- Read Stk Offset
1109 _ORA_spiY:
1110     BRV0    _RO_SPIY,, Inc, IF, OP1                -- Read Stk Offset
1111 _AND_sp:
1112     BRV0    _RO_SP,, Inc, IF, OP1                  -- Read Stk Offset
1113 _AND_spiY:
1114     BRV0    _RO_SPIY,, Inc, IF, OP1                -- Read Stk Offset
1115 _EOR_sp:
1116     BRV0    _RO_SP,, Inc, IF, OP1                  -- Read Stk Offset
1117 _EOR_spiY:
1118     BRV0    _RO_SPIY,, Inc, IF, OP1                -- Read Stk Offset
1119 _ADC_sp:
1120     BRV0    _RO_SP,, Inc, IF, OP1                  -- Read Stk Offset
1121 _ADC_spiY:
1122     BRV0    _RO_SPIY,, Inc, IF, OP1                -- Read Stk Offset
1123 _STA_sp:
```

```
1124     BRV0    _WO_SP,, Inc, IF, OP1                  -- Read Stk Offset
1125 _STA_spiY:
1126     BRV0    _WO_SPIY,, Inc, IF, OP1                -- Read Stk Offset
1127 _LDA_sp:
1128     BRV0    _RO_SP,, Inc, IF, OP1                  -- Read Stk Offset
1129 _LDA_spiY:
1130     BRV0    _RO_SPIY,, Inc, IF, OP1                -- Read Stk Offset
1131 _CMP_sp:
1132     BRV0    _RO_SP,, Inc, IF, OP1                  -- Read Stk Offset
1133 _CMP_spiY:
1134     BRV0    _RO_SPIY,, Inc, IF, OP1                -- Read Stk Offset
1135 _SBC_sp:
1136     BRV0    _RO_SP,, Inc, IF, OP1                  -- Read Stk Offset
1137 _SBC_spiY:
1138     BRV0    _RO_SPIY,, Inc, IF, OP1                -- Read Stk Offset
1139 ------------------------------------------------------------------------------
1140 -- Row 4 : 0x04-0xF4
1141 --  I   BA, uP, NA, IO, DI, Reg_WE, ISR
1142 ------------------------------------------------------------------------------
1143 _TSB_dp:
1144     BMW3    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1145 _TRB_dp:
1146     BMW3    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1147 _BIT_dp:
1148     BMW3    _RO_DP,, Inc, IF, OP1                  -- Read ZP Pointer
1149 _BIT_dpX:
1150     BMW3    _RO_DPX,, Inc, IF, OP1                 -- Read ZP Pointer
1151 _NOP_44:
1152     BRV3    $,, Inc, IF, IR                        -- Skip Invalid Instruction
1153 _NOP_54:
1154     BRV3    $,, Inc, IF, IR                        -- Skip Invalid Instruction
1155 _STZ_dp:
1156     BMW3    _WO_DP,, Inc, IF, OP1                  -- Read ZP Pointer
1157 _STZ_dpX:
1158     BMW3    _WO_DPX,, Inc, IF, OP1                 -- Read ZP Pointer
1159 _STY_dp:
1160     BMW3    _WO_DP,, Inc, IF, OP1                  -- Read ZP Pointer
1161 _STY_dpX:
1162     BMW3    _WO_DPX,, Inc, IF, OP1                 -- Read ZP Pointer
1163 _LDY_dp:
1164     BMW3    _RO_DP,, Inc, IF, OP1                  -- Read ZP Pointer
1165 _LDY_dpX:
1166     BMW3    _RO_DPX,, Inc, IF, OP1                 -- Read ZP Pointer
1167 _CPY_dp:
1168     BMW3    _RO_DP,, Inc, IF, OP1                  -- Read ZP Pointer
1169 _PHW_dp:
1170     BMW3    _PHW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1171 _CPX_dp:
1172     BMW3    _RO_DP,, Inc, IF, OP1                  -- Read ZP Pointer
1173 _PHW_imm:
1174     BRV0    _PHW_Imm,, Inc, IF, OP1                -- Read low(imm16)
1175 ------------------------------------------------------------------------------
1176 -- Row 5 : 0x05-0xF5
1177 --  I   BA, uP, NA, IO, DI, Reg_WE, ISR
1178 ------------------------------------------------------------------------------
1179 _ORA_dp:
1180     BRV0    _RO_DP,, Inc, IF, OP1                  -- Read ZP Pointer
1181 _ORA_dpX:
1182     BRV0    _RO_DPX,, Inc, IF, OP1                 -- Read ZP Pointer
1183 _AND_dp:
1184     BRV0    _RO_DP,, Inc, IF, OP1                  -- Read ZP Pointer
1185 _AND_dpX:
1186     BRV0    _RO_DPX,, Inc, IF, OP1                 -- Read ZP Pointer
1187 _EOR_dp:
1188     BRV0    _RO_DP,, Inc, IF, OP1                  -- Read ZP Pointer
1189 _EOR_dpX:
1190     BRV0    _RO_DPX,, Inc, IF, OP1                 -- Read ZP Pointer
1191 _ADC_dp:
1192     BRV0    _RO_DP,, Inc, IF, OP1                  -- Read ZP Pointer
1193 _ADC_dpX:
1194     BRV0    _RO_DPX,, Inc, IF, OP1                 -- Read ZP Pointer
1195 _STA_dp:
1196     BRV0    _WO_DP,, Inc, IF, OP1                  -- Read ZP Pointer
1197 _STA_dpX:
1198     BRV0    _WO_DPX,, Inc, IF, OP1                 -- Read ZP Pointer
1199 _LDA_dp:
1200     BRV0    _RO_DP,, Inc, IF, OP1                  -- Read ZP Pointer
1201 _LDA_dpX:
1202     BRV0    _RO_DPX,, Inc, IF, OP1                 -- Read ZP Pointer
```

```
1203 _CMP_dp:
1204     BRV0    _RO_DP,, Inc, IF, OP1                  -- Read ZP Pointer
1205 _CMP_dpX:
1206     BRV0    _RO_DPX,, Inc, IF, OP1                 -- Read ZP Pointer
1207 _SBC_dp:
1208     BRV0    _RO_DP,, Inc, IF, OP1                  -- Read ZP Pointer
1209 _SBC_dpX:
1210     BRV0    _RO_DPX,, Inc, IF, OP1                 -- Read ZP Pointer
1211 ------------------------------------------------------------------------------
1212 -- Row 6 : 0x06-0xF6
1213 --  I   BA, uP, NA, IO, DI, Reg_WE, ISR
1214 ------------------------------------------------------------------------------
1215 _ASL_dp:
1216     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1217 _ASL_dpX:
1218     BRV0    _RMW_DPX,, Inc, IF, OP1                -- Read ZP Pointer
1219 _ROL_dp:
1220     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1221 _ROL_dpX:
1222     BRV0    _RMW_DPX,, Inc, IF, OP1                -- Read ZP Pointer
1223 _LSR_dp:
1224     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1225 _LSR_dpX:
1226     BRV0    _RMW_DPX,, Inc, IF, OP1                -- Read ZP Pointer
1227 _ROR_dp:
1228     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1229 _ROR_dpX:
1230     BRV0    _RMW_DPX,, Inc, IF, OP1                -- Read ZP Pointer
1231 _STX_dp:
1232     BMW3    _WO_DP,, Inc, IF, OP1                  -- Read ZP Pointer
1233 _STX_dpY:
1234     BMW3    _WO_DPY,, Inc, IF, OP1                 -- Read ZP Pointer
1235 _LDX_dp:
1236     BMW3    _RO_DP,, Inc, IF, OP1                  -- Read ZP Pointer
1237 _LDX_dpY:
1238     BMW3    _RO_DPY,, Inc, IF, OP1                 -- Read ZP Pointer
1239 _DEC_dp:
1240     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1241 _DEC_dpX:
1242     BRV0    _RMW_DPX,, Inc, IF, OP1                -- Read ZP Pointer
1243 _INC_dp:
1244     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1245 _INC_dpX:
1246     BRV0    _RMW_DPX,, Inc, IF, OP1                -- Read ZP Pointer
1247 ------------------------------------------------------------------------------
1248 -- Row 7 : 0x07-0xF7 (Rockwell Instructions: RMBx/SMBx dp)
1249 --  I   BA, uP, NA, IO, DI, Reg_WE, ISR
1250 ------------------------------------------------------------------------------
1251 _RMB0_dp:
1252     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1253 _RMB1_dp:
1254     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1255 _RMB2_dp:
1256     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1257 _RMB3_dp:
1258     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1259 _RMB4_dp:
1260     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1261 _RMB5_dp:
1262     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1263 _RMB6_dp:
1264     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1265 _RMB7_dp:
1266     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1267 _SMB0_dp:
1268     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1269 _SMB1_dp:
1270     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1271 _SMB2_dp:
1272     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1273 _SMB3_dp:
1274     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1275 _SMB4_dp:
1276     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1277 _SMB5_dp:
1278     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1279 _SMB6_dp:
1280     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1281 _SMB7_dp:
```

```
1282     BRV0    _RMW_DP,, Inc, IF, OP1                 -- Read ZP Pointer
1283 ------------------------------------------------------------------------------
1284 -- Row 8 : 0x08-0xF8
1285 --  I   BA, uP, NA, IO, DI, Reg_WE, ISR
1286 ------------------------------------------------------------------------------
1287 _PHP:
1288     BMW     _Psh,SP, Psh, WR, ALU, WE_R            -- Push P
1289 _CLC:
1290     BRV3    $,, Inc, IF, IR, WE_P                  -- Clear Carry Flag
1291 _PLP:
1292     BMW     _Pop,SP, Pop, RD, OP1                  -- Pop P
1293 _SEC:
1294     BRV3    $,, Inc, IF, IR, WE_P                  -- Set Carry Flag
1295 _PHA:
1296     BMW     _Psh,SP, Psh, WR, ALU, WE_R            -- Push A
1297 _CLI:
1298     BRV1    $,, Inc, IF, IR, WE_P                  -- Clear Interrupt Mask Flg
1299 _PLA:
1300     BMW     _Pop,SP, Pop, RD, OP1                  -- Pop A
1301 _SEI:
1302     BRV1    $,, Inc, IF, IR, WE_P                  -- Set Interrupt Mask Flag
1303 _DEY:
1304     BRV3    $,, Inc, IF, IR, WE_Y                  -- Decrement Y
1305 _TYA:
1306     BRV3    $,, Inc, IF, IR, WE_A                  -- Transfer Y to A
1307 _TAY:
1308     BRV3    $,, Inc, IF, IR, WE_Y                  -- Transfer A to Y
1309 _CLV:
1310     BRV3    $,, Inc, IF, IR, WE_P                  -- Clear oVerflow Flag
1311 _INY:
1312     BRV3    $,, Inc, IF, IR, WE_Y                  -- Increment Y
1313 _CLD:
1314     BRV3    $,, Inc, IF, IR, WE_P                  -- Clear Decimal Mode Flag
1315 _INX:
1316     BRV3    $,, Inc, IF, IR, WE_X                  -- Increment X
1317 _SED:
1318     BRV3    $,, Inc, IF, IR, WE_P                  -- Set Decimal Mode Flag
1319 ------------------------------------------------------------------------------
1320 -- Row 9 : 0x09-0xF9
1321 --  I   BA, uP, NA, IO, DI, Reg_WE, ISR
1322 ------------------------------------------------------------------------------
1323 _ORA_imm:
1324     BMW     _Imm,, Inc, IF, OP1                    -- Read Immediate Operand
1325 _ORA_absY:
1326     BRV0    _RO_AbsY,, Inc, IF, OP1                -- Read Mem Ptr Lo
1327 _AND_imm:
1328     BMW     _Imm,, Inc, IF, OP1                    -- Read Immediate Operand
1329 _AND_absY:
1330     BRV0    _RO_AbsY,, Inc, IF, OP1                -- Read Mem Ptr Lo
1331 _EOR_imm:
1332     BMW     _Imm,, Inc, IF, OP1                    -- Read Immediate Operand
1333 _EOR_absY:
1334     BRV0    _RO_AbsY,, Inc, IF, OP1                -- Read Mem Ptr Lo
1335 _ADC_imm:
1336     BMW     _Imm,, Inc, IF, OP1                    -- Read Immediate Operand
1337 _ADC_absY:
1338     BRV0    _RO_AbsY,, Inc, IF, OP1                -- Read Mem Ptr Lo
1339 _BIT_imm:
1340     BMW     _Imm,, Inc, IF, OP1                    -- Read Immediate Operand
1341 _STA_absY:
1342     BRV0    _WO_AbsY,, Inc, IF, OP1                -- Read Mem Ptr Lo
1343 _LDA_imm:
1344     BMW     _Imm,, Inc, IF, OP1                    -- Read Immediate Operand
1345 _LDA_absY:
1346     BRV0    _RO_AbsY,, Inc, IF, OP1                -- Read Mem Ptr Lo
1347 _CMP_imm:
1348     BMW     _Imm,, Inc, IF, OP1                    -- Read Immediate Operand
1349 _CMP_absY:
1350     BRV0    _RO_AbsY,, Inc, IF, OP1                -- Read Mem Ptr Lo
1351 _SBC_imm:
1352     BMW     _Imm,, Inc, IF, OP1                    -- Read Immediate Operand
1353 _SBC_absY:
1354     BRV0    _RO_AbsY,, Inc, IF, OP1                -- Read Mem Ptr Lo
1355 ------------------------------------------------------------------------------
1356 -- Row A : 0x0A-0xFA
1357 --  I   BA, uP, NA, IO, DI, Reg_WE, ISR
1358 ------------------------------------------------------------------------------
1359 _ASL_A:
1360     BRV3    $,, Inc, IF, IR, WE_A                  -- Arithmetic Shift A Left
```

```
1361 _INC_A:
1362     BRV3    $,, Inc, IF, IR, WE_A                  -- Increment A
1363 _ROL_A:
1364     BRV3    $,, Inc, IF, IR, WE_A                  -- Rotate A Left
1365 _DEC_A:
1366     BRV3    $,, Inc, IF, IR, WE_A                  -- Decrement A
1367 _LSR_A:
1368     BRV3    $,, Inc, IF, IR, WE_A                  -- Logical Shift A Right
1369 _PHY:
1370     BMW     _Psh,SP, Psh, WR, ALU, WE_R            -- Push Y
1371 _ROR_A:
1372     BRV3    $,, Inc, IF, IR, WE_A                  -- Rotate A Right
1373 _PLY:
1374     BMW     _Pop,SP, Pop, RD, OP1                  -- Pop Y
1375 _TXA:
1376     BRV3    $,, Inc, IF, IR, WE_A                  -- Transfer X to A
1377 _TXS:
1378     BRV3    $,, Inc, IF, IR, WE_S                  -- Transfer X to S
1379 _TAX:
1380     BRV3    $,, Inc, IF, IR, WE_X                  -- Transfer A to X
1381 _TSX:
1382     BRV3    $,, Inc, IF, IR, WE_X                  -- Transfer S to X
1383 _DEX:
1384     BRV3    $,, Inc, IF, IR, WE_X                  -- Decrement X
1385 _PHX:
1386     BMW     _Psh,SP, Psh, WR, ALU, WE_R            -- Push X
1387 _NOP:   -- the real NOP
1388     BRV3    $,, Inc, IF, IR                        -- Skip True NOP Instruction
1389 _PLX:
1390     BMW     _Pop,SP, Pop, RD, OP1                  -- Pop X
1391 ------------------------------------------------------------------------------
1392 -- Row B : 0x0B-0xFB
1393 --  I   BA, uP, NA, IO, DI, Reg_WE, ISR
1394 ------------------------------------------------------------------------------
1395 _NOP_0B:
1396     BRV3    $,, Inc, IF, IR                        -- Skip Invalid Instruction
1397 _NOP_1B:
1398     BRV3    $,, Inc, IF, IR                        -- Skip Invalid Instruction
1399 _NOP_2B:
1400     BRV3    $,, Inc, IF, IR                        -- Skip Invalid Instruction
1401 _NOP_3B:
1402     BRV3    $,, Inc, IF, IR                        -- Skip Invalid Instruction
1403 _NOP_4B:
1404     BRV3    $,, Inc, IF, IR                        -- Skip Invalid Instruction
1405 _NOP_5B:
1406     BRV3    $,, Inc, IF, IR                        -- Skip Invalid Instruction
1407 _NOP_6B:
1408     BRV3    $,, Inc, IF, IR                        -- Skip Invalid Instruction
1409 _NOP_7B:
1410     BRV3    $,, Inc, IF, IR                        -- Skip Invalid Instruction
1411 _NOP_8B:
1412     BRV1    $,, Inc, IF, IR                        -- Prefix 0 - IND, Indirect
1413 _NOP_9B:
1414     BRV1    $,, Inc, IF, IR                        -- Prefix 1 - SIZ, Word Ops
1415 _NOP_AB:
1416     BRV1    $,, Inc, IF, IR                        -- Prefix 2 - OAX
1417 _NOP_BB:
1418     BRV1    $,, Inc, IF, IR                        -- Prefix 4 - OAY
1419 _WAI_CB:
1420     BRV0    _WAI,, PC                              -- Wait for Interrupt
1421 _STP_DB:
1422     BRV0    $,, PC                                 -- Stop Processor Execution
1423 _NOP_EB:
1424     BRV3    $,, Inc, IF, IR                        -- Skip Invalid Instruction
1425 _NOP_FB:
1426     BRV3    $,, Inc, IF, IR                        -- Skip Invalid Instruction
1427 ------------------------------------------------------------------------------
1428 -- Row C : 0x0C-0xFC
1429 --  I   BA, uP, NA, IO, DI, Reg_WE, ISR
1430 ------------------------------------------------------------------------------
1431 _TSB_abs:
1432     BRV0    _RMW_Abs,, Inc, IF, OP1                -- Read Dst Ptr Lo
1433 _TRB_abs:
1434     BRV0    _RMW_Abs,, Inc, IF, OP1                -- Read Dst Ptr Lo
1435 _BIT_abs:
1436     BRV0    _RO_Abs,, Inc, IF, OP1                 -- Read Dst Ptr Lo
1437 _BIT_absX:
1438     BRV0    _RO_AbsX,, Inc, IF, OP1                -- Read Dst Ptr Lo
1439 _JMP_abs:
```

```
1440     BRV0    _Jmp,, Inc, IF, OP1                    -- Read Dst Ptr Lo
1441 _BRA_rel16:
1442     BRV0    _BRA_Rel16,, Inc, IF, OP1              -- Read low(rel16)
1443 _JMP_absi:
1444     BRV0    _JmpI,, Inc, IF, OP1                   -- Read Dst Ptr Lo
1445 _JMP_absXi:
1446     BRV0    _JmpXI,, Inc, IF, OP1                  -- Read Dst Ptr Lo
1447 _STY_abs:
1448     BRV0    _WO_Abs,, Inc, IF, OP1                 -- Read Dst Ptr Lo
1449 _STZ_abs:
1450     BRV0    _WO_Abs,, Inc, IF, OP1                 -- Read Dst Ptr Lo
1451 _LDY_abs:
1452     BRV0    _RO_Abs,, Inc, IF, OP1                 -- Read Dst Ptr Lo
1453 _LDY_absX:
1454     BRV0    _RO_AbsX,, Inc, IF, OP1                -- Read Dst Ptr Lo
1455 _CPY_abs:
1456     BRV0    _RO_Abs,, Inc, IF, OP1                 -- Read Dst Ptr Lo
1457 _BSR_rel16:
1458     BRV0    _BSR_Rel16,, Inc, IF, OP1              -- Read Dst Ptr Lo (Rel16)
1459 _CPX_abs:
1460     BRV0    _RO_Abs,, Inc, IF, OP1                 -- Read Dst Ptr Lo
1461 _PHW_abs:
1462     BRV0    _PHW_Abs,, Inc, IF, OP1                -- Read Dst Ptr Lo
1463 ------------------------------------------------------------------------------
1464 -- Row D : 0x0D-0xFD
1465 --  I   BA, uP, NA, IO, DI, Reg_WE, ISR
1466 ------------------------------------------------------------------------------
1467 _ORA_abs:
1468     BRV0    _RO_Abs,, Inc, IF, OP1                 -- Read Dst Ptr Lo
1469 _ORA_absX:
1470     BRV0    _RO_AbsX,, Inc, IF, OP1                -- Read Dst Ptr Lo
1471 _AND_abs:
1472     BRV0    _RO_Abs,, Inc, IF, OP1                 -- Read Dst Ptr Lo
1473 _AND_absX:
1474     BRV0    _RO_AbsX,, Inc, IF, OP1                -- Read Dst Ptr Lo
1475 _EOR_abs:
1476     BRV0    _RO_Abs,, Inc, IF, OP1                 -- Read Dst Ptr Lo
1477 _EOR_absX:
1478     BRV0    _RO_AbsX,, Inc, IF, OP1                -- Read Dst Ptr Lo
1479 _ADC_abs:
1480     BRV0    _RO_Abs,, Inc, IF, OP1                 -- Read Dst Ptr Lo
1481 _ADC_absX:
1482     BRV0    _RO_AbsX,, Inc, IF, OP1                -- Read Dst Ptr Lo
1483 _STA_abs:
1484     BRV0    _WO_Abs,, Inc, IF, OP1                 -- Read Dst Ptr Lo
1485 _STA_absX:
1486     BRV0    _WO_AbsX,, Inc, IF, OP1                -- Read Dst Ptr Lo
1487 _LDA_abs:
1488     BRV0    _RO_Abs,, Inc, IF, OP1                 -- Read Dst Ptr Lo
1489 _LDA_absX:
1490     BRV0    _RO_AbsX,, Inc, IF, OP1                -- Read Dst Ptr Lo
1491 _CMP_abs:
1492     BRV0    _RO_Abs,, Inc, IF, OP1                 -- Read Dst Ptr Lo
1493 _CMP_absX:
1494     BRV0    _RO_AbsX,, Inc, IF, OP1                -- Read Dst Ptr Lo
1495 _SBC_abs:
1496     BRV0    _RO_Abs,, Inc, IF, OP1                 -- Read Dst Ptr Lo
1497 _SBC_absX:
1498     BRV0    _RO_AbsX,, Inc, IF, OP1                -- Read Dst Ptr Lo
1499 ------------------------------------------------------------------------------
1500 -- Row E : 0x0E-0xFE
1501 --  I   BA, uP, NA, IO, DI, Reg_WE, ISR
1502 ------------------------------------------------------------------------------
1503 _ASL_abs:
1504     BRV0    _RMW_Abs,, Inc, IF, OP1                -- Read Dst Ptr Lo
1505 _ASL_absX:
1506     BRV0    _RMW_AbsX,, Inc, IF, OP1               -- Read Dst Ptr Lo
1507 _ROL_abs:
1508     BRV0    _RMW_Abs,, Inc, IF, OP1                -- Read Dst Ptr Lo
1509 _ROL_absX:
1510     BRV0    _RMW_AbsX,, Inc, IF, OP1               -- Read Dst Ptr Lo
1511 _LSR_abs:
1512     BRV0    _RMW_Abs,, Inc, IF, OP1                -- Read Dst Ptr Lo
1513 _LSR_absX:
1514     BRV0    _RMW_AbsX,, Inc, IF, OP1               -- Read Dst Ptr Lo
1515 _ROR_abs:
1516     BRV0    _RMW_Abs,, Inc, IF, OP1                -- Read Dst Ptr Lo
1517 _ROR_absX:
1518     BRV0    _RMW_AbsX,, Inc, IF, OP1               -- Read Dst Ptr Lo
```

```
1519 _STX_abs:
1520     BRV0    _WO_Abs,, Inc, IF, OP1                 -- Read Dst Ptr Lo
1521 _STZ_absX:
1522     BRV0    _WO_AbsX,, Inc, IF, OP1                -- Read Dst Ptr Lo
1523 _LDX_abs:
1524     BRV0    _RO_Abs,, Inc, IF, OP1                 -- Read Dst Ptr Lo
1525 _LDX_absY:
1526     BRV0    _RO_AbsY,, Inc, IF, OP1                -- Read Dst Ptr Lo
1527 _DEC_abs:
1528     BRV0    _RMW_Abs,, Inc, IF, OP1                -- Read Dst Ptr Lo
1529 _DEC_absX:
1530     BRV0    _RMW_AbsX,, Inc, IF, OP1               -- Read Dst Ptr Lo
1531 _INC_abs:
1532     BRV0    _RMW_Abs,, Inc, IF, OP1                -- Read Dst Ptr Lo
1533 _INC_absX:
1534     BRV0    _RMW_AbsX,, Inc, IF, OP1               -- Read Dst Ptr Lo
1535 ------------------------------------------------------------------------------
1536 -- Row F : 0x0F-0xFF (Rockwell Instructions: BBRx/BBSx dp,rel)
1537 --  I   BA, uP, NA, IO, DI, Reg_WE, ISR
1538 ------------------------------------------------------------------------------
1539 _BBR0_dp_rel:
1540     BRV0    _BByx_dp_rel,, Inc, IF, OP1            -- Read ZP Pointer
1541 _BBR1_dp_rel:
1542     BRV0    _BByx_dp_rel,, Inc, IF, OP1            -- Read ZP Pointer
1543 _BBR2_dp_rel:
1544     BRV0    _BByx_dp_rel,, Inc, IF, OP1            -- Read ZP Pointer
1545 _BBR3_dp_rel:
1546     BRV0    _BByx_dp_rel,, Inc, IF, OP1            -- Read ZP Pointer
1547 _BBR4_dp_rel:
1548     BRV0    _BByx_dp_rel,, Inc, IF, OP1            -- Read ZP Pointer
1549 _BBR5_dp_rel:
1550     BRV0    _BByx_dp_rel,, Inc, IF, OP1            -- Read ZP Pointer
1551 _BBR6_dp_rel:
1552     BRV0    _BByx_dp_rel,, Inc, IF, OP1            -- Read ZP Pointer
1553 _BBR7_dp_rel:
1554     BRV0    _BByx_dp_rel,, Inc, IF, OP1            -- Read ZP Pointer
1555 _BBS0_dp_rel:
1556     BRV0    _BByx_dp_rel,, Inc, IF, OP1            -- Read ZP Pointer
1557 _BBS1_dp_rel:
1558     BRV0    _BByx_dp_rel,, Inc, IF, OP1            -- Read ZP Pointer
1559 _BBS2_dp_rel:
1560     BRV0    _BByx_dp_rel,, Inc, IF, OP1            -- Read ZP Pointer
1561 _BBS3_dp_rel:
1562     BRV0    _BByx_dp_rel,, Inc, IF, OP1            -- Read ZP Pointer
1563 _BBS4_dp_rel:
1564     BRV0    _BByx_dp_rel,, Inc, IF, OP1            -- Read ZP Pointer
1565 _BBS5_dp_rel:
1566     BRV0    _BByx_dp_rel,, Inc, IF, OP1            -- Read ZP Pointer
1567 _BBS6_dp_rel:
1568     BRV0    _BByx_dp_rel,, Inc, IF, OP1            -- Read ZP Pointer
1569 _BBS7_dp_rel:
1570     BRV0    _BByx_dp_rel,, Inc, IF, OP1            -- Read ZP Pointer
1571 ------------------------------------------------------------------------------
1572 --  End of Instruction Decode Table
1573 ------------------------------------------------------------------------------
1574
1575 _Last:  .org 512
1576
1577 _end:
```

```
header
Project: M65C02A_IDecode_ROM
File Revision: 0008
Author(s): Michael A. Morris
Description: M65C02A Instruction Decoder ROM
endh

--------------------------------------------------------------------------------
--
--  Copyright 2013-2014 by Michael A. Morris, dba M. A. Morris & Associates
--
--  All rights reserved. The source code contained herein is publicly released
--  under the terms and conditions of the GNU General Public License as conveyed
--  in the license provided below.
--
--  This program is free software: you can redistribute it and/or modify it
--  under the terms of the GNU General Public License as published by the Free
--  Software Foundation, either version 3 of the License, or any later version.
--
--  This program is distributed in the hope that it will be useful, but WITHOUT
--  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
--  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
--  more details.
--
--  You should have received a copy of the GNU General Public License along with
--  this program.  If not, see <http://www.gnu.org/licenses/>, or write to
--
--  Free Software Foundation, Inc.
--  51 Franklin Street, Fifth Floor
--  Boston, MA  02110-1301 USA
--
--  Further, no use of this source code is permitted in any form or means
--  without inclusion of this banner prominently in any derived works.
--
--  Michael A. Morris <morrisma_at_mchsi_dot_com>
--  164 Raleigh Way
--  Huntsville, AL 35811
--  USA
--
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
-- Revision History:
--------------------------------------------------------------------------------
--
--  0001    13I16   mam     Initial conversion of M65C02_Decoder_ROM used with
--                          the M65C02_ALU.v module. New version changes the
--                          Op[3:0] field into a 5-bit FU_Sel field and a 2-bit
--                          Op field. The new FU_Sel[4:0] field selects/enables
--                          the ALU functional unit, and the new Op[1:0] field
--                          controls the ALU functional unit operations. Removed
--                          Rockwell instructions. Instruction set supported now
--                          only supports original 65C02, otherwise known as the
--                          65SC02, plus the WAI and STP instructions from the
--                          WDC W65C02S.
--
--  0002    13I21   mam     Changed the Q multiplexer select codes. M is now the
--                          the default, and other codes match the other fields.
--
--  0003    13J05   mam     Changed width and encoding of the CCSel fields. The
--                          PSW set/clr instructions will now be implemented in
--                          logic unit of ALU, and the opcode field of this ROM
--                          will be used to generate the mask for the bit in the
--                          P register. As a result of moving these 7 instruc-
--                          out of the PSW multiplexer and into the ALU, the
--                          CCSel field has been narrowed from 5 to 4 bits. The
--                          bit eliminated will be retained as a spare bit in
--                          the same position as it has occupied as part of the
--                          CCSel field.
--
--  0004    13K09   mam     Removed Rsvd (1-bit) field. Assigned that bit to the
--                          FU_Sel field as msb. The new FU_Sel bit functions as
--                          the OR reduction of the 5 least significant bits. In
--                          this manner another logic level is removed from the
--                          control logic of the core. Converted the structure
--                          from sequential opcodes into row order opcodes, i.e.
--                          swapped order of high nibble and low nibble. Matches
--                          the order of the instruction decode table in the
--                          microprogram. Completed the removal of the Rockwell
```

```
 80 --                          bit instructions and the new W65C02S instructions.
 81 --                          Added note to the NOP instructions that indicates
 82 --                          the number of cycles (PC increment operations) that
 83 --                          each requires. (In all cases, the invalid/undefined
 84 --                          instructions do not cause changes in the ALU regis-
 85 --                          ters, but may advance the PC by a defined amount,
 86 --                          and perform a number of dummy cycles. The number of
 87 --                          cycles for this feature is determined by the data-
 88 --                          sheet for 65SC02.)
 89 --
 90 --  0005    14F17   mam     Normalized the implementation to match the M65C02Duo
 91 --                          version of the instruction decoder. Accepted changes
 92 --                          to the mode field encoding, and some of the correct-
 93 --                          ed instructions.
 94 --
 95 --  0006    14F21   mam     Modified the R multiplexer to select {P, K, A, M}
 96 --                          instead of {P, 0, A, M}. Adds flexibility to the
 97 --                          microprogram. AU can be made to support INC/DEC by
 98 --                          any value other than +1/-1. Similarly, CMP can sup-
 99 --                          port comparisons against a memory value or a con-
100 --                          stant. The LU operations are similarly affected.
101 --
102 --  0007    14H09   mam     Modified the definitions of the Mode field. Deleted
103 --                          MMU=. VAL is now defined as 0 and INV is defined as
104 --                          1. Duplicated the table in order to support Kernel
105 --                          and User modes: User mode is the first 256 locations
106 --                          and Kernel (default) the second 256 locations. Added
107 --                          in stack relative addressing for ORA/AND/EOR/ADC/
108 --                          LDA/STA/SBC, RMBx/SMBx/BBRx/BBSx, PEA/PEI/PER,
109 --                          COP #imm, COP dp, JMP/JSR (sp,S),Y, BRA/BSR rel16,
110 --                          MWT/MWF, IND and SIZ (addressing mode and/or opera-
111 --                          tion size override/prefix/escape) instructions.
112 --                          MWT/MWF instructions are privileged instructions
113 --                          available only in Kernel mode. They move a 16-bit
114 --                          word between zero page and the IO page. The Y regis-
115 --                          ter indexes the IO page. The single byte operand
116 --                          provides the zero page address of the 16-bit
117 --                          src/dst, MWT (Y),dp or MWF dp,(Y). Also included in
118 --                          advanced instruction set are 16-bit stack pop opera-
119 --                          tions: PLW dp, and PLW abs. PEA is actually listed
120 --                          as PHW #imm16, and PEI is listed as PHW dp. In addi-
121 --                          tion, the instruction PHW abs has been added instead
122 --                          of JSR (abs,X). There now only 12 unused opcodes re-
123 --                          maining in the instruction set. These twelve are re-
124 --                          served for the user. The expectation is that these
125 --                          opcodes will be used for implementing primitive
126 --                          operations/words useful in a FORTH virtual machine.
127 --
128 --  0008    14H23   mam     Changed project description to M65C02Axxx.
129 --
130 ------------------------------------------------------------------------------
131 -- Mode Field
132 ------------------------------------------------------------------------------
133
134 VAL     .asm    0   -- Valid Instruction (not otherwise decoded)
135 INV     .asm    1   -- Invalid/Unused Instructions
136 COP     .asm    2   -- COP Instruction  (CO-Processor trap, software trap)
137 BRK     .asm    3   -- BRK Instruction  (BReaK execution, software trap  )
138 PFX     .asm    4   -- PFX Instructions (SIZe and INDirect Overrides     )
139 PHR     .asm    5   -- PHR Instruction  (PHR rel16                       )
140 PHW     .asm    6   -- PHW Instructions (PHW #imm16, PHW dp, PHW abs     )
141 WAI     .asm    7   -- WAI Instruction  (WAit for Interrupt              )
142
143 ------------------------------------------------------------------------------
144 -- ROM ( output ) Field definitions
145 ------------------------------------------------------------------------------
146
147 Mode    .def    3   -- Instruction Class or Specific Instruction
148 RMW     .def    1   -- Read-Modify-Write Instruction Type Field
149 FU_Sel  .def    6   -- ALU Functional Unit Select (one-hot)
150 ALU_OP  .def    2   -- ALU Operation
151 QSel    .def    2   -- ALU Q Operand Select
152 RSel    .def    2   -- ALU R Operand Select
153 CSel    .def    2   -- ALU Arithmetic Unit Carry Input Select
154 WSel    .def    3   -- ALU Register Write Select
155 OSel    .def    3   -- ALU Register Output Select
156 CCSel   .def    4   -- ALU Condition Code Operation
157 Opcode  .def    8   -- Instruction Opcode, bit mask, etc.
158
```

```
159 ----------------------------------------------------------------------------
160 --  Constant definitions
161 ----------------------------------------------------------------------------
162
163 ----------------------------------------------------------------------------
164 --  ALU Functional Unit Select definitions
165
166 LST     .equ    48  -- Select Load/Store/Transfer Functional Unit
167 LU      .equ    40  -- Select Logic Unit (AND/ORA/EOR, BIT/TRB/TSB,
168                     --                   RMB/SMB/BBR/BBS,
169                     --                   SEC/CLC/SEI/CLI/SED/CLD/CLV
170 SU      .equ    36  -- Select Shift/Rotate Unit (ASL/ROL/LSR/ROR)
171 ADD     .equ    34  -- Select Decimal/Binary Adder (ADC/SBC)
172 IDC     .equ    33  -- Select Bin. Add (INC/DEC/CMP/INX/DEX/CPX/INY/DEY/CPY)
173
174 ----------------------------------------------------------------------------
175 --  General Definitions
176 ----------------------------------------------------------------------------
177
178 NOP     .equ    0   -- No operation and/or default operation
179 XFR     .equ    0   -- No operation and/or default operation
180
181 ----------------------------------------------------------------------------
182 --  ALU Operation Field (ALU_Op) Definitions
183 ----------------------------------------------------------------------------
184
185 --  Logic Unit Operations
186
187 AND     .equ    1   -- ALU <=  A & M;   N <= ALU[7]; Z <= ~|ALU;
188 ORA     .equ    2   -- ALU <=  A | M;   N <= ALU[7]; Z <= ~|ALU;
189 EOR     .equ    3   -- ALU <=  A ^ M;   N <= ALU[7]; Z <= ~|ALU;
190
191 BIT     .equ    1   -- ALU <=  A & M;   N <= M[7]; V <= M[6]; Z <= ~|(A & M)
192 -- BIT #imm         -- ALU <=  A & M;                          Z <= ~|(A & M)
193 TRB     .equ    0   -- ALU <= ~A & M;                          Z <= ~|(A & M)
194 TSB     .equ    2   -- ALU <=  A | M;                          Z <= ~|(A & M)
195
196 RMB     .equ    0   -- ALU <= ~K & M;  (K <= (1 << bit))
197 SMB     .equ    2   -- ALU <=  K | M;  (K <= (1 << bit))
198 BBR     .equ    1   -- ALU <=  K & M;  (K <= (1 << bit))
199 BBS     .equ    1   -- ALU <=  K & M;  (K <= (1 << bit))
200
201 CLC     .equ    0   -- ALU <= ~K & P;   C <= 0; (K <= 0x01)
202 SEC     .equ    2   -- ALU <=  K | P;   C <= 1; (K <= 0x01)
203 CLI     .equ    0   -- ALU <= ~K & P;   I <= 0; (K <= 0x04)
204 SEI     .equ    2   -- ALU <=  K | P;   I <= 1; (K <= 0x04)
205 CLD     .equ    0   -- ALU <= ~K & P;   D <= 0; (K <= 0x08)
206 SED     .equ    2   -- ALU <=  K | P;   D <= 1; (K <= 0x08)
207 CLV     .equ    0   -- ALU <= ~K & P;   V <= 0; (K <= 0x40)
208
209 --REP     .equ    0   -- ALU <= ~M & P;   P <= ALU
210 --SEP     .equ    2   -- ALU <=  M | P;   P <= ALU
211
212 --  Shift Unit Operations (Note: for ASL/LSR, Ci <= 0; for ROL/ROR, Ci <= C)
213
214 ASL     .equ    0   -- ALU <= {R[6:0],Ci};N <= ALU[7]; Z <= ~|ALU; C <= R[7]
215 ROL     .equ    0   -- ALU <= {R[6:0],Ci};N <= ALU[7]; Z <= ~|ALU; C <= R[7]
216 LSR     .equ    1   -- ALU <= {Ci,R[7:1]};N <= ALU[7]; Z <= ~|ALU; C <= R[0]
217 ROR     .equ    1   -- ALU <= {Ci,R[7:1]} N <= ALU[7]; Z <= ~|ALU; C <= R[0]
218
219 --  Arithmetic Unit Operations
220
221 ADC     .equ    0   -- ALU <= Q +  M + C; N <= ALU[7]; Z <= ~|ALU;
222 --                                        V <= OVF;    C <= COut;
223 SBC     .equ    1   -- ALU <= Q + ~M + C; N <= ALU[7]; Z <= ~|ALU;
224 --                                        V <= OVF;    C <= COut;
225 INC     .equ    0   -- ALU <= Q +  0 + 1; N <= ALU[7]; Z <= ~|ALU;
226 DEC     .equ    1   -- ALU <= Q + ~0 + 0; N <= ALU[7]; Z <= ~|ALU;
227 CMP     .equ    1   -- ALU <= Q + ~M + 1; N <= ALU[7]; Z <= ~|ALU;
228 --                                                    C <= COut
229
230 ----------------------------------------------------------------------------
231 --  ALU Left (L/Q) Operand Select
232 ----------------------------------------------------------------------------
233
234 L_M     .equ    0   -- L <= M (default)
235 L_A     .equ    1   -- L <= A
236 L_K     .equ    2   -- L <= K
237 L_P     .equ    3   -- L <= P
```

```
238
239 Q_M     .equ    0   -- Q <= M (default)
240 Q_A     .equ    1   -- Q <= A
241 Q_X     .equ    2   -- Q <= X
242 Q_Y     .equ    3   -- Q <= Y
243
244 ------------------------------------------------------------------------------
245 --  ALU Right (R) Operand Select
246 ------------------------------------------------------------------------------
247
248 R_M     .equ    0   -- R <= M (default)
249 R_A     .equ    1   -- R <= A
250 R_K     .equ    2   -- R <= K
251 R_P     .equ    3   -- R <= P
252
253 ------------------------------------------------------------------------------
254 --  ALU Carry Input Multiplexer Select
255 ------------------------------------------------------------------------------
256
257 Ci_C    .equ    0   -- Ci <= C (default)
258 Ci_S    .equ    1   -- Ci <= Q[7]
259 Ci_0    .equ    2   -- Ci <= 0
260 Ci_1    .equ    3   -- Ci <= 1
261
262 ------------------------------------------------------------------------------
263 --  ALU Register Write Select Definitions
264 ------------------------------------------------------------------------------
265
266 WS_A    .equ    1   -- Write Accumulator (Binary)
267 WS_X    .equ    2   -- Write X (Pre-Index Register)
268 WS_Y    .equ    3   -- Write Y (Post-Index Register)
269 WS_R    .equ    4   -- Write Registers
270 WS_S    .equ    5   -- Write S (Stack Pointer)
271 WS_P    .equ    6   -- Write P (Processor Status Word)
272 WS_M    .equ    7   -- Write M (ALU Output)
273
274 ------------------------------------------------------------------------------
275 --  ALU Register Output Select Definitions
276 ------------------------------------------------------------------------------
277
278 OS_A    .equ    1   -- ALU <= A
279 OS_X    .equ    2   -- ALU <= X
280 OS_Y    .equ    3   -- ALU <= Y
281 OS_T    .equ    4   -- ALU <= Tmp (Operand Register 2)
282 OS_S    .equ    5   -- ALU <= S   (Stack Pointer)
283 OS_P    .equ    6   -- ALU <= P   (Processor Status Word)
284 OS_M    .equ    7   -- ALU <= M   (Memory Data Input, Operand Register 1)
285
286 ------------------------------------------------------------------------------
287 --  Condition Code Operation/Output Select Definitions
288 --      Note: CC_Out = 1 unless (CCSel[4:3] != 2'b01)
289 ------------------------------------------------------------------------------
290
291 TRUE    .equ    0   -- CC_Out <=  1
292
293 BRA     .equ    0   -- CC_Out <=  1
294 BCC     .equ    8   -- CC_Out <= ~C;
295 BCS     .equ    9   -- CC_Out <=  C;
296 BNE     .equ    10  -- CC_Out <= ~Z;
297 BEQ     .equ    11  -- CC_Out <=  Z;
298 BVC     .equ    12  -- CC_Out <= ~V;
299 BVS     .equ    13  -- CC_Out <=  V;
300 BPL     .equ    14  -- CC_OUT <= ~N;
301 BMI     .equ    15  -- CC_Out <=  N;
302
303 PSW     .equ    0   -- P <= ALU;
304 Trap    .equ    1   -- P.4 <= 1 on push P during BRK
305 Z       .equ    2   --                          Z <= ~|(A & M);
306 NVZ     .equ    3   -- N <= M[7];  V <= M[6]; Z <= ~|(A & M);
307 PHP     .equ    4   -- P.4 <= 1 on push P during PHP
308 NZ      .equ    5   -- N <= ALU[7];           Z <= ~|ALU;
309 NZC     .equ    6   -- N <= ALU[7];           Z <= ~|ALU; C <= COut;
310 NVZC    .equ    7   -- N <= ALU[7]; V <= OV;  Z <= ~|ALU; C <= COut;
311
312 ------------------------------------------------------------------------------
313 -- Mask Settings
314 ------------------------------------------------------------------------------
315
316 K_0     .equ    1   -- Bit 0
```

```
317 K_1     .equ    2   -- Bit 1
318 K_2     .equ    4   -- Bit 2
319 K_3     .equ    8   -- Bit 3
320 K_4     .equ    16  -- Bit 4
321 K_5     .equ    32  -- Bit 5
322 K_6     .equ    64  -- Bit 6
323 K_7     .equ    128 -- Bit 7
324
325 K_C     .equ    1   -- Mask for P.C (Carry Flag)
326 K_Z     .equ    2   -- Mask for P.Z (Zero Flag)
327 K_I     .equ    4   -- Mask for P.I (Interrupt Mask)
328 K_D     .equ    8   -- Mask for P.D (Decimal Mode Flag)
329 K_B     .equ    16  -- Mask for P.B (BRK/PHP Flag)
330 K_M     .equ    32  -- Mask for P.M (Unused bit - always 1)
331 K_V     .equ    64  -- Mask for P.V (oVerflow Flag)
332 K_N     .equ    128 -- Mask for P.N (Negative Flag)
333
334 ------------------------------------------------------------------------------
335 --
336 --  Added annotations regarding the implementation of the various opcodes in re-
337 --  lation to the various microprocessors. As currently defined in this file,
338 --  the opcodes being decoded represent the instruction set of the original
339 --  65C02. In other words, the non-NOP instructions are for the W65C02/G65SC02.
340 --  In the right hand margin comments, opcodes that were added by these two
341 --  processors to the base MOS6502 instruction set are marked with an asterisk.
342 --
343 --  Also placed in parentheses the instructions added by the Rockwell R65C02,
344 --  and marked those instructions with an addition symbol (+). WDC also added
345 --  two instructions from the W65C816/W65C802 processor instruction set, and
346 --  they are marked with an ampersand (&).
347 --
348 --  The instruction set of the M65C02 starts from the instruction set of G65SC02
349 --  and adds the Rockwell (+), and the W65C816/W65C802 (&) instructions. The
350 --  resulting instruction set is equivalent to the instruction set of the WDC
351 --  W65C02S microprocessor.
352 --
353 --  There are additional instructions marked by the octothorpe (#) which will be
354 --  added to the M65C02 instruction set to form the M65C02A's instruction set.
355 --  The new instructions for the M65C02A are generally taken from the W65C816
356 --  instruction set. There are a few unique M65C02A instructions, but generally,
357 --  the new instructions add the W65C816's stack relative addresssing mode in-
358 --  structions.
359 --
360 --  The additional instructions discussed above can be included without any
361 --  modifications to the core logic; only updates to the microcode ROMs are re-
362 --  quired.
363 --
364 ------------------------------------------------------------------------------
365
366 _start:
367 _User:      .org    0                                   -- User Mode
368
369 ------------------------------------------------------------------------------
370
371     BRK     0,LST,XFR,Q_M,R_M,Ci_C,WS_P,    ,Trap,0x00  -- 00: BRK #imm
372     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,    ,BPL ,0x10  -- 10: BPL rel
373     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,    ,    ,0x20  -- 20: JSR abs
374     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,    ,BMI ,0x30  -- 30: BMI rel
375     VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_P,OS_M,PSW ,0x40  -- 40: RTI
376     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,    ,BVC ,0x50  -- 50: BVC rel
377     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,    ,    ,0x60  -- 60: RTS
378     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,    ,BVS ,0x70  -- 70: BVS rel
379     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,    ,BRA ,0x80  -- 80:*BRA rel
380     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,    ,BCC ,0x90  -- 90: BCC rel
381     VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_Y,OS_M,NZ  ,0xA0  -- A0: LDY #imm
382     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,    ,BCS ,0xB0  -- B0: BCS rel
383     VAL     0,IDC,CMP,Q_Y,R_M,Ci_1,WS_P,    ,NZC ,0xC0  -- C0: CPY #imm
384     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,    ,BNE ,0xD0  -- D0: BNE rel
385     VAL     0,IDC,CMP,Q_X,R_M,Ci_1,WS_P,    ,NZC ,0xE0  -- E0: CPX #imm
386     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,    ,BEQ ,0xF0  -- F0: BEQ rel
387
388 ------------------------------------------------------------------------------
389
390     VAL     0,LU ,ORA,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x01  -- 01: ORA (dp,X)
391     VAL     0,LU ,ORA,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x11  -- 11: ORA (dp),Y
392     VAL     0,LU ,AND,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x21  -- 21: AND (dp,X)
393     VAL     0,LU ,AND,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x31  -- 31: AND (dp),Y
394     VAL     0,LU ,EOR,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x41  -- 41: EOR (dp,X)
395     VAL     0,LU ,EOR,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x51  -- 51: EOR (dp),Y
```

```
396     VAL     0,ADD,ADC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0x61  -- 61: ADC (dp,X)
397     VAL     0,ADD,ADC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0x71  -- 71: ADC (dp),Y
398     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,OS_A,    ,0x81  -- 81: STA (dp,X)
399     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,OS_A,    ,0x91  -- 91: STA (dp),Y
400     VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_A,OS_M,NZ  ,0xA1  -- A1: LDA (dp,X)
401     VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_A,OS_M,NZ  ,0xB1  -- B1: LDA (dp),Y
402     VAL     0,IDC,CMP,Q_A,R_M,Ci_1,WS_P,    ,NZC ,0xC1  -- C1: CMP (dp,X)
403     VAL     0,IDC,CMP,Q_A,R_M,Ci_1,WS_P,    ,NZC ,0xD1  -- D1: CMP (dp),Y
404     VAL     0,ADD,SBC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0xE1  -- E1: SBC (dp,X)
405     VAL     0,ADD,SBC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0xF1  -- F1: SBC (dp),Y
406
407 --------------------------------------------------------------------------------
408
409     COP     0,LST,XFR,Q_M,R_M,Ci_C,WS_X,OS_M,NZ  ,0x02  -- 02:#COP dp
410     VAL     0,LU ,ORA,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x12  -- 12:*ORA (dp)
411     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,    ,    ,0x22  -- 22:#JSR (sp,S),Y
412     VAL     0,LU ,AND,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x32  -- 32:*AND (dp)
413     COP     0,LST,XFR,Q_M,R_M,Ci_C,WS_X,OS_M,NZ  ,0x42  -- 42:#COP #imm
414     VAL     0,LU ,EOR,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x52  -- 52:*EOR (dp)
415     PHR     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0x62  -- 62:#PHR rel16
416     VAL     0,ADD,ADC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0x72  -- 72:*ADC (dp)
417     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,    ,    ,0x82  -- 82:#JMP (sp,S),Y
418     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,OS_A,    ,0x92  -- 92:*STA (dp)
419     VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_X,OS_M,NZ  ,0xA2  -- A2: LDX #imm
420     VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_A,OS_M,NZ  ,0xB2  -- B2:*LDA (dp)
421     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,OS_M,    ,0xC2  -- C2:#PLW dp
422     VAL     0,IDC,CMP,Q_A,R_M,Ci_1,WS_P,    ,NZC ,0xD2  -- D2:*CMP (dp)
423     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,OS_M,    ,0xE2  -- E2:#PLW abs
424     VAL     0,ADD,SBC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0xF2  -- F2:*SBC (dp)
425
426 --------------------------------------------------------------------------------
427
428     VAL     0,LU ,ORA,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x03  -- 03:#ORA sp,S
429     VAL     0,LU ,ORA,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x13  -- 13:#ORA (sp,S),Y
430     VAL     0,LU ,AND,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x23  -- 23:#AND sp,S
431     VAL     0,LU ,AND,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x33  -- 33:#AND (sp,S),Y
432     VAL     0,LU ,EOR,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x43  -- 43:#EOR sp,S
433     VAL     0,LU ,EOR,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x53  -- 53:#EOR (sp,S),Y
434     VAL     0,ADD,ADC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0x63  -- 63:#ADC sp,S
435     VAL     0,ADD,ADC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0x73  -- 73:#ADC (sp,S),Y
436     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,OS_A,    ,0x83  -- 83:#STA sp,S
437     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,OS_A,    ,0x93  -- 93:#STA (sp,S),Y
438     VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_A,OS_M,NZ  ,0xA3  -- A3:#LDA sp,S
439     VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_A,OS_M,NZ  ,0xB3  -- B3:#LDA (sp,S),Y
440     VAL     0,IDC,CMP,Q_A,R_M,Ci_1,WS_P,    ,NZC ,0xC3  -- C3:#CMP sp,S
441     VAL     0,IDC,CMP,Q_A,R_M,Ci_1,WS_P,    ,NZC ,0xD3  -- D3:#CMP (sp,S),Y
442     VAL     0,ADD,SBC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0xE3  -- E3:#SBC sp,S
443     VAL     0,ADD,SBC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0xF3  -- F3:#SBC (sp,S),Y
444
445 --------------------------------------------------------------------------------
446
447     VAL     1,LU ,TSB,L_A,R_M,Ci_C,WS_P,    ,Z   ,0x04  -- 04:*TSB dp
448     VAL     1,LU ,TRB,L_A,R_M,Ci_C,WS_P,    ,Z   ,0x14  -- 14:*TRB dp
449     VAL     0,LU ,BIT,L_A,R_M,Ci_C,WS_P,    ,NVZ ,0x24  -- 24: BIT dp
450     VAL     0,LU ,BIT,L_A,R_M,Ci_C,WS_P,    ,NVZ ,0x34  -- 34:*BIT dp,X
451     INV     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0x44  -- 44:#NOP (MWT dp,(Y))
452     INV     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0x54  -- 54:#NOP (MWF dp,(Y))
453     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,    ,    ,0x64  -- 64:*STZ dp
454     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,    ,    ,0x74  -- 74:*STZ dp,X
455     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,OS_Y,    ,0x84  -- 84: STY dp
456     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,OS_Y,    ,0x94  -- 94: STY dp,X
457     VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_Y,OS_M,NZ  ,0xA4  -- A4: LDY dp
458     VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_Y,OS_M,NZ  ,0xB4  -- B4: LDY dp,X
459     VAL     0,IDC,CMP,Q_Y,R_M,Ci_1,WS_P,    ,NZC ,0xC4  -- C4: CPY dp
460     PHW     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0xD4  -- D4:#PHW dp
461     VAL     0,IDC,CMP,Q_X,R_M,Ci_1,WS_P,    ,NZC ,0xE4  -- E4: CPX dp
462     PHW     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0xF4  -- F4:#PHW #imm16
463
464 --------------------------------------------------------------------------------
465
466     VAL     0,LU ,ORA,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x05  -- 05: ORA dp
467     VAL     0,LU ,ORA,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x15  -- 15: ORA dp,X
468     VAL     0,LU ,AND,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x25  -- 25: AND dp
469     VAL     0,LU ,AND,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x35  -- 35: AND dp,X
470     VAL     0,LU ,EOR,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x45  -- 45: EOR dp
471     VAL     0,LU ,EOR,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x55  -- 55: EOR dp,X
472     VAL     0,ADD,ADC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0x65  -- 65: ADC dp
473     VAL     0,ADD,ADC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0x75  -- 75: ADC dp,X
474     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,OS_A,    ,0x85  -- 85: STA dp
```

```
475     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,OS_A,    ,0x95  -- 95: STA dp,X
476     VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_A,OS_M,NZ  ,0xA5  -- A5: LDA dp
477     VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_A,OS_M,NZ  ,0xB5  -- B5: LDA dp,X
478     VAL     0,IDC,CMP,Q_A,R_M,Ci_1,WS_P,    ,NZC ,0xC5  -- C5: CMP dp
479     VAL     0,IDC,CMP,Q_A,R_M,Ci_1,WS_P,    ,NZC ,0xD5  -- D5: CMP dp,X
480     VAL     0,ADD,SBC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0xE5  -- E5: SBC dp
481     VAL     0,ADD,SBC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0xF5  -- F5: SBC dp,X
482
483 --------------------------------------------------------------------------------
484
485     VAL     1,SU ,ASL,Q_M,R_M,Ci_0,WS_P,    ,NZC ,0x06  -- 06: ASL dp
486     VAL     1,SU ,ASL,Q_M,R_M,Ci_0,WS_P,    ,NZC ,0x16  -- 16: ASL dp,X
487     VAL     1,SU ,ROL,Q_M,R_M,Ci_C,WS_P,    ,NZC ,0x26  -- 26: ROL dp
488     VAL     1,SU ,ROL,Q_M,R_M,Ci_C,WS_P,    ,NZC ,0x36  -- 36: ROL dp,X
489     VAL     1,SU ,LSR,Q_M,R_M,Ci_0,WS_P,    ,NZC ,0x46  -- 46: LSR dp
490     VAL     1,SU ,LSR,Q_M,R_M,Ci_0,WS_P,    ,NZC ,0x56  -- 56: LSR dp,X
491     VAL     1,SU ,ROR,Q_M,R_M,Ci_C,WS_P,    ,NZC ,0x66  -- 66: ROR dp
492     VAL     1,SU ,ROR,Q_M,R_M,Ci_C,WS_P,    ,NZC ,0x76  -- 76: ROR dp,X
493     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,OS_X,    ,0x86  -- 86: STX dp
494     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,OS_X,    ,0x96  -- 96: STX dp,Y
495     VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_X,OS_M,NZ  ,0xA6  -- A6: LDX dp
496     VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_X,OS_M,NZ  ,0xB6  -- B6: LDX dp,Y
497     VAL     1,IDC,DEC,Q_M,R_K,Ci_0,WS_P,    ,NZ  ,0x00  -- C6: DEC dp
498     VAL     1,IDC,DEC,Q_M,R_K,Ci_0,WS_P,    ,NZ  ,0x00  -- D6: DEC dp,X
499     VAL     1,IDC,INC,Q_M,R_K,Ci_1,WS_P,    ,NZ  ,0x00  -- E6: INC dp
500     VAL     1,IDC,INC,Q_M,R_K,Ci_1,WS_P,    ,NZ  ,0x00  -- F6: INC dp,X
501
502 --------------------------------------------------------------------------------
503
504     VAL     1,LU ,RMB,L_K,R_M,Ci_C,    ,    ,    ,K_0   -- 07:+RMB0 dp
505     VAL     1,LU ,RMB,L_K,R_M,Ci_C,    ,    ,    ,K_1   -- 17:+RMB1 dp
506     VAL     1,LU ,RMB,L_K,R_M,Ci_C,    ,    ,    ,K_2   -- 27:+RMB2 dp
507     VAL     1,LU ,RMB,L_K,R_M,Ci_C,    ,    ,    ,K_3   -- 37:+RMB3 dp
508     VAL     1,LU ,RMB,L_K,R_M,Ci_C,    ,    ,    ,K_4   -- 47:+RMB4 dp
509     VAL     1,LU ,RMB,L_K,R_M,Ci_C,    ,    ,    ,K_5   -- 57:+RMB5 dp
510     VAL     1,LU ,RMB,L_K,R_M,Ci_C,    ,    ,    ,K_6   -- 67:+RMB6 dp
511     VAL     1,LU ,RMB,L_K,R_M,Ci_C,    ,    ,    ,K_7   -- 77:+RMB7 dp
512     VAL     1,LU ,SMB,L_K,R_M,Ci_C,    ,    ,    ,K_0   -- 87:+SMB0 dp
513     VAL     1,LU ,SMB,L_K,R_M,Ci_C,    ,    ,    ,K_1   -- 97:+SMB1 dp
514     VAL     1,LU ,SMB,L_K,R_M,Ci_C,    ,    ,    ,K_2   -- A7:+SMB2 dp
515     VAL     1,LU ,SMB,L_K,R_M,Ci_C,    ,    ,    ,K_3   -- B7:+SMB3 dp
516     VAL     1,LU ,SMB,L_K,R_M,Ci_C,    ,    ,    ,K_4   -- C7:+SMB4 dp
517     VAL     1,LU ,SMB,L_K,R_M,Ci_C,    ,    ,    ,K_5   -- D7:+SMB5 dp
518     VAL     1,LU ,SMB,L_K,R_M,Ci_C,    ,    ,    ,K_6   -- E7:+SMB6 dp
519     VAL     1,LU ,SMB,L_K,R_M,Ci_C,    ,    ,    ,K_7   -- F7:+SMB7 dp
520
521 --------------------------------------------------------------------------------
522
523     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,OS_P,PHP ,0x08  -- 08: PHP
524     VAL     0,LU ,CLC,L_K,R_P,Ci_C,WS_P,    ,PSW ,K_C   -- 18: CLC
525     VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_P,OS_M,PSW ,0x28  -- 28: PLP
526     VAL     0,LU ,SEC,L_K,R_P,Ci_C,WS_P,    ,PSW ,K_C   -- 38: SEC
527     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,OS_A,    ,0x48  -- 48: PHA
528     VAL     0,LU ,CLI,L_K,R_P,Ci_C,WS_P,    ,PSW ,K_I   -- 58: CLI
529     VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_A,OS_M,NZ  ,0x68  -- 68: PLA
530     VAL     0,LU ,SEI,L_K,R_P,Ci_C,WS_P,    ,PSW ,K_I   -- 78: SEI
531     VAL     0,IDC,DEC,Q_Y,R_K,Ci_0,WS_Y,    ,NZ  ,0x00  -- 88: DEY
532     VAL     0,LST,XFR,Q_A,R_M,Ci_C,WS_A,OS_Y,NZ  ,0x98  -- 98: TYA
533     VAL     0,LST,XFR,Q_A,R_M,Ci_C,WS_Y,OS_A,NZ  ,0xA8  -- A8: TAY
534     VAL     0,LU ,CLV,L_K,R_P,Ci_C,WS_P,    ,PSW ,K_V   -- B8: CLV
535     VAL     0,IDC,INC,Q_Y,R_K,Ci_1,WS_Y,    ,NZ  ,0x00  -- C8: INY
536     VAL     0,LU ,CLD,L_K,R_P,Ci_C,WS_P,    ,PSW ,K_D   -- D8: CLD
537     VAL     0,IDC,INC,Q_X,R_K,Ci_1,WS_X,    ,NZ  ,0x00  -- E8: INX
538     VAL     0,LU ,SED,L_K,R_P,Ci_C,WS_P,    ,PSW ,K_D   -- F8: SED
539
540 --------------------------------------------------------------------------------
541
542     VAL     0,LU ,ORA,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x09  -- 09: ORA #imm
543     VAL     0,LU ,ORA,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x19  -- 19: ORA abs,Y
544     VAL     0,LU ,AND,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x29  -- 29: AND #imm
545     VAL     0,LU ,AND,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x39  -- 39: AND abs,Y
546     VAL     0,LU ,EOR,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x49  -- 49: EOR #imm
547     VAL     0,LU ,EOR,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x59  -- 59: EOR abs,Y
548     VAL     0,ADD,ADC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0x69  -- 69: ADC #imm
549     VAL     0,ADD,ADC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0x79  -- 79: ADC abs,Y
550     VAL     0,LU ,BIT,L_A,R_M,Ci_C,WS_P,    ,Z   ,0x89  -- 89:*BIT #imm
551     VAL     0,LST,XFR,Q_A,R_M,Ci_C,    ,OS_A,    ,0x99  -- 99: STA abs,Y
552     VAL     0,LST,XFR,Q_A,R_M,Ci_C,WS_A,OS_M,NZ  ,0xA9  -- A9: LDA #imm
553     VAL     0,LST,XFR,Q_A,R_M,Ci_C,WS_A,OS_M,NZ  ,0xB9  -- B9: LDA abs,Y
```

```
554      VAL     0,IDC,CMP,Q_A,R_M,Ci_1,WS_P,    ,NZC ,0xC9  -- C9: CMP #imm
555      VAL     0,IDC,CMP,Q_A,R_M,Ci_1,WS_P,    ,NZC ,0xD9  -- D9: CMP abs,Y
556      VAL     0,ADD,SBC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0xE9  -- E9: SBC #imm
557      VAL     0,ADD,SBC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0xF9  -- F9: SBC abs,Y
558
559  --------------------------------------------------------------------------------
560
561      VAL     0,SU ,ASL,Q_A,R_M,Ci_0,WS_A,    ,NZC ,0x0A  -- 0A: ASL A
562      VAL     0,IDC,INC,Q_A,R_K,Ci_1,WS_A,    ,NZ  ,0x00  -- 1A:*INC A
563      VAL     0,SU ,ROL,Q_A,R_M,Ci_C,WS_A,    ,NZC ,0x2A  -- 2A: ROL A
564      VAL     0,IDC,DEC,Q_A,R_K,Ci_0,WS_A,    ,NZ  ,0x00  -- 3A:*DEC A
565      VAL     0,SU ,LSR,Q_A,R_M,Ci_0,WS_A,    ,NZC ,0x4A  -- 4A: LSR A
566      VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,OS_Y,    ,0x5A  -- 5A:*PHY
567      VAL     0,SU ,ROR,Q_A,R_M,Ci_C,WS_A,    ,NZC ,0x6A  -- 6A: ROR A
568      VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_Y,OS_M,NZ  ,0x7A  -- 7A:*PLY
569      VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_A,OS_X,NZ  ,0x8A  -- 8A: TXA
570      VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_S,OS_X,    ,0x9A  -- 9A: TXS
571      VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_X,OS_A,NZ  ,0xAA  -- AA: TAX
572      VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_X,OS_S,NZ  ,0xBA  -- BA: TSX
573      VAL     0,IDC,DEC,Q_X,R_K,Ci_0,WS_X,    ,NZ  ,0x00  -- CA: DEX
574      VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,OS_X,    ,0xDA  -- DA:*PHX
575      VAL     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0xEA  -- EA: NOP
576      VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_X,OS_M,NZ  ,0xFA  -- FA:*PLX
577
578  --------------------------------------------------------------------------------
579  -- 0B...BB: 1 byte, 1 cycle
580      INV     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0x0B  -- 0B: NOP
581      INV     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0x1B  -- 1B: NOP
582      INV     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0x2B  -- 2B: NOP
583      INV     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0x3B  -- 3B: NOP
584      INV     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0x4B  -- 4B: NOP
585      INV     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0x5B  -- 5B: NOP
586      INV     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0x6B  -- 6B: NOP
587      INV     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0x7B  -- 7B: NOP
588      PFX     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0x8B  -- 8B:#IND  -- Indirect
589      PFX     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0x9B  -- 9B:#OAX  -- A<=>X
590      PFX     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0xAB  -- AB:#OAY  -- A<=>Y
591      PFX     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0xBB  -- BB:#SIZ  -- Size OVR
592      WAI     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0xCB  -- CB:&WAI
593      VAL     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0xDB  -- DB:&STP
594      INV     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0xEB  -- EB: NOP
595      INV     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0xFB  -- FB: NOP
596
597  --------------------------------------------------------------------------------
598
599      VAL     1,LU ,TSB,L_A,R_M,Ci_C,WS_P,    ,Z   ,0x0C  -- 0C:*TSB abs
600      VAL     1,LU ,TRB,L_A,R_M,Ci_C,WS_P,    ,Z   ,0x1C  -- 1C:*TRB abs
601      VAL     0,LU ,BIT,L_A,R_M,Ci_C,WS_P,    ,NVZ ,0x2C  -- 2C: BIT abs
602      VAL     0,LU ,BIT,L_A,R_M,Ci_C,WS_P,    ,NVZ ,0x3C  -- 3C:*BIT abs,X
603      VAL     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0x4C  -- 4C: JMP abs
604      VAL     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0x5C  -- 5C:#BRA rel16
605      VAL     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0x6C  -- 6C: JMP (abs)
606      VAL     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0x7C  -- 7C:*JMP (abs,X)
607      VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,OS_Y,    ,0x8C  -- 8C: STY abs
608      VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,    ,    ,0x9C  -- 9C:*STZ abs
609      VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_Y,OS_M,NZ  ,0xAC  -- AC: LDY abs
610      VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_Y,OS_M,NZ  ,0xBC  -- BC: LDY abs,X
611      VAL     0,IDC,CMP,Q_Y,R_M,Ci_1,WS_P,    ,NZC ,0xCC  -- CC: CPY abs
612      VAL     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0xDC  -- DC:#BSR rel16
613      VAL     0,IDC,CMP,Q_X,R_M,Ci_1,WS_P,    ,NZC ,0xEC  -- EC: CPX abs
614      PHW     0,NOP,NOP,Q_M,R_M,Ci_C,    ,    ,    ,0xFC  -- FC:#PHW abs
615
616  --------------------------------------------------------------------------------
617
618      VAL     0,LU ,ORA,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x0D  -- 0D: ORA abs
619      VAL     0,LU ,ORA,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x1D  -- 1D: ORA abs,X
620      VAL     0,LU ,AND,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x2D  -- 2D: AND abs
621      VAL     0,LU ,AND,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x3D  -- 3D: AND abs,X
622      VAL     0,LU ,EOR,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x4D  -- 4D: EOR abs
623      VAL     0,LU ,EOR,L_A,R_M,Ci_C,WS_A,    ,NZ  ,0x5D  -- 5D: EOR abs,X
624      VAL     0,ADD,ADC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0x6D  -- 6D: ADC abs
625      VAL     0,ADD,ADC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0x7D  -- 7D: ADC abs,X
626      VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,OS_A,    ,0x8D  -- 8D: STA abs
627      VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,OS_A,    ,0x9D  -- 9D: STA abs,X
628      VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_A,OS_M,NZ  ,0xAD  -- AD: LDA abs
629      VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_A,OS_M,NZ  ,0xBD  -- BD: LDA abs,X
630      VAL     0,IDC,CMP,Q_A,R_M,Ci_1,WS_P,    ,NZC ,0xCD  -- CD: CMP abs
631      VAL     0,IDC,CMP,Q_A,R_M,Ci_1,WS_P,    ,NZC ,0xDD  -- DD: CMP abs,X
632      VAL     0,ADD,SBC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0xED  -- ED: SBC abs
```

```
633     VAL     0,ADD,SBC,Q_A,R_M,Ci_C,WS_A,    ,NVZC,0xFD  -- FD: SBC abs,X
634
635 -------------------------------------------------------------------------------
636
637     VAL     1,SU ,ASL,Q_M,R_M,Ci_0,WS_P,    ,NZC ,0x0E  -- 0E: ASL abs
638     VAL     1,SU ,ASL,Q_M,R_M,Ci_0,WS_P,    ,NZC ,0x1E  -- 1E: ASL abs,X
639     VAL     1,SU ,ROL,Q_M,R_M,Ci_C,WS_P,    ,NZC ,0x2E  -- 2E: ROL abs
640     VAL     1,SU ,ROL,Q_M,R_M,Ci_C,WS_P,    ,NZC ,0x3E  -- 3E: ROL abs,X
641     VAL     1,SU ,LSR,Q_M,R_M,Ci_0,WS_P,    ,NZC ,0x4E  -- 4E: LSR abs
642     VAL     1,SU ,LSR,Q_M,R_M,Ci_0,WS_P,    ,NZC ,0x5E  -- 5E: LSR abs,X
643     VAL     1,SU ,ROR,Q_M,R_M,Ci_C,WS_P,    ,NZC ,0x6E  -- 6E: ROR abs
644     VAL     1,SU ,ROR,Q_M,R_M,Ci_C,WS_P,    ,NZC ,0x7E  -- 7E: ROR abs,X
645     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,OS_X,    ,0x8E  -- 8E: STX abs
646     VAL     0,LST,XFR,Q_M,R_M,Ci_C,    ,   ,     ,0x9E  -- 9E:*STZ abs,X
647     VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_X,OS_M,NZ  ,0xAE  -- AE: LDX abs
648     VAL     0,LST,XFR,Q_M,R_M,Ci_C,WS_X,OS_M,NZ  ,0xBE  -- BE: LDX abs,Y
649     VAL     1,IDC,DEC,Q_M,R_K,Ci_0,WS_P,    ,NZ  ,0x00  -- CE: DEC abs
650     VAL     1,IDC,DEC,Q_M,R_K,Ci_0,WS_P,    ,NZ  ,0x00  -- DE: DEC abs,X
651     VAL     1,IDC,INC,Q_M,R_K,Ci_1,WS_P,    ,NZ  ,0x00  -- EE: INC abs
652     VAL     1,IDC,INC,Q_M,R_K,Ci_1,WS_P,    ,NZ  ,0x00  -- FE: INC abs,X
653
654 -------------------------------------------------------------------------------
655
656     VAL     0,LU ,BBR,L_K,R_M,Ci_C,    ,    ,BEQ ,K_0   -- 0F:+BBR0 dp,rel
657     VAL     0,LU ,BBR,L_K,R_M,Ci_C,    ,    ,BEQ ,K_1   -- 1F:+BBR1 dp,rel
658     VAL     0,LU ,BBR,L_K,R_M,Ci_C,    ,    ,BEQ ,K_2   -- 2F:+BBR2 dp,rel
659     VAL     0,LU ,BBR,L_K,R_M,Ci_C,    ,    ,BEQ ,K_3   -- 3F:+BBR3 dp,rel
660     VAL     0,LU ,BBR,L_K,R_M,Ci_C,    ,    ,BEQ ,K_4   -- 4F:+BBR4 dp,rel
661     VAL     0,LU ,BBR,L_K,R_M,Ci_C,    ,    ,BEQ ,K_5   -- 5F:+BBR5 dp,rel
662     VAL     0,LU ,BBR,L_K,R_M,Ci_C,    ,    ,BEQ ,K_6   -- 6F:+BBR6 dp,rel
663     VAL     0,LU ,BBR,L_K,R_M,Ci_C,    ,    ,BEQ ,K_7   -- 7F:+BBR7 dp,rel
664     VAL     0,LU ,BBS,L_K,R_M,Ci_C,    ,    ,BNE ,K_0   -- 8F:+BBS0 dp,rel
665     VAL     0,LU ,BBS,L_K,R_M,Ci_C,    ,    ,BNE ,K_1   -- 9F:+BBS1 dp,rel
666     VAL     0,LU ,BBS,L_K,R_M,Ci_C,    ,    ,BNE ,K_2   -- AF:+BBS2 dp,rel
667     VAL     0,LU ,BBS,L_K,R_M,Ci_C,    ,    ,BNE ,K_3   -- BF:+BBS3 dp,rel
668     VAL     0,LU ,BBS,L_K,R_M,Ci_C,    ,    ,BNE ,K_4   -- CF:+BBS4 dp,rel
669     VAL     0,LU ,BBS,L_K,R_M,Ci_C,    ,    ,BNE ,K_5   -- DF:+BBS5 dp,rel
670     VAL     0,LU ,BBS,L_K,R_M,Ci_C,    ,    ,BNE ,K_6   -- EF:+BBS6 dp,rel
671     VAL     0,LU ,BBS,L_K,R_M,Ci_C,    ,    ,BNE ,K_7   -- FF:+BBS7 dp,rel
672
673 -------------------------------------------------------------------------------
674
675 _end:
```