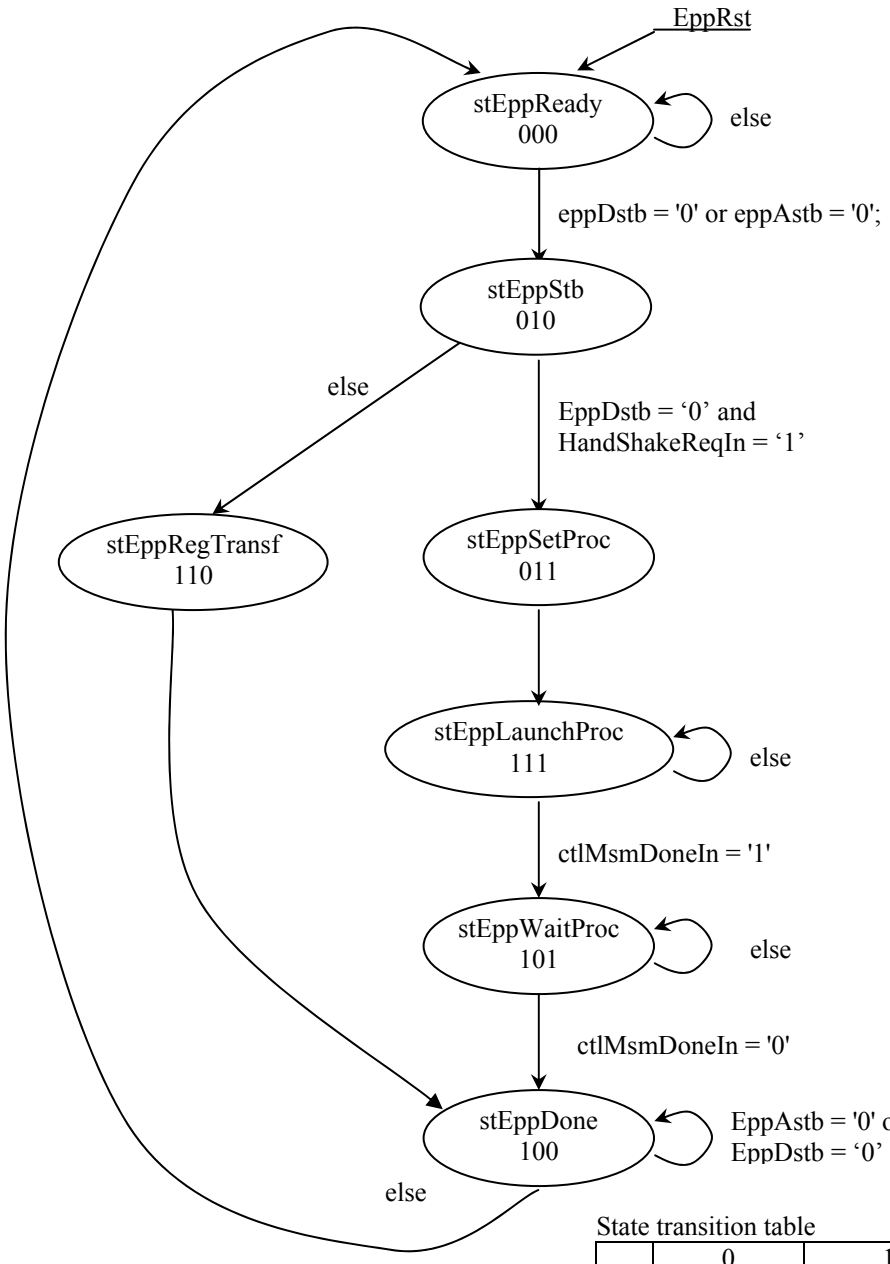
EppCtrl State diagram and state transition table



State transition table

	0	1
00	Ready ↓	
01	Stb ↓ →	SetProc ↓
11	RegTransf ↓	LaunchProc ↓
10	Done	WaitProc

Port signals

```
-- Epp-like bus signals
clk             : in std_logic;      -- system clock (50MHz)
EppAstb         : in std_logic;      -- Address strobe
EppDstb         : in std_logic;      -- Data strobe
EppWr           : in std_logic;      -- Port write signal
EppRst          : in std_logic;      -- Port reset signal
EppDB           : inout std_logic_vector(7 downto 0); -- port data bus
EppWait: out std_logic;              -- Port wait signal
-- User signals
busEppOut       : out std_logic_vector(7 downto 0);-- Data Output bus
busEppIn        : in std_logic_vector(7 downto 0);   -- Data Input bus
ctlEppDwrOut    : out std_logic;     -- Data Write pulse
ctlEppRdCycleOut: inout std_logic;-- Indicates a READ Epp cycle
regEppAdrOut    : inout std_logic_vector(7 downto 0) := "00000000"; -- Epp Address Register content
HandShakeReqIn : in std_logic;       -- User Handshake Request
ctlEppStartOut  : out std_logic;     -- Automatic process Start
ctlEppDoneIn    : in std_logic       -- Automatic process Done
```

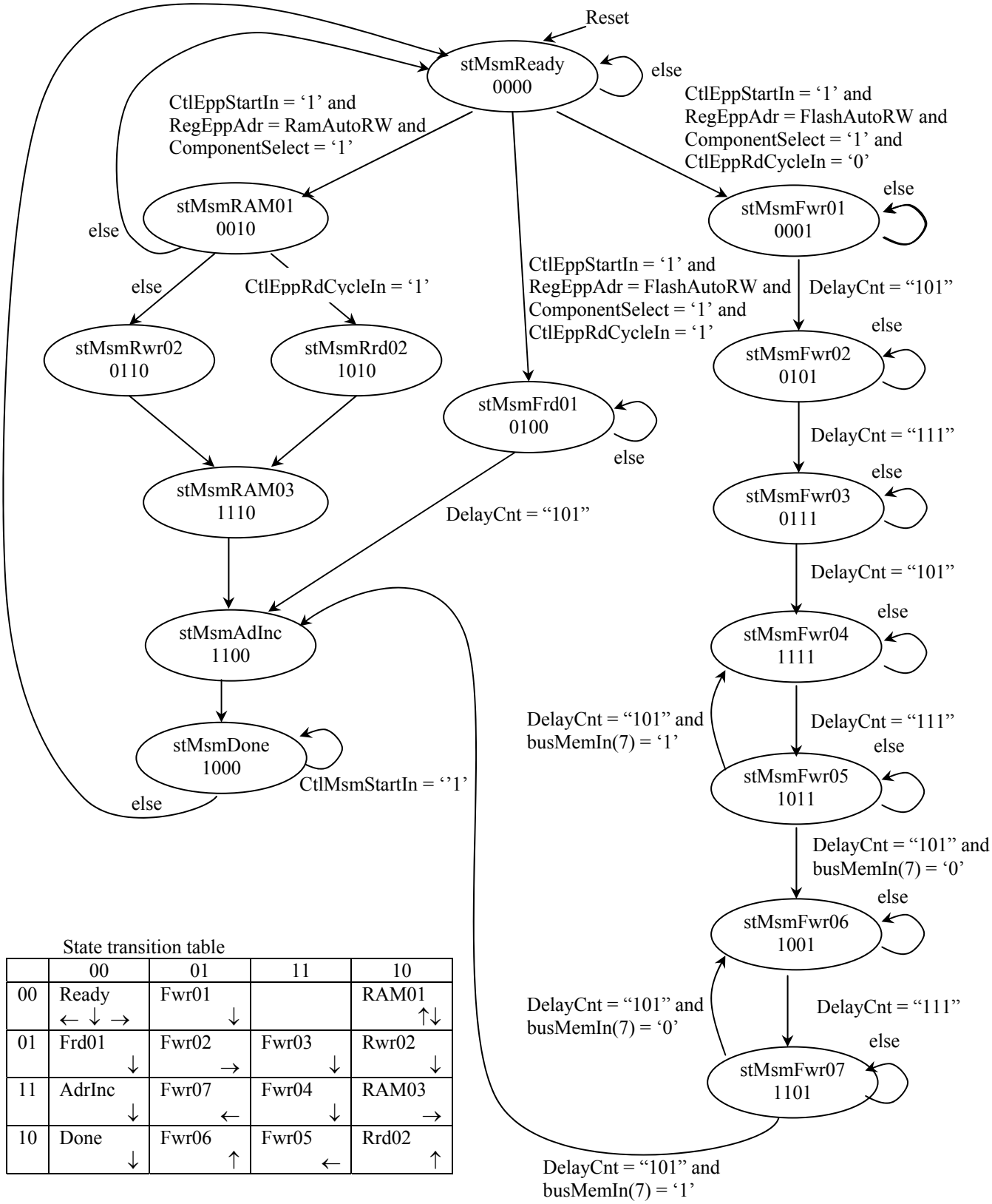
Internal Signals

```
busEppInternal: std_logic_vector(7 downto 0);
ctlEppAwr       : std_logic;
```

Signal Assignments:

```
-- Synchronized Epp outputs:
        process(clk)
        begin
                if clk'event and clk='1' then
                        if stEppCur = stEppReady then
                                ctlEppRdCycleOut <= '0';
                        elsif stEppCur = stEppStb then
                                ctlEppRdCycleOut <= EppWr;      -- not equivalent to EppWr due to
default state
                        end if;
                end if;
        end process;

        busEppOut <= EppDB;

        EppDB <=        busEppInternal when (ctlEppRdCycleOut = '1') else "ZZZZZZZZ";

        busEppInternal <= regEppAdrOut when EppAstb = '0' else busEppIn;

        -- Epp State machine related signals

        EppWait <= '1' when stEppCur = stEppDone else '0';
        ctlEppAwr  <= '1' when stEppCur = stEppRegTransf and EppAstb = '0' and EppWr = '0' else '0';
        ctlEppDwrOut  <= '1' when (stEppCur = stEppRegTransf or stEppCur = stEppSetProc)
                                                and EppDstb = '0' and EppWr = '0' else '0';
        ctlEppStartOut <= '1' when stEppCur = stEpplaunchProc else '0';
```

C1MemCtrl State diagram and state transition table



State transition table

	00	01	11	10
00	Ready ← ↓ →	Fwr01 ↓		RAM01 ↑↓
01	Frd01 ↓	Fwr02 →	Fwr03 ↓	Rwr02 ↓
11	AdrInc ↓	Fwr07 ←	Fwr04 ↓	RAM03 →
10	Done ↓	Fwr06 ↑	Fwr05 ←	Rrd02 ↑

Port signals

```
ck                  : in std_logic;                          -- system clock (50MHz)

-- Epp interface signals
HandShakeReqOut:    out std_logic;                           -- User Handshake Request
ctlMsmStartIn:      in std_logic;                            -- Automatic process Start
ctlMsmDoneOut:      out std_logic;                           -- Automatic process Done
ctlMsmDwrIn:        in std_logic;                            -- Data Write pulse
ctlEppRdCycleIn:    in std_logic;                            -- Indicates a READ Epp cycle
EppRdDataOut:       out std_logic_vector(7 downto 0);        -- Data Input bus
EppWrDataIn:        in std_logic_vector(7 downto 0);         -- Data Output bus
regEppAdrIn:      in std_logic_vector(7 downto 0) ;  -- Epp Address Register content (bits 7:3 ignored)
ComponentSelect: in std_logic;                             -- active HIGH, selects the current MemCtrl instance

-- Memory bus signals
MemDB           : inout std_logic_vector(7 downto 0); -- Memory data bus
MemAdr          : out std_logic_vector(18 downto 0);  -- Memory Address bus
RamCS           : out std_logic;    -- RAM CS
FlashCS         : out std_logic;    -- Flash CS
MemWR           : out std_logic;    -- memory write
MemOE           : out std_logic     -- memory read (Output Enable), also controls the MemDB direction
```

Internal Signals

```
signal    DelayCnt          : std_logic_vector(2 downto 0);

-- Signals dealing with memory chips
signal    regMemCtl:        std_logic_vector(4 downto 0) := "11111";-- Memory Control register
signal    regMemAdr:        std_logic_vector(18 downto 0);      -- Memory Address register
signal    carryoutL:        std_logic := '0';                   -- Carry out for memory address low byte
signal    carryoutM:        std_logic := '0';                   -- Carry out for memory address middle byte
signal    regMemWrData:     std_logic_vector(7 downto 0);       -- Memory Write Data register
signal    regMemRdData:     std_logic_vector(7 downto 0);       -- Memory Read Data register
signal    busMemIn:         std_logic_vector(7 downto 0);
signal    busMemOut:        std_logic_vector(7 downto 0);

-- Signals in the memory control register
signal    ctlMcrOe          : std_logic;        -- Output enable (read strobe)
signal    ctlMcrWr          : std_logic;        -- Write enable (write strobe)
signal    ctlMcrRAMCs       : std_logic;        -- RAM chip select
signal    ctlMcrFlashCs     : std_logic;        -- Flash chip select
signal    ctlMcrDir         : std_logic;        -- composed out of previous ones

-- Signals used by Memory control state machine
signal    ctlMsmOe          : std_logic;
signal    ctlMsmWr          : std_logic;
signal    ctlMsmRAMCs       : std_logic;
signal    ctlMsmFlashCs     : std_logic;
signal    ctlMsmDir         : std_logic;
signal    ctlMsmAdrInc      : std_logic;
signal    ctlMsmWrCmd       : std_logic;
```

Signal assignment

```
-- Memory signals
--          Memory control register
            ctlMcrOe        <= regMemCtl(0);        -- Output enable (read strobe)
            ctlMcrWr        <= regMemCtl(1);        -- Write enable (write strobe)
            ctlMcrRAMCs     <= regMemCtl(2);        -- RAM chip select
            ctlMcrFlashCs   <= regMemCtl(3);        -- Flash chip select

-- Memory control bus driven either by automatic state machine or by memory control register
            RamCS           <= ctlMsmRAMCs and ctlMcrRAMCs;     -- PC generated RAM CS;
            FlashCS         <= ctlMsmFlashCs and ctlMcrFlashCs;     -- PC generated Flash CS;
            MemWR           <= ctlMsmWr and ctlMcrWr;           -- PC generated MemWr;
            MemOE           <= ctlMsmOe and ctlMcrOe;           -- PC generated MemOe;
            busMemIn        <= MemDB;
            busMemOut       <= "01000000" when ctlMsmWrCmd = '1' else regMemWrData;
            MemAdr          <= regMemAdr;
            ctlMcrDir       <= ctlMcrOe and ((not ctlMcrFlashCs) or (not ctlMcrRAMCs));
            MemDB           <= busMemOut when (ctlMsmDir = '1' or ctlMcrDir = '1') else  "ZZZZZZZZ";

-- Handshake signal
HandShakeReqOut <=
'1'         when (regEppAdrIn(2 downto 0) = RamAutoRW or regEppAdrIn(2 downto 0) = FlashAutoRW)
            and ComponentSelect = '1' else
'0';

-- Delay Counter
process (clk)
            begin
                        if clk'event and clk = '1' then
                                    if stMsmCur = stMsmReady then DelayCnt <= "000";
                                    else DelayCnt <= DelayCnt + 1;
                                    end if;
                        end if;
end process;

-- Memory Control Register
process (clk, ctlMsmDwrIn)
begin
if clk = '1' and clk'Event then
  if ctlMsmDwrIn = '1' and regEppAdrIn(2 downto 0) = MemCtrlReg and ComponentSelect = '1' then
            regMemCtl <= EppWrDataIn(4 downto 0);
  end if;
end if;
end process;

-- Memory Address Register/Counter
MsmAdrL:    process (clk, ctlMsmDwrIn, ctlMsmAdrInc)
begin
if clk = '1' and clk'Event then
   if ctlMsmAdrInc = '1' then
            regMemAdr(7 downto 0) <= regMemAdr(7 downto 0) + 1; --"00000001";
   elsif ctlMsmDwrIn = '1' and regEppAdrIn(2 downto 0) = MemAdrL and ComponentSelect = '1' then
            regMemAdr(7 downto 0) <= EppWrDataIn;
   end if;
end if;
end process;
```

```
carryoutL <= '1' when regMemAdr(7 downto 0) = "11111111" else '0';

MsmAdrM:        process (clk, ctlMsmDwrIn, ctlMsmAdrInc)
begin
if clk = '1' and clk'Event then
  if ctlMsmAdrInc = '1' and carryoutL = '1' then
        regMemAdr(15 downto 8) <= regMemAdr(15 downto 8) + 1; --"00000001";
  elsif ctlMsmDwrIn = '1' and regEppAdrIn(2 downto 0) = MemAdrM and ComponentSelect = '1' then
        regMemAdr(15 downto 8) <= EppWrDataIn;
  end if;
end if;
end process;
carryoutM <= '1' when regMemAdr(15 downto 8) = "11111111" else '0';

MsmAdrH:        process (clk, ctlMsmDwrIn, ctlMsmAdrInc)
begin
if clk = '1' and clk'Event then
  if ctlMsmAdrInc = '1' and carryoutL = '1' and carryoutM = '1' then
        regMemAdr(18 downto 16) <= regMemAdr(18 downto 16) + 1; --"001";
  elsif ctlMsmDwrIn = '1' and regEppAdrIn(2 downto 0) = MemAdrH and ComponentSelect = '1' then
        regMemAdr(18 downto 16) <= EppWrDataIn(2 downto 0);
  end if;
end if;
end process;

-- Memory write data holding register
process (clk, ctlMsmDwrIn)
begin
if clk = '1' and clk'Event then
  if ctlMsmDwrIn = '1' and
        (regEppAdrIn(2 downto 0) = RamAutoRW or
         regEppAdrIn(2 downto 0) = FlashAutoRW or
         regEppAdrIn(2 downto 0) = MemDataWr) and ComponentSelect = '1' then
                regMemWrData <= EppWrDataIn;
  end if;
end if;
end process;

-- Memory read register: - holds data after an automatic read
process (clk)
begin
if clk = '1' and clk'Event then
  if    stMsmCur = stMsmFrd01 or stMsmCur = stMsmRrd02 then
                regMemRdData <= busMemIn;
  end if;
end if;
end process;
```

Combinatorial Moore Outputs tables

Signal	stMsmFrd01 or stMsmFwr05 or stMsmFwr07 or stMsmFrd02	Others
ctlMsmOe	‘0’	‘1’

Signal	stMsmFwr01 or stMsmFwr03 or stMsmRwr02	Others
ctlMsmWr	‘0’	‘1’

Signal	stMsmRAM01 or stMsmRAM03 or stMsmRwr02 or stMsmRrd02	Others
ctlMsmRAMCs	‘0’	‘1’

Signal	stMsmFwr0X or stMsmFrd01	Others
ctlMsmFlashCs	‘0’	‘1’

Signal	stMsmFwr01 or stMsmFwr02 or stMsmFwr03 or stMsmRwr02	Others
ctlMsmDir	‘1’	‘0’

Signal	stMsmAdInc	Others
ctlAdrInc	‘1’	‘0’

Signal	stMsmFwr01 or stMsmFwr02	Others
ctlMsmWrCmd	‘1’	‘0’

Signal	stMsmDone	Others
ctlMsmDoneOut	‘1’	‘0’