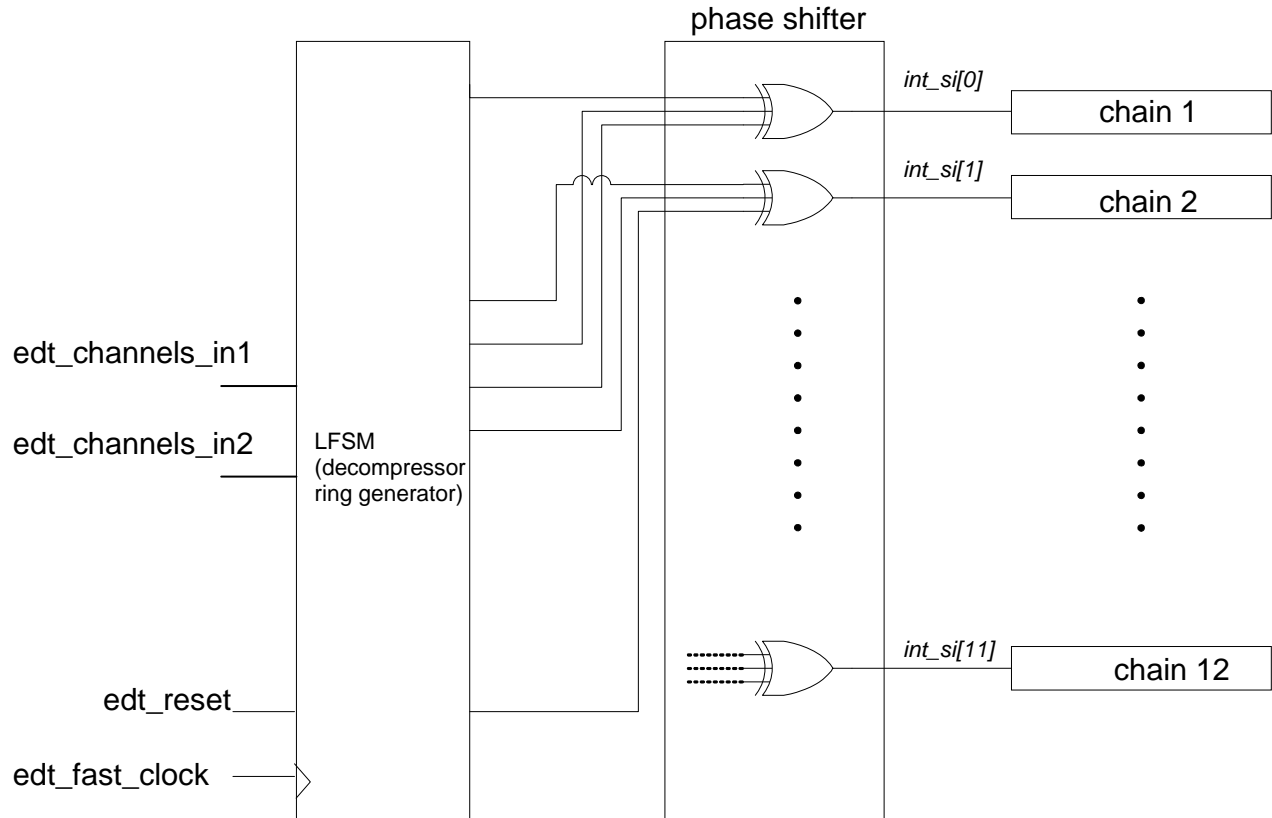


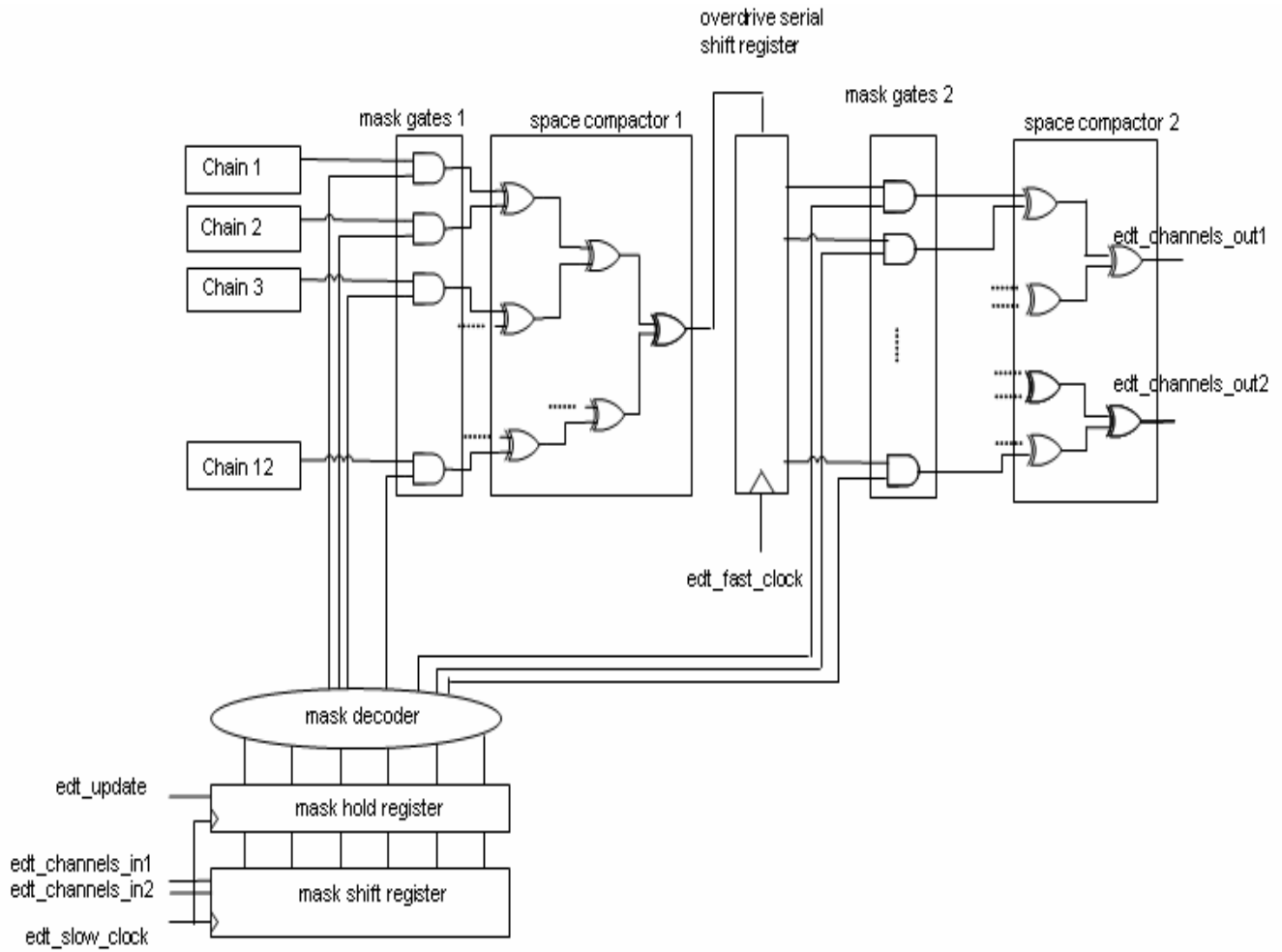
Accellera OCI example  
CTL describing "TestKompres Xpress with Overdrive"-like  
compression circuitry

Last updated January 25, 2008

Decompressor



# Compactor (including mask logic)



```

// Accellera OCI example based on TestKompress Xpress with Overdrive
// like compression circuitry.
//
// TestKompress Xpress compression is covered by several U.S. and
// International patents held by Mentor Graphics Corporation. They
// include coverage of the compression method, the decompressor,
// and space compactors with masking.
//
// New OCI syntax is shown in bold.
//
// For simplicity, test signals are not shared or internally generated.
//
// Terminology:
// - Chains: Refers to internal scan chains.
// - Channels: Refers to external compressed "virtual" scan chains.
//
// Design characteristics:
// - 2 compressed scan channels
// - 12 internal scan chains
// - 64 Mux-D scan cells per scan chain
// - 8x Overdrive
// - Design functional pins: fin[2:0], fout[1:0], clk

```

```

STIL 1.0 {
  Design 2005;
  CTL 2005;
}

```

```

Header {
  Title "CTL describing TestKompress Xpress with Overdrive compression
circuitry";
  Date "Mon Jan 24 20:23:05 PST 2008";
}

```

```

Signals {

  // functional signals
  //
  clk      In;
  fin[2..0] In;
  fout[1..0] Out;

  // test signals
  //
  edt_slow_clock    In;
  edt_fast_clock    In;
  edt_reset         In;
  edt_update        In;
  scan_en           In;
  edt_channels_in1  In;    // ScanIn, scan length defined in ScanGroups
  edt_channels_in2  In;    // ScanIn, scan length defined in ScanGroups
  edt_channels_out1 Out;   // ScanOut, scan length defined in ScanGroups
  edt_channels_out2 Out;   // ScanOut, scan length defined in ScanGroups

```

```

// internal signals
//
int_si[0..11]          Pseudo;
int_so[0..11]          Pseudo;
}

SignalGroups {
// Lengths of compressed scan input and output signals must be the same.
// Patterns generated must be padded.
// Specifying "12" below is redundant since this is specified
// in DataRateForProtocol.
_si_ = 'edt_channels_in1 + edt_channels_in2' { ScanIn 12; }
_so_ = 'edt_channels_out1 + edt_channels_out2' { ScanOut 12; }
_clks_ = ' edt_fast_clock + edt_slow_clock + clk';
_pi_ = '_si_ + edt_reset + edt_update + scan_en + fin[2..0]';
_po_ = '_so_ + fout[1..0]';
}

Timing comp_timing {
WaveformTable time {
Period '100ns';
Waveforms {
_pi_ {01X { '0ns' D/U/N; }}
_clks_ {0P { '0ns' D; '40ns' D/U; '50ns' D; }}
_po_ {LHXZ { '0ns' X; '10ns' l/h/X/t; '40ns' X; }}
}
}
}

MacroDefs comp_macros {
load_unload_and_update_masks {
W time;
C { edt_update = 1; edt_reset = 1; scan_en = 1; }
V { clk = 0; edt_fast_clock = P; edt_slow_clock = P; }
C { edt_update = 0; edt_reset = 0; scan_en = 1; }
Shift {
V { clk = P; edt_fast_clock = P; edt_slow_clock = P; _si_ = #; _so_ = #; }
V { clk = P; edt_fast_clock = P; edt_slow_clock = 0; }
V { clk = P; edt_fast_clock = P; edt_slow_clock = 0; }
V { clk = P; edt_fast_clock = P; edt_slow_clock = 0; }
V { clk = P; edt_fast_clock = P; edt_slow_clock = 0; }
V { clk = P; edt_fast_clock = P; edt_slow_clock = 0; }
V { clk = P; edt_fast_clock = P; edt_slow_clock = 0; }
V { clk = P; edt_fast_clock = P; edt_slow_clock = 0; }
}
}

combinational_pattern {
Macro load_unload_and_update_masks { _si_ = #; _so_ = #; }
W time;
C { edt_fast_clock = 0; edt_slow_clock = 0; }
V { _pi_ = #; clk = #; _po_ = #; }
}
}

```

```

ScanStructures comp_internal_chains {
  ScanChain chain1 { ScanCells      chain1_cells[63..0];
                    ScanLength     64;
                    ScanIn          int_si[0];
                    ScanOut         int_so[0];
                    ScanMasterClock clk; }
  ScanChain chain2 { ScanCells      chain2_cells[63..0];
                    ScanLength     64;
                    ScanIn          int_si[1];
                    ScanOut         int_so[1];
                    ScanMasterClock clk; }
  ScanChain chain3 { ScanCells      chain3_cells[63..0];
                    ScanLength     64;
                    ScanIn          int_si[2];
                    ScanOut         int_so[2];
                    ScanMasterClock clk; }
  ScanChain chain4 { ScanCells      chain4_cells[63..0];
                    ScanLength     64;
                    ScanIn          int_si[3];
                    ScanOut         int_so[3];
                    ScanMasterClock clk; }
  ScanChain chain5 { ScanCells      chain5_cells[63..0];
                    ScanLength     64;
                    ScanIn          int_si[4];
                    ScanOut         int_so[4];
                    ScanMasterClock clk; }
  ScanChain chain6 { ScanCells      chain6_cells[63..0];
                    ScanLength     64;
                    ScanIn          int_si[5];
                    ScanOut         int_so[5];
                    ScanMasterClock clk; }
  ScanChain chain7 { ScanCells      chain7_cells[63..0];
                    ScanLength     64;
                    ScanIn          int_si[6];
                    ScanOut         int_so[6];
                    ScanMasterClock clk; }
  ScanChain chain8 { ScanCells      chain8_cells[63..0];
                    ScanLength     64;
                    ScanIn          int_si[7];
                    ScanOut         int_so[7];
                    ScanMasterClock clk; }
  ScanChain chain9 { ScanCells      chain9_cells[63..0];
                    ScanLength     64;
                    ScanIn          int_si[8];
                    ScanOut         int_so[8];
                    ScanMasterClock clk; }
  ScanChain chain10 { ScanCells     chain10_cells[63..0];
                    ScanLength     64;
                    ScanIn          int_si[9];
                    ScanOut         int_so[9];
                    ScanMasterClock clk; }
  ScanChain chain11 { ScanCells     chain11_cells[63..0];
                    ScanLength     64;
                    ScanIn          int_si[10];
                    ScanOut         int_so[10];
                    ScanMasterClock clk; }
}

```

```

ScanChain chain12 { ScanCells      chain12_cells[63..0];
                   ScanLength    64;
                   ScanIn        int_si[11];
                   ScanOut       int_so[11];
                   ScanMasterClock clk; }
}

CompressionStructures comp_compstructures {
  Decompressor decompressor {
    Vendor MentorGraphics {
      // Vendor-specific information. Ignored by other vendors.
      ip_version = "1";
      n_channels = "2";
      ...
    }

    // LFSM (decompressor ring generator)
    //
    State lfsm[0] { Data '~edt_reset & (lfsm[1])';
                  Clock edt_fast_clock; }
    State lfsm[1] { Data '~edt_reset & (lfsm[2] ^ edt_channels_in2)';
                  Clock edt_fast_clock; }
    State lfsm[2] { Data '~edt_reset & (lfsm[3])';
                  Clock edt_fast_clock; }
    State lfsm[3] { Data '~edt_reset & (lfsm[4] ^ lfsm[3] ^ edt_channels_in2)';
                  Clock edt_fast_clock; }
    State lfsm[4] { Data '~edt_reset & (lfsm[5] ^ lfsm[2] ^ lfsm[3])';
                  Clock edt_fast_clock; }
    State lfsm[5] { Data '~edt_reset & (lfsm[6] ^ lfsm[2] ^ edt_channels_in1)';
                  Clock edt_fast_clock; }
    State lfsm[6] { Data '~edt_reset & (lfsm[7] ^ lfsm[1])';
                  Clock edt_fast_clock; }
    State lfsm[7] { Data '~edt_reset & (lfsm[0] ^ edt_channels_in1)';
                  Clock edt_fast_clock; }

    // phase shifter
    //
    int_si[0] 'lfsm[0] ^ lfsm[3] ^ lfsm[4]';
    int_si[1] 'lfsm[5] ^ lfsm[6] ^ lfsm[7]';
    int_si[2] 'lfsm[2] ^ lfsm[3] ^ lfsm[7]';
    int_si[3] 'lfsm[0] ^ lfsm[1] ^ lfsm[6]';
    int_si[4] 'lfsm[1] ^ lfsm[2] ^ lfsm[4]';
    int_si[5] 'lfsm[0] ^ lfsm[2] ^ lfsm[5]';
    int_si[6] 'lfsm[4] ^ lfsm[5] ^ lfsm[6]';
    int_si[7] 'lfsm[0] ^ lfsm[6] ^ lfsm[7]';
    int_si[8] 'lfsm[1] ^ lfsm[5] ^ lfsm[7]';
    int_si[9] 'lfsm[2] ^ lfsm[3] ^ lfsm[4]';
    int_si[10] 'lfsm[3] ^ lfsm[6] ^ lfsm[7]';
    int_si[11] 'lfsm[0] ^ lfsm[2] ^ lfsm[3]';
  }
}

Compressor compressor {
  Vendor MentorGraphics;

  // mask shift register
  //
  State mask_shift[5] { Data edt_channels_in1; Clock edt_slow_clock; }
}

```

```

State mask_shift[4] { Data mask_shift[5];      Clock edt_slow_clock; }
State mask_shift[3] { Data mask_shift[4];      Clock edt_slow_clock; }
State mask_shift[2] { Data edt_channels_in2;   Clock edt_slow_clock; }
State mask_shift[1] { Data mask_shift[2];      Clock edt_slow_clock; }
State mask_shift[0] { Data mask_shift[1];      Clock edt_slow_clock; }

// mask hold register
//
State mask_hold[0] { Data 'edt_update ? mask_shift[0] : mask_hold[0]';
                   Clock edt_slow_clock; }
State mask_hold[1] { Data 'edt_update ? mask_shift[1] : mask_hold[1]';
                   Clock edt_slow_clock; }
State mask_hold[2] { Data 'edt_update ? mask_shift[2] : mask_hold[2]';
                   Clock edt_slow_clock; }
State mask_hold[3] { Data 'edt_update ? mask_shift[3] : mask_hold[3]';
                   Clock edt_slow_clock; }
State mask_hold[4] { Data 'edt_update ? mask_shift[4] : mask_hold[4]';
                   Clock edt_slow_clock; }
State mask_hold[5] { Data 'edt_update ? mask_shift[5] : mask_hold[5]';
                   Clock edt_slow_clock; }

// mask1 decoder
//
Wire channell_chain_masks[0] 'Function of mask_hold[0..5]';
Wire channell_chain_masks[1] 'Function of mask_hold[0..5]';
Wire channell_chain_masks[2] 'Function of mask_hold[0..5]';
Wire channell_chain_masks[3] 'Function of mask_hold[0..5]';
Wire channell_chain_masks[4] 'Function of mask_hold[0..5]';
Wire channell_chain_masks[5] 'Function of mask_hold[0..5]';
Wire channell_chain_masks[6] 'Function of mask_hold[0..5]';
Wire channell_chain_masks[7] 'Function of mask_hold[0..5]';
Wire channell_chain_masks[8] 'Function of mask_hold[0..5]';
Wire channell_chain_masks[9] 'Function of mask_hold[0..5]';
Wire channell_chain_masks[10] 'Function of mask_hold[0..5]';
Wire channell_chain_masks[11] 'Function of mask_hold[0..5]';

// space compactor 1 (XOR trees)
//
Wire space_compactor_out1 = '(int_so[0] & channell_chain_masks[0]) ^
                             (int_so[1] & channell_chain_masks[1]) ^
                             (int_so[2] & channell_chain_masks[2]) ^
                             (int_so[3] & channell_chain_masks[3]) ^
                             (int_so[4] & channell_chain_masks[4]) ^
                             (int_so[5] & channell_chain_masks[5]) ^
                             (int_so[6] & channell_chain_masks[6]) ^
                             (int_so[7] & channell_chain_masks[7]) ^
                             (int_so[8] & channell_chain_masks[8]) ^
                             (int_so[9] & channell_chain_masks[9]) ^
                             (int_so[10] & channell_chain_masks[10]) ^
                             (int_so[11] & channell_chain_masks[11])';

State overdrive_reg[0] { Data space_compactor_out1; Clock edt_fast_clock; }
State overdrive_reg[1] { Data overdrive_reg[0];      Clock edt_fast_clock; }
State overdrive_reg[2] { Data overdrive_reg[1];      Clock edt_fast_clock; }
State overdrive_reg[3] { Data overdrive_reg[2];      Clock edt_fast_clock; }
State overdrive_reg[4] { Data overdrive_reg[3];      Clock edt_fast_clock; }
State overdrive_reg[5] { Data overdrive_reg[4];      Clock edt_fast_clock; }

```

```

State overdrive_reg[6] { Data overdrive_reg[5];      Clock edt_fast_clock; }
State overdrive_reg[7] { Data overdrive_reg[6];      Clock edt_fast_clock; }

Wire channel2_chain_masks[0] 'Function of mask_hold[0..5]';
Wire channel2_chain_masks[1] 'Function of mask_hold[0..5]';
Wire channel2_chain_masks[2] 'Function of mask_hold[0..5]';
Wire channel2_chain_masks[3] 'Function of mask_hold[0..5]';
Wire channel2_chain_masks[4] 'Function of mask_hold[0..5]';
Wire channel2_chain_masks[5] 'Function of mask_hold[0..5]';
Wire channel2_chain_masks[6] 'Function of mask_hold[0..5]';
Wire channel2_chain_masks[7] 'Function of mask_hold[0..5]';

// space compactor 2 (XOR trees)
//
edt_channels_out1 = '(overdrive_reg[0] & channel2_chain_masks[0]) ^
                    (overdrive_reg[1] & channel2_chain_masks[1]) ^
                    (overdrive_reg[2] & channel2_chain_masks[2]) ^
                    (overdrive_reg[3] & channel2_chain_masks[3])';

edt_channels_out2 = '(overdrive_reg[4] & channel2_chain_masks[4]) ^
                    (overdrive_reg[5] & channel2_chain_masks[5]) ^
                    (overdrive_reg[6] & channel2_chain_masks[6]) ^
                    (overdrive_reg[7] & channel2_chain_masks[7])';
}
}

```

```
Environment edt_compression {
```

```
  CTLMode comp_atpg {
```

```
    DomainReferences {
```

```
      Timing           comp_timing;
      MacroDefs        comp_macros;
      ScanStructures   comp_internal_chains;
      CompressionStructures comp_compstructures;
    }

```

```
    Internal {
```

```
      _si_ { DataType TestData ScanDataIn { DataRateForProtocol Minimum 12; } }
      _so_ { DataType TestData ScanDataOut { DataRateForProtocol Minimum 12; } }
      scan_en { DataType TestControl ScanEnable
                { ActiveState ForceUp; AssumedInitialState ForceDown; } }
      clk { DataType TestControl ScanMasterClock MasterClock CaptureClock
            { ActiveState U; AssumedInitialState D; } }
      edt_fast_clock { DataType TestControl ScanMasterClock MasterClock
                      { ActiveState U; AssumedInitialState D;
                        ConstrainedDuringCapture D; } }
      edt_slow_clock { DataType TestControl ScanMasterClock MasterClock
                      { ActiveState U; AssumedInitialState D;
                        ConstrainedDuringCapture D; } }
      edt_update { DataType TestControl
                   { ActiveState U; AssumedInitialState D; } }
      edt_reset { DataType TestControl
                  { ActiveState U; AssumedInitialState D; } }
    }

```

```

// Specify padding rules for compressed scan channels
//
External {
    _si_ { PadType PrePad; PadValue D; }
    _so_ { PadType PostPad; PadValue X; }
}

// Define anchor points for internal scan inputs and outputs
//
NameMaps scan_chain_anchor_points {
    Signals {
        int_si[0..11]   edt.scan_in[0..11];
        int_so[0..11]   edt.scan_out[0..11];
    }
}

PatternInformation {
    Macro load_unload_and_update_masks { Purpose ControlObserve LoadMask; }
    Macro combinational_pattern       { Purpose DoTestOverlap LoadMask; }
    PatternExec scanexec;
}
}

PatternExec scanexec {
    Timing      comb_timing;
    PatternBurst scanpats;
}

PatternBurst scanpats {
    Purpose Scan;
    Identifiers {
        EventType TestPatternUnit {
            Label pattern { Prefix; Begin; }
        }
    }
    PatList {
        pattern_0 { Protocol Macro "combinational_pattern"; }
    }
}

Pattern pattern_0 {
    Ann {* Pattern:0 *}
    "pattern 0": P {
        edt_channels_in1 = 000000110000;
        edt_channels_out1 = XXXXXXXXXXXXX;
        edt_channels_in2 = 000001010000;
        edt_channels_out2 = XXXXXXXXXXXXX;
        _pi_ = XXXX0XXX;
        clk = P;
        _po_ = XXXX;
    }
    Ann {* Pattern:1 *}
}

```

```
"pattern 1": P {  
  edt_channels_in1 = 101100011000;  
  edt_channels_out1 = XLLLHHHLHXXX;  
  edt_channels_in2 = 000011100110;  
  edt_channels_out2 = XLLHLHHLLXXX;  
  _pi_ = XXXX0XXX;  
  clk = P;  
  _po_ = XXXX;  
}  
}
```