Bottom of Chip

Gerrit Meddeler

June 6, 2001

Bottom of Chip

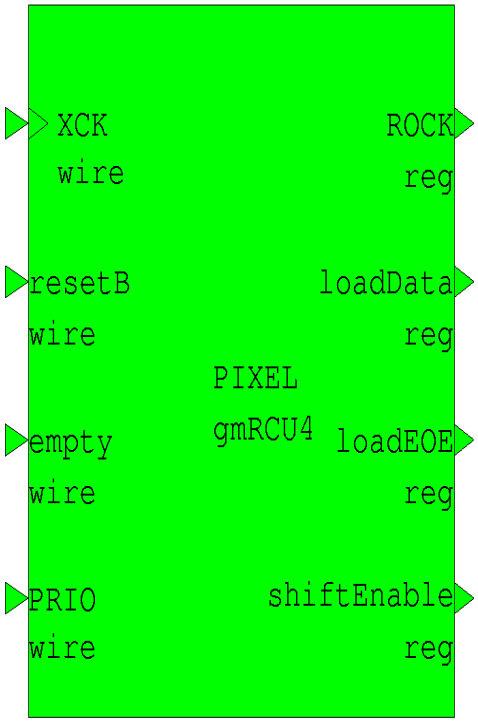
Basic Functions

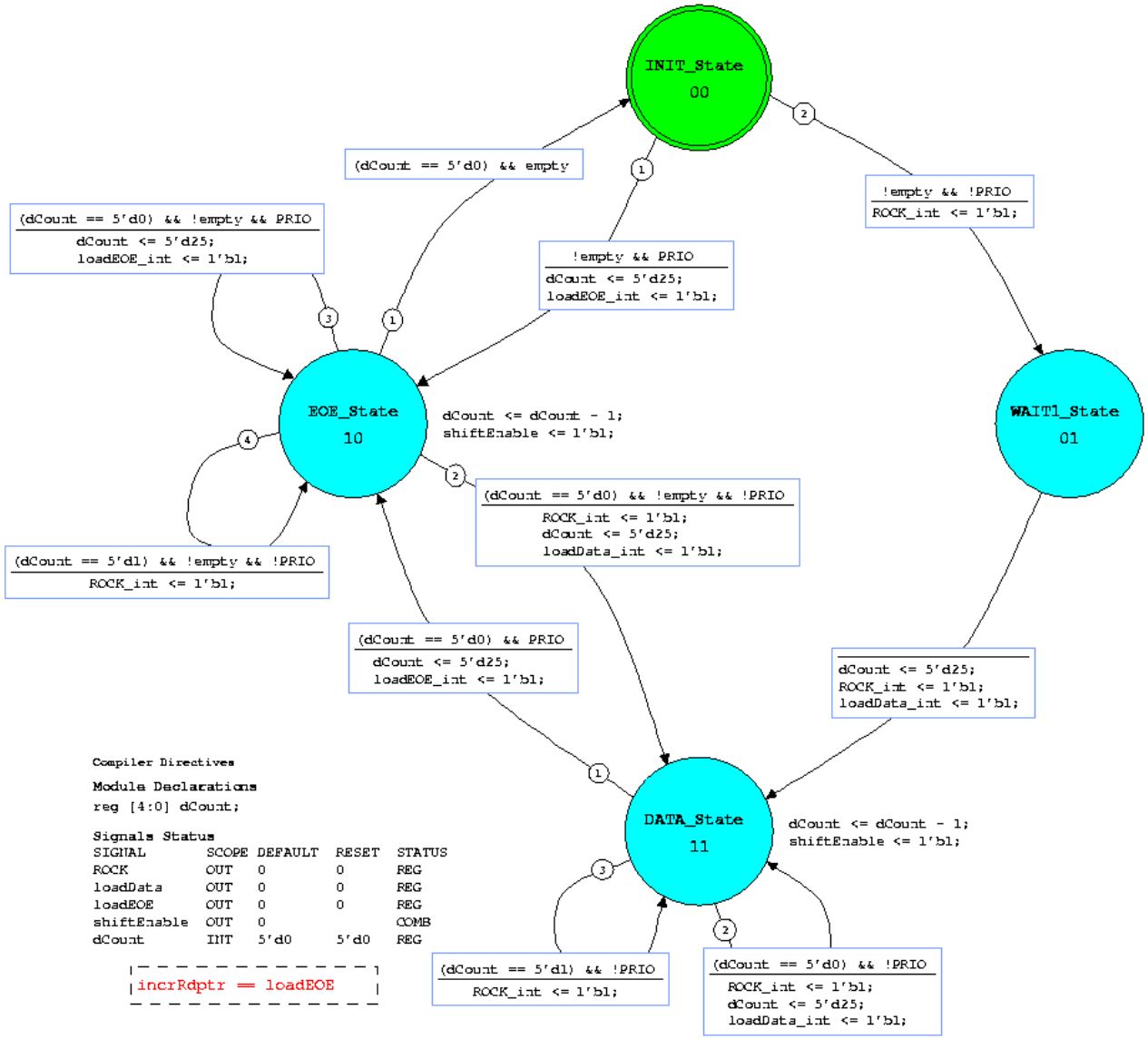
- Read Out Controller
- CEU Clock Generator
- Command Decoder
- Configuration Register
- Time Stamp Generator
- Reset Generator
- Test Logic

Read Out Controller

- Readout Controller Statemachine
- Serializer
- Trigger Manager
- Overflow Handler
- Trigger Address Index Counter (TAI/TAC)

Read Out Controller Statemachine





Compiler Directives

Module Declarations

```
reg [4:0] dCount;
```

Signals Status

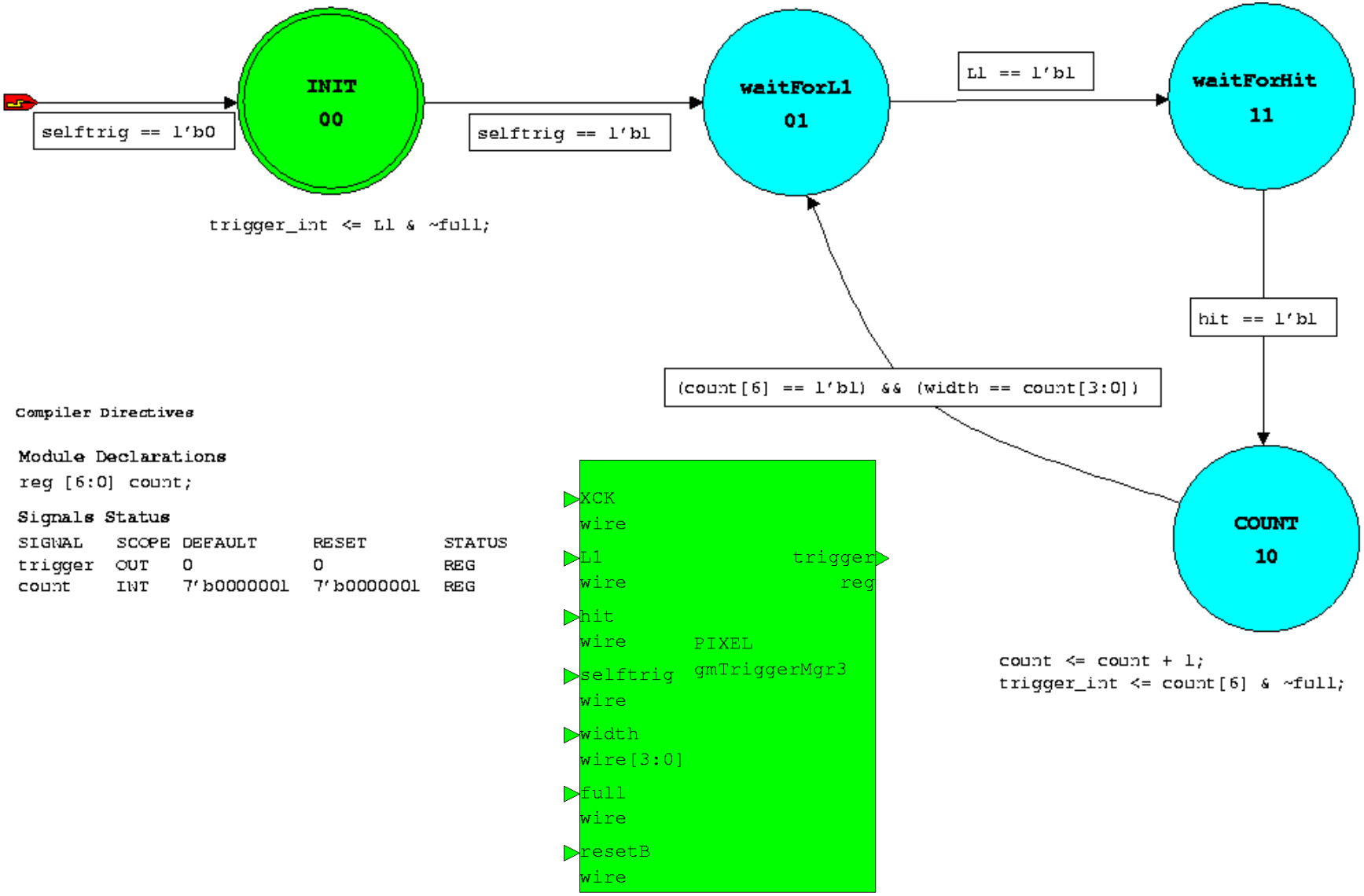
SIGNAL	SCOPE	DEFAULT	RESET	STATUS
ROCK	OUT	0	0	REG
loadData	OUT	0	0	REG
loadEOE	OUT	0	0	REG
shiftEnable	OUT	0		COMB
dCount	INT	5'd0	5'd0	REG

incrRdptr == loadEOE

Serializer

```
module gmSDOut2(SDO, XCK, resetB, loadData, loadEOE, ADDRout, TAC,
        Tot, shiftEnable, fifoErr, parity_warning);
output       SDO;
input        XCK, resetB, loadData, loadEOE, shiftEnable,
            fifoErr, parity_warning;
input [12:0] ADDRout;
input  [7:0] Tot;
input  [3:0] TAC;

reg  [25:0] shiftOut;

wire        fErrPar;
wire [25:0] dataInt;

assign SDO = shiftOut[25];
assign fErrPar = ~(fifoErr | parity_warning);

assign dataInt = {1'b1, TAC[3:0],
  loadEOE ? {3'b111, fErrPar, 2'b00, parity_warning, fifoErr} : ADDRout[7:0], ADDRout[12:8],Tot[7:0]};

always @(posedge XCK or negedge resetB) begin
        if (!resetB) shiftOut <= 26'b0;
        else if ( loadData | loadEOE ) shiftOut <= dataInt;
        else if ( shiftEnable ) shiftOut <= shiftOut << 1;
        else shiftOut <= 26'b0;
end

endmodule
```

Trigger Manager + Self Trigger



Overflow Handler

```
module gmOverflowHandler(XCK, resetB, Ovfl, Error);
output Error;
input  XCK, resetB, Ovfl;

reg [7:0] q;

assign Error = ~q[7];

always @(posedge XCK or negedge resetB)
    if (!resetB)
        q = 8'b10000000;
    else if (Ovfl)
        q = 8'b00000000;
    else if (Error)
        q = q + 1;

endmodule
```

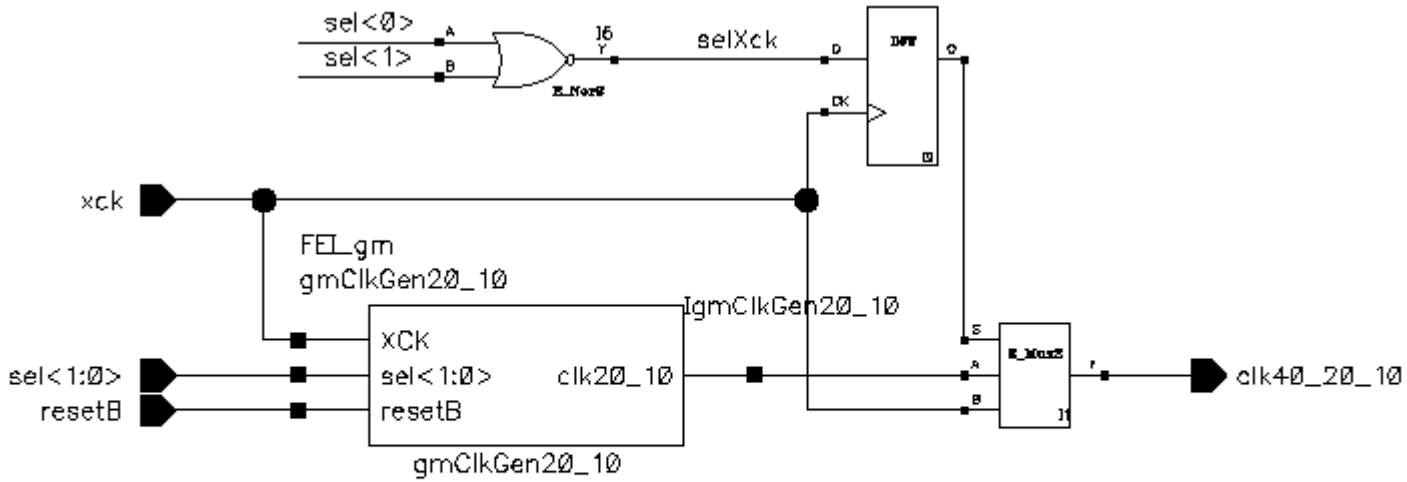
Trigger Address

Maximum # of events waiting for readout is 16.

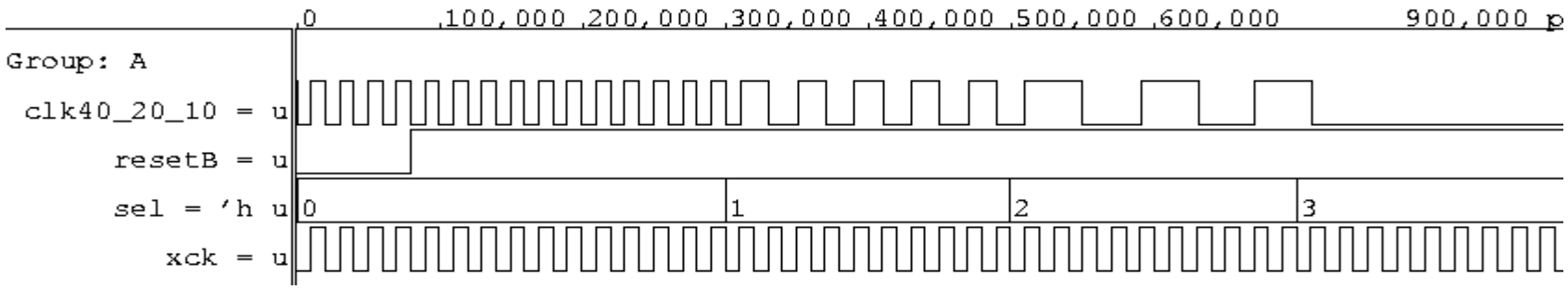
2 pointers implemented as 2 5-bit counters keep track of the # of events which are waiting to be read out. The 5th bit is used to determine if an overflow has occurred.

In previous versions of the chip this was also used as the read/write pointer for the FIFO to store the LE data for the TOT calculation. The TOT calculation has been moved to the column, so this will most likely change …

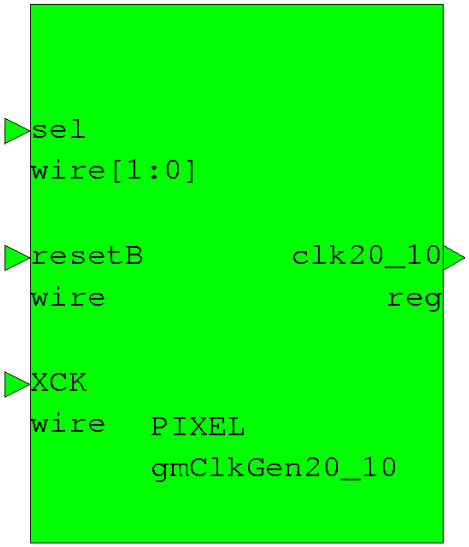
CEU Clock Generator

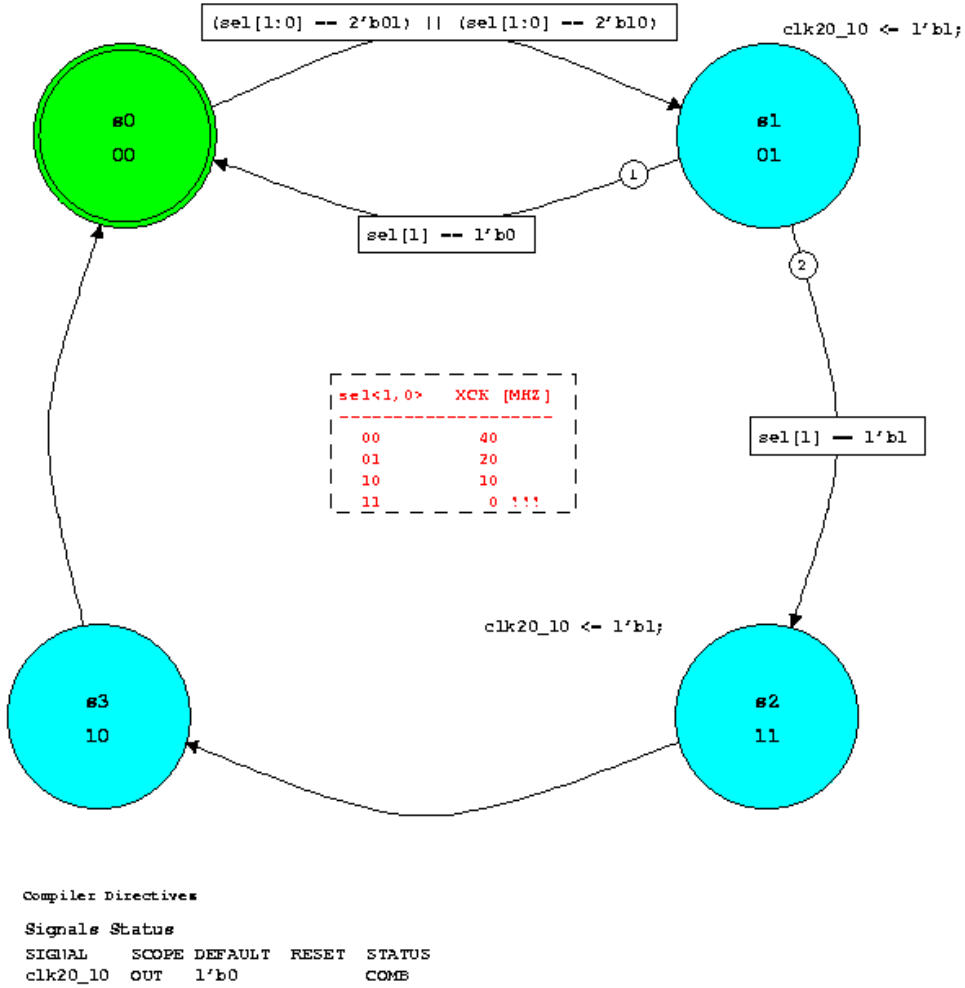


sel<1>	sel<0>	clk[MHz]
0	0	40
0	1	20
1	0	10
1	1	Off



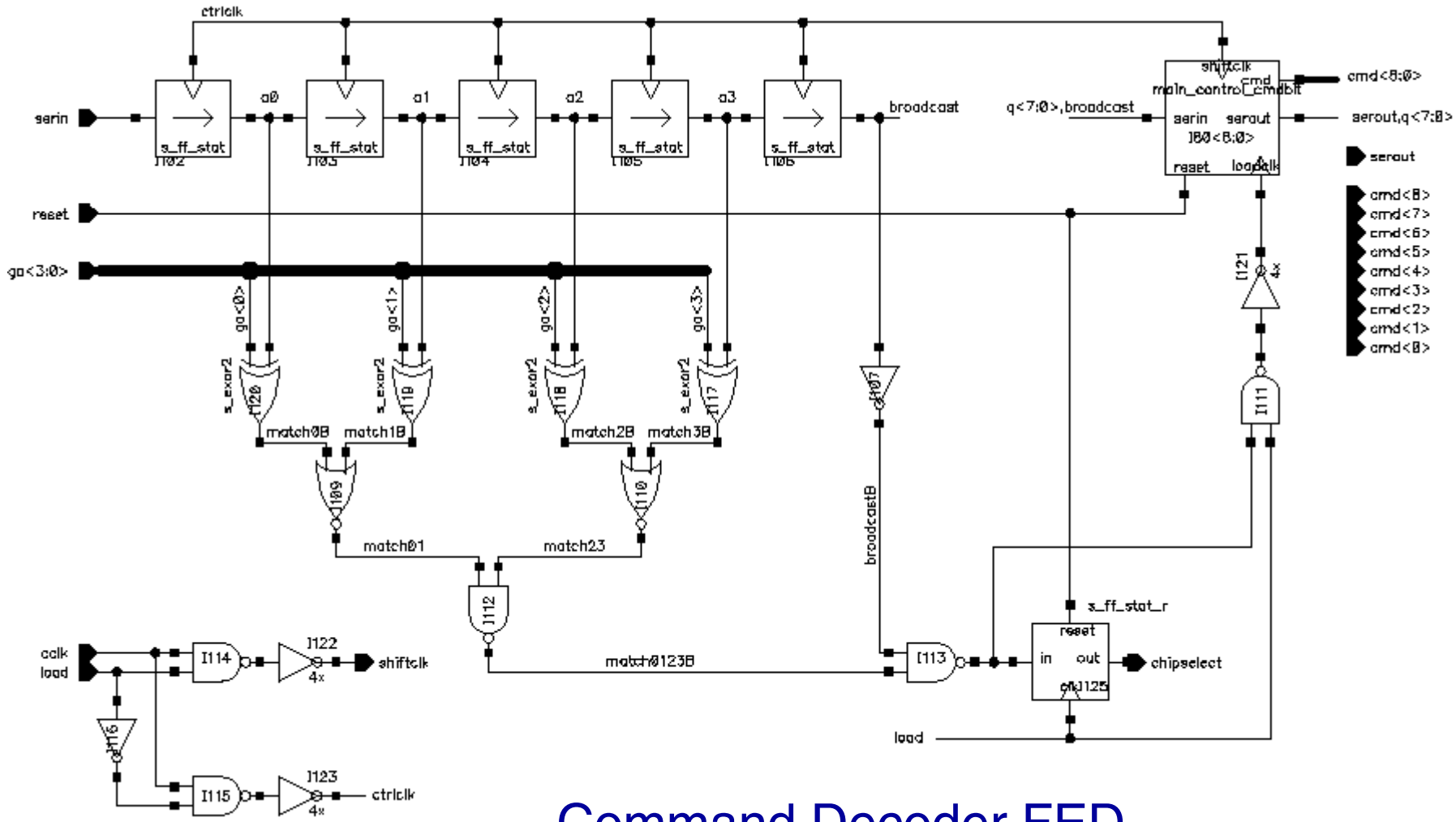
CEU Clock Generator





Move to columns, same state-machine as "Write Pulse Generator"

Command Decoder



Command Decoder FED

Command Decoder

```
module gmMainCtrl(cck, resetB, load, ga, serin, pixClk, regClk, cmd,
                  chipSelect, serout); input       cck, resetB, load, serin;
input  [3:0] ga;
output       pixClk, regClk, chipSelect, serout;
output [18:0] cmd;
reg          chipSelect;
reg    [18:0] cmd;
reg    [23:0] shiftReg;

wire         ctrlClk, shiftClk, match, loadClk;

assign ctrlClk  = cck & ~load;
assign shiftClk = cck &  load;
assign match    = (ga == shiftReg[3:0]) | shiftReg[4];
assign loadClk  = match & load;
assign serout   = shiftReg[23];
assign regClk   = cmd[1] & shiftClk;
assign pixClk   = cmd[4] & shiftClk;

always @(posedge ctrlClk or negedge resetB)
begin
   if (!resetB) shiftReg <= 24'b000000000000000000000000;
   else shiftReg <= {shiftReg[22:0],serin};
end

always @(posedge load or negedge resetB)
begin
   if (!resetB) chipSelect <= 1'b0;
   else chipSelect <= match;
end

always @(posedge loadClk or negedge resetB)
begin
   if (!resetB) cmd <= 19'b0000000000000000000;
   else cmd <= shiftReg[23:5];
end

endmodule
```

Configuration Register

```
module gmConfigReg(clk, resetB, load, readBack, serIn, serOut, parityIn,parityOut, configReg);
input        clk, resetB, load, readBack, serIn, parityIn;
output       serOut, parityOut;
output [22:0] configReg;

reg [22:0] shiftReg, configReg;

assign serOut = shiftReg[22];
assign parityOut = ^{parityIn, configReg};
always @(posedge clk or negedge resetB or posedge readBack)
begin
            if (!resetB)      shiftReg <= 23'b00000000000000000000000;
I           else if (readBack) shiftReg <= configReg;
            else              shiftReg <= {shiftReg[21:0], serIn};
end
always @(load or shiftReg or resetB)
  if (!resetB)   configReg <= 23'b00000000000000000000000;
  else if (load) configReg <= shiftReg;

endmodule
```

Time Stamp Generator

```
module gmGrayGenerator(xck, resetB, latency, tsc, tsi);
input       xck, resetB;
input  [7:0] latency;
output [7:0] tsc, tsi;

reg [7:0] tsc, tsi;
reg [7:0] q;

always @(posedge xck or negedge resetB)
begin
  if (!resetB) begin
    tsc <= bin2gray(-latency);
    tsi <= 8'b00000000;
    q   <= 8'b00000000;
  end
  else begin
    tsi <= bin2gray(q);
    tsc <= bin2gray(q - latency);
    q   <= q + 1;
  end
end

function [7:0] bin2gray;
input [7:0] b;
integer i;
begin
  bin2gray[7] = b[7];
  for ( i = 6; i >= 0; i = i - 1)
     bin2gray[i] = b[i] ^ b[i+1];
end
endfunction

endmodule
```

Reset Generator

```
module gmResetGenerator(xck, resetpinB, syncB, cmdReset, hardResetB, softResetB);
input xck, resetpinB, syncB, cmdReset;
output hardResetB, softResetB;

reg [4:0] q;
reg hardResetInt, softResetInt;

assign hardResetB = resetpinB & ~hardResetInt;
assign softResetB = resetpinB & ~softResetInt & ~cmdReset;

always @(posedge xck or negedge resetpinB)
   if (!resetpinB) q <= 5'b00000;
   else if (syncB) q <= 5'b00000;
   else q <= q + 1;

always @(posedge xck or negedge resetpinB)
   if (!resetpinB) begin
        hardResetInt <= 1'b0;
        softResetInt <= 1'b0;
   end
   else begin
        hardResetInt <= ~syncB & (q[4] | hardResetInt);
        softResetInt <= ~syncB & (q[3] | softResetInt);
   end

endmodule
```

Test Logic

In the previous chip two multiplexors were added to be able to look at some of the internal nodes of the chip.

Not decided yet what to do for the FEI chip.

Summary

To do:

- Adapt some of the Verilog descriptions
- Detailed Verilog Simulations
- Synthesis
- Place & Route
- Verilog Simulations with back-annotation