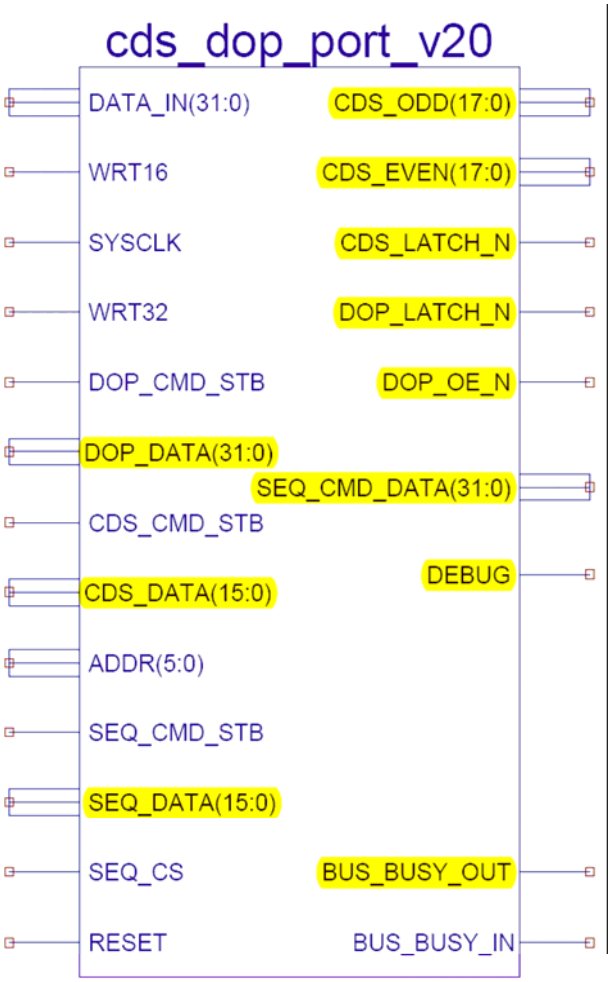
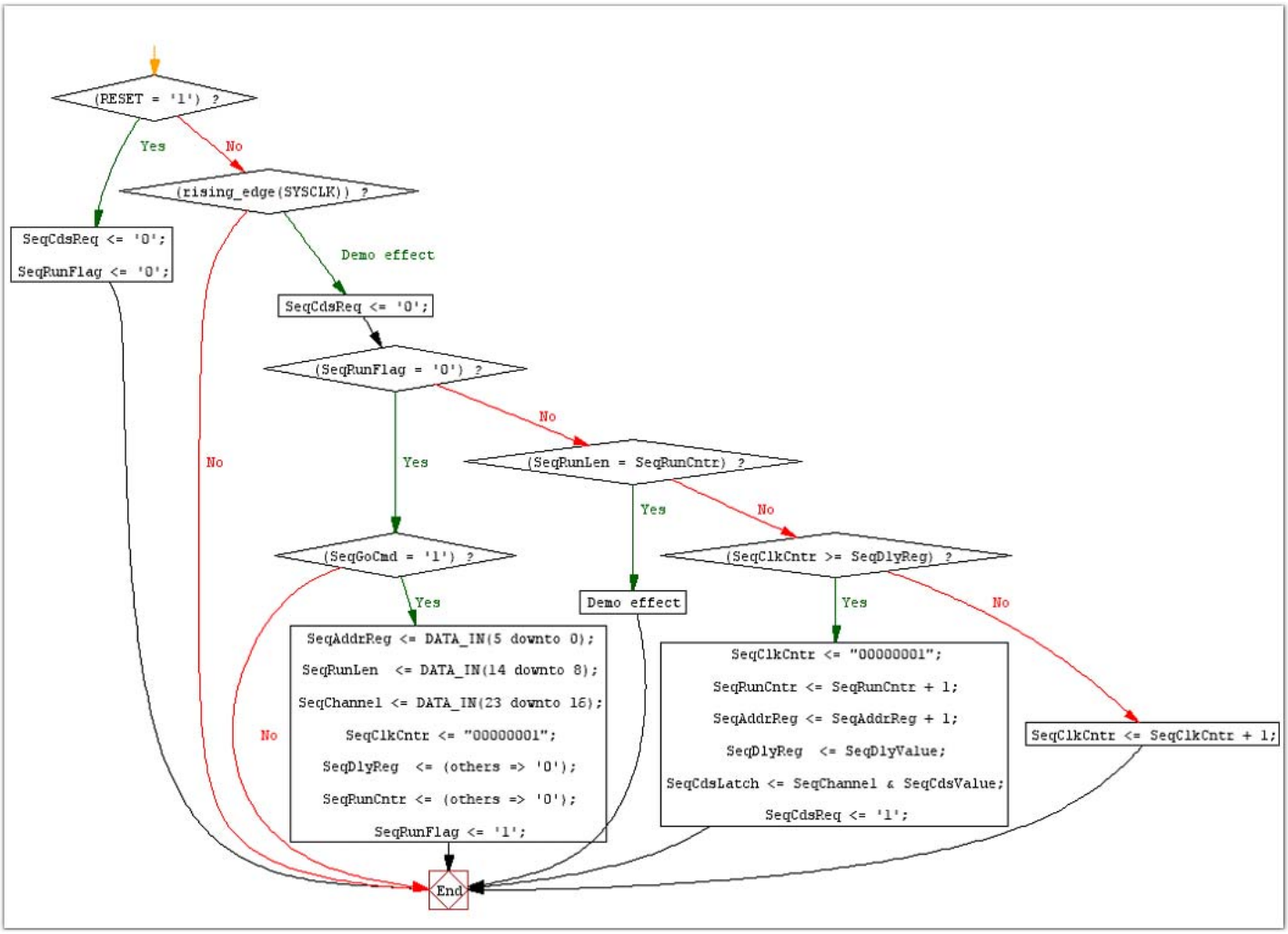
```
-- DESCRIPTION  :
-- PROVIDES THE CONTROL MECHANISM TO SENT 32 BIT DATA TO THE CDS CONTROL
-- LOGIC IN THE CPLD AND THE DIGITAL PORT ON THE CCD ACQUISITION BOARD
-- MICRO SEQUENCER INCORPORATES 64 x 16 BIT WORDS WHICH ARE STORED IN THE
-- RAM. THE FORMAT OF THE WORD IS:
-- ---------------------------------------------------------------------------
-- | BITS 15 DOWNTO 8 | BIT 7 | BIT 6 | BIT 5 | BIT 4 | BIT 3 | BIT 2 | BIT 1 |
-- |------------------|-------|-------|-------|-------|-------|-------|-------|
-- | TIME DELAY VALUE | SPARE |  EXT  | RESET |   DC  | INTEG |  NON  |  INV  |
-- | IN CLOCK TICKS   |       |  TRIG | INTEG | CLAMP |       |  INV  |       |
-- ---------------------------------------------------------------------------
-- THE SEQUENCER IS TRIGGERED BY WRITTING TO THE SEQ_CMD_STB ADDRESS. THE FORMAT
-- OF THIS 32 BIT WORD IS:
-- ---------------------------------------------------------------------------
-- | BITS 31 DOWNTO 24 | BITS 23 DOWNTO 16 | BITS 15 DOWNTO 8| BITS 7 DOWNTO 0|
-- |-------------------|-------------------|-----------------|----------------|
-- |     NOT USED      | CDS CHANNEL MASK  |    LENGTH OF    | STARTING ADDR  |
-- |                   |                   |    SEQUENCE     | OF SEQ IN RAM  |
-- ---------------------------------------------------------------------------
-- EACH TIME THE DELAY VALUE EQUALS THE CLOCK COUNTER THE CDS BITS ARE OUTPUT TO
-- THE CPLD AND THE RAM ADDRESS INCREMENTED. WHEN THE NUMBER OF CDS CODES OUTPUT
-- EQUALS THE LENGTH OF SEQUENCE VALUE, THE SEQUENCER STOPS.
--
-- IMPLEMENTATION NOTES :
-- Note that two writes, one to cds on the first clock edge then the next to
-- the dop on the next clock edge will cause a glitch in the dop data before being
-- successfully latched on the falling edge of the dop strobe. In real life ('!') this
-- should never occur since the sequencer would have to change the address pointer
-- then issue another board select stobe (i.e at least one clock between writes) so no
-- extra logic has been included to supress this situation ... But one day it will bite
us !
```

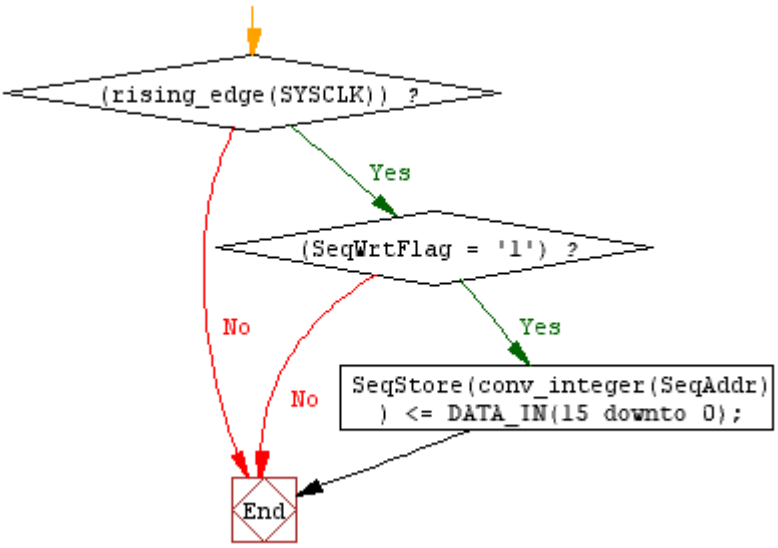



```
ADDR    : in std_logic_vector (5 downto 0);
BUS_BUSY_IN : in  std_logic;
BUS_BUSY_OUT: out std_logic;
CDS_CMD_STB : in  std_logic;
CDS_DATA    : out std_logic_vector(15 downto 0);
CDS_EVEN    : out std_logic_vector(17 downto 0);
CDS_LATCH_N : out std_logic;
CDS_ODD     : out std_logic_vector(17 downto 0);
DATA_IN     : in  std_logic_vector(31 downto 0);
DEBUG       : out std_logic
DOP_CMD_STB : in  std_logic;
DOP_DATA    : out std_logic_vector(31 downto 0);
DOP_LATCH_N : out std_logic;
DOP_OE_N    : out std_logic;
RESET  : in std_logic;
SEQ_CMD_DATA: out std_logic_vector(31 downto 0);
SEQ_CMD_STB : in  std_logic;
SEQ_CS : in std_logic;
SEQ_DATA    : out std_logic_vector(15 downto 0);
SYSCLK : in std_logic;
WRT16  : in std_logic;
WRT32  : in std_logic;
```

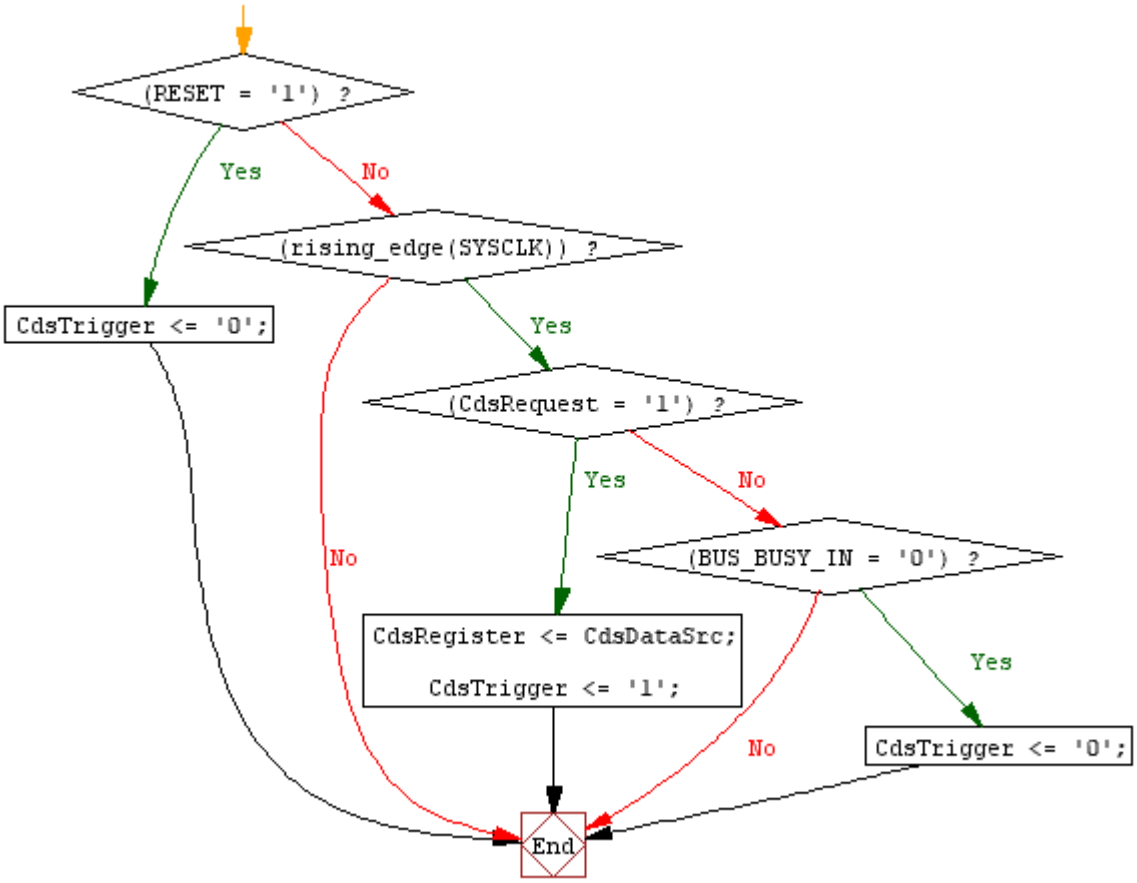
```
SeqCntrlProc : process (SYSCLK, RESET)
```



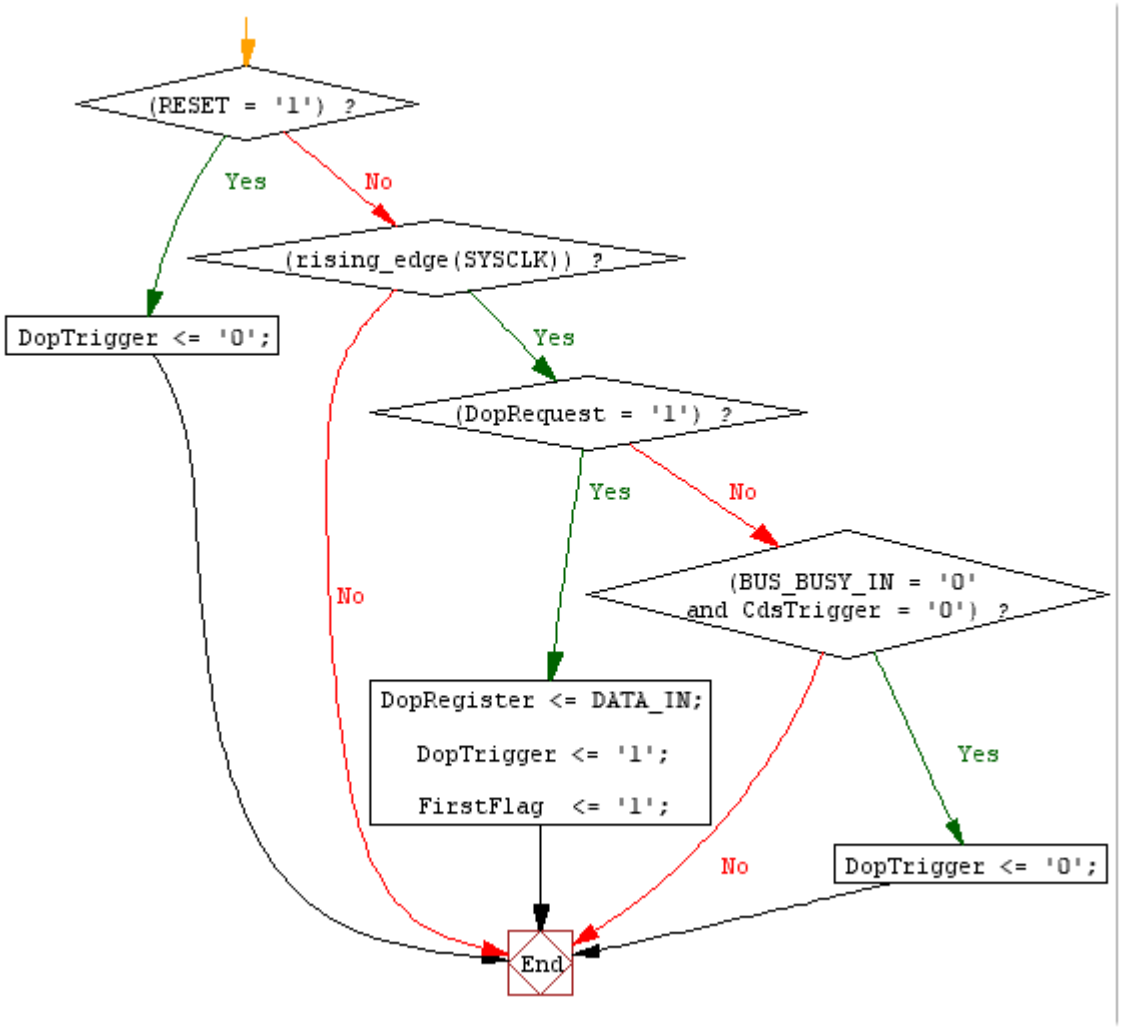
```
SeqRamWrite : process (SYSCLK)
```



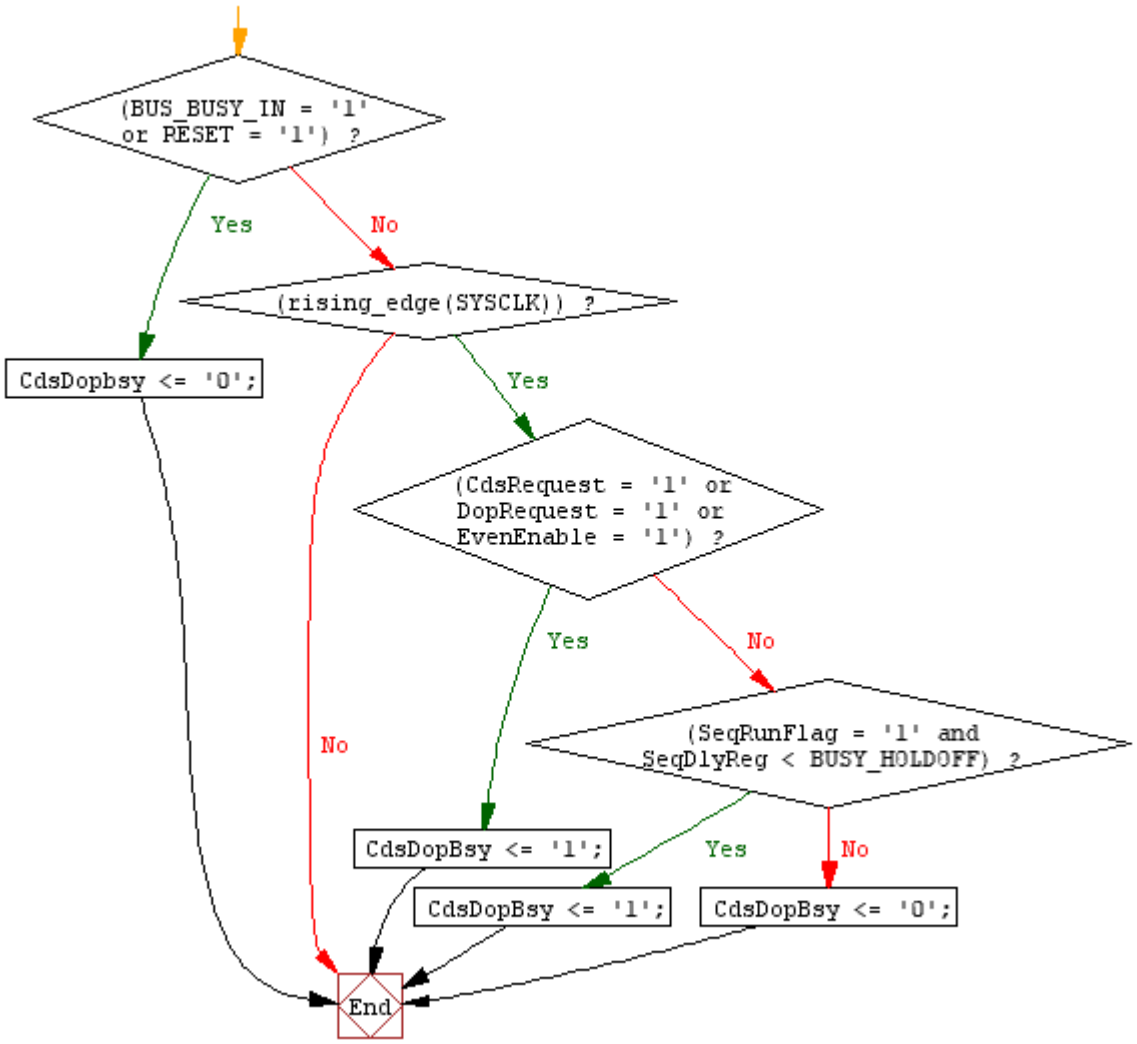

```
WrtCdsPortProc : process (SYSCLK, RESET)
```



```
WrtDopPortProc : process (SYSCLK, RESET)
```



```
WrtDataProc : process(SYSCLK, BUS_BUSY_IN, RESET)
```



```
     SeqGoCmd <= SEQ_CMD_STB and WRT32;
     SeqAddr <= DATA_IN(5 downto 0) when (SeqGoCmd = '1') else
                ADDR when (SEQ_CS = '1') else
                SeqAddrReg ;
     SEQ_CMD_DATA <= x"00" & SeqChannel & '0' & SeqRunLen & "00" & SeqAddrReg;
-- Sequencer store writes
    SeqWrtFlag <= not SeqRunFlag and WRT16 and SEQ_CS;
    SeqData <= SeqStore(conv_integer(SeqAddr));
    SeqDlyValue <= SeqData(15 downto 8);
    SeqCdsValue <= SeqData( 7 downto 0);
    SEQ_DATA <= SeqData when (SeqRunFlag = '0') else x"1234";
-- Generate bus lock signal
    DEBUG <= CdsDopBsy or SeqCdsLatch(7);
    BUS_BUSY_OUT <= CdsDopBsy or SeqCdsLatch(7);
-- Generate CDS port write request and data source
    CdsRequest <= (CDS_CMD_STB and WRT32) or SeqCdsReq;
     CdsDataSrc <= DATA_IN(15 downto 0) when (CDS_CMD_STB = '1' and WRT32 = '1') else
SeqCdsLatch;
     CDS_DATA <= '0' & CdsRegister(14 downto 0);      -- Mask out the global ctc bit
     DopRequest <= DOP_CMD_STB and WRT32;
     DOP_DATA <= DopRegister;
     OddEnable  <= DopTrigger and not BUS_BUSY_IN; -- Use the ODD bus for DOP data only
     EvenEnable <= (CdsTrigger or DopTrigger) and not BUS_BUSY_IN; -- Use the EVEN bus for CDS
and DOP data
-- Data output mux
     CDS_ODD  <= (others => 'Z') when (OddEnable = '0') else
                 ("0000" & DopRegister(31 downto 18));
     CDS_EVEN <= (others => 'Z') when (EvenEnable = '0') else
                 ("00" & CdsRegister(15 downto 0)) when (CdsTrigger = '1') else
                 (DopRegister(17 downto 0));
     DOP_OE_N <= not FirstFlag;       -- Make sure that DOP data valid before enabling to device.
-- Generate strobes
     CDS_LATCH_N <= SYSCLK or not CdsTrigger;
     DOP_LATCH_N <= SYSCLK or not DopTrigger;
```