



MONSOON-DHE WAVEFORM SPECIFICATION

Kaviraj Chopra



OVERVIEW

- New specification Why?
- Studying Control Requirements.
- Specification (DRAFT)
- A Small Example.



NEW SPECIFICATION WHY ?

"WILDFIRE" WAVE-FORM SPECIFICATION

- A modified subset of 'C' is used to model the control:-
Proprietary Construct "Wave": to specify a waveform

```
wave ( <waveform_name> )  
{  
    /* Control waits for specified Delay (clock cycles) */  
    wait (<clock_cycles>);  
    /* Assigns the specified 'value' to the specified 'Bit-Vector' */  
    set (<bit_vector_label>, <value> ) ;  
    .../* Assert the specified 'Bit' high for specified 'clock_cycles' */  
    toggle_hi (<bit_label> , <clock_cycles> ) ;  
    ...  
}
```

- Classical Control-Constructs: to specify Logic flow:
 - Conditional Branching:
'switch-case' & 'if-else'.
 - Repetition:
'while' & 'for'.



WHY NOT USE THE SAME ?

- Design Implemented for “WILDFIRE CONTROL SYSTEM” :- is essentially ‘Centralized’ and ‘Sequential’.
- On the contrary “MONSOON-DHE CONTROL” design is essentially ‘Distributed’ and ‘Concurrent’.
- So to model the ‘Distributed & Concurrent’ control behavior, previously used programming language(sequential) ‘C’ can’t be used.
- Now currently used hardware description language VHDL, to design the DHE Digital Hardware (FPGA & CPLD) is meant to describe ‘distributed and concurrent control logic.
- So VHDL was the obvious solution.



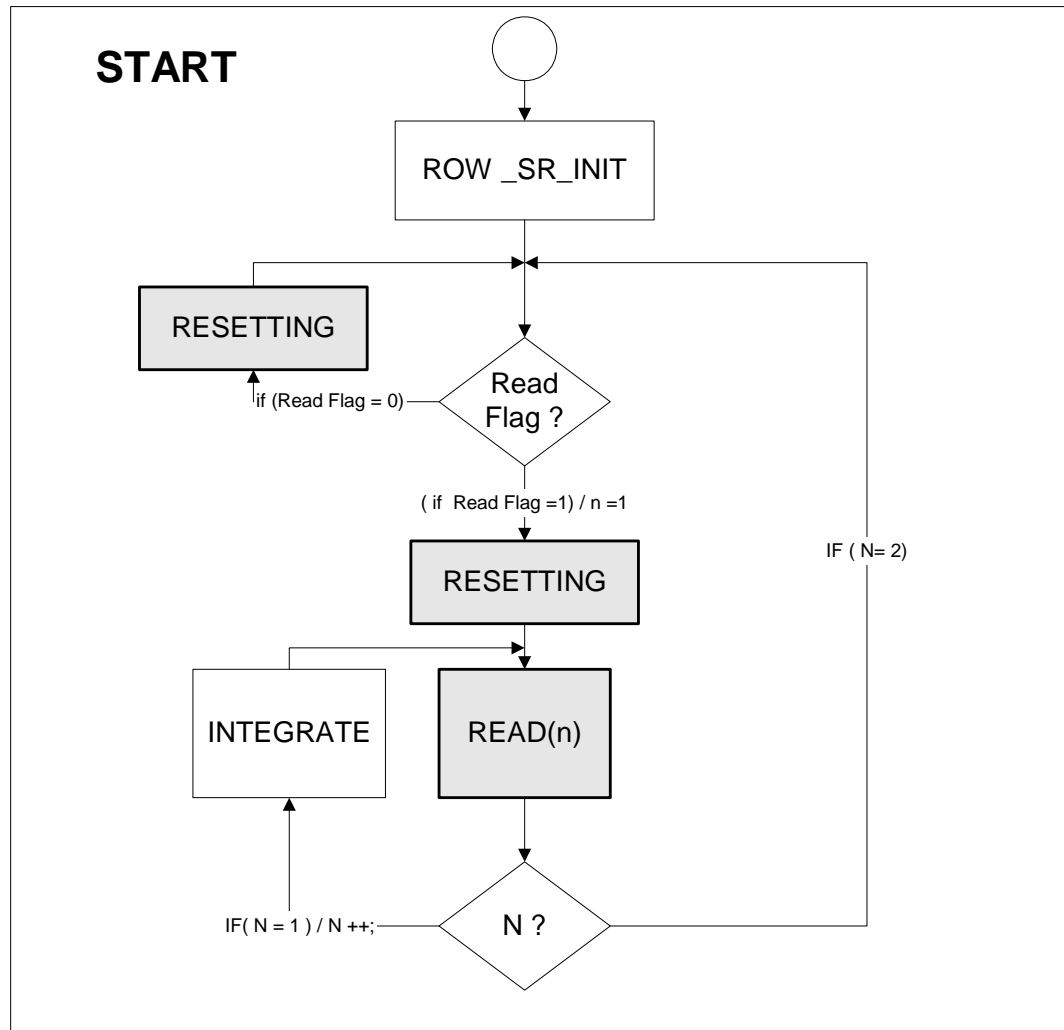
WHY VHDL?

- + Moreover it appeals to use a **Consistent** Specification for both modeling behavior and design implementation.
- + Also the same **free simulation** tools can be used to view the described behavior. So software-overhead to write, maintain and distribute a Waveform Specification Simulator which was needed previously can be avoided.
- 0. However in-order to *compile* waveform patterns and control information a Proprietary **compiler** is still needed. (which was also needed previously)
- - The only drawback of using VHDL is the overhead of **learning** a **very small** subset of VHDL syntax. But this might have been the case for any other choice.



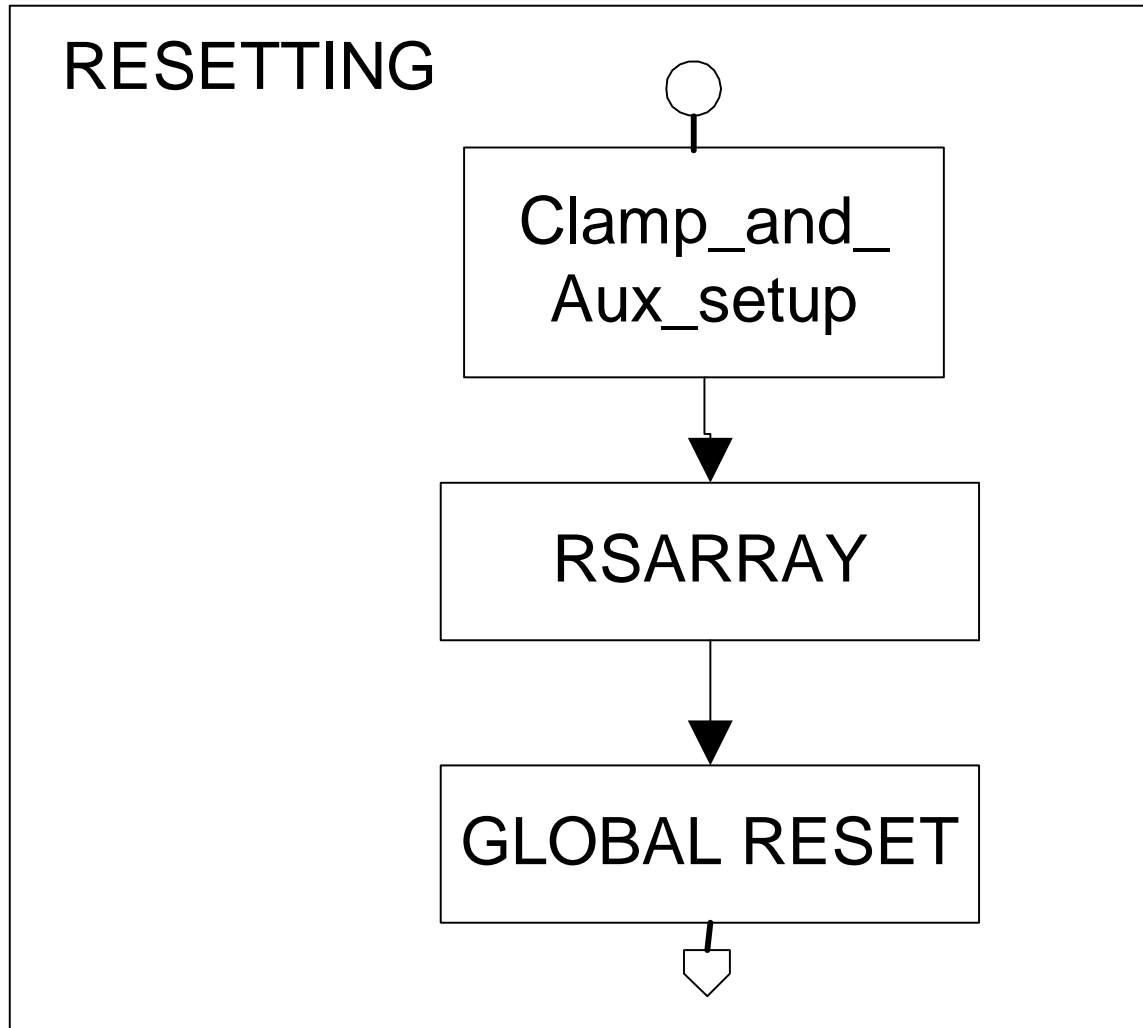
STUDYING CONTROL REQUIREMENTS

TYPICAL CONTROL BEHAVIOR REQUIRED-I

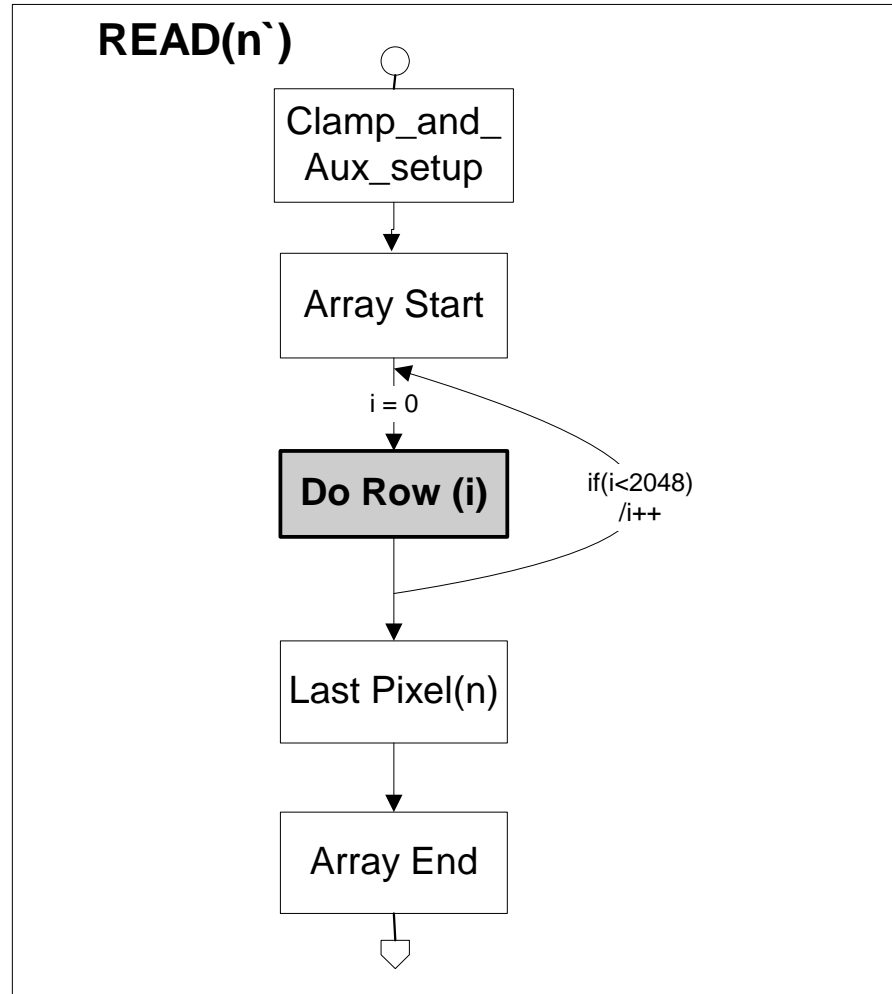


MONSOON

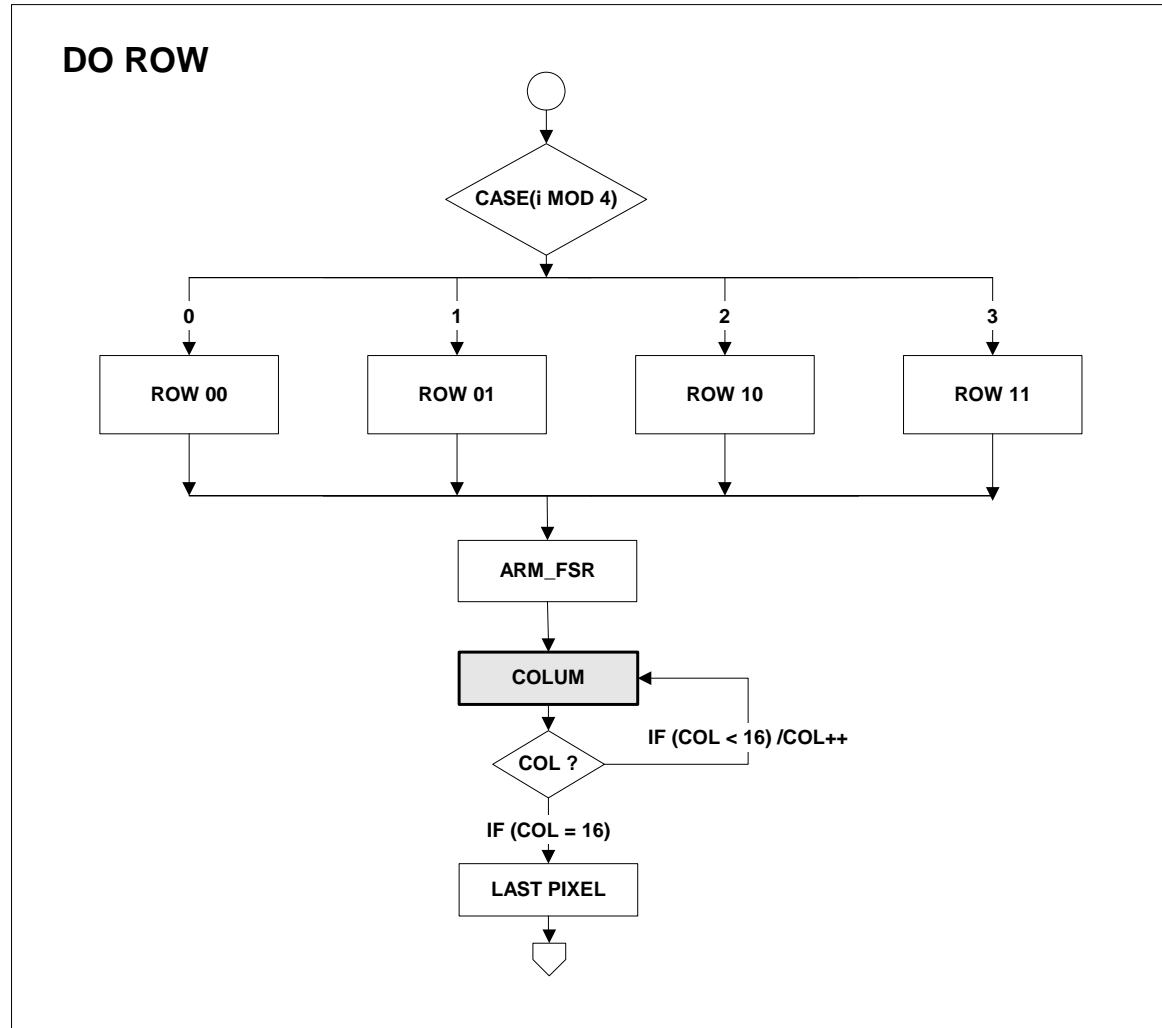
TYPICAL CONTROL BEHAVIOR REQUIRED-II



TYPICAL CONTROL BEHAVIOR REQUIRED-III

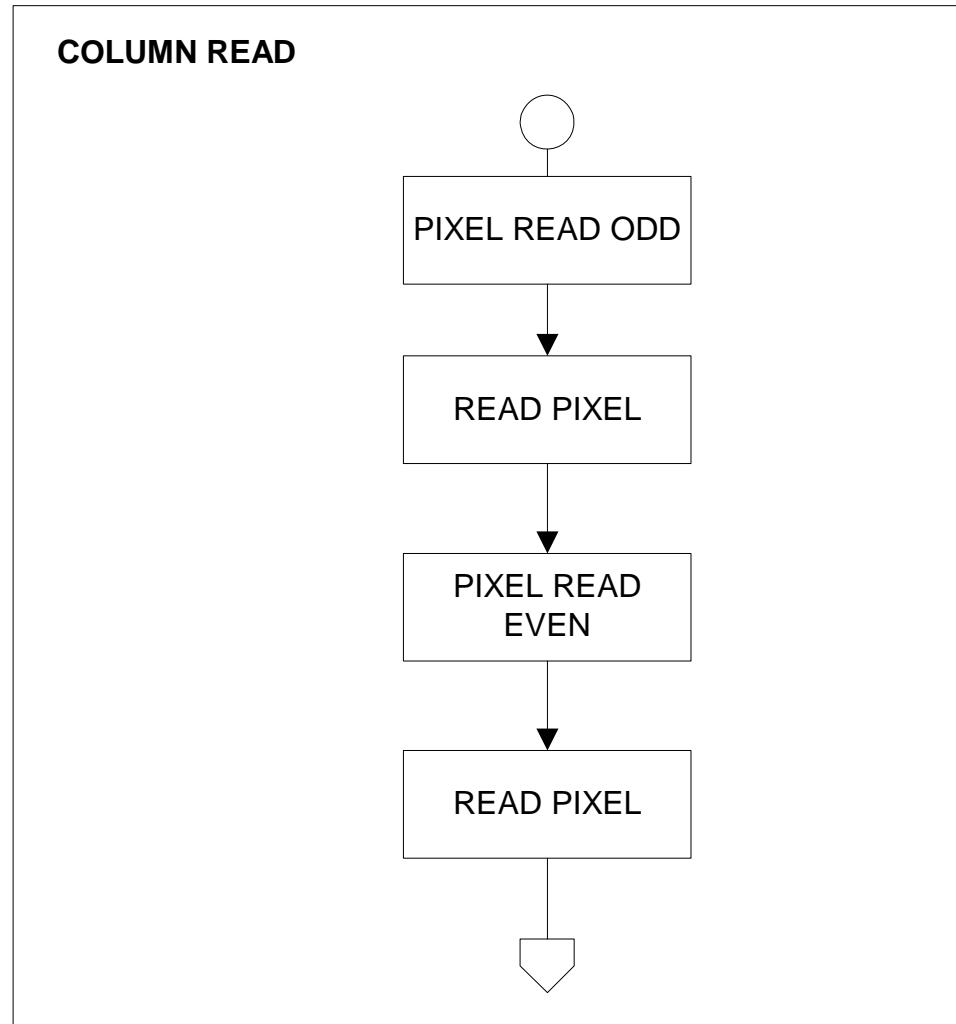


TYPICAL CONTROL BEHAVIOR REQUIRED-IV



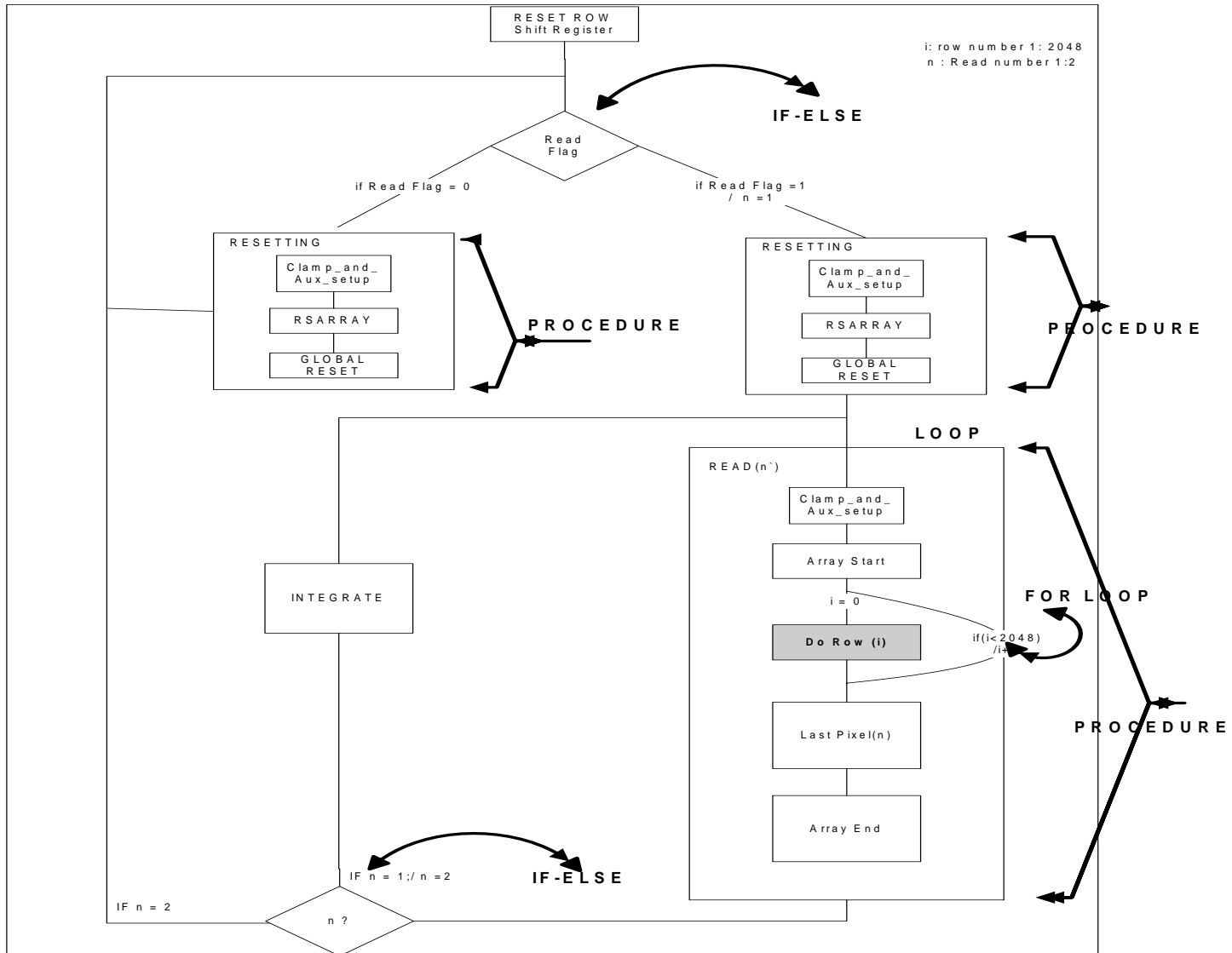
MONSOON

TYPICAL CONTROL BEHAVIOR REQUIRED-V



MONSOON

ALL TOGETHER:



MONSOON



CONCLUSION:

- BEHAVIOR DESCRIPTION REQUIRMENTS:
 - Iteration
 - Conditional branching
 - SEQUENCE (or WAVE)
 - Time Delay
 - And Concurrency.(for Distributed Control)



SPECIFICATION (DRAFT)



MAPPING CONTROL: VHDL SUBSET USED:

■ REQUIREMENTS

- Concurrency
- SEQUENCE (or WAVE)
- Iteration
- Conditional branching
- Time Delay

■ CONSTRUCTS

- PROCESS
- PROCEDURE
- FOR & WHILE
- IF-ELSE & CASE
- WAIT

VHDL SYNTAX - I

PROCESS

<process_label >: **process** (<sensitivity list>)

-- A change in any parameter in the sensitivity list triggers the process.

begin

<statements>

end process;

PROCEDURE:

procedure < label>({<type> <par_name>: <mode>}) **is**

--<type>: **signal** | **variable** | **constant** --<mode>: **in** | **inout** | **out**

begin

<statements>

end;

ASSIGNMENT:

SIG_NAME <= <expression>;

DELAY:

Wait for <Time> **ps/ns/us/ms ;**

VHDL SYNTAX II

ITERATION:

```
while (<condition>) loop
    <statement>
end loop;
```

```
for I In <lower_limit> to <upper_limit> loop
    <statement>
end loop;
```

CONDITIONAL BRANCHING:

```
case <expression> is
    when <choices> =>
        <statements>
    when <choices> =>
        <statements>
    when others =>
        <statements>
end case;
```

```
if (<condition>) then
    <statement>
elsif (<condition>) then
    <statement>
else
    <statement>
end if;
```



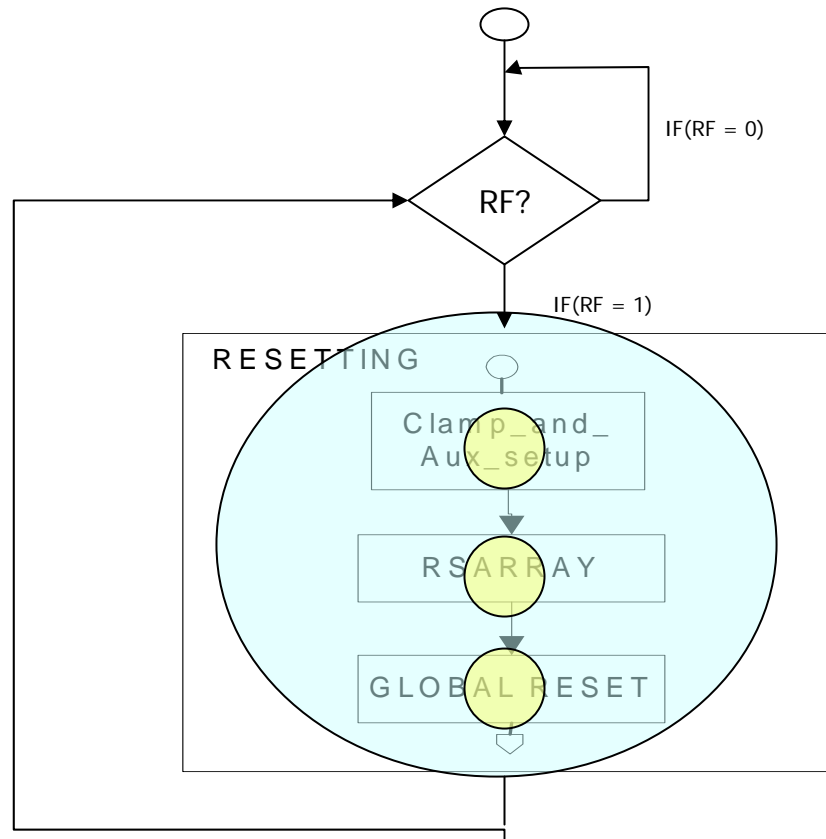
STEPS

- Identify the distributed concurrent processes in the system.
- Based on hierarchy of distribution of concurrent processes classify and group control signals (clocks, fast bias and biases) by processes.
- For every concurrent process, specify the required 'sequence' of events (previously called 'wave').
- Using the same classic control constructs to specify the required branching and looping sequential control behavior for a process. A concurrent process at higher level of abstraction controls the lower level processes thus the sequence for such a process can be called Macro-Sequence(Wave)/Super-Sequence and controlled process 'sequence' can be called 'micro-sequence'.



WAVEFORM SPECIFICATION EXAMPLE

"RESETTING"



Identify the distributed concurrent processes in the system.

MONSOON-DHE CONTROL

PAN

FIBER FPGA
 Recieve, Pass and Return Command
 Generate Pattern Counter
 Transmit Pixel Data

SEQUENCER FPGA
 Execute / Pass Command
 Assemble Data
 Rabbit Interface ?

MACRO_MCB

CLK & BIAS (BRD -I)
 Write to CLK & Fast Bias Register (parallel 32 bit)
 Read HK ADC Data (serial)
 Write to DACs, OE & MUX (serial)

MICRO CLOCK

VIDEO (BRD -II)
 Read Pixel/Sig Data (parallel 32 bit)
 Write to DACs & ADCs (serial)

MICRO VIDEO

SEQ CPLD
 Control Clock Register
 Control Fast Bias Register
 Control Fast Bias Write Register

CONFIG. CPLD
 Sample & Read HK ADCs Transfer
 Configure DACs
 Control OE & CLK-MUX-SEL

ACQ CPLD 1
 Sample & Read ADCs
 Configure DACs

ACQ CPLD 2
 Sample & Read ADCs
 Configure DACs

ACQ CPLD 3
 Sample & Read ADCs
 Configure DACs

MONSOON



STEPS

- Identify the distributed concurrent processes in the system.
- Based on hierarchy of distribution of concurrent processes classify and group control signals (clocks, fast bias and biases) by processes.
- For every concurrent process, specify the required 'sequence' of events (previously called 'wave').
- Using the same classic control constructs to specify the required branching and looping sequential control behavior for a process. A concurrent process at higher level of abstraction controls the lower level processes thus the sequence for such a process can be called Macro-Sequence(Wave)/Super-Sequence and controlled process 'sequence' can be called 'micro-sequence'.



SIGNALS TO BE CONTROLLED

CREN	-- CLAMP AND RESET ENABLE
Clamp	-- CLAMP BIT
Phi1Fast	-- PHASE 1 FAST (CLOCK)
Phi2Fast	-- PHASE 2 FAST (CLOCK)
PhiSFast	-- PHASE SYNC FAST (CLOCK)
Phi1Slow	-- PHASE 1 SLOW (CLOCK)
Phi2Slow	-- PHASE 2 SLOW (CLOCK)
PhiSSlow	-- PHASE SYNC SLOW (CLOCK)
PhiRowO_E	-- ODD/EVEN ROW SEL (CLOCK)
PhiDes	-- ROW DESELECT (CLOCK)
VrowEn	-- CTRL FOR ROW EN (BIAS)
VrstR	-- ROW RESET (BIAS)
VrstG	-- GLOBAL RESET (BIAS)
M_CLK	-- Master Clk Trig. (CONTROL)
SEN	-- Slow Clk Enable (CONTROL)
FEN	-- Fast Clock Enable (CONTROL)

CLASSIFY CONTROL SIGNALS

■ MICRO-I CLOCK

Phi1Fast -- PHASE 1 FAST (CLOCK)
Phi2Fast -- PHASE 2 FAST (CLOCK)
PhiSFast -- PHASE SYNC FASTCLOCK)

Phi1Slow -- PHASE 1 SLOW (CLOCK)
Phi2Slow -- PHASE 2 SLOW (CLOCK)
PhiSSlow -- PHASE SYNC SLOW (CLOCK)
PhiRowO_E -- ODD/EVEN ROW SEL (CLOCK)
PhiDes -- ROW DESELCT (CLOCK)

VrowEn -- CTRL FOR ROW EN (BIAS)
VrstR -- ROW RESET (BIAS)
VrstG -- GLOBAL RESET (BIAS)

M_CLK -- Master Clk Trig. (CONTROL)
SEN -- Slow Clk Enable (CONTROL)
FEN -- Fast Clock Enable (CONTROL)

■ MICRO-II VIDEO

CTC -- COMMAND TO CONVERT
CREN -- CLAMP AND RESET ENABLE
Clamp -- CLAMP BIT

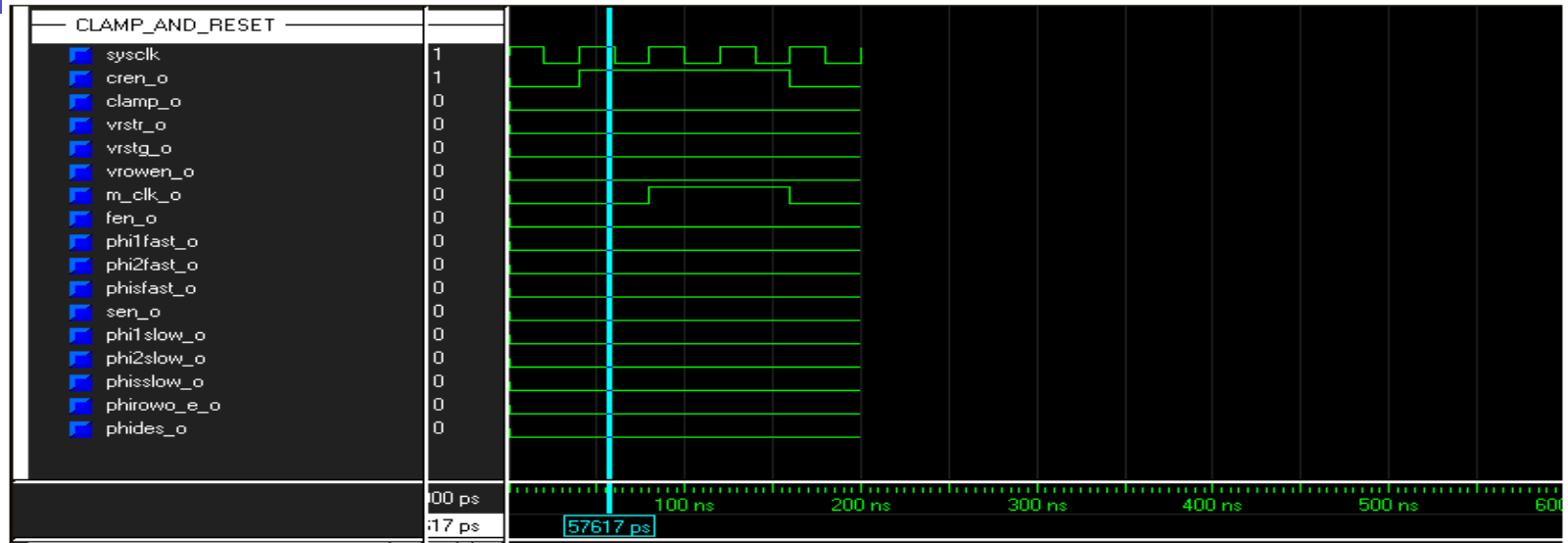
M_CLK -- Master Clk Trig. (CONTROL)



STEPS

- Identify the distributed concurrent processes in the system.
- Based on hierarchy of distribution of concurrent processes classify and group control signals (clocks, fast bias and biases) by processes.
- For every concurrent process, specify the required 'sequence' of events (previously called 'wave').
- Using the same classic control constructs to specify the required branching and looping sequential control behavior for a process. A concurrent process at higher level of abstraction controls the lower level processes thus the sequence for such a process can be called Macro-Sequence(Wave)/Super-Sequence and controlled process 'sequence' can be called 'micro-sequence'.

CLAMP_AND_RESET



```
procedure MicSeq_ClampAndReset2 (signal Set: out std_logic_vector(15 downto 0)) IS
```

```
begin
```

```
    Delay (1) ;      Set (CREN ) <= '1';
```

```
    Delay (1) ;      Set (M_CLK ) <= '1';
```

```
    Delay (2) ;      Set (M_CLK ) <= '0'; Set (CREN ) <= '0';
```

```
end;
```

```
procedure Delay (constant Ticks : integer) is
```

```
begin
```

```
    for I in 1 to Ticks loop
```

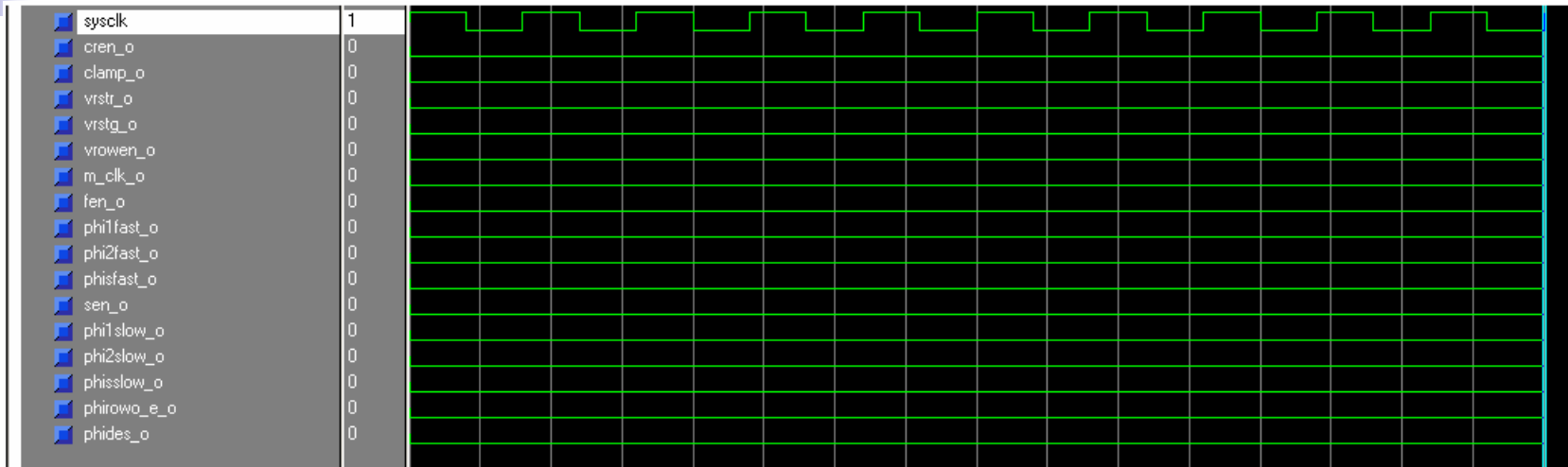
```
        wait for 25ns; -- Clock Period
```

```
    end loop;
```

```
end;
```

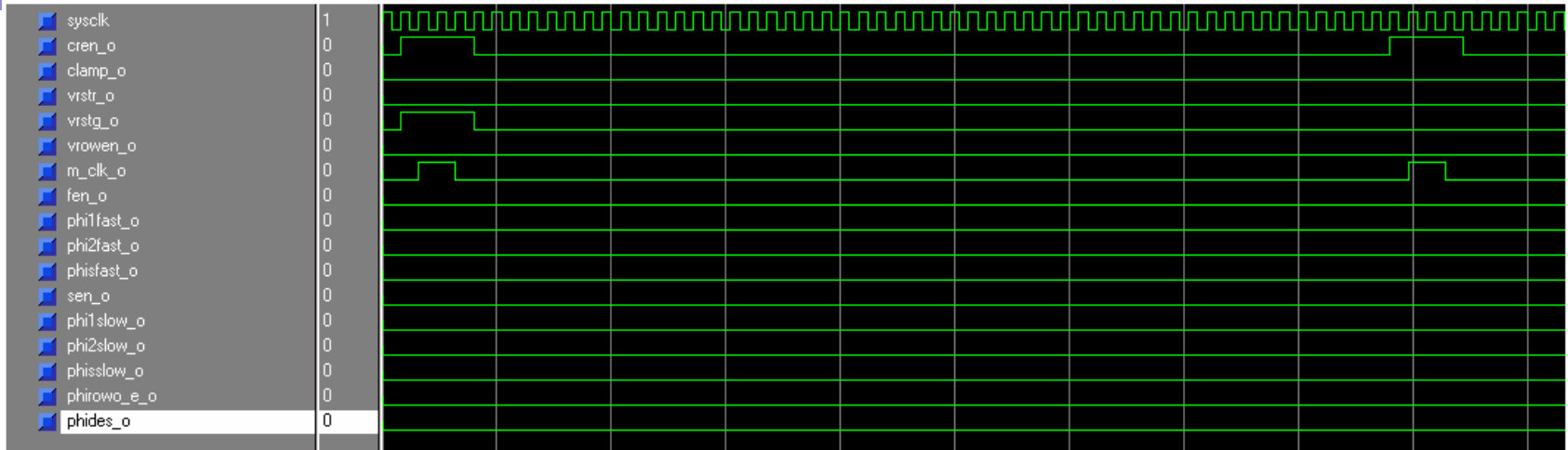
MONSOON

RSARRAY



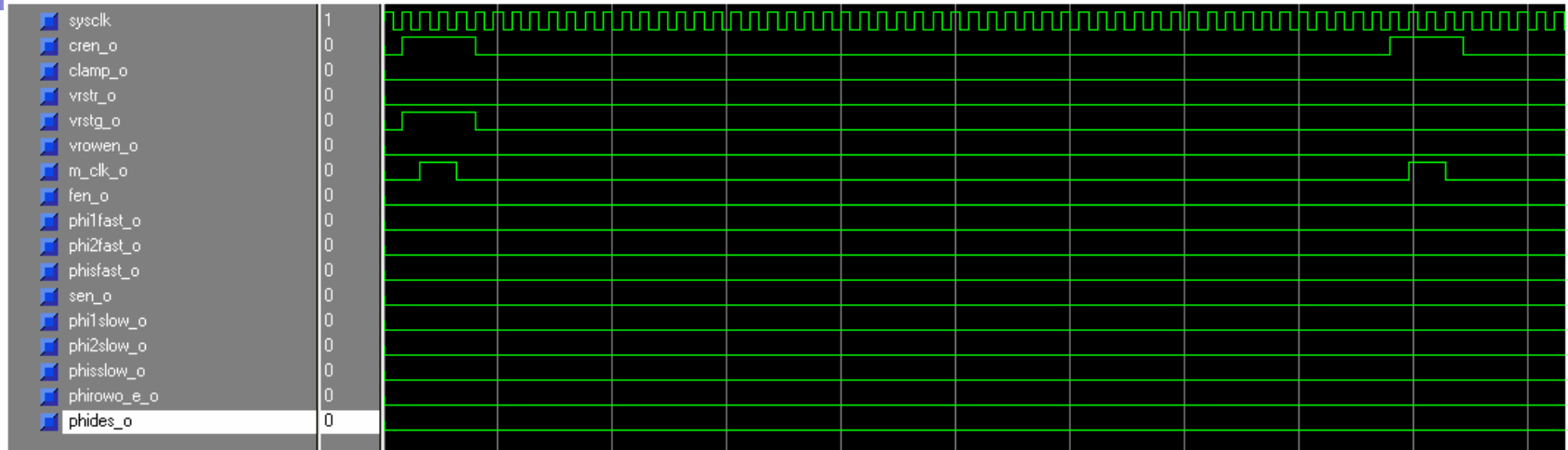
```
procedure MicSeq_rsarray (signal Set: out std_logic_vector (15 downto 0)) IS
begin
    Delay (10) ;
end;
```

GLOBAL_RESET1 (CLK)



```
procedure MicSeq_GlobalReset1 ( signal Set : out std_logic_vector (15 downto 0)) IS
begin
    Delay (1) ;      Set (VrstG   ) <= '1';
    Delay (1) ;      Set (M_CLK   ) <= '1';
    Delay (2) ;      Set (M_CLK   ) <= '0';
    Delay (1) ;      Set (VrstG   ) <= '0';
    Delay (501) ;    Set (M_CLK   ) <= '1';
    Delay (2) ;      Set (M_CLK   ) <= '0';
    Delay (74) ;
end;
```

GLOBAL_RESET2(VIDEO)



procedure MicSeq_GlobalReset2 (**signal** Set : **out std_logic_vector** (15 downto 0)) **IS**

begin

Delay (1) ; **Set** (CREN) <= '1';

Delay (1) ; **Set** (M_CLK) <= '1';

Delay (2) ; **Set** (M_CLK) <= '0';

Delay (1) ; **Set** (CREN) <= '0';

Delay (500); **Set** (CREN) <= '1';

Delay (1) ; **Set** (M_CLK) <= '1';

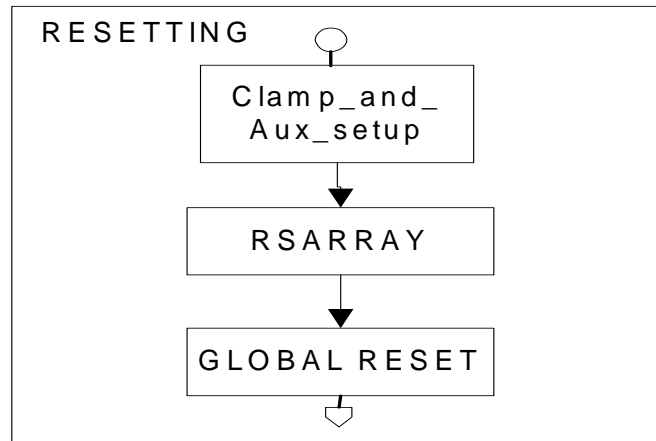
Delay (2) ; **Set** (M_CLK) <= '0';

Delay (1) ; **Set** (CREN) <= '0';

Delay (73) ;

end;

RESETTING & MACRO_SEQ PROCESS



```
procedure MacSeq_RESETTING (  
    signal Seq : out std_logic_vector (15 downto 0) ;  
    signal Board : out std_logic_vector (8 downto 2) ) IS  
begin  
    Seq <= ClampAndReset; Board <= Video; Delay (4) ;  
    Seq <= rsarray ; Board <= VidAndClock; Delay (10) ;  
    Seq <= GlobalReset ; Board <= VidAndClock; Delay (582) ;  
end;
```



STEPS

- Identify the distributed concurrent processes in the system.
- Based on hierarchy of distribution of concurrent processes classify and group control signals (clocks, fast bias and biases) by processes.
- For every concurrent process, specify the required 'sequence' of events (previously called 'wave').
- Using the same classic control constructs to specify the required branching and looping sequential control behavior for a process. A concurrent process at higher level of abstraction controls the lower level processes thus the sequence for such a process can be called Macro-Sequence(Wave)/Super-Sequence and controlled process 'sequence' can be called 'micro-sequence'.

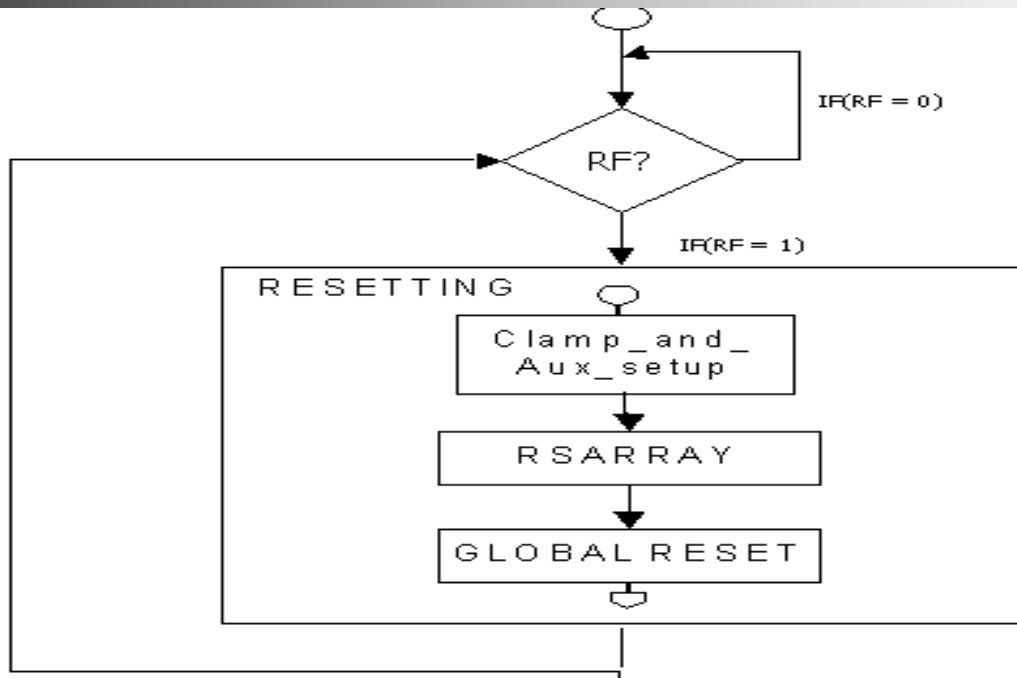


MICRO_SEQ PROCESSES

```
MICRO_SEQ_CLK: process (MacroSequence, Board(ClockBias)) Begin
  if (Board ( Clock) = '1') then
    case MacroSequence is
      when rsarray =>
        MicSeq_rsarray( ClockPattern);
      when GlobalReset =>
        MicSeq_GlobalReset1 (ClockPattern);
      when others =>
    end case;
  end if;
end process;
```

```
MICRO_SEQ_VIDEO: process (MacroSequence, Board(Video)) Begin
  if (Board ( Video) = '1') then
    case MacroSequence is
      when ClampAndReset =>
        MicSeq_ClampAndReset (VideoPattern);
      when rsarray =>
        MicSeq_rsarray(VideoPattern);
      when GlobalReset =>
        MicSeq_GlobalReset1 (VideoPattern);
      when others =>
    end case;
  end if;
end process;
```

MACRO_SEQ PROCESSES



MACRO_SEQ: **process** (RF)

begin

if (RF = '1') **then**

 MacSeq_RESETTING (MacroSequence, Board);

end if;

end process;



USEFUL NOTES

- VHDL is not case sensitive
- All processes are executed concurrently. But within a process the execution is sequential.
- Assignment operator is '`<=`' and not '`=`'
- But the conditional operator is '`=`' and not '`==`'
- The braces '`{`' '`}`' are replaced by constructs '`begin`' and '`End`'.
- **Logic Signals** are explicitly declared by a data-type `std_logic` and `std_logic_vector` (preferably) or `bit` and `bit_vector`.



References:

- “Orion 2048 Code Using the COB Kludge System” A.M. Fowler.
- MONSOON: Image Acquisition System or “Pixel Server”B. Starr, N. Buchholz, G. Rahmer, R. Schmidt, M. Warner, K.M. Merrill, C. Claver, G. Penegor, E. Mondaca, Y. Ho, K. Chopra, C. Shroff, D. Shroff
-