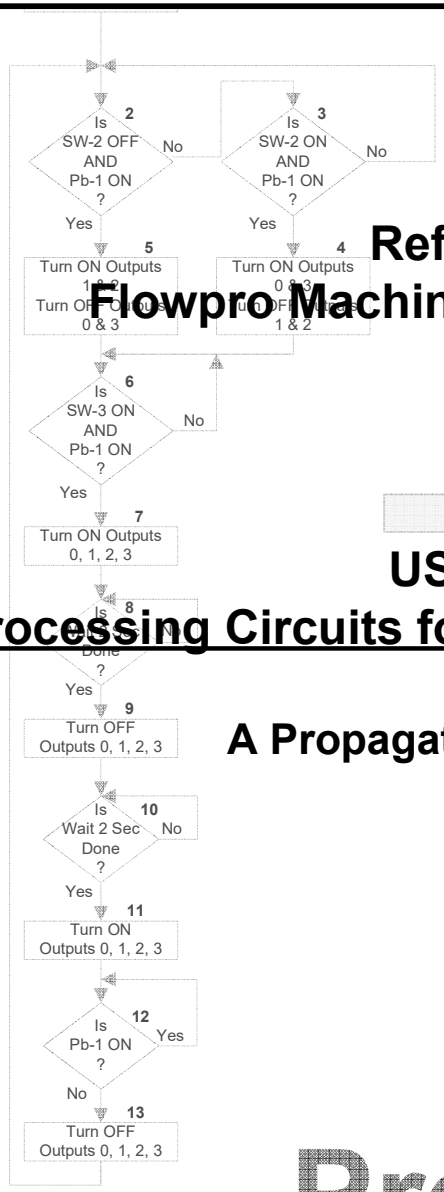
* This is preliminary V2.0 documentation. Version 1.0 of the Flowpro Machine to Verilog Utility did not support Flowpro Software objects. The circuits and Verilog code shown are V1.0. The documentation covers both.




```
output fc1clock, fc1enablk_out);
reg DFF_fc1enablk;
assign fc1enablk_criteria = input_0;
always @(negedge master_clk or negedge fc1enablk_criteria)
  begin
    if (!fc1enablk_criteria)
      DFF_fc1enablk <= 0;
    else
      begin
        if (!master_clk)
          DFF_fc1enablk <= 1;
      end
  end
assign fc1enablk_out = DFF_fc1enablk & fc1enablk_criteria;
assign fc1clock = master_clk & fc1enablk_out;
endmodule
```

```
////// fc1actblk1   T.OFF O-0   T.OFF O-1   T.OFF O-2   T.OFF O-3
output fc1actblk1_spk, fc1actblk1_out);
reg fc1actblk1_Q1, fc1actblk1_D2, fc1actblk1_Q2;
assign fc1actblk1_out = fc1clock & fc1actblk1_out;
assign fc1actblk1_spk = ~fc1actblk1_Q2 & fc1actblk1_out;
always @(posedge fc1clock or negedge fc1actblk1_out)
  begin
    if (fc1actblk1_out == 0)
      begin
        fc1actblk1_Q1 <= 0;
        fc1actblk1_Q2 <= 0;
      end
    else
      begin
        fc1actblk1_Q1 <= fc1actblk1_out;
        fc1actblk1_Q2 <= fc1actblk1_Q1;
      end
  end
endmodule
```

Reference Design: Rev 2.0 *

Flowpro Machine Processing Circuits Using Verilog

April 2024

US Patent No. 10,181,003

Processing Circuits for Asynchronous Modeling and Execution

A Propagation Flow Computational Machine

Preliminary

What is a Flowpro Machine?

From US patent No. 10181003 Column 2, Line 63 with emphasis added

The presently disclosed Flowpro Machine design model is a parallel asynchronous Stateless event model and a Flowpro Machine implementation model is also an event model. A Flowpro Machine Computation is an evolution of events (not States) that determines a computation. The process of asynchronous design may now involve drawing parallel asynchronous flowcharts of a process, synthesizing those flowcharts to Action, Test and Task Object flowchart devices, and downloading those devices to a substrate for execution as parallel asynchronous flowcharts.

Restated, A Flowpro Machine is a propagation flow based computational machine and a common software/hardware approach to parallel asynchronous design and implementation of intelligent systems, by synthesizing design flowcharts as parallel hierarchical flowchart Action Object, Test Object and Task Object constructs.

How does a Flowpro Machine work?

By propagating multiple decision flowcharts simultaneously and starting event functions as the propagation proceeds. This propagation processing can be in a physical, biological or chemical substrate.

Please explain?

Picture a decision flowchart
It is constructed using graphic symbols for an Enable bubble, Action rectangles and Test diamonds each representing event functions. These symbols are referred to as Blocks or Objects and sometimes as elements
These Action (Control) and Test (Decision) BLcks are assembled into rectangle Action Objects, diamond Test Objects and parallel flowcharts called Task Objects. A Flowpro Machine project is an assembly of flowcharts constructed in a hierarchical fashion using Task Objects
All flowchart blocks are ordered (numbered) according to the flow lines and an algorithm that attempts to number all ordered blocks higher than any blocks leading to it.
The leading edge of the propagation signal travels along the flow lines and Starts (triggers without stopping) Action or Test functions as the flowchart Blocks or Objects are encountered
All functions are built from a few Atomic Action and Test function blocks
Other than the propagation signal, spike signals are used to Start (initiate) functions

Flowpro Computational Machine Genesis

The 1980s	* US patent 4,852,047 granted, describes flowcharts to control factory automation
	* Micro-controller and I/O developed for factory automation control
The 1990s	- FloPro PC Control Software adopted by General Motors Powertrain and deployed
	- Flowpro Control Software and US patent 4,852,047 sold
The 2000s	* US Patent 6,421,821 granted, describes flowchart objects
	* FloPro Software is reborn as Flowpro Software
	* US patent 9,003,383 granted, Parallelizing serial code into parallel flowcharts
The 2010s	- Flowpro Software focused on educational Vex and Lego robots
	- US patent 1,181,003 granted (2019), Flowpro computational machine
The 2020s	* Flowpro software becomes an open system
	* A Flowpro Software to Verilog utility for FPGAs is available

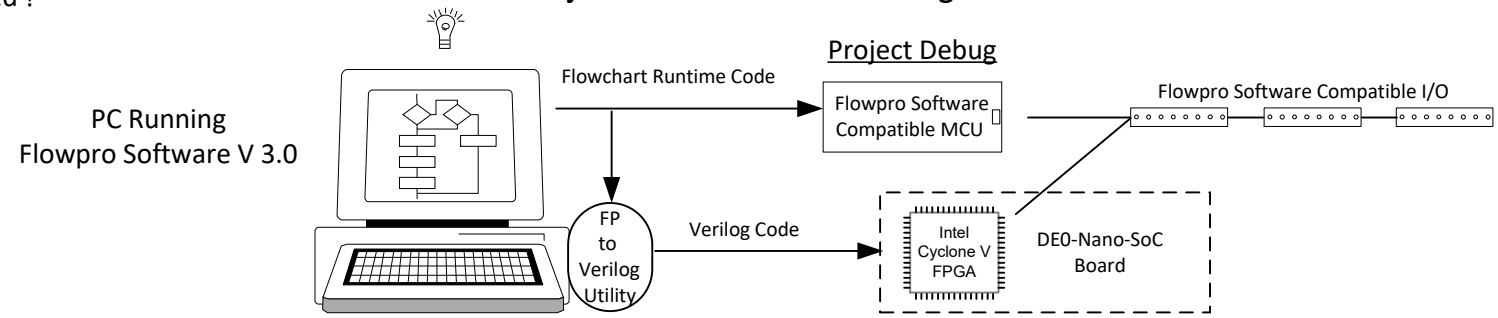
Flowpro Machine to Verilog 'Reference Design' Objective

1. Demonstrate the fundamental concepts of a Flowpro Computational Machine design and implementation including ….
   - Flowchart to RTL FPGA synthesis using Verilog
   - Provide a software utility to translate parallel Flowpro Software flowcharts to parallel Verilog flowchart structures
   - Provide a basic design of flowchart Enable, Decision, and Control blocks
   - Show Flowpro Machine Action, Test, and Task Objects and it's hierarchy of execution within Verilog
2. Promote Flowpro Software as an open system with source code and documentation
3. Provide a basic development and evaluation system for potential licensees

Flowpro Machine 'Reference Design' Implementation Flow

1. Input the application's parallel design flowcharts using PC based **open** Flowpro Software V 3.0
2. Debug the application functionality using Flowpro Software with Flowpro compatible MCU and Flowpro style SPI I/O
3. Download your compiled Flowpro Software application to disc
4. Using the **proprietary** Flowpro to Verilog utility, add real time monitor points to your design and translate the flowcharts to a Verilog v file
5. Manually install the Verilog file into a project you have created with the Intel Quartus FPGA programming system
6. Compile the Verilog file using System Verilog
7. Download the program to the Intel FPGA
8. Be amazed !

Project Flow for Reference Design: Rev 2.0

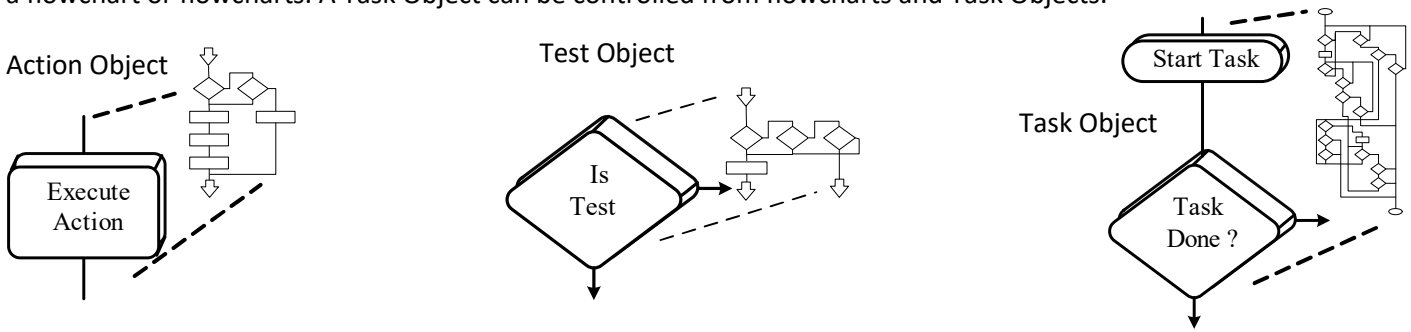


From FloPro Software to a Flowpro Machine

The original FloPro was a Microsoft DOS based application when it was sold. In the early 2000s the original Flowpro team and new members developed Windows-based Flowpro Software that carried forward fundamental execution rules from FloPro. These fundamental execution rules ensure that there is some compatibility between Flowpro Software that is a multitasked execution and a Flowpro Machine that is true parallel execution. As the Flowpro Software was enhanced and newly patented art (i.e. objects) added the art implementation was flexible as the new ideas were refined. Therefore there are slight differences between the hierarchy rules for Flowpro Software and the hierarchy rules when implementing a Flowpro Machine in hardware. The Flowpro Software to Verilog utility implements the processing of Flowpro Software objects in a manner described in US patent 10,181,003 and below.

From FloPro Software to a Flowpro Machine - Objects

Flowpro objects consists of three types, Action Object, Test Object and Task Object. An Action Object is the encapsulation of a portion of a flowchart that has one entry point, one exit point and does not contain any flowline loopback's in the encapsulation. All Blocks on a flowchart are ordered and numbered and so are the blocks inside of Objects. In effect, placing an Action Object block on a flowchart is equivalent to expanding the contents of that Object encapsulation in the flowchart between the object block's entry and exit points. A Test Object is the encapsulation of a portion of a flowchart that has one entry point, two exit points and does not contain any flowline loopback's in the encapsulation. Placing a Test Object on a flowchart is equivalent to expanding the contents of the Test Object encapsulation on the flowchart between the object block entry point and both exit points. A Task Object is equivalent to a flowchart except that a Task Object cannot be the top level of a project hierarchy. Flowpro Software and Flowpro Machine projects must always begins with a flowchart or flowcharts. A Task Object can be controlled from flowcharts and Task Objects.



A Flowpro Machine has only one copy of a Task Object and that Task Object can be controlled and monitored from any flowchart or Task Object within the Flowpro Machine project. A Flowpro Software project will potentially contain multiple copies of the same Task Object. This was early Flowpro Software thinking, this thinking is incorrect, and should be rectified in the open system Flowpro Software V3.0.

The Flowpro Machine US patent 10,181,003 describes controlling and monitoring of Task Objects. The Task Object controls are: Start (begin, resume operation after a Stop), Restart (begin, resume operation from the beginning of the Task Object), Stop (pause operation), Abort (disable task operation), and the monitoring is: Done (task complete), and Running (task operation enabled).The Flowpro Software supports Start, Restart, Stop, Abort and Done. Task Running is not supported although the Flowpro Software debugger shows status of a Task Object by applying colors to the Enable bubble of a Task Object. Deep blue if the Task Object is executing, light blue if the Task Object Is Done, orange if the Task Object is Stopped (paused) and red if the task object is not running. Rev 0 of the Flowpro to Verilog utility only supports Start, Abort and Done. A Restart function can be accomplished using an 'Abort - Wait Block - Start' sequence in a user's program. Re-issuing the same Task Object command while the command is processing has no effect on the processing. The FP to Verilog utility has a monitor point feature. A monitor point can be added to any flowchart flowline in the Flowpro project. This means that a monitor point will be routed to an output pin or LED as specified by the user.

Hierarchy of Flowpro Task Objects is an important concept to understand and it differs slightly between Flowpro Software and the Verilog implementation. This is another example of 'early thinking' that has been corrected when running the Verilog code. In Flowpro Software when a Task Object becomes Done it Aborts all Task Objects under it. This may be an issue in some applications but the workaround is easy if you are aware of it. In the Verilog code when a Task Object is Done, it is just that. A user must implicitly Abort the desired underlying Task Objects when Done. When a Flowchart or Task Object is disabled it generates a Hierarchy Abort spike (**#hracyabrt_spk) that is connected to the Enables bubble of all Task Objects immediately under it. Their Hierarchy Abort spike (**#hracyabrt_spk) is connected to the Enable bubble of all Task Objects immediately under it, and so on, and so on. Although one top level Abort spike could be used for all Task Objects under it, the cascade method above avoids fan out issues.

There can be a hierarchy of Action and Test Objects when either one is used inside one another or inside themselves. The Verilog utility allows sixteen levels of this kind of encapsulation and sixteen levels of Task Objects. Action and Test Objects are not controlled, they live and die with the Flowchart or Task Object they are within.

From FloPro Software to a Flowpro Machine - Design

The philosophy behind our approach to implementing the Flowpro to Verilog reference design utility was not to make any changes to the Flowpro Software source files. This decision was made because of unfamiliarity with the Flowpro Software development code. A secondary reason was that the Flowpro Software compiler output guarantees that the Flowpro project is ready to run. The Flowpro Software run time code is an image of the project flowcharts along with many other parameters, but, if the code format is known, the flowcharts can be recovered. We have been able to recover most the parts of the project flowcharts with the exception of the Test Object. The Flowpro Machine patent describes the Block numbering inside an Action or Test Object as ordered beginning with number 1. Flowpro Software's early thinking did not do this. Action and Tesk Objects are expanded and then the entire flowchart or Task Object is renumbered in a normal fashion. The Action and Test Object original block number entry and exit points are lost. For this reason the Flowpro Software does not show real-time flowchart status of Action and Test objects.. This should be corrected in Flowpro Software but will require multiple module updates.

Flowpro Software is a fairly complete hobbyist control system and parallel programming educational platform. The Flowpro to Verilog 'reference design' utility is intended to show flowchart to hardware structures and not build a control system or product. The following table lists the features of Flowpro Software and which features are supported in the Verilog code.

Flowpro Software to Verilog Features Supported

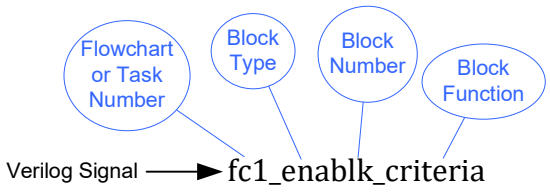
Flowpro Software Atomic Blocks	FP to Verilog Atomic Blocks
Enable Bubble - I/O, Flags	Enable Bubble - I/O, Flags
Control Block - I/O, Flags, T/C , Register	Control Block - I/O, Flags
Math Block - Integer Math	
Move Block - Parallel Data Move	
Decision Block - I/O, Flags, T/C, Register	Decision Block - I/O, Flags
Wait Block - 1 ms or sec	Wait Block - 1 ms or 1 us Time Base
Compare Block - T/C, Register, Fixed Value	
Objects	**Objects**
Action Object	Action Object
Test Object	Test Object
Task Object	Task Object
Start, Restart, Stop, Abort, Done	Start, Abort, Done
General Flowpro Software Specifications	**General FP to Verilog Specifications**
Capacity	Capacity
Blocks/Project - 1400 Maximum	Blocks/Project - 1400 Maximum
Blocks/Flowchart - 480 Maximum	Blocks/Flowchart - 480 Maximum
Tasks/Project - 128 Maximum	Tasks/Project - 128 Maximum
16 Object Levels	16 Object Levels
Data Types	Data Types
Inputs (256) - Binary (on/off) value	Inputs (256) - Binary (on/off) value
Outputs (256) - Binary (on/off) value	Outputs (256) - Binary (on/off) value
Flags (256) - Binary (on/off) value	Flags (256) - Binary (on/off) value
Timers/Counters (64) - 8 digit, xxxx.xxx sec., h:m:s	
Registers (64) - for holding or counting values, integer, time, date	

Flowpro to Verilog Utility Signal Naming Conventions

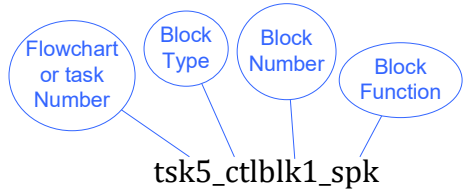
Signal names used in Verilog originate with Flowpro Software flowchart and object features
Each flowchart and Task Object has a unique name and number (see Flowpro Software 'Controller-Verify-Object Map')
All blocks and objects on Flowcharts and Objects are numbered beginning with number 1
There are only three types of flowchart blocks, Enable 'ena', Control 'ctl', Decision 'dec' and three types of objects, Action 'act', Test 'tst', Task 'tsk'
The signal identifier field is followed by an underscore and the signal's function
Flowpro can use upper and lower case characters but only lower case characters are used for Verilog signal naming
Flowpro characters / and \ and (space) and - (dash) are converted to _ (underscore) when named in the Verilog utility
Input and Output are key words in Verilog, Flowpro therefore uses fm_in# and fm_out# are Flowpro I/O system references used in Verilog

Examples- Flowchart or Task Object Block Verilog Signals

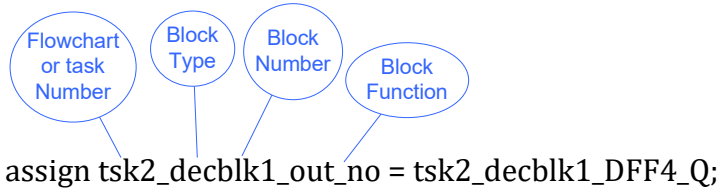
The numbering of all flowcharts and objects begins with the number 1 and in order to have a unique ID for each Verilog signal in all objects, a prefix can be added to each Verilog signal to achieve a unique ID. Each Flowchart and Task Object in a Flowpro Software project has an identifying number assigned to it and shown in the Flowpro Software Object Map. Flowcharts and Task Object Verilog signals are labeled with the prefix of the Flowchart number or the Task Object number assigned to it by the Flowpro Software Object Map.



Signal Definition: Flowchart #1, Enable, bubble, enabling signal



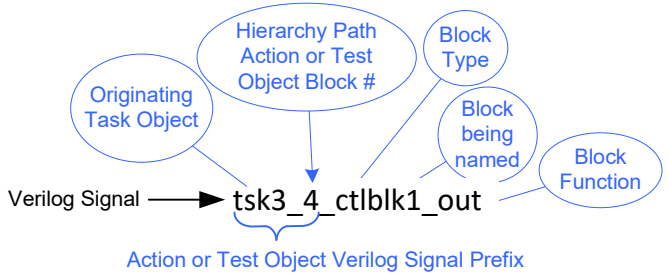
Signal Definition: Task Objet #5,Control Block #1, spike output from the block that sets and resets output registers

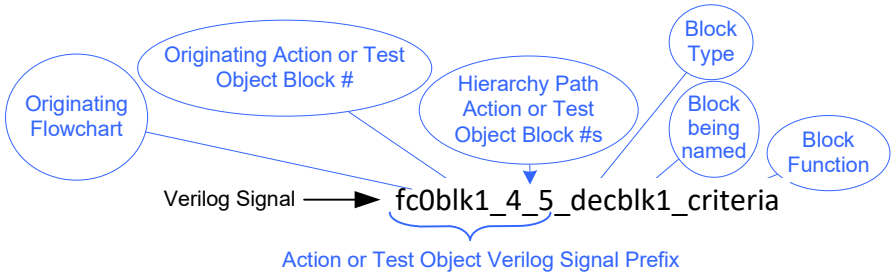


Signal Definition: Task Object #2, Decision Block #2 No out from the block

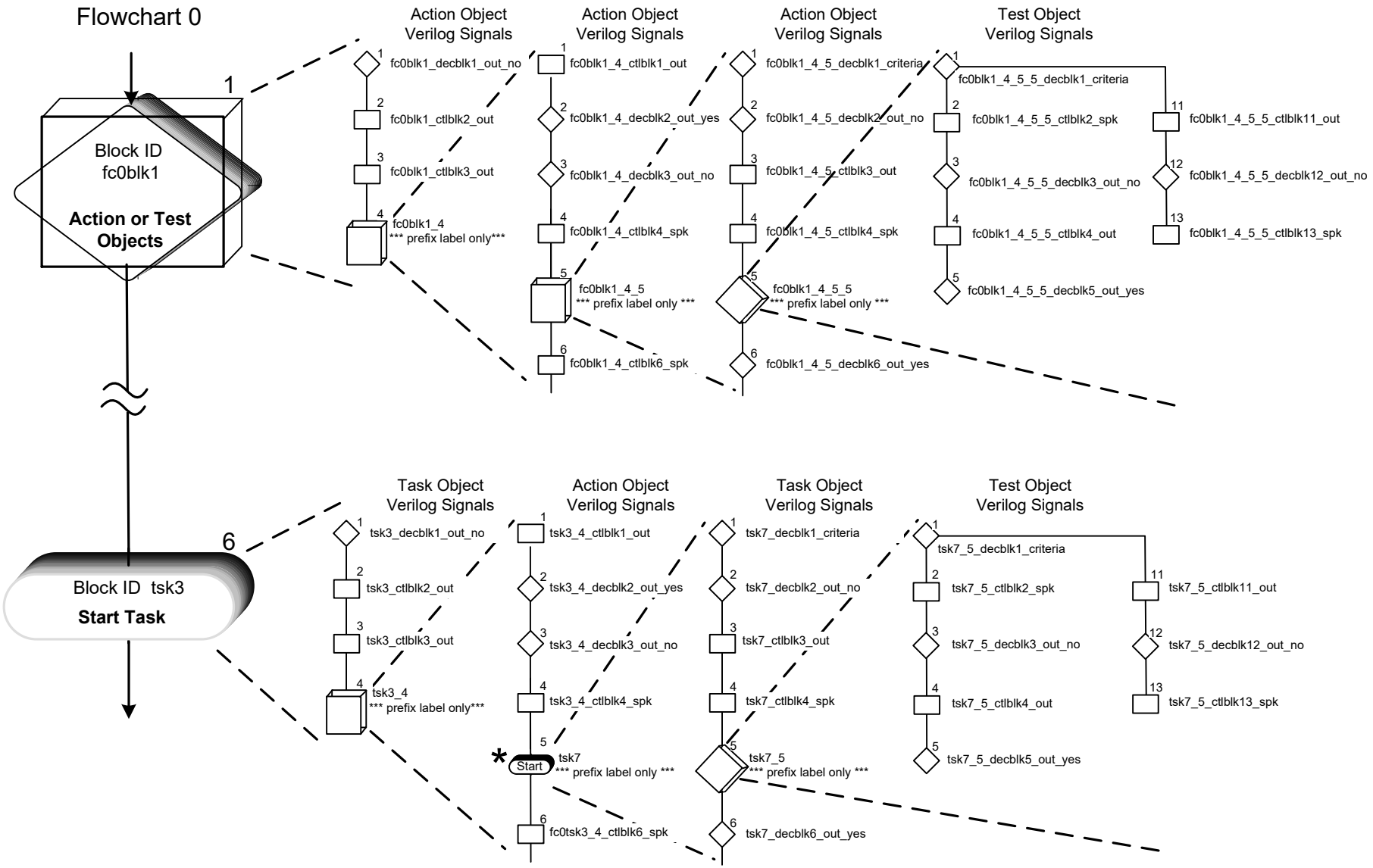
Examples- Action or Test Object Block Verilog Signals

Action and Test Objects are not assigned a number by the Flowpro Software, so one must be developed and assigned to each instantiation of these objects. One way to do this is to use the hierarchy path to each Action and Test Object instantiation. All hierarchies begin at the flowchart level and go through a Task Object, which is identifiable, and/or Action and/or Test Objects which are not. Each Action and Test Object will have a unique element (Block) number that can be used as part of a unique signal naming prefix. The prefix can be easily expanded as the hierarchy gets deeper and deeper, by using the object element number at each level of the hierarchy. (See below and next page)





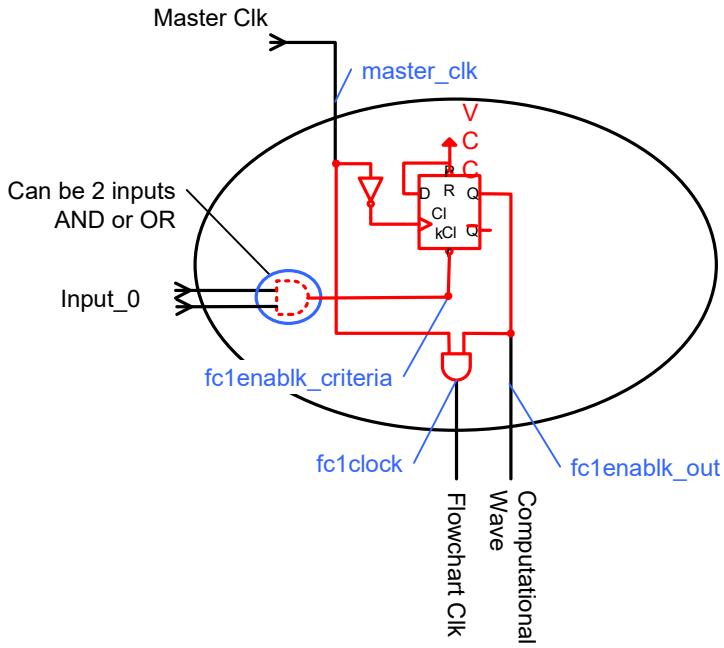
Verilog Signal Naming through Flowpro Objects



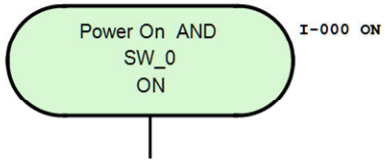
* Flowpro Software currently does not allow control or monitoring of a Task Object. US patent 10,181,003 calls for it, so it is shown to highlight that prefixes begin again with a new Task Object.

Flowpro to Verilog Enable Block Conversion
REV 1.0

Original Flowpro Machine Concept



Flowpro Software

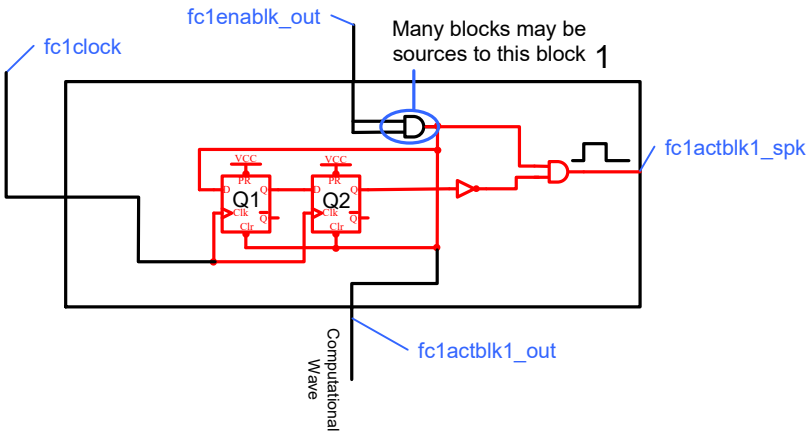


Verilog

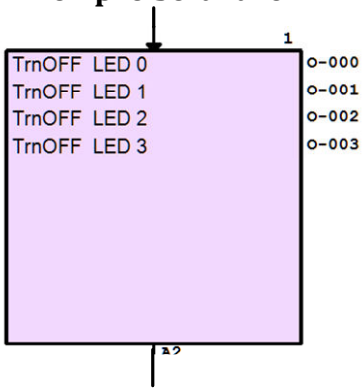
```
///// fc1enablk     I-0 ON

module fc1enablk (input master_clk, input_0,
                  output fc1clock, fc1enablk_out);
                  reg DFF_fc1enablk;
assign fc1enablk_criteria = input_0;
always @(negedge master_clk or negedge fc1enablk_criteria)
   begin
        if (!fc1enablk_criteria)
            DFF_fc1enablk <= 0;
        else
            begin
                if (!master_clk)
                    DFF_fc1enablk <= 1;
            end
   end
assign fc1enablk_out = DFF_fc1enablk & fc1enablk_criteria;
assign fc1clock = master_clk & fc1enablk_out;
endmodule
```

Flowpro to Verilog Action Block Conversion
REV 1.0

Original Flowpro Machine Concept



Flowpro Software



Verilog

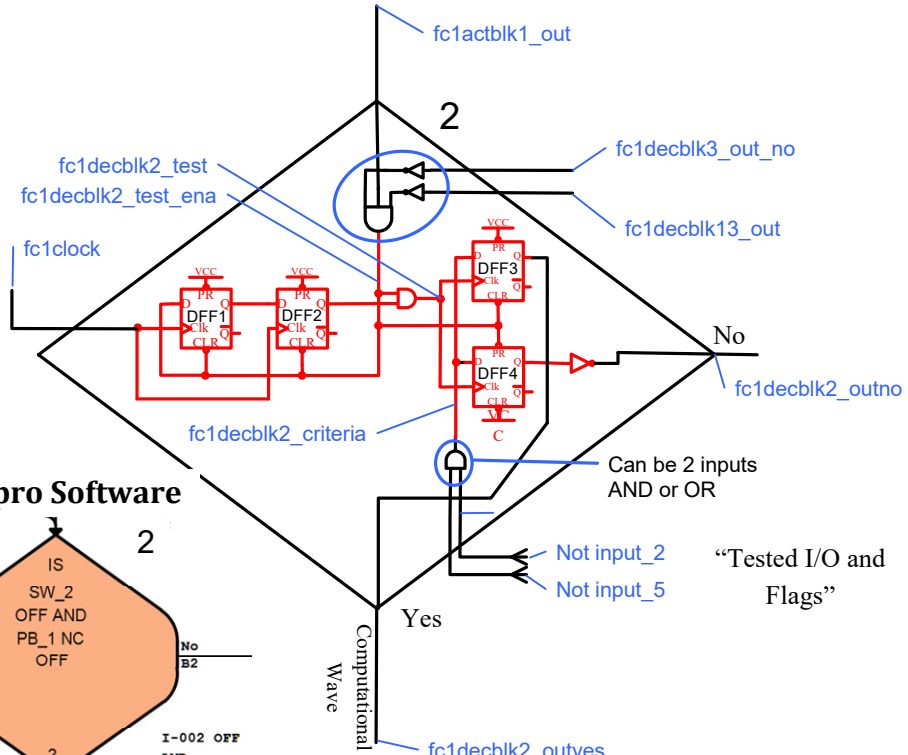
```
///// fc1actblk1    T.OFF O-0    T.OFF O-1    T.OFF O-2    T.OFF O-3

module fc1actblk1 (input fc1clock, fc1enablk_out,
                   output fc1actblk1_spk, fc1actblk1_out);
                   reg fc1actblk1_Q1, fc1actblk1_D2, fc1actblk1_Q2;
assign fc1actblk1_out = fc1enablk_out;
assign fc1actblk1_spk = ~fc1actblk1_Q2 & fc1actblk1_out;
always @(posedge fc1clock or negedge fc1actblk1_out)
   begin
        if (fc1actblk1_out == 0)
            begin
                  fc1actblk1_Q1 <= 0;
                  fc1actblk1_Q2 <= 0;
            end
        else
            begin
                  fc1actblk1_Q1 <= fc1actblk1_out;
                  fc1actblk1_Q2 <= fc1actblk1_Q1;
            end
    end
endmodule
```

Flowpro to Verilog Decision Block Conversion
REV 1.0

Verilog

```
///// fc1decblk2    I-2 OFF   AND   I-5 OFF

module fc1decblk2 (input fc1clock, input_2, input_5, fc1actblk1_out, fc1decblk3_out_no, fc1actblk14_out,
                   output fc1decblk2_out_yes, fc1decblk2_out_no);
                   reg fc1decblk2_DFF1_Q, fc1decblk2_DFF2_Q, fc1decblk2_DFF3_Q, fc1decblk2_DFF4_Q;
assign fc1decblk2_test_ena = fc1actblk1_out ^ fc1decblk3_out_no ^ fc1actblk14_out;
assign fc1decblk2_test = fc1decblk2_DFF2_Q & fc1decblk2_test_ena;
assign fc1decblk2_criteria = ~input_2 & ~input_5;
always @(posedge fc1clock or negedge fc1decblk2_test_ena)
   begin
       if (fc1decblk2_test_ena == 0)
           begin
               fc1decblk2_DFF1_Q <= 0;
               fc1decblk2_DFF2_Q <= 0;
           end
       else
           begin
               fc1decblk2_DFF1_Q <= fc1decblk2_test_ena;
               fc1decblk2_DFF2_Q <= fc1decblk2_DFF1_Q;
           end
    end
always @(posedge fc1decblk2_test or negedge fc1decblk2_test_ena)
   begin
       if (fc1decblk2_test_ena == 0)
           begin
               fc1decblk2_DFF3_Q = 0;
               fc1decblk2_DFF4_Q = 0;
           end
       else
           begin
               fc1decblk2_DFF3_Q <= fc1decblk2_criteria;
               fc1decblk2_DFF4_Q <= ~fc1decblk2_criteria;
           end
    end
        assign fc1decblk2_out_no = fc1decblk2_DFF4_Q;
        assign fc1decblk2_out_yes = fc1decblk2_DFF3_Q;
endmodule
```

Original Flowpro Machine Concept

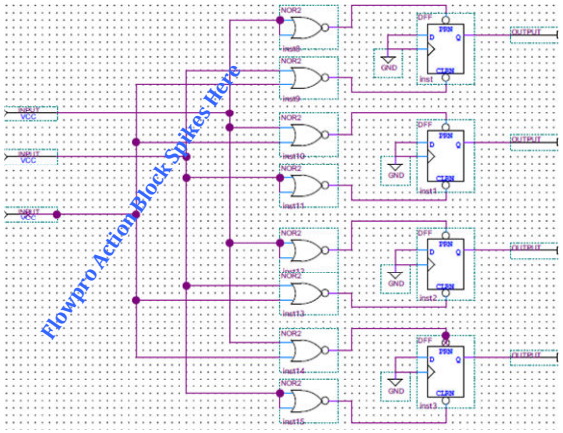


Flowpro Software

Flowpro to Verilog Output Registers

Original Flowpro Machine Concept

Four Registers shown



Flowpro Software

XRef	O-000	LED 0
XRef	O-001	LED 1
XRef	O-002	LED 2
XRef	O-003	LED 3
XRef	O-004	LED 4
XRef	O-005	LED 5
XRef	O-006	LED 6
XRef	O-007	LED 7
XRef	O-008	Monitor PIN_Y15
XRef	O-009	Monitor PIN_AG28
XRef	O-010	Flag 0
XRef	O-011	Flag 1
XRef	O-012	I/O Load-Low True
XRef	O-013	I/O Clock
XRef	O-014	MOSI Serial Out
XRef	O-015	Output 015
XRef	O-016	Output 016
XRef	O-017	Output 017
XRef	O-018	Output 018
XRef	O-019	Output 019
XRef	O-020	Output 020
XRef	O-021	Output 021
XRef	O-022	Output 022
XRef	O-023	Output 023

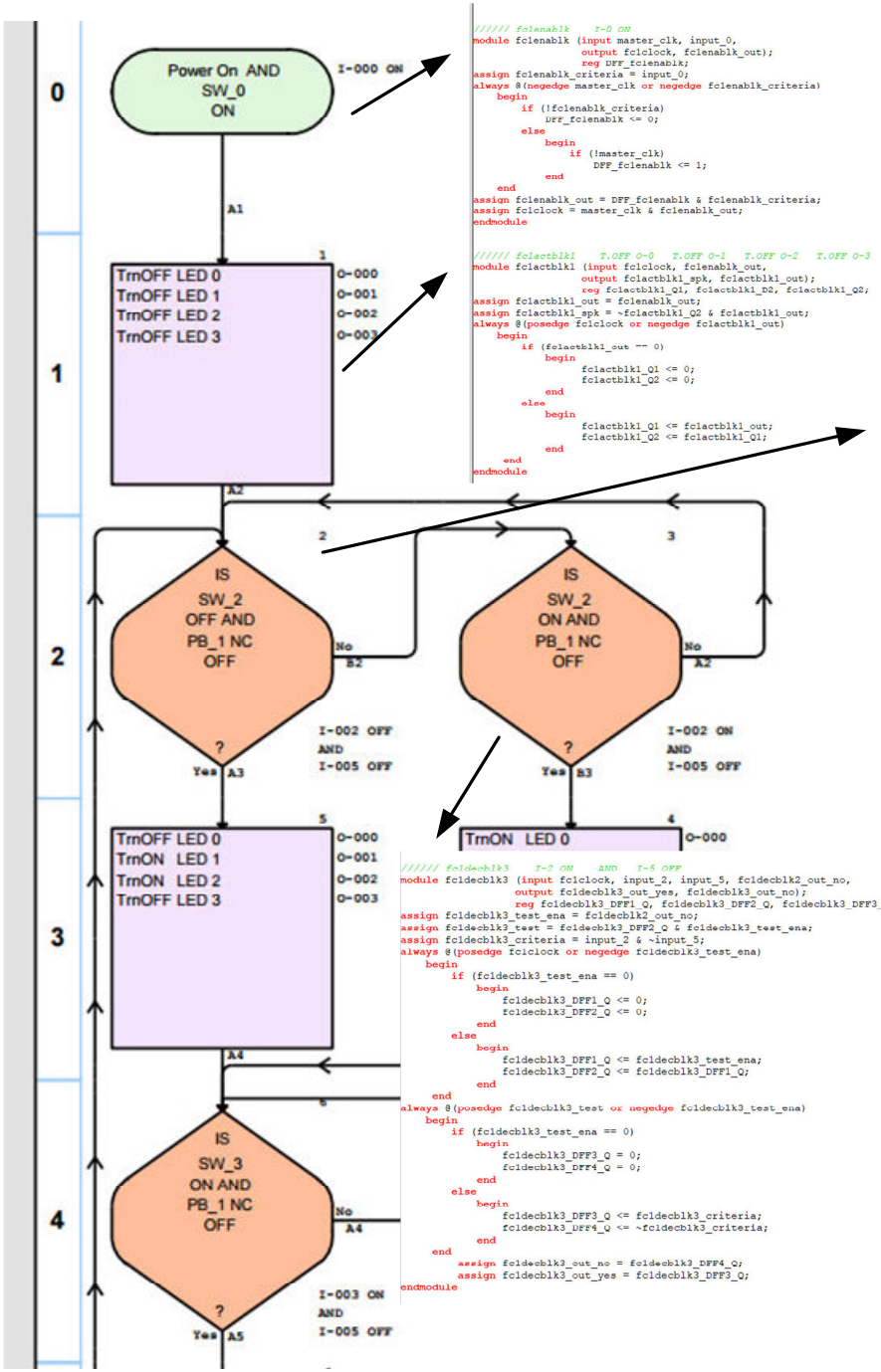
Verilog

Code for One Register Shown

```
///// Output registers


module out0 (input wire fc1actblk1_spk, fc1actblk4_spk, fc1actblk5_spk, fc1actblk7_spk, fc
output wire output_0, fm_out0);
reg DFF_out0;

wire DFF_out0_clr;
wire DFF_out0_q;
wire DFF_out0_set;

assign DFF_out0_q = 0;

always@(posedge DFF_out0_q or negedge DFF_out0_clr or negedge DFF_out0_set)
begin
   if (!DFF_out0_clr)
       begin
       output_0 <= 0;
       end
   else
   if (!DFF_out0_set)
       begin
       output_0 <= 1;
       end
   else
       begin
       output_0 <= DFF_out0_q;
       end
end

assign fm_out0 = output_0;
assign DFF_out0_set = ~(fc1actblk4_spk | fc1actblk7_spk | fc1actblk12_spk);
assign DFF_out0_clr = ~(fc1actblk1_spk | fc1actblk5_spk | fc1actblk9_spk | fc1actblk14_spk
endmodule
```

Not All Spikes are Shown

Not All Spikes are Shown




```
////// fclenablk    I-0 ON
module fclenablk (input master_clk, input_0,
                  output fclclock, fclenablk_out);
                  reg DFF_fclenablk;
assign fclenablk_criteria = input_0;
always @(negedge master_clk or negedge fclenablk_criteria)
    begin
        if (!fclenablk_criteria)
            DFF_fclenablk <= 0;
        else
            begin
                if (!master_clk)
                    DFF_fclenablk <= 1;
            end
    end
assign fclenablk_out = DFF_fclenablk & fclenablk_criteria;
assign fclclock = master_clk & fclenablk_out;
endmodule
```

```
////// fclactblk1    T.OFF O-0   T.OFF O-1   T.OFF O-2   T.OFF O-3
module fclactblk1 (input fclclock, fclenablk_out,
                   output fclactblk1_spk, fclactblk1_out);
                   reg fclactblk1_Q1, fclactblk1_D2, fclactblk1_Q2;
assign fclactblk1_out = fclenablk_out;
assign fclactblk1_spk = ~fclactblk1_Q2 & fclactblk1_out;
always @(posedge fclclock or negedge fclactblk1_out)
    begin
        if (fclactblk1_out == 0)
            begin
                fclactblk1_Q1 <= 0;
                fclactblk1_Q2 <= 0;
            end
        else
            begin
                fclactblk1_Q1 <= fclactblk1_out;
                fclactblk1_Q2 <= fclactblk1_Q1;
            end
    end
endmodule
```

```
////// fcldecblk2    I-2 OFF   AND   I-5 OFF
module fcldecblk2 (input fclclock, input_2, input_5, fclactblk1_out, fcldecblk3_out_no, fclactblk14_out,
                   output fcldecblk2_out_yes, fcldecblk2_out_no);
                   reg fcldecblk2_DFF1_Q, fcldecblk2_DFF2_Q, fcldecblk2_DFF3_Q, fcldecblk2_DFF4_Q;
assign fcldecblk2_test_ena = fclactblk1_out ^ fcldecblk3_out_no ^ fclactblk14_out;
assign fcldecblk2_test = fcldecblk2_DFF2_Q & fcldecblk2_test_ena;
assign fcldecblk2_criteria = ~input_2 & ~input_5;
always @(posedge fclclock or negedge fcldecblk2_test_ena)
    begin
        if (fcldecblk2_test_ena == 0)
            begin
                fcldecblk2_DFF1_Q <= 0;
                fcldecblk2_DFF2_Q <= 0;
            end
        else
            begin
                fcldecblk2_DFF1_Q <= fcldecblk2_test_ena;
                fcldecblk2_DFF2_Q <= fcldecblk2_DFF1_Q;
            end
    end
always @(posedge fcldecblk2_test or negedge fcldecblk2_test_ena)
    begin
        if (fcldecblk2_test_ena == 0)
            begin
                fcldecblk2_DFF3_Q = 0;
                fcldecblk2_DFF4_Q = 0;
            end
        else
            begin
                fcldecblk2_DFF3_Q <= fcldecblk2_criteria;
                fcldecblk2_DFF4_Q <= ~fcldecblk2_criteria;
            end
    end
    assign fcldecblk2_out_no = fcldecblk2_DFF4_Q;
    assign fcldecblk2_out_yes = fcldecblk2_DFF3_Q;
endmodule
```

```
////// fcldecblk3    I-2 ON    AND   I-5 OFF
module fcldecblk3 (input fclclock, input_2, input_5, fcldecblk2_out_no,
                   output fcldecblk3_out_yes, fcldecblk3_out_no);
                   reg fcldecblk3_DFF1_Q, fcldecblk3_DFF2_Q, fcldecblk3_DFF3_Q, fcldecblk3_DFF4_Q;
assign fcldecblk3_test_ena = fcldecblk2_out_no;
assign fcldecblk3_test = fcldecblk3_DFF2_Q & fcldecblk3_test_ena;
assign fcldecblk3_criteria = input_2 & ~input_5;
always @(posedge fclclock or negedge fcldecblk3_test_ena)
    begin
        if (fcldecblk3_test_ena == 0)
            begin
                fcldecblk3_DFF1_Q <= 0;
                fcldecblk3_DFF2_Q <= 0;
            end
        else
            begin
                fcldecblk3_DFF1_Q <= fcldecblk3_test_ena;
                fcldecblk3_DFF2_Q <= fcldecblk3_DFF1_Q;
            end
    end
always @(posedge fcldecblk3_test or negedge fcldecblk3_test_ena)
    begin
        if (fcldecblk3_test_ena == 0)
            begin
                fcldecblk3_DFF3_Q = 0;
                fcldecblk3_DFF4_Q = 0;
            end
        else
            begin
                fcldecblk3_DFF3_Q <= fcldecblk3_criteria;
                fcldecblk3_DFF4_Q <= ~fcldecblk3_criteria;
            end
    end
    assign fcldecblk3_out_no = fcldecblk3_DFF4_Q;
    assign fcldecblk3_out_yes = fcldecblk3_DFF3_Q;
endmodule
```

Flowchart to Verilog Code
V1.0