# Reference Design: V 2.0

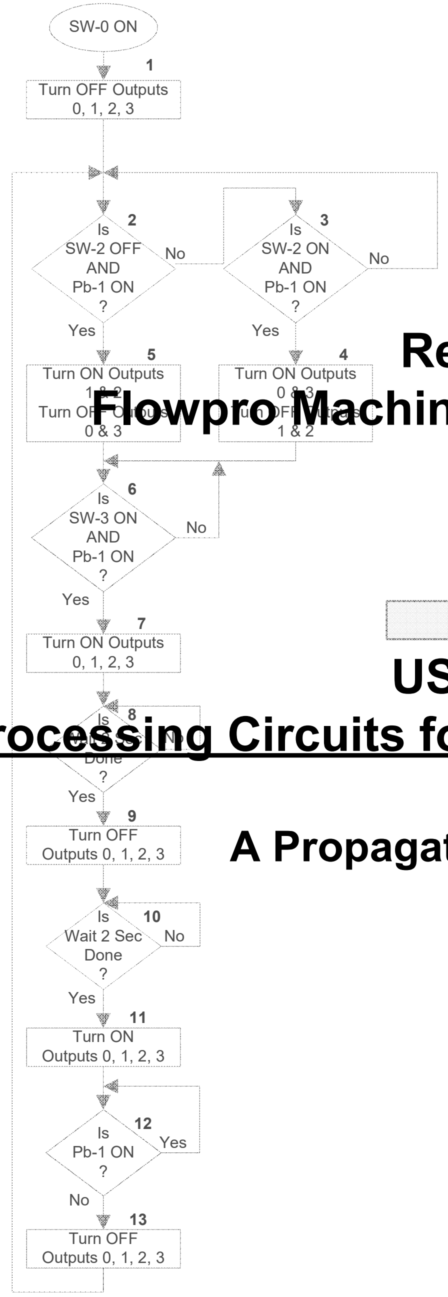
# Flowpro Machine Processing Circuits Using Verilog

**April 2025**

## US Patent No. 10,181,003

## <u>Processing Circuits for Asynchronous Modeling and Execution</u>

## A Propagation Flow Computational Machine


SW-0 ON
1
Turn OFF Outputs 0, 1, 2, 3
2
Is SW-2 OFF AND Pb-1 ON ?
No
Yes
3
Is SW-2 ON AND Pb-1 ON ?
No
Yes
5
Turn ON Outputs 1 & 2 Turn OFF Outputs 0 & 3
4
Turn ON Outputs 0 & 3 Turn OFF Outputs 1 & 2
6
Is SW-3 ON AND Pb-1 ON ?
No
Yes
7
Turn ON Outputs 0, 1, 2, 3
8
Is Wait 2 Sec Done ?
No
Yes
9
Turn OFF Outputs 0, 1, 2, 3
10
Is Wait 2 Sec Done ?
No
Yes
11
Turn ON Outputs 0, 1, 2, 3
12
Is Pb-1 ON ?
Yes
No
13
Turn OFF Outputs 0, 1, 2, 3

```
//////  Block functions  //////

////// 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


////// 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
```

# 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 these propagations proceed. This propagation processing can be in a physical, biological or chemical substrate.

### Please explain?

See Figure 1 Page 14 “FM-8BA 8 Bit Adder Control” flowchart

A Flowpro Machine 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) blocks are assembled into 3D rectangle Action Objects, 3D diamond Test Objects and 3D 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) and Abort (cancel) 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 - (Expired)
* FloPro Software is reborn as Flowpro Software
* US patent 9,003,383 granted, Parallelizing serial code into
parallel flowcharts - (Public Domain) The 2010s * Flowpro Software focused on educational
Vex and Lego robots
* US patent 10,181,003 granted (2019), Flowpro computational machine The
2020s * Flowpro Software becomes an open system
* A Flowpro Software to Verilog utility for FPGAs is available

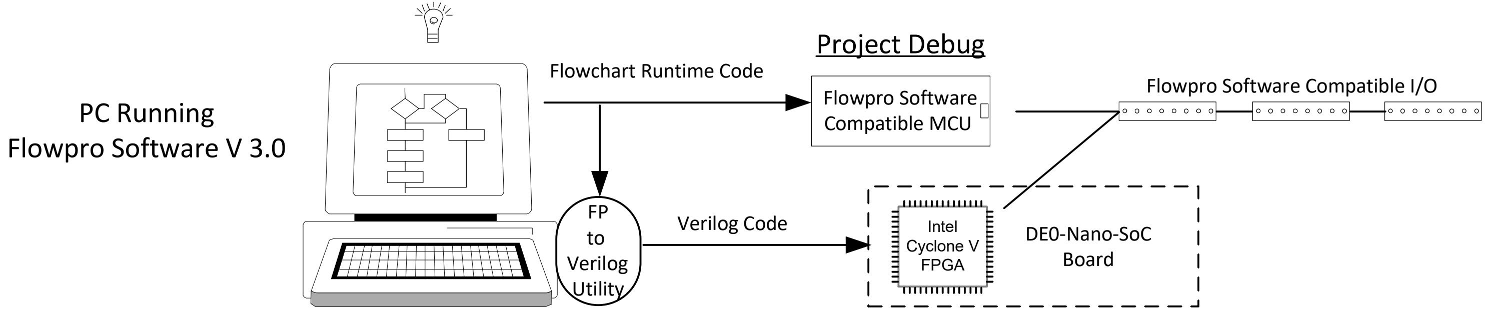
## Flowpro Machine to Verilog ‘Reference Design’ Objective

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 an FPGA project you have created with the Intel Quartus FPGA programming system
6. Compile the Verilog file using <u>System Verilog</u>
7. Download the program to the Intel FPGA
8. Be amazed !

**Project Flow for Reference Design: Rev 2.0**

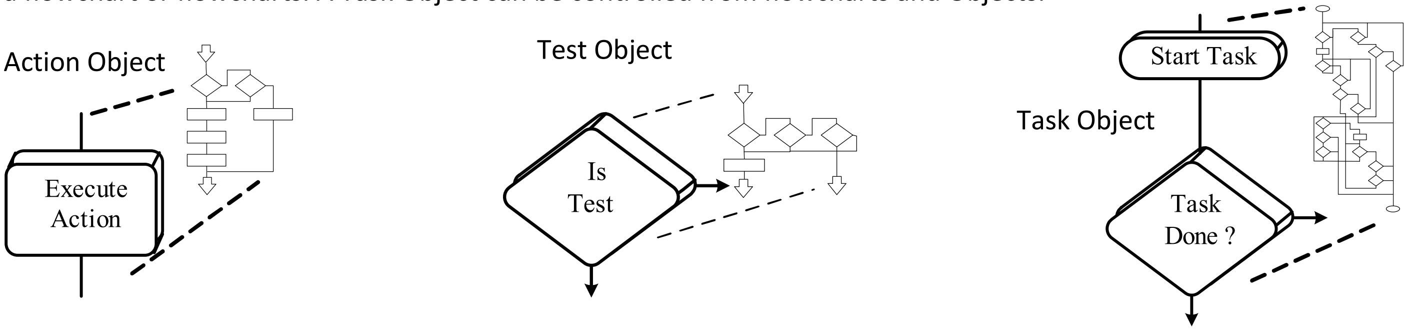


## From FloPro Software to a Flowpro Machine

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 DOS 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 ideas (i.e. objects) were added the implementation was flexible as these 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, 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 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 2 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 (see page 12).

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 Aborted, 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. 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 by the utility to generate the Verilog code.

### Flowpro Software to Verilog Features Supported

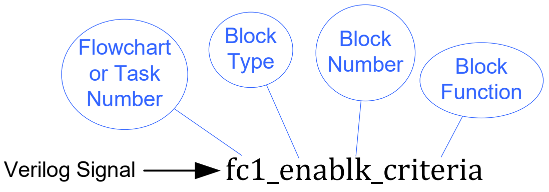
| **Flowpro Software Atomic Blocks** | **FP to Verilog Atomic Blocks** |
|---|---|
| Enable Bubble - I/O, Flags | Enable Bubble - I/O, Flags |
| Control Block - I/O, Flags, T/C , Register | Control Block - I/O, Flags |
| Math Block - Integer Math | |
| Move Block - Parallel Data Move | |
| Decision Block - 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' Pg. 13)
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
Flowpro allows a name to begin with a number but Verilog does not. Be sure that your Flowpro names don't begin with a number
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

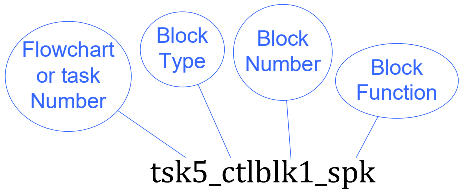
## Examples- Flowchart or Task Object Block Verilog Signals

The numbering of all Blocks used in 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.



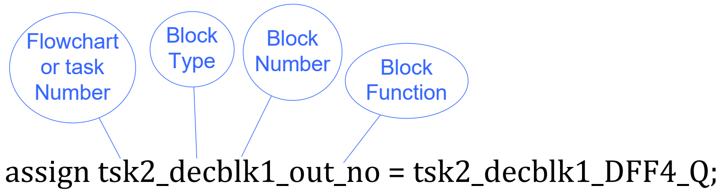
Verilog Signal → fc1_enablk_criteria

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



tsk5_ctlblk1_spk

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

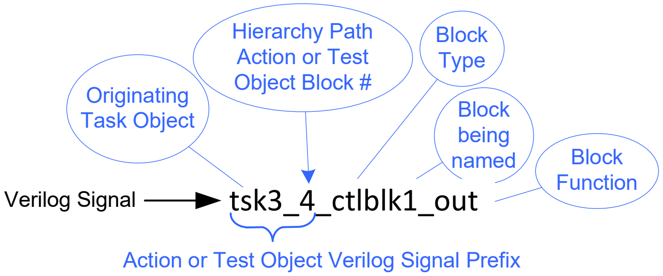


assign tsk2_decblk1_out_no = tsk2_decblk1_DFF4_Q;

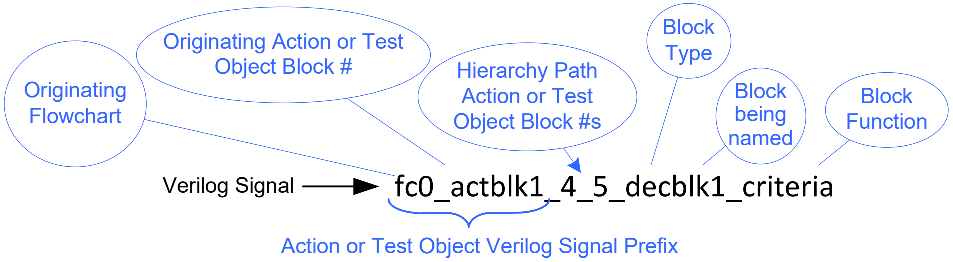
Signal Definition: Task Object #2, Decision Block #1
No out from the block

## 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.


Originating Task Object
Hierarchy Path Action or Test Object Block #
Block Type
Block being named
Block Function
Action or Test Object Verilog Signal Prefix

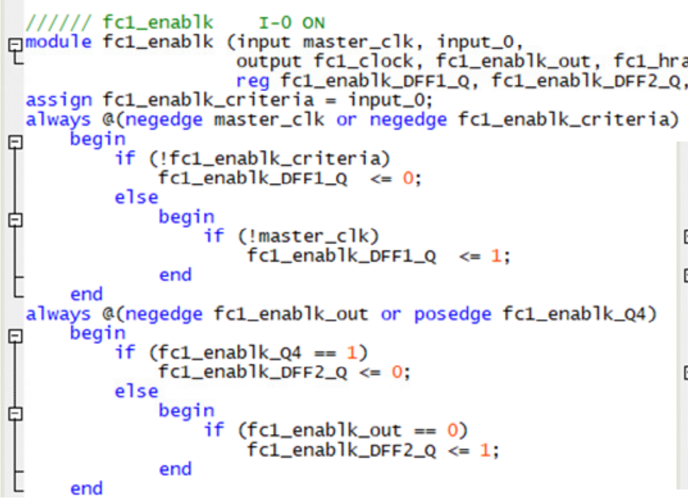
Verilog Signal → tsk3_4_ctlblk1_out


Originating Flowchart
Originating Action or Test Object Block #
Hierarchy Path Action or Test Object Block #s
Block Type
Block being named
Block Function
Action or Test Object Verilog Signal Prefix

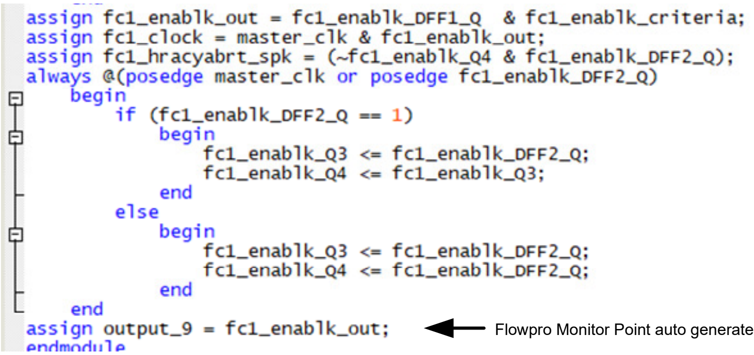
Verilog Signal → fc0_actblk1_4_5_decblk1_criteria

# Flowpro to Verilog: Flowchart Enable Block Conversion
# REV 2.0

## System Verilog



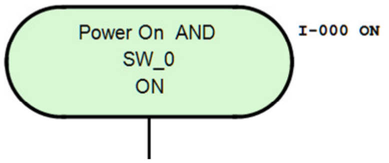
```
////// fc1_enablk    I-0 ON
module fc1_enablk (input master_clk, input_0,
                   output fc1_clock, fc1_enablk_out, fc1_hracyabrt_spk, output_9);
                   reg fc1_enablk_DFF1_Q, fc1_enablk_DFF2_Q, fc1_enablk_Q3, fc1_enablk_Q4;
assign fc1_enablk_criteria = input_0;
always @(negedge master_clk or negedge fc1_enablk_criteria)
    begin
        if (!fc1_enablk_criteria)
            fc1_enablk_DFF1_Q  <= 0;
        else
            begin
                if (!master_clk)
                    fc1_enablk_DFF1_Q  <= 1;
            end
    end
always @(negedge fc1_enablk_out or posedge fc1_enablk_Q4)
    begin
        if (fc1_enablk_Q4 == 1)
            fc1_enablk_DFF2_Q <= 0;
        else
            begin
                if (fc1_enablk_out == 0)
                    fc1_enablk_DFF2_Q <= 1;
            end
    end
```



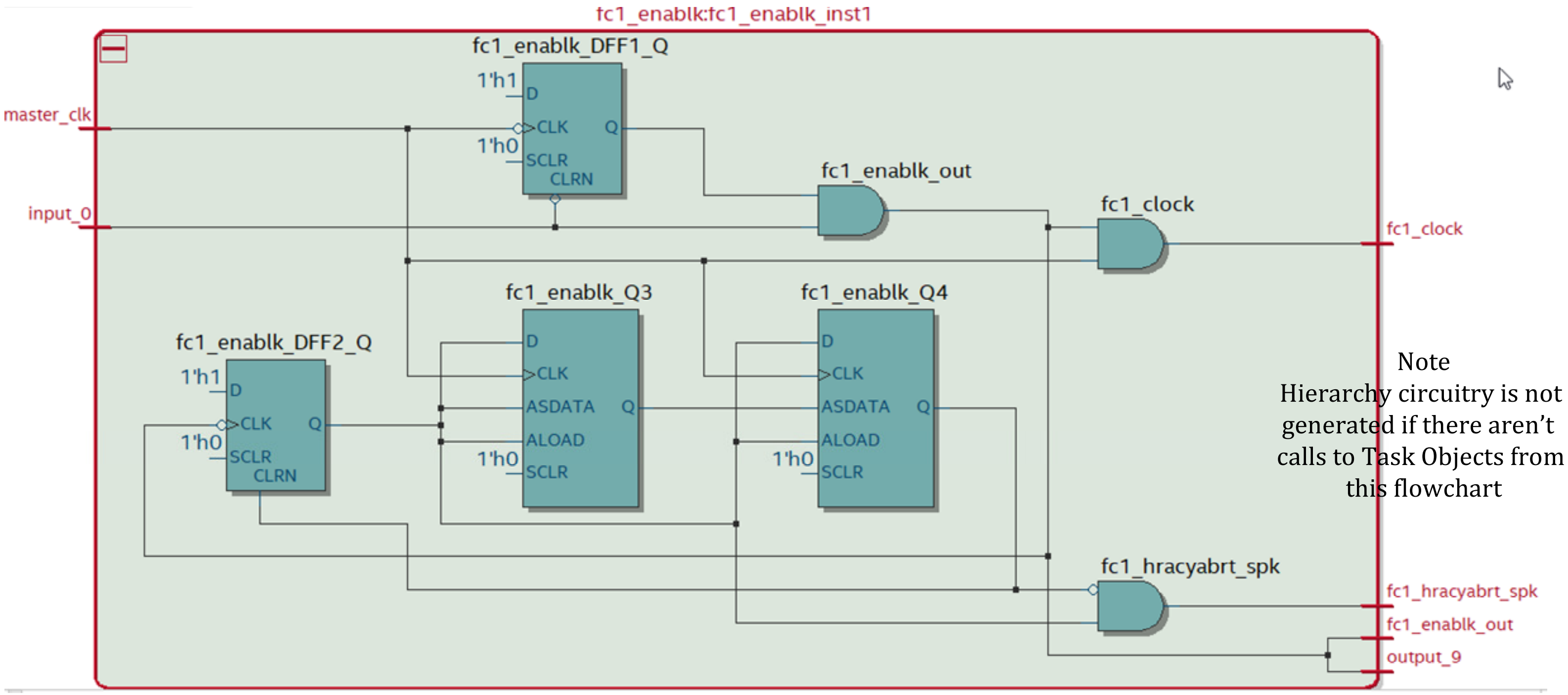
```
assign fc1_enablk_out = fc1_enablk_DFF1_Q  & fc1_enablk_criteria;
assign fc1_clock = master_clk & fc1_enablk_out;
assign fc1_hracyabrt_spk = (~fc1_enablk_Q4 & fc1_enablk_DFF2_Q);
always @(posedge master_clk or posedge fc1_enablk_DFF2_Q)
    begin
        if (fc1_enablk_DFF2_Q == 1)
            begin
                fc1_enablk_Q3 <= fc1_enablk_DFF2_Q;
                fc1_enablk_Q4 <= fc1_enablk_Q3;
            end
        else
            begin
                fc1_enablk_Q3 <= fc1_enablk_DFF2_Q;
                fc1_enablk_Q4 <= fc1_enablk_DFF2_Q;
            end
    end
assign output_9 = fc1_enablk_out;
endmodule
```

Flowpro Monitor Point auto generated  for testing

## Flowpro Software



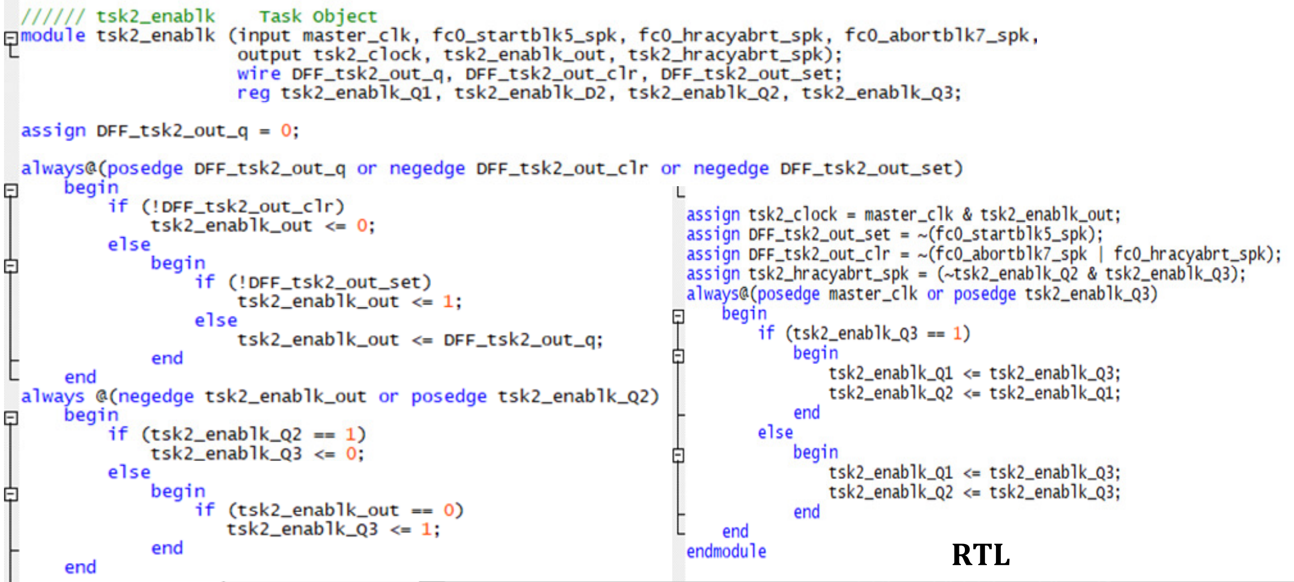
## RTL



Note
Hierarchy circuitry is not generated if there aren't calls to Task Objects from this flowchart

# Flowpro to Verilog: Task Object Enable Block Conversion
## REV 2.0

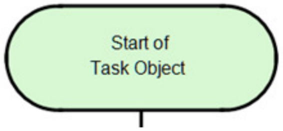
## System Verilog



```
////// tsk2_enablk    Task Object
module tsk2_enablk (input master_clk, fc0_startblk5_spk, fc0_hracyabrt_spk, fc0_abortblk7_spk,
                    output tsk2_clock, tsk2_enablk_out, tsk2_hracyabrt_spk);
                    wire DFF_tsk2_out_q, DFF_tsk2_out_clr, DFF_tsk2_out_set;
                    reg tsk2_enablk_Q1, tsk2_enablk_D2, tsk2_enablk_Q2, tsk2_enablk_Q3;

assign DFF_tsk2_out_q = 0;

always@(posedge DFF_tsk2_out_q or negedge DFF_tsk2_out_clr or negedge DFF_tsk2_out_set)
    begin
        if (!DFF_tsk2_out_clr)
            tsk2_enablk_out <= 0;
        else
            begin
                if (!DFF_tsk2_out_set)
                    tsk2_enablk_out <= 1;
                else
                    tsk2_enablk_out <= DFF_tsk2_out_q;
            end
    end
always @(negedge tsk2_enablk_out or posedge tsk2_enablk_Q2)
    begin
        if (tsk2_enablk_Q2 == 1)
            tsk2_enablk_Q3 <= 0;
        else
            begin
                if (tsk2_enablk_out == 0)
                   tsk2_enablk_Q3 <= 1;
            end
    end
```

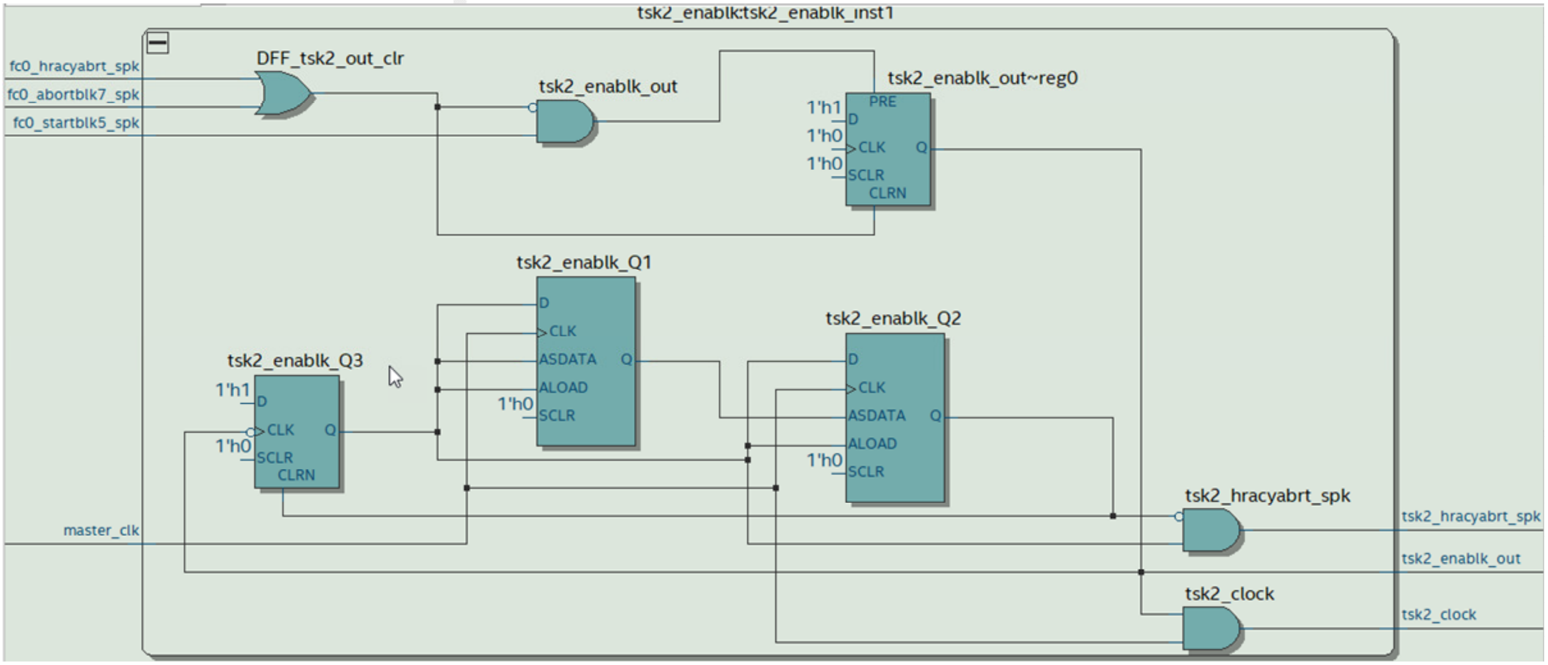
```
assign tsk2_clock = master_clk & tsk2_enablk_out;
assign DFF_tsk2_out_set = ~(fc0_startblk5_spk);
assign DFF_tsk2_out_clr = ~(fc0_abortblk7_spk | fc0_hracyabrt_spk);
assign tsk2_hracyabrt_spk = (~tsk2_enablk_Q2 & tsk2_enablk_Q3);
always@(posedge master_clk or posedge tsk2_enablk_Q3)
    begin
        if (tsk2_enablk_Q3 == 1)
            begin
                tsk2_enablk_Q1 <= tsk2_enablk_Q3;
                tsk2_enablk_Q2 <= tsk2_enablk_Q1;
            end
        else
            begin
                tsk2_enablk_Q1 <= tsk2_enablk_Q3;
                tsk2_enablk_Q2 <= tsk2_enablk_Q3;
            end
    end
endmodule
```

## Flowpro Software



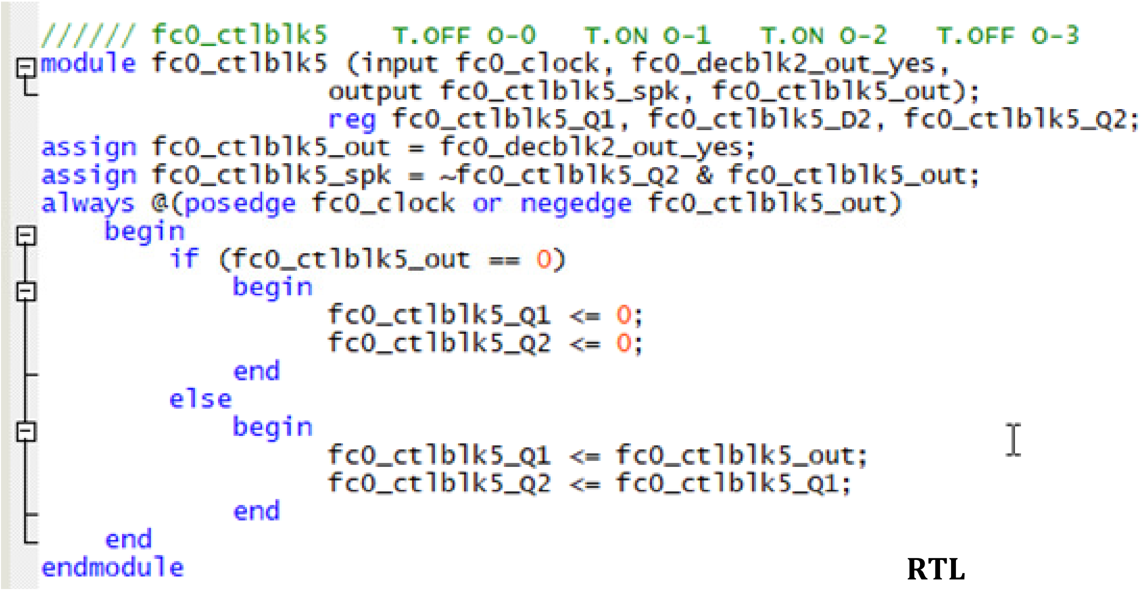
Note
Hierarchy circuitry is not generated if there aren't calls to other Task Objects from this Task Object

## RTL



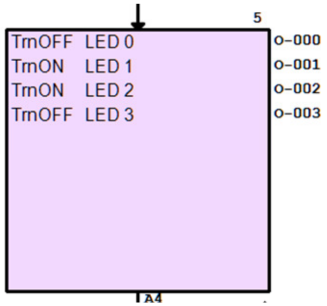
# Flowpro to Verilog: Control Block Conversion
REV 2.0

## System Verilog

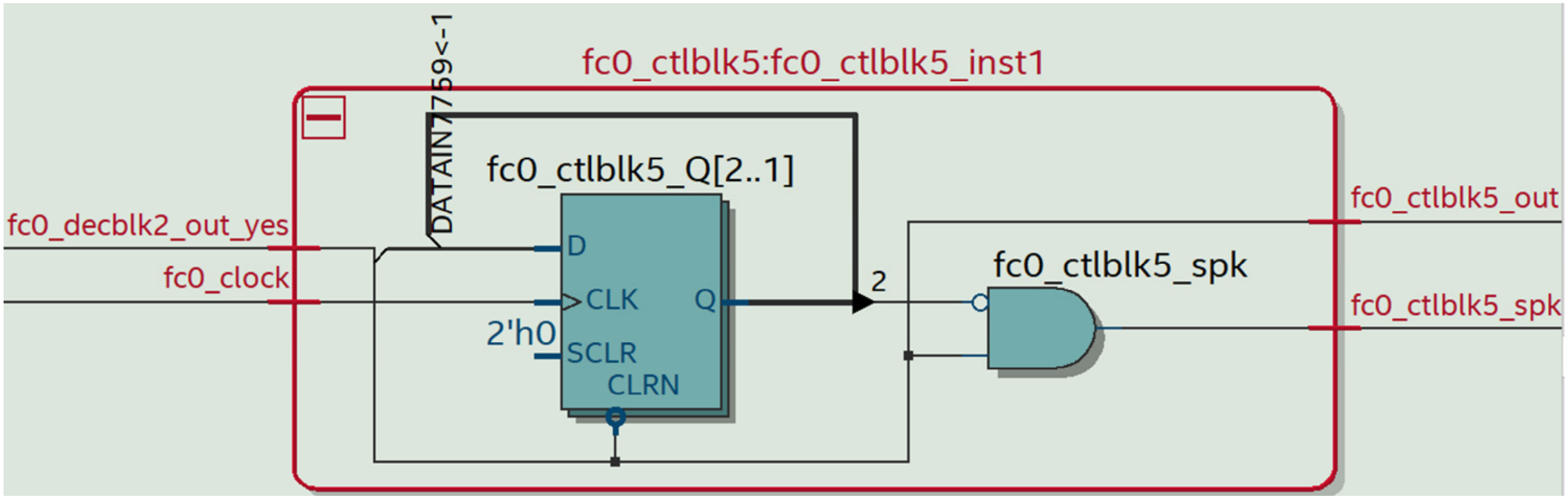


```
////// fc0_ctlblk5    T.OFF O-0   T.ON O-1   T.ON O-2   T.OFF O-3
module fc0_ctlblk5 (input fc0_clock, fc0_decblk2_out_yes,
                    output fc0_ctlblk5_spk, fc0_ctlblk5_out);
                    reg fc0_ctlblk5_Q1, fc0_ctlblk5_D2, fc0_ctlblk5_Q2;
assign fc0_ctlblk5_out = fc0_decblk2_out_yes;
assign fc0_ctlblk5_spk = ~fc0_ctlblk5_Q2 & fc0_ctlblk5_out;
always @(posedge fc0_clock or negedge fc0_ctlblk5_out)
    begin
        if (fc0_ctlblk5_out == 0)
            begin
                  fc0_ctlblk5_Q1 <= 0;
                  fc0_ctlblk5_Q2 <= 0;
            end
        else
            begin
                  fc0_ctlblk5_Q1 <= fc0_ctlblk5_out;
                  fc0_ctlblk5_Q2 <= fc0_ctlblk5_Q1;
            end
    end
endmodule
```

## Flowpro Software

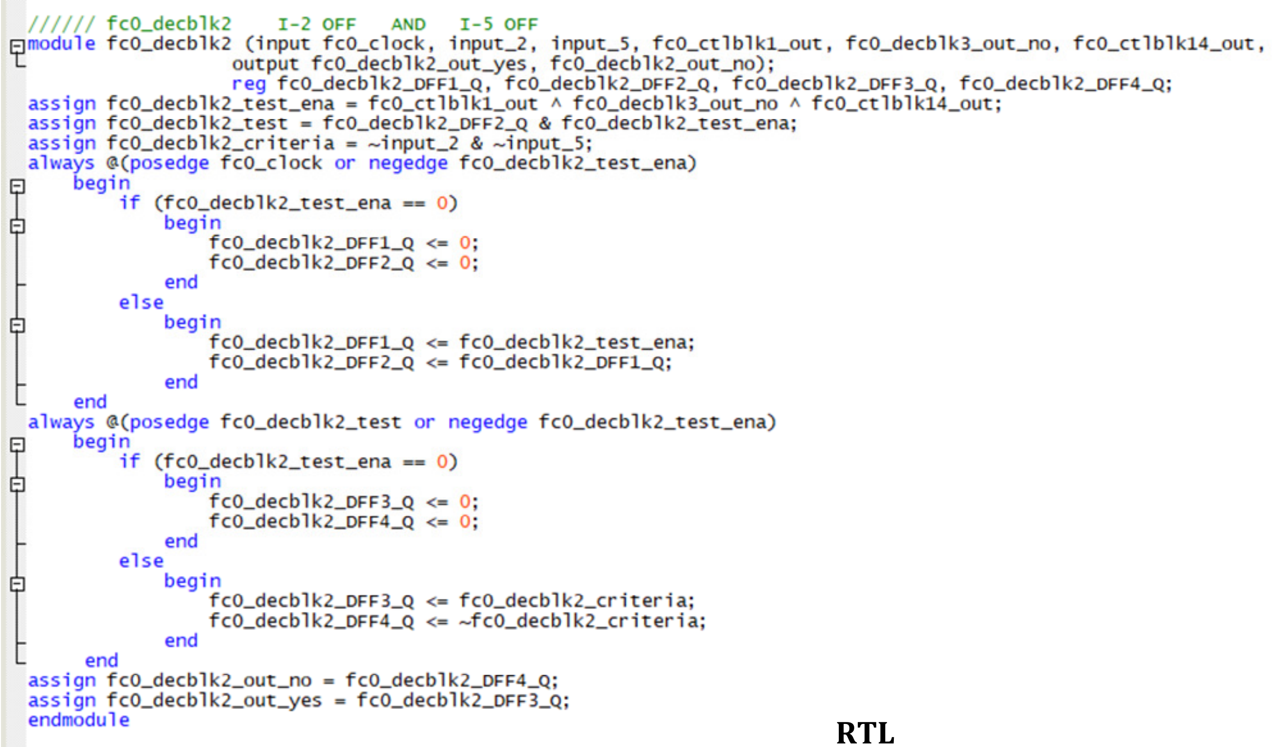


## RTL



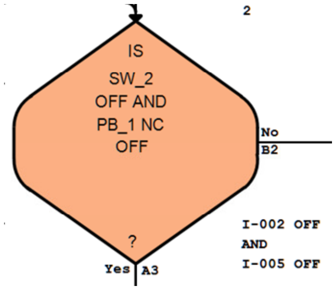
# Flowpro to Verilog: Decision Block Conversion
## REV 2.0

### System Verilog

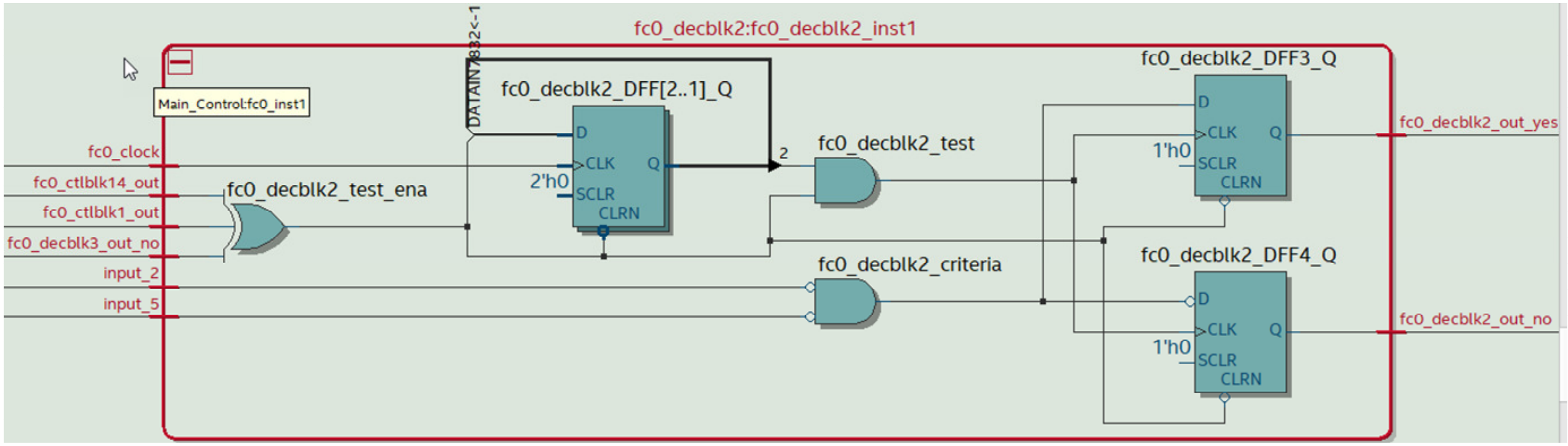


```
////// fc0_decblk2    I-2 OFF   AND   I-5 OFF
module fc0_decblk2 (input fc0_clock, input_2, input_5, fc0_ctlblk1_out, fc0_decblk3_out_no, fc0_ctlblk14_out,
                  output fc0_decblk2_out_yes, fc0_decblk2_out_no);
                  reg fc0_decblk2_DFF1_Q, fc0_decblk2_DFF2_Q, fc0_decblk2_DFF3_Q, fc0_decblk2_DFF4_Q;
assign fc0_decblk2_test_ena = fc0_ctlblk1_out ^ fc0_decblk3_out_no ^ fc0_ctlblk14_out;
assign fc0_decblk2_test = fc0_decblk2_DFF2_Q & fc0_decblk2_test_ena;
assign fc0_decblk2_criteria = ~input_2 & ~input_5;
always @(posedge fc0_clock or negedge fc0_decblk2_test_ena)
    begin
        if (fc0_decblk2_test_ena == 0)
            begin
                fc0_decblk2_DFF1_Q <= 0;
                fc0_decblk2_DFF2_Q <= 0;
            end
        else
            begin
                fc0_decblk2_DFF1_Q <= fc0_decblk2_test_ena;
                fc0_decblk2_DFF2_Q <= fc0_decblk2_DFF1_Q;
            end
    end
always @(posedge fc0_decblk2_test or negedge fc0_decblk2_test_ena)
    begin
        if (fc0_decblk2_test_ena == 0)
            begin
                fc0_decblk2_DFF3_Q <= 0;
                fc0_decblk2_DFF4_Q <= 0;
            end
        else
            begin
                fc0_decblk2_DFF3_Q <= fc0_decblk2_criteria;
                fc0_decblk2_DFF4_Q <= ~fc0_decblk2_criteria;
            end
     end
assign fc0_decblk2_out_no = fc0_decblk2_DFF4_Q;
assign fc0_decblk2_out_yes = fc0_decblk2_DFF3_Q;
endmodule
```

### Flowpro Software

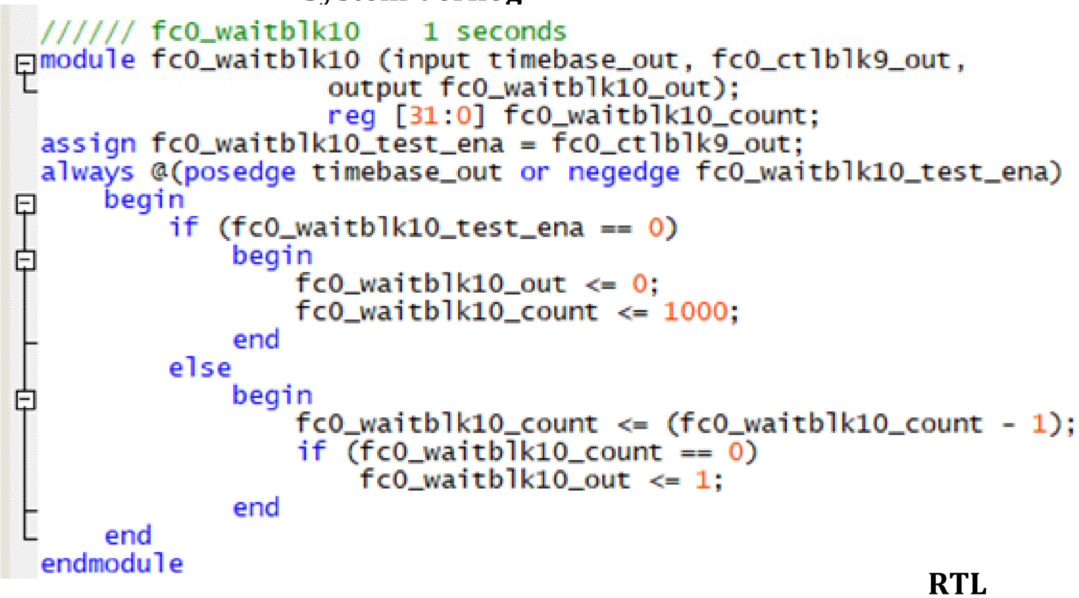


### RTL



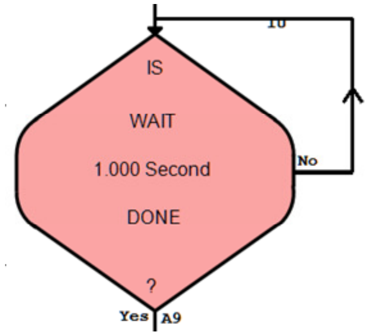
# Flowpro to Verilog: Wait Block Conversion
## REV 2.0

### System Verilog

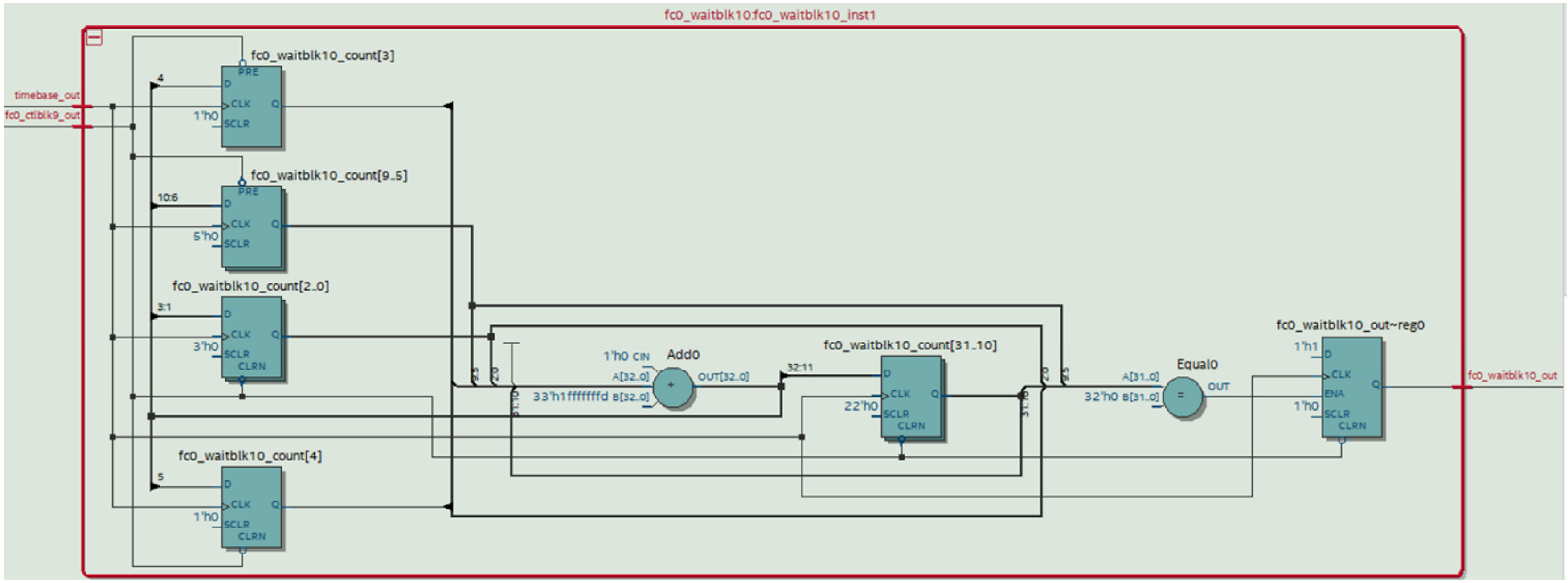


```
////// fc0_waitblk10    1 seconds
module fc0_waitblk10 (input timebase_out, fc0_ctlblk9_out,
                      output fc0_waitblk10_out);
                      reg [31:0] fc0_waitblk10_count;
assign fc0_waitblk10_test_ena = fc0_ctlblk9_out;
always @(posedge timebase_out or negedge fc0_waitblk10_test_ena)
    begin
        if (fc0_waitblk10_test_ena == 0)
            begin
                fc0_waitblk10_out <= 0;
                fc0_waitblk10_count <= 1000;
            end
        else
            begin
                fc0_waitblk10_count <= (fc0_waitblk10_count - 1);
                if (fc0_waitblk10_count == 0)
                    fc0_waitblk10_out <= 1;
            end
    end
endmodule
```

### Flowpro Software

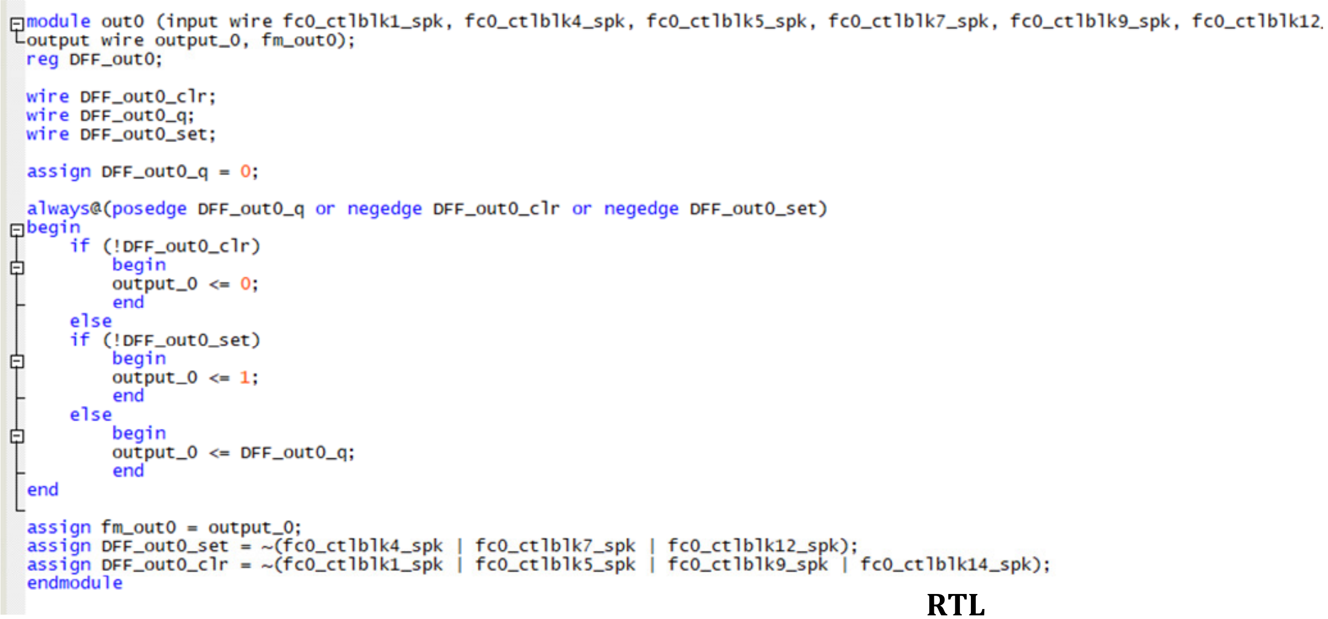


### RTL



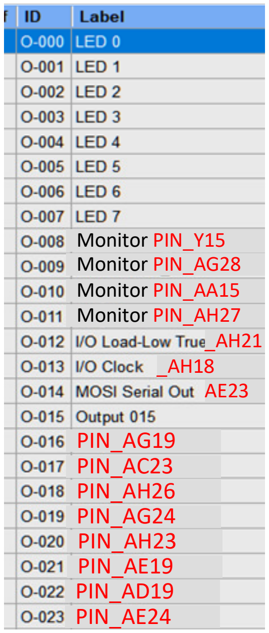
# Flowpro to Verilog: Outputs Conversion
# REV 2.0

## System Verilog



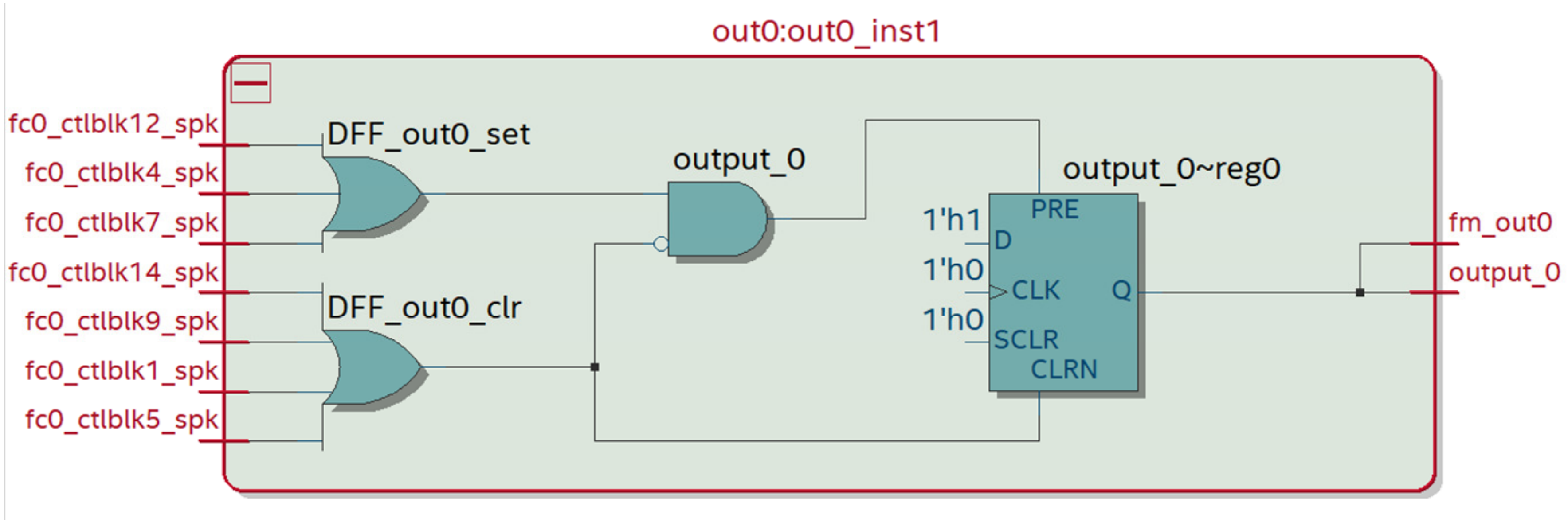
```
module out0 (input wire fc0_ctlblk1_spk, fc0_ctlblk4_spk, fc0_ctlblk5_spk, fc0_ctlblk7_spk, fc0_ctlblk9_spk, fc0_ctlblk12.
output wire output_0, fm_out0);
reg DFF_out0;

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 = ~(fc0_ctlblk4_spk | fc0_ctlblk7_spk | fc0_ctlblk12_spk);
assign DFF_out0_clr = ~(fc0_ctlblk1_spk | fc0_ctlblk5_spk | fc0_ctlblk9_spk | fc0_ctlblk14_spk);
endmodule
```

## Flowpro Software



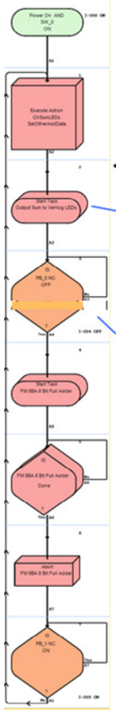
| f | ID | Label |
|---|---|---|
| | O-000 | LED 0 |
| | O-001 | LED 1 |
| | O-002 | LED 2 |
| | O-003 | LED 3 |
| | O-004 | LED 4 |
| | O-005 | LED 5 |
| | O-006 | LED 6 |
| | O-007 | LED 7 |
| | O-008 | Monitor PIN_Y15 |
| | O-009 | Monitor PIN_AG28 |
| | O-010 | Monitor PIN_AA15 |
| | O-011 | Monitor PIN_AH27 |
| | O-012 | I/O Load-Low True_AH21 |
| | O-013 | I/O Clock _AH18 |
| | O-014 | MOSI Serial Out AE23 |
| | O-015 | Output 015 |
| | O-016 | PIN_AG19 |
| | O-017 | PIN_AC23 |
| | O-018 | PIN_AH26 |
| | O-019 | PIN_AG24 |
| | O-020 | PIN_AH23 |
| | O-021 | PIN_AE19 |
| | O-022 | PIN_AD19 |
| | O-023 | PIN_AE24 |

## RTL



# Flowpro Machine - Parallel 8 Bit Adder FPGA Flow

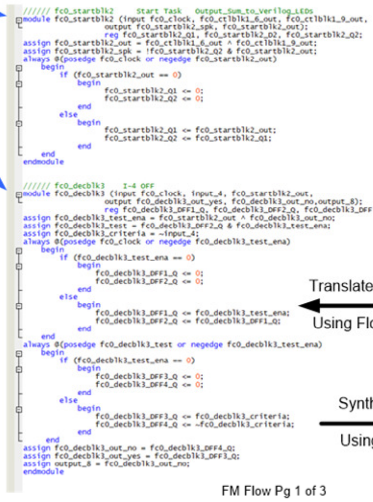
**Flowpro Machine Flow**
**"8 Bit Adder"**



Data Entry and Bit Carry Flags

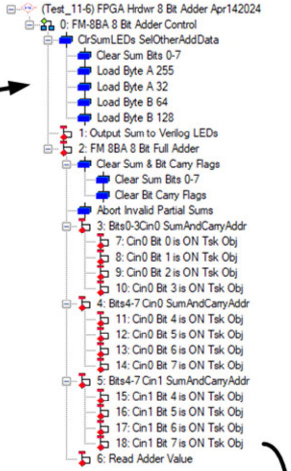
Compile High Level Flowcharts to an Object File

Using Flowpro Software



```
////// fc0_startblk2     Start Task   Output_Sum_to_Verilog_LEDs
module fc0_startblk2 (input fc0_clock, fc0_ctlblk1_6_out, fc0_ctlblk1_9_out,
                output fc0_startblk2_spk, fc0_startblk2_out);
                reg fc0_startblk2_Q1, fc0_startblk2_D2, fc0_startblk2_Q2;
assign fc0_startblk2_out = fc0_ctlblk1_6_out ^ fc0_ctlblk1_9_out;
assign fc0_startblk2_spk = !fc0_startblk2_Q2 & fc0_startblk2_out;
always @(posedge fc0_clock or negedge fc0_startblk2_out)
    begin
        if (fc0_startblk2_out == 0)
            begin
                fc0_startblk2_Q1 <= 0;
                fc0_startblk2_Q2 <= 0;
            end
        else
            begin
                fc0_startblk2_Q1 <= fc0_startblk2_out;
                fc0_startblk2_Q2 <= fc0_startblk2_Q1;
            end
    end
endmodule

////// fc0_decblk3    I-4 OFF
module fc0_decblk3 (input fc0_clock, input_4, fc0_startblk2_out,
                output fc0_decblk3_out_yes, fc0_decblk3_out_no,output_8);
                reg fc0_decblk3_DFF1_Q, fc0_decblk3_DFF2_Q, fc0_decblk3_DFF
assign fc0_decblk3_test_ena = fc0_startblk2_out ^ fc0_decblk3_out_no;
assign fc0_decblk3_test = fc0_decblk3_DFF2_Q & fc0_decblk3_test_ena;
assign fc0_decblk3_criteria = ~input_4;
always @(posedge fc0_clock or negedge fc0_decblk3_test_ena)
    begin
        if (fc0_decblk3_test_ena == 0)
            begin
               fc0_decblk3_DFF1_Q <= 0;
               fc0_decblk3_DFF2_Q <= 0;
            end
        else
            begin
               fc0_decblk3_DFF1_Q <= fc0_decblk3_test_ena;
               fc0_decblk3_DFF2_Q <= fc0_decblk3_DFF1_Q;
            end
    end
always @(posedge fc0_decblk3_test or negedge fc0_decblk3_test_ena)
    begin
        if (fc0_decblk3_test_ena == 0)
            begin
               fc0_decblk3_DFF3_Q <= 0;
               fc0_decblk3_DFF4_Q <= 0;
            end
        else
            begin
               fc0_decblk3_DFF3_Q <= fc0_decblk3_criteria;
               fc0_decblk3_DFF4_Q <= ~fc0_decblk3_criteria;
            end
     end
assign fc0_decblk3_out_no = fc0_decblk3_DFF4_Q;
assign fc0_decblk3_out_yes = fc0_decblk3_DFF3_Q;
assign output_8 = fc0_decblk3_out_no;
endmodule
```

FM Flow Pg 1 of 3



- (Test_11-6) FPGA Hrdwr 8 Bit Adder Apr142024
  - 0: FM-8BA 8 Bit Adder Control
    - ClrSumLEDs SelOtherAddData
      - Clear Sum Bits 0-7
      - Load Byte A 255
      - Load Byte A 32
      - Load Byte B 64
      - Load Byte B 128
    - 1: Output Sum to Verilog LEDs
    - 2: FM 8BA 8 Bit Full Adder
      - Clear Sum & Bit Carry Flags
        - Clear Sum Bits 0-7
        - Clear Bit Carry Flags
      - Abort Invalid Partial Sums
      - 3: Bits0-3Cin0 SumAndCarryAddr
        - 7: Cin0 Bit 0 is ON Tsk Obj
        - 8: Cin0 Bit 1 is ON Tsk Obj
        - 9: Cin0 Bit 2 is ON Tsk Obj
        - 10: Cin0 Bit 3 is ON Tsk Obj
      - 4: Bits4-7 Cin0 SumAndCarryAddr
        - 11: Cin0 Bit 4 is ON Tsk Obj
        - 12: Cin0 Bit 5 is ON Tsk Obj
        - 13: Cin0 Bit 6 is ON Tsk Obj
        - 14: Cin0 Bit 7 is ON Tsk Obj
      - 5: Bits4-7 Cin1 SumAndCarryAddr
        - 15: Cin1 Bit 4 is ON Tsk Obj
        - 16: Cin1 Bit 5 is ON Tsk Obj
        - 17: Cin1 Bit 6 is ON Tsk Obj
        - 18: Cin1 Bit 7 is ON Tsk Obj
      - 6: Read Adder Value

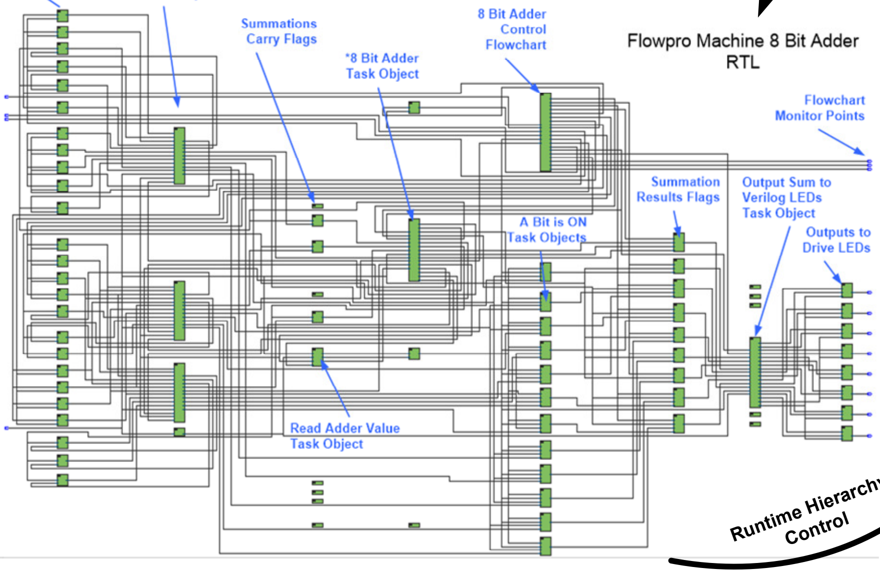
Translate Flowpro Software Object File to Verilog

Using Flowpro Machine to Verilog Utility FM2VU

Synthesize Verilog Code to RTL

Using FPGA Vendor's Software

Flowpro Machine 8 Bit Adder RTL



Three 4 Bit Adder Task Objects

Summations Carry Flags

*8 Bit Adder Task Object

8 Bit Adder Control Flowchart

Flowchart Monitor Points

Summation Results Flags

Output Sum to Verilog LEDs Task Object

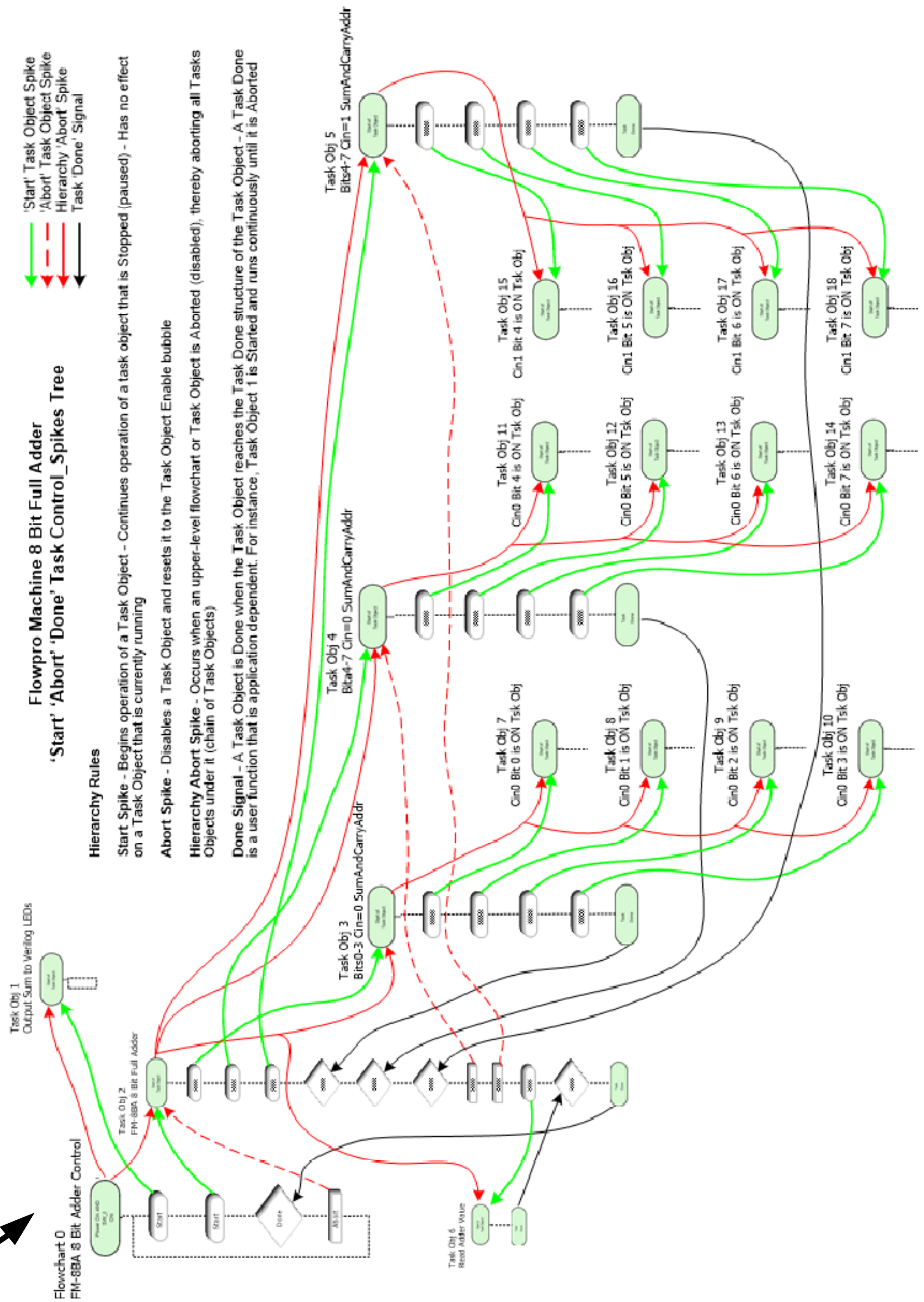
Outputs to Drive LEDs

A Bit is ON Task Objects

Read Adder Value Task Object

FM Flow Pg 2 of 3

Runtime Hierarchy Control



**Flowpro Machine 8 Bit Full Adder**
**'Start' 'Abort' 'Done' Task Control_Spikes Tree**

- 'Start' Task Object Spike
- 'Abort' Task Object Spike
- Hierarchy 'Abort' Spike
- Task 'Done' Signal

**Hierarchy Rules**

**Start Spike** - Begins operation of a Task Object - Continues operation of a task object that is Stopped (paused) - Has no effect on a Task Object that is currently running

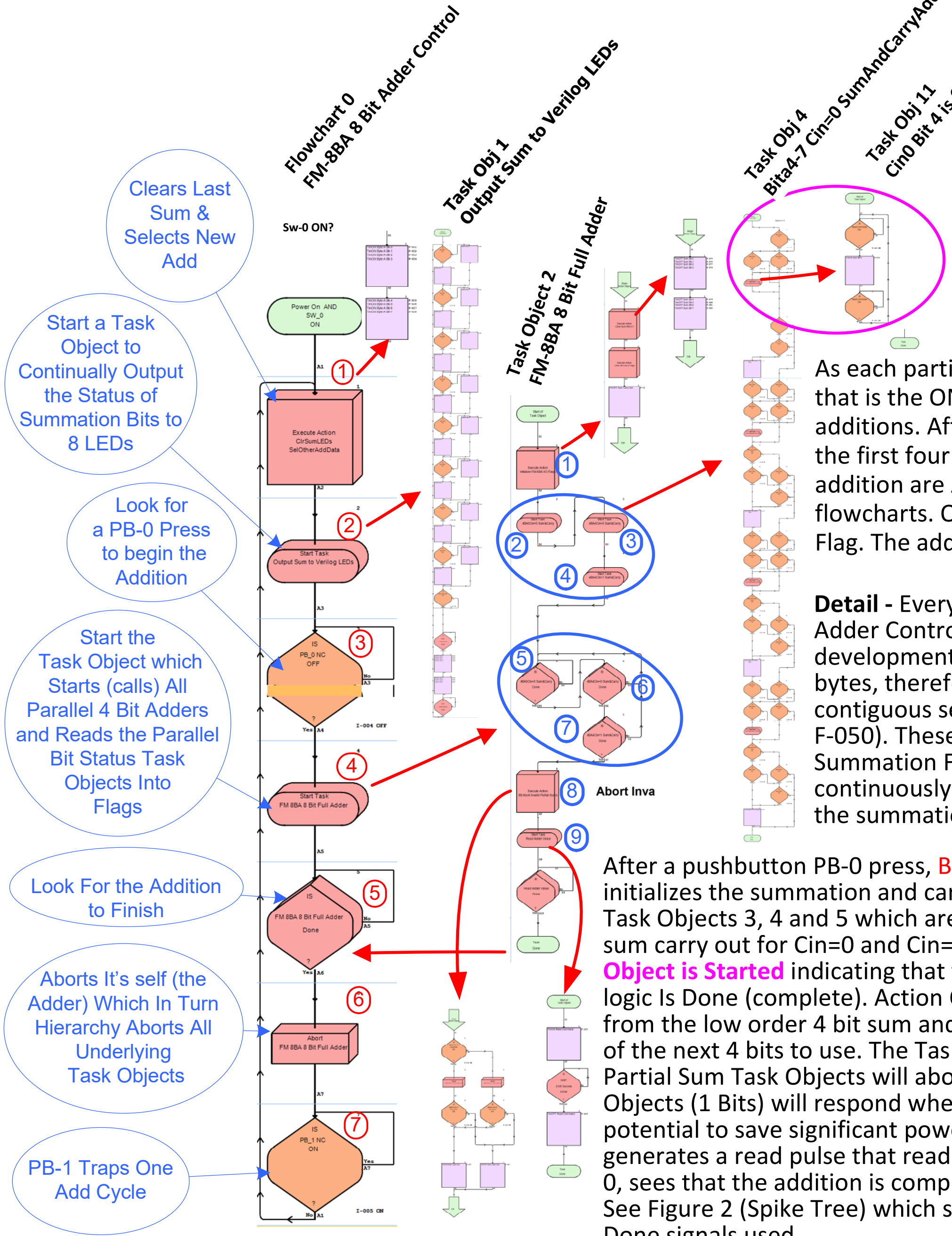
**Abort Spike** - Disables a Task Object and resets it to the Task Object Enable bubble

**Hierarchy Abort Spike** - Occurs when an upper-level flowchart or Task Object is Aborted (disabled), thereby aborting **all** Tasks Objects under it (chain of Task Objects)

**Done Signal** - A Task Object is Done when the **Task** Object reaches the Task Done structure of the Task Object - A Task Done is a user function that is application dependent. For instance, Task Object 1 is Started and runs continuously until it is Aborted

Task Obj 5 Bits4-7 Cin=1 SumAndCarryAddr
Task Obj 15 Cin1 Bit 4 is ON Tsk Obj
Task Obj 16 Cin1 Bit 5 is ON Tsk Obj
Task Obj 17 Cin1 Bit 6 is ON Tsk Obj
Task Obj 18 Cin1 Bit 7 is ON Tsk Obj
Task Obj 11 Cin0 Bit 4 is ON Tsk Obj
Task Obj 12 Cin0 Bit 5 is ON Tsk Obj
Task Obj 13 Cin0 Bit 6 is ON Tsk Obj
Task Obj 14 Cin0 Bit 7 is ON Tsk Obj
Task Obj 4 Bits4-7 Cin=0 SumAndCarryAddr
Task Obj 7 Cin0 Bit 0 is ON Tsk Obj
Task Obj 8 Cin0 Bit 1 is ON Tsk Obj
Task Obj 9 Cin0 Bit 2 is ON Tsk Obj
Task Obj 10 Cin0 Bit 3 is ON Tsk Obj
Task Obj 3 Bits0-3 Cin=0 SumAndCarryAddr
Task Obj 1 Output Sum to Verilog LEDs
Task Obj 2 FM-8BA 8 Bit Full Adder
Flowchart 0 FM-8BA 8 Bit Adder Control
Task Obj 6 Read Adder Value
Start
Start
Done
Abort

# FM-8BA 8 Bit Adder
# Figure 1


Flowchart 0
FM-8BA 8 Bit Adder Control
Task Obj 1
Output Sum to Verilog LEDs
Task Object 2
FM-8BA 8 Bit Full Adder
Task Obj 4
Bita4-7 Cin=0 SumAndCarryAddr
Task Obj 11
Cin0 Bit 4 is ON Tsk Obj
Sw-0 ON?
Power On AND SW_0 ON
Execute Action ClrSumLEDs SelOtherAddData
Start Task Output Sum to Verilog LEDs
IS PB_0 NC OFF
Start Task FM 8BA 8 Bit Full Adder
IS FM 8BA 8 Bit Full Adder Done
Abort FM 8BA 8 Bit Full Adder
IS PB_1 NC ON
Abort Inva
1
2
3
4
5
6
7
1
2
3
4
5
6
7
8
9
Clears Last Sum & Selects New Add
Start a Task Object to Continually Output the Status of Summation Bits to 8 LEDs
Look for a PB-0 Press to begin the Addition
Start the Task Object which Starts (calls) All Parallel 4 Bit Adders and Reads the Parallel Bit Status Task Objects Into Flags
Look For the Addition to Finish
Aborts It's self (the Adder) Which In Turn Hierarchy Aborts All Underlying Task Objects
PB-1 Traps One Add Cycle

## Flowpro Machine 8 Bit Adder Description

This description assumes that you have knowledge of Flowpro Software or have read the introductory sections of the Flowpro to Verilog Utility document.

**Preface -** The Flowpro parallel 8 bit adder consists of 19 parallel propagating tasks, one Flowchart and 18 Task Objects (Figure 2). It is a good example of flowchart hierarchy and selectively running portions of a parallel asynchronous function. With this flowchart architecture, adding more bits to the addition width simply requires adding more derivative parallel flowcharts to the Flowpro Machine project..

**Overview -** The addition of the two binary values begins with a pushbutton press on the FPGA development system. All possible '4 bit partial-sums' (ie. carry in, Cin=0 and Cin=1) are calculated in parallel.
As each partial-sum flowchart is determining its 4 bit result it Starts an individual flowchart that is the ON status (1) for that bit. There are four ON flowcharts per 4 bit partial-sum additions. After the partial-sums are complete and with the status of the 'carry out' from the first four bit partial-sum known, the remaining partial-sums that are not valid for this addition are Aborted. A read command is then generated and read by all of the ON flowcharts. Only the ON flowcharts that have been Started will turn on a partial-sum Bit Flag. The addition sum is then represented by nine contiguous Flowpro Flags.

**Detail -** Every Flowpro Machine project begins with a flowchart. Flowchart "FM-8BA 8 Bit Adder Control" (fc0) begins operation when power is applied and SW-0 is ON. The FPGA development system does not have enough switches for programmable input of two 8 bit bytes, therefore, a Flowpro Machine Action Object (Block ①)is used to initialize two contiguous sets of Flowpro Machine Flags (Byte A, F-011 thru F-018 and Byte B, F-043 thru F-050). These will be used as Input values to the 8 bit adder. Block① also clears the Summation Flags from the previous add cycle. Block②Starts Task Object 1 which runs continuously monitoring the summation flags and updating the LEDs. If the carry out from the summation is ON the LEDs will blink at a 1 Hz rate.

After a pushbutton PB-0 press, Block④Starts Task Object 2, FM 8BA 8 Bit Full Adder, in which it's Block① initializes the summation and carrying flags from the previous addition cycle. Blocks②③, and④parallel Start Task Objects 3, 4 and 5 which are the partial sum and carry logic to compute in parallel all partial sums and sum carry out for Cin=0 and Cin=1. If the summation logic determines a bit should be a 1, a separate Task Object is Started indicating that this bit is a 1. Blocks⑤⑥and⑦are checking to ensure that all partial sum logic Is Done (complete). Action Object Block⑧determines which partial sums are valid by using the Carry Out from the low order 4 bit sum and carry, Task Obj 3, to determine which partial sum and carry, Cin=0 or Cin=1, of the next 4 bits to use. The Task Objects of partial sums not used are Aborted. Aborting of the incorrect Partial Sum Task Objects will abort any On Bit Task Objects under them. This way only valid On Bit Task Objects (1 Bits) will respond when a read command is given. Only running the necessary Task Objects has the potential to save significant power long-term, especially when expanding from 8 to 64 bit data. Block⑨ generates a read pulse that reads all of the active On Bit Task Objects into summation Flags. Block⑤ Flowchart 0, sees that the addition is complete and Aborts itself Block⑥which in turn Aborts all Task Objects under it. See Figure 2 (Spike Tree) which shows the Starting and Aborting of Task Objects along with the Task Object Done signals used.

## Flowpro Machine 8 Bit Full Adder
## 'Start' 'Abort' 'Done' Task Control_Spikes Tree

| | |
|---|---|
| 🟩 → | 'Start' Task Object Spike |
| 🟥 - - → | 'Abort' Task Object Spike |
| 🟥 → | Hierarchy 'Abort' Spike |
| ⬛ → | Task 'Done' Signal |

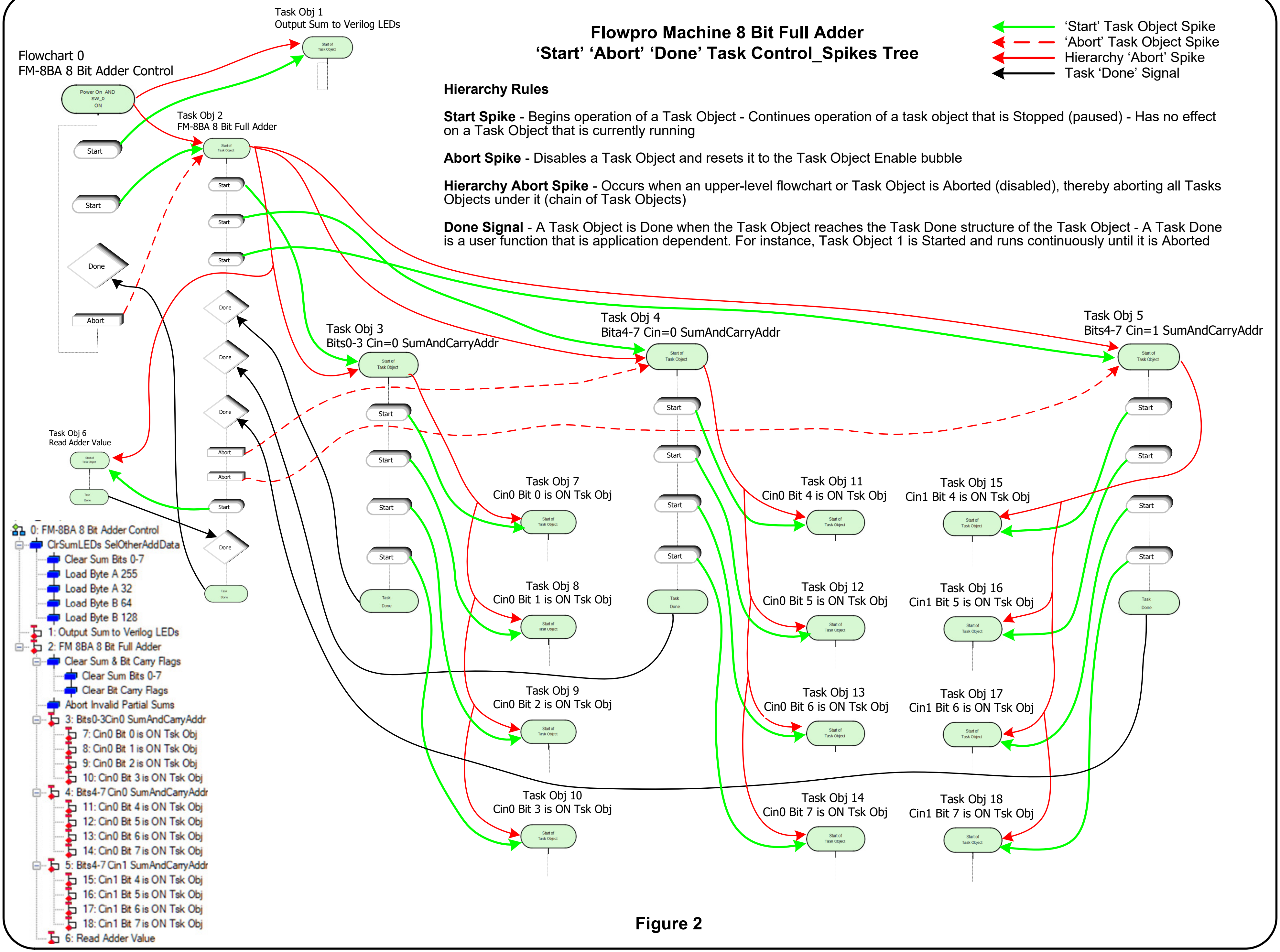
**Hierarchy Rules**

**Start Spike** - Begins operation of a Task Object - Continues operation of a task object that is Stopped (paused) - Has no effect on a Task Object that is currently running

**Abort Spike** - Disables a Task Object and resets it to the Task Object Enable bubble

**Hierarchy Abort Spike** - Occurs when an upper-level flowchart or Task Object is Aborted (disabled), thereby aborting all Tasks Objects under it (chain of Task Objects)

**Done Signal** - A Task Object is Done when the Task Object reaches the Task Done structure of the Task Object - A Task Done is a user function that is application dependent. For instance, Task Object 1 is Started and runs continuously until it is Aborted



- 0: FM-8BA 8 Bit Adder Control
  - ClrSumLEDs SelOtherAddData
    - Clear Sum Bits 0-7
    - Load Byte A 255
    - Load Byte A 32
    - Load Byte B 64
    - Load Byte B 128
  - 1: Output Sum to Verilog LEDs
  - 2: FM 8BA 8 Bit Full Adder
    - Clear Sum & Bit Carry Flags
      - Clear Sum Bits 0-7
      - Clear Bit Carry Flags
    - Abort Invalid Partial Sums
    - 3: Bits0-3Cin0 SumAndCarryAddr
      - 7: Cin0 Bit 0 is ON Tsk Obj
      - 8: Cin0 Bit 1 is ON Tsk Obj
      - 9: Cin0 Bit 2 is ON Tsk Obj
      - 10: Cin0 Bit 3 is ON Tsk Obj
    - 4: Bits4-7 Cin0 SumAndCarryAddr
      - 11: Cin0 Bit 4 is ON Tsk Obj
      - 12: Cin0 Bit 5 is ON Tsk Obj
      - 13: Cin0 Bit 6 is ON Tsk Obj
      - 14: Cin0 Bit 7 is ON Tsk Obj
    - 5: Bits4-7 Cin1 SumAndCarryAddr
      - 15: Cin1 Bit 4 is ON Tsk Obj
      - 16: Cin1 Bit 5 is ON Tsk Obj
      - 17: Cin1 Bit 6 is ON Tsk Obj
      - 18: Cin1 Bit 7 is ON Tsk Obj
  - 6: Read Adder Value

**Figure 2**