

```
samaksh19200@edaserver3:~$ cd Ass2
samaksh19200@edaserver3:~/Ass2$ ls
file.c func.v HLS_output synthesize_Synthesis_func.sh
samaksh19200@edaserver3:~/Ass2$
```

## 2) Verilog RTL code generated (date and time present)

```
// Politecnico di Milano
// Code created using PandaA - Version: Panda 0.9.7-dev - Revision 4fe142b5c340716f8bd751323655c01057141ffd-SROA_TCAD_release - Date 2022-02-01T20:07:06
// /tmp/.mount_bambu-BoUm5c/usr/bin/bambu executed with: /tmp/.mount_bambu-BoUm5c/usr/bin/bambu --top-fname=func --range-analysis-mode=skip --compiler=I3
86_CLANG4 /home/samaksh19200/Ass2/file.c
//
// Send any bug to: panda-info@polimi.it
// *****
// The following text holds for all the components tagged with PANDA_LGPLv3.
// They are all part of the BAMBU/PANDA IP LIBRARY.
// This library is free software; you can redistribute it and/or
// modify it under the terms of the GNU Lesser General Public
// License as published by the Free Software Foundation; either
// version 3 of the License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with the PandaA framework; see the files COPYING.LIB
// If not, see <http://www.gnu.org/licenses/>.
// *****
```

### RTL CODE

```
`ifdef __ICARUS__
  `define _SIM_HAVE_CLOG2
`endif
`ifdef VERILATOR
  `define _SIM_HAVE_CLOG2
`endif
`ifdef MODEL_TECH
  `define _SIM_HAVE_CLOG2
`endif
`ifdef VCS
  `define _SIM_HAVE_CLOG2
`endif
`ifdef NCVERILOG
  `define _SIM_HAVE_CLOG2
`endif
`ifdef XILINX_SIMULATOR
  `define _SIM_HAVE_CLOG2
`endif
`ifdef XILINX_ISIM
  `define _SIM_HAVE_CLOG2
`endif

// This component is part of the BAMBU/PANDA IP LIBRARY
// Copyright (C) 2004-2020 Politecnico di Milano
// Author(s): Fabrizio Ferrandi <fabrizio.ferrandi@polimi.it>
// License: PANDA_LGPLv3
`timescale 1ns / 1ps
module ui_cond_expr_FU(in1, in2, in3, out1);
  parameter BITSIZE_in1=1, BITSIZE_in2=1, BITSIZE_in3=1, BITSIZE_out1=1;
  // IN
```

```

    input [BITSIZE_in1-1:0] in1;
    input [BITSIZE_in2-1:0] in2;
    input [BITSIZE_in3-1:0] in3;
    // OUT
    output [BITSIZE_out1-1:0] out1;
    assign out1 = in1 != 0 ? in2 : in3;
endmodule

// This component is part of the BAMBU/PANDA IP LIBRARY
// Copyright (C) 2004-2020 Politecnico di Milano
// Author(s): Fabrizio Ferrandi <fabrizio.ferrandi@polimi.it>
// License: PANDA_LGPLv3
`timescale 1ns / 1ps
module ui_negate_expr_FU(in1, out1);
    parameter BITSIZE_in1=1, BITSIZE_out1=1;
    // IN
    input [BITSIZE_in1-1:0] in1;
    // OUT
    output [BITSIZE_out1-1:0] out1;
    assign out1 = -in1;
endmodule

// This component is part of the BAMBU/PANDA IP LIBRARY
// Copyright (C) 2004-2020 Politecnico di Milano
// Author(s): Fabrizio Ferrandi <fabrizio.ferrandi@polimi.it>
// License: PANDA_LGPLv3
`timescale 1ns / 1ps
module ui_plus_expr_FU(in1, in2, out1);
    parameter BITSIZE_in1=1, BITSIZE_in2=1, BITSIZE_out1=1;
    // IN
    input [BITSIZE_in1-1:0] in1;
    input [BITSIZE_in2-1:0] in2;
    // OUT
    output [BITSIZE_out1-1:0] out1;
    assign out1 = in1 + in2;
endmodule

// Datapath RTL description for func
// This component has been derived from the input source code and so it does
// not fall under the copyright of Panda framework, but it follows the input
// source code copyright, and may be aggregated with components of the
// BAMBU/PANDA IP LIBRARY.
// Author(s): Component automatically generated by bambu
// License: THIS COMPONENT IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR
// IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF
// MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
`timescale 1ns / 1ps

```

```

module datapath_func(clock, reset, in_port_j, in_port_k, in_port_c,
return_port);
    // IN
    input clock;
    input reset;
    input [31:0] in_port_j;
    input [31:0] in_port_k;
    input [7:0] in_port_c;
    // OUT
    output [31:0] return_port;
    // Component and signal declarations
    wire [31:0] out_ui_cond_expr_FU_32_32_32_32_3_i0_fu_func_417475_417504;
    wire [31:0] out_ui_negate_expr_FU_32_32_4_i0_fu_func_417475_417503;
    wire [31:0] out_ui_plus_expr_FU_32_32_32_5_i0_fu_func_417475_417505;

    ui_negate_expr_FU #(.BITSIZE_in1(32), .BITSIZE_out1(32))
fu_func_417475_417503
(.out1(out_ui_negate_expr_FU_32_32_4_i0_fu_func_417475_417503),
.in1(in_port_k));
    ui_cond_expr_FU #(.BITSIZE_in1(8), .BITSIZE_in2(32), .BITSIZE_in3(32),
.BITSIZE_out1(32)) fu_func_417475_417504
(.out1(out_ui_cond_expr_FU_32_32_32_32_3_i0_fu_func_417475_417504),
.in1(in_port_c), .in2(in_port_k),
.in3(out_ui_negate_expr_FU_32_32_4_i0_fu_func_417475_417503));
    ui_plus_expr_FU #(.BITSIZE_in1(32), .BITSIZE_in2(32), .BITSIZE_out1(32))
fu_func_417475_417505
(.out1(out_ui_plus_expr_FU_32_32_32_5_i0_fu_func_417475_417505),
.in1(out_ui_cond_expr_FU_32_32_32_32_3_i0_fu_func_417475_417504),
.in2(in_port_j));
    // io-signal post fix
    assign return_port =
out_ui_plus_expr_FU_32_32_32_5_i0_fu_func_417475_417505;

endmodule

// FSM based controller description for func
// This component has been derived from the input source code and so it does
not fall under the copyright of Panda framework, but it follows the input
source code copyright, and may be aggregated with components of the
BAMBU/PANDA IP LIBRARY.
// Author(s): Component automatically generated by bambu
// License: THIS COMPONENT IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
`timescale 1ns / 1ps
module controller_func(done_port, clock, reset, start_port);
    // IN
    input clock;

```

```

input reset;
input start_port;
// OUT
output done_port;
parameter [0:0] S_0 = 1'b1;
reg [0:0] _present_state, _next_state;
reg done_port;

always @(posedge clock)
    if (reset == 1'b0) _present_state <= S_0;
    else _present_state <= _next_state;

always @(*)
begin
    done_port = 1'b0;
    case (_present_state)
        S_0 :
            if(start_port == 1'b1)
            begin
                _next_state = S_0;
                done_port = 1'b1;
            end
            else
            begin
                _next_state = S_0;
            end
        default :
            begin
                _next_state = S_0;
            end
    endcase
end
endmodule

// Top component for func
// This component has been derived from the input source code and so it does
not fall under the copyright of Panda framework, but it follows the input
source code copyright, and may be aggregated with components of the
BAMBU/PANDA IP LIBRARY.
// Author(s): Component automatically generated by bambu
// License: THIS COMPONENT IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
`timescale 1ns / 1ps
module _func(clock, reset, start_port, done_port, j, k, c, return_port);
    // IN
    input clock;
    input reset;

```

```

input start_port;
input [31:0] j;
input [31:0] k;
input [7:0] c;
// OUT
output done_port;
output [31:0] return_port;
// Component and signal declarations

    controller_func Controller_i (.done_port(done_port), .clock(clock),
.reset(reset), .start_port(start_port));
    datapath_func Datapath_i (.return_port(return_port), .clock(clock),
.reset(reset), .in_port_j(j), .in_port_k(k), .in_port_c(c));

endmodule

// This component is part of the BAMBU/PANDA IP LIBRARY
// Copyright (C) 2004-2020 Politecnico di Milano
// Author(s): Fabrizio Ferrandi <fabrizio.ferrandi@polimi.it>
// License: PANDA_LGPLv3
`timescale 1ns / 1ps
module ui_view_convert_expr_FU(in1, out1);
    parameter BITSIZE_in1=1, BITSIZE_out1=1;
    // IN
    input [BITSIZE_in1-1:0] in1;
    // OUT
    output [BITSIZE_out1-1:0] out1;
    assign out1 = in1;
endmodule

// Minimal interface for function: func
// This component has been derived from the input source code and so it does
not fall under the copyright of Panda framework, but it follows the input
source code copyright, and may be aggregated with components of the
BAMBU/PANDA IP LIBRARY.
// Author(s): Component automatically generated by bambu
// License: THIS COMPONENT IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
`timescale 1ns / 1ps
module func(clock, reset, start_port, j, k, c, done_port, return_port);
    // IN
    input clock;
    input reset;
    input start_port;
    input [31:0] j;
    input [31:0] k;
    input [7:0] c;

```

```

// OUT
output done_port;
output [31:0] return_port;
// Component and signal declarations
wire [31:0] out_return_port_ui_view_convert_expr_FU;

_func _func_i0 (.done_port(done_port),
.return_port(out_return_port_ui_view_convert_expr_FU), .clock(clock),
.reset(reset), .start_port(start_port), .j(j), .k(k), .c(c));
    ui_view_convert_expr_FU #(.BITSIZE_in1(32), .BITSIZE_out1(32))
return_port_ui_view_convert_expr_FU (.out1(return_port),
.in1(out_return_port_ui_view_convert_expr_FU));

endmodule

```

## **MAIN IDEA OF THE CODE**

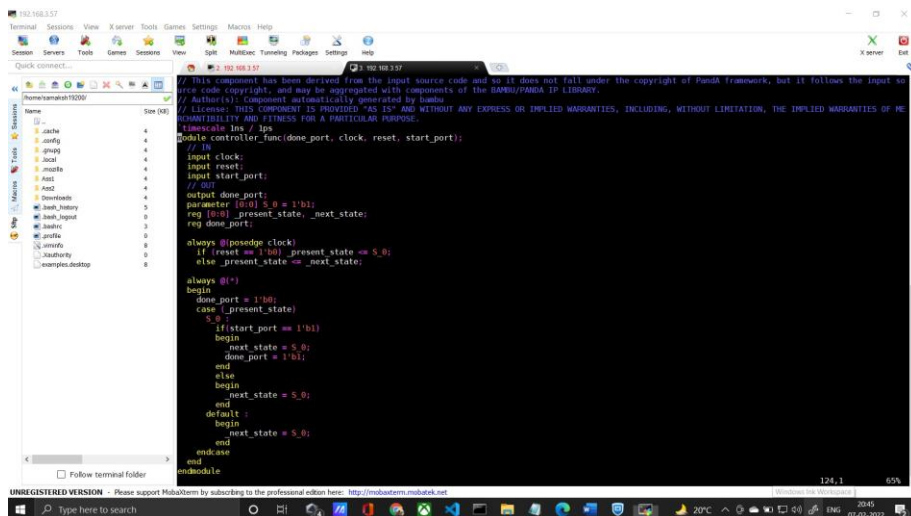
3 Inputs: j,k,c

1 output

The code is responsible for either performing  $j+k$  or  $j-k$ . The arithmetic operation is dependent upon the third input 'c'. If 'c' is False (0) then we perform  $j-k$ , if 'c' is True (1) then we perform  $j+k$ .

1 module in the data path will convert  $k$  to  $-k$ . Another module will pick either  $+k$  or  $-k$ , depending on the value of  $c$ . The third module will take 'j' and either  $+k$  or  $-k$  as supplied by the previous model and simply perform addition. This added value would be the final answer.

### 3) Control Path Explanation and FSM



```
// This component has been derived from the input source code and so it does not fall under the copyright of Panda framework, but it follows the input source code copyright, and may be aggregated with components of the BAMBI/PANDA IP LIBRARY.
// Author(s): Component automatically generated by bambo
// License: THIS COMPONENT IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
timescale 1ns / 1ps
module single_controller_func(done_port, clock, reset, start_port);
    // In
    input clock;
    input reset;
    input start_port;
    // Out
    output done_port;
    parameter [0:0] S_0 = 1'b1;
    reg [0:0] present_state, _next_state;
    reg done_port;

    always @(posedge clock)
    if (reset == 1'b0) present_state <= S_0;
    else present_state <= _next_state;

    always @(*)
    begin
        done_port = 1'b0;
        case (present_state)
            S_0 :
            if (start_port == 1'b1)
            begin
                _next_state = S_0;
                done_port = 1'b1;
            end
            else
            begin
                _next_state = S_0;
            end
            default :
            begin
                _next_state = S_0;
            end
        endcase
    end
endmodule
```

The following control path module describes a single state FSM. The single state is  $S_0$ .  $S_0$  is assigned as  $1'b1$  (1 in binary... single bit number) initially. The module has 3 input signals of 1 bit each (clock, reset, start\_port) and 1 output signal (done\_port). Depending on the present state and the current input, the next state and the output is described. Hence, we can describe this module easily using a Mealey FSM model. At every positive edge of the clock, if reset is 0 then present\_state is  $S_0$  only. Otherwise, if the reset is 1 then the present state is assigned as the next state which has been calculated. The reset is therefore an active low signal. The calculation of next\_state is done as follows--> If present state is  $S_0$  and the input signal start\_port is 1, then next\_port is assigned as  $S_0$  only. If present state is  $S_0$  and the input signal start\_port is 0, then also the next\_port is assigned as  $S_0$  (single state FSM). However, the output done\_port depends on the current state as well the input. If the current\_state is  $S_0$  and the input signal start\_port is 1, then output signal done\_port is assigned as 1. In all other cases done\_port is 0.

#### FSM





## 4) Explanation of every other Module

### DATAPATH

```
// Datapath RTL description for func
// This component has been derived from the input source code and so it does not fall under the copyright of Panda framework, but it follows the input so
// the code copyright, and may be aggregated with components of the SAMS/PANDA IP LIBRARY.
// Author(s): Component automatically generated by bambu
// License: THIS COMPONENT IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF RE
// SCHABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
timescale 1ns / 1ps
module datapath_func(clock, reset, in_port_j, in_port_k, in_port_c, return_port);
// in
input clock;
input reset;
input [31:0] in_port_j;
input [31:0] in_port_k;
input [7:0] in_port_c;
// out
output [31:0] return_port;
// Component and signal declarations
wire [31:0] out_ui_cond_expr_FU_32_32_32_3_10_fu_func_417475_417504;
wire [31:0] out_ui_negate_expr_FU_32_32_4_10_fu_func_417475_417503;
wire [31:0] out_ui_plus_expr_FU_32_32_32_5_10_fu_func_417475_417505;
// in
ui_negate_expr_FU #(BITSIZE_in1(32), BITSIZE_out1(32)) fu_func_417475_417503 (.out1(out_ui_negate_expr_FU_32_32_4_10_fu_func_417475_417503), .in1(in
port_k));
ui_cond_expr_FU #(BITSIZE_in1(0), BITSIZE_in2(32), BITSIZE_in3(32), BITSIZE_out1(32)) fu_func_417475_417504 (.out1(out_ui_cond_expr_FU_32_32_32
_3_10_fu_func_417475_417504), .in1(in_port_c), .in2(in_port_k), .in3(out_ui_negate_expr_FU_32_32_4_10_fu_func_417475_417503));
ui_plus_expr_FU #(BITSIZE_in1(32), BITSIZE_in2(32), BITSIZE_out1(32)) fu_func_417475_417505 (.out1(out_ui_plus_expr_FU_32_32_32_5_10_fu_func_417475
_417505), .in1(out_ui_cond_expr_FU_32_32_32_3_10_fu_func_417475_417504), .in2(in_port_j));
// in-signal: out1
assign return_port = out_ui_plus_expr_FU_32_32_32_5_10_fu_func_417475_417505;
endmodule
```

```
timescale 1ns / 1ps
module func(clock, reset, start_port, j, k, c, done_port, return_port);
// in
input clock;
input reset;
input start_port;
input [31:0] j;
input [31:0] k;
input [7:0] c;
// out
output [31:0] return_port;
// Component and signal declarations
wire [31:0] out_return_port_ui_view_convert_expr_FU;
// in
func_fnc_10 (.done_port(done_port), .return_port(out_return_port_ui_view_convert_expr_FU), .clock(clock), .reset(reset), .start_port(start_port), .j
j), .k(k), .c(c));
// in-signal: out1
ui_view_convert_expr_FU #(BITSIZE_in1(32), BITSIZE_out1(32)) return_port_ui_view_convert_expr_FU (.out1(return_port), .in1(out_return_port_ui_view_co
vert_expr_FU));
endmodule
```

Top module: \_func → data path → 3 helper modules to perform the operation.

The \_func module is the heart of the RTL which calls the other modules. It contains the input signals and the output. The data path is called using the \_func module and the data path is further divided into 3 parts.

1. **ui\_negate\_expr\_FU**: This module generates the negate (negative value) of a number and stores it as an output.
2. **ui\_cond\_expr\_FU**: This module takes 3 inputs. 1 is the value of 'c' which is responsible for the condition, 2nd input is the original number 'k', 3<sup>rd</sup> input is the negate of 'k' which was created by the previous module. Depending on the value of 'c' we decide what the output of this module will be. If 'c' takes the value 0 then the output is negative of the input 'k', if c takes the value 1 then 'k' is the output of this module.
3. **ui\_plus\_expr\_FU**: This module takes 2 inputs, 1<sup>st</sup> is 'j' and second is the output of the previous module (cond). Depending on cond we could either be performing j+k or j+(-k) in this module and the result is the final output.  
(Since we are dealing with integers → All signals are 32bit in the data path.)

```
// Component: ui_negate_expr_FU
timescale 1ns / 1ps
module ui_negate_expr_FU(in1, in2, out1);
parameter BITSIZE_in1, BITSIZE_in2, BITSIZE_out1;
// in
input [BITSIZE_in1-1:0] in1;
input [BITSIZE_in2-1:0] in2;
input [BITSIZE_out1-1:0] out1;
// out
output [BITSIZE_out1-1:0] out1;
assign out1 = in1 < 0 ? in2 : in1;
endmodule

// This component is part of the SAMS/PANDA IP LIBRARY
// Copyright (c) 2020-2020 Politecnico di Milano
// Author(s): Fabrizio Perem, Stefano Foranopoli, It
// License: Public (GPL)
timescale 1ns / 1ps
module ui_negate_expr_FU(in1, out1);
parameter BITSIZE_in1, BITSIZE_out1;
// in
input [BITSIZE_in1-1:0] in1;
// out
output [BITSIZE_out1-1:0] out1;
assign out1 = -in1;
endmodule

// This component is part of the SAMS/PANDA IP LIBRARY
// Copyright (c) 2020-2020 Politecnico di Milano
// Author(s): Fabrizio Perem, Stefano Foranopoli, It
// License: Public (GPL)
timescale 1ns / 1ps
module ui_plus_expr_FU(in1, in2, out1);
parameter BITSIZE_in1, BITSIZE_in2, BITSIZE_out1;
// in
input [BITSIZE_in1-1:0] in1;
input [BITSIZE_in2-1:0] in2;
// out
output [BITSIZE_out1-1:0] out1;
assign out1 = in1 + in2;
endmodule
```