NOKIA

Rapid Prototyping Using HDL Coder

Who Are We?

Esa-Matti Turtinen

R&D Manager, SoC Prototyping, Nokia Oulu

- M.Sc., Electrical Engineering
- 31 years old
- About 6 years of experience working on different roles related to SoC development

Joonas Järviluoma

Prototype Engineer, SoC, Nokia Oulu

- M.Sc., Electrical Engineering
- 26 years old
- Just graduated
- Currently working on FPGA lab testing

Nokia vision

Expanding the human possibilities of the connected world

Nokia has been at the forefront of every fundamental change in how we communicate and connect

Telephony begins

Bell Telephone Laboratories formed in 1925

Analog revolution

Long distance voice communication

- Copper networks
- Circuit switches
- Amplifiers

Digital revolution

Voice, data, and video communication

- Laser
- Satellite communications
- UNIX
- DWDM
- 100Gbps optical transport
- 400G routers

Mobile revolution

Wireless communication

- First ever calls on GSM and LTE
- First car phone
- Commercialization of Small Cells
- MIMO

The new connectivity

Intelligent and seamless connectivity through the Cloud

- 5G
- G.Fast: 1Gbps over copper
- Optical super channels
- Terabit IP routing
- Datacenter infrastructure and applications for the Cloud
- Smart sensors for the Internet of Things

A financially strong leader

Revenue*

€26.6bn

R&D spend*

€4.5bn

Net cash*

€10.0bn

Employees

106,000

* Combined Nokia and Alcatel-Lucent 2015 numbers according to Nokia accounting policies, non-IFRS

R&D professionals

~40,000

Services professionals

~40,000

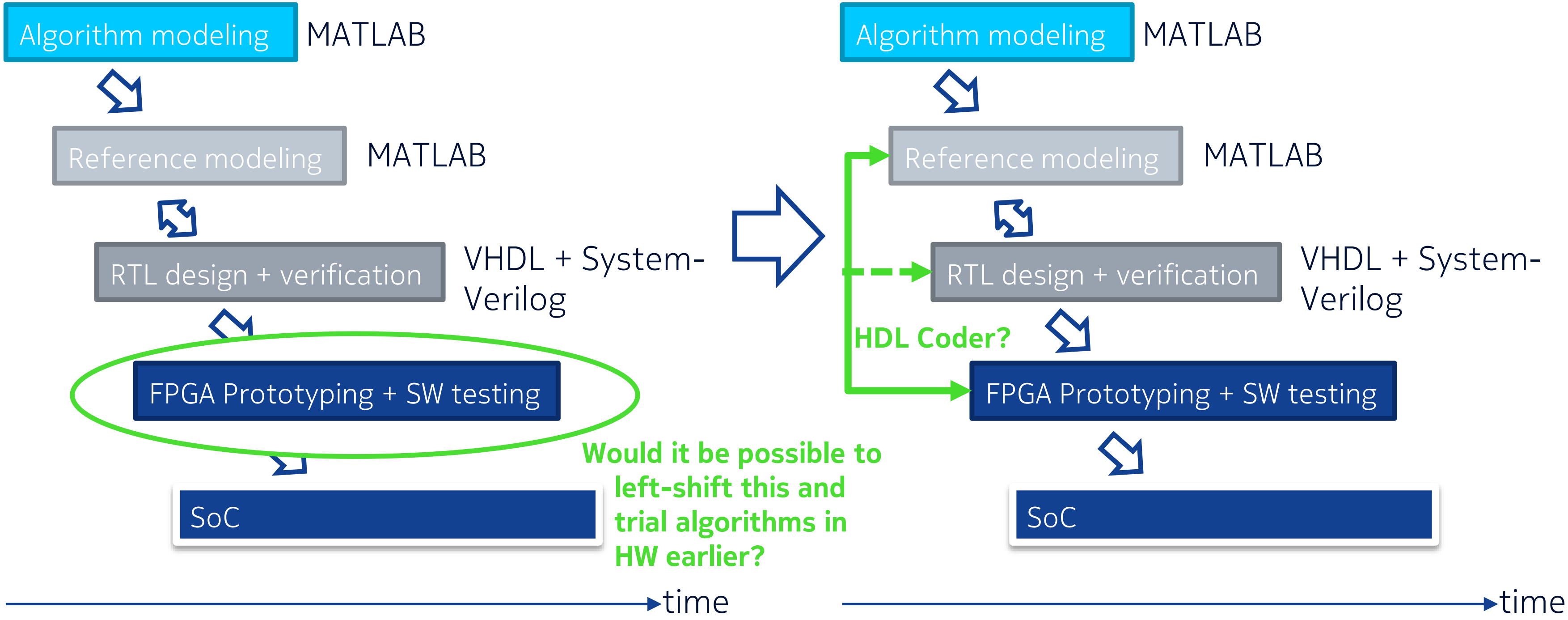
World leading intellectual property (patent families)

~31,000

Bell Labs

Nokia Technologies

Challenge



Algorithm modeling MATLAB

Reference modeling MATLAB

RTL design + verification VHDL + System-Verilog

FPGA Prototyping + SW testing

SoC

Would it be possible to left-shift this and trial algorithms in HW earlier?

time

Algorithm modeling MATLAB

Reference modeling MATLAB

RTL design + verification VHDL + System-Verilog

HDL Coder?

FPGA Prototyping + SW testing

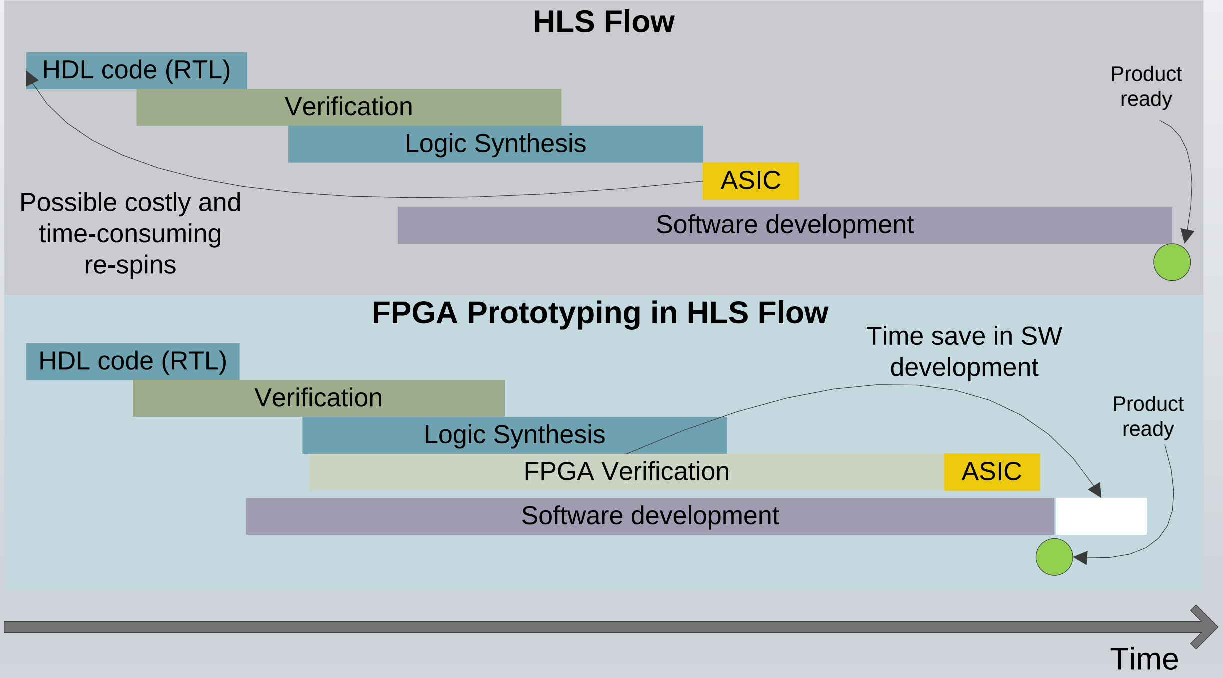
SoC

time

NOKIA

FPGA Prototyping Flow Timeline

Proportional Estimation in Generic HLS flow

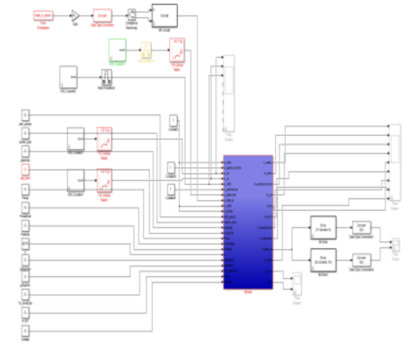


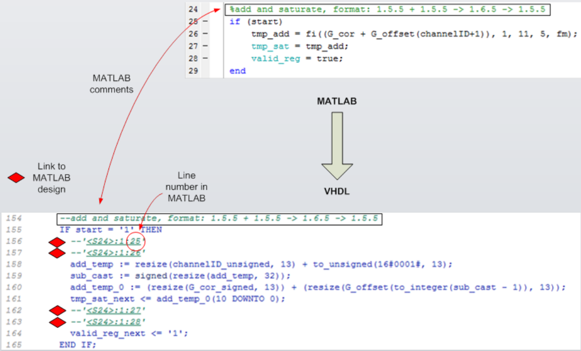
HDL Coder Flow

From Algorithm to FPGA Programmable Model

HLS

Logic Synthesis





MATLAB comments

```
%add and saturate, format: 1.5.5 + 1.5.5 -> 1.6.5 -> 1.5.5
if (start)
    tmp_add = fi((G_cor + G_offset(channelID+1)), 1, 11, 5, fm);
    tmp_sat = tmp_add;
    valid_reg = true;
end
```

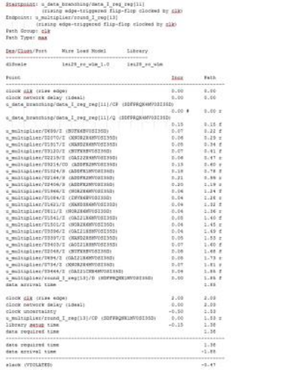
MATLAB

VHDL

◆ Link to MATLAB design

Line number in MATLAB

```
--add and saturate, format: 1.5.5 + 1.5.5 -> 1.6.5 -> 1.5.5
IF start = '1' THEN
  --'<S24>:1:25'
  --'<S24>:1:26'
  add_temp := resize(channelID_unsigned, 13) + to_unsigned(16#0001#, 13);
  sub_cast := signed(resize(add_temp, 32));
  add_temp_0 := (resize(G_cor_signed, 13)) + (resize(G_offset(to_integer(sub_cast - 1)), 13));
  tmp_sat_next <= add_temp_0(10 DOWNTO 0);
  --'<S24>:1:27'
  --'<S24>:1:28'
  valid_reg_next <= '1';
END IF;
```



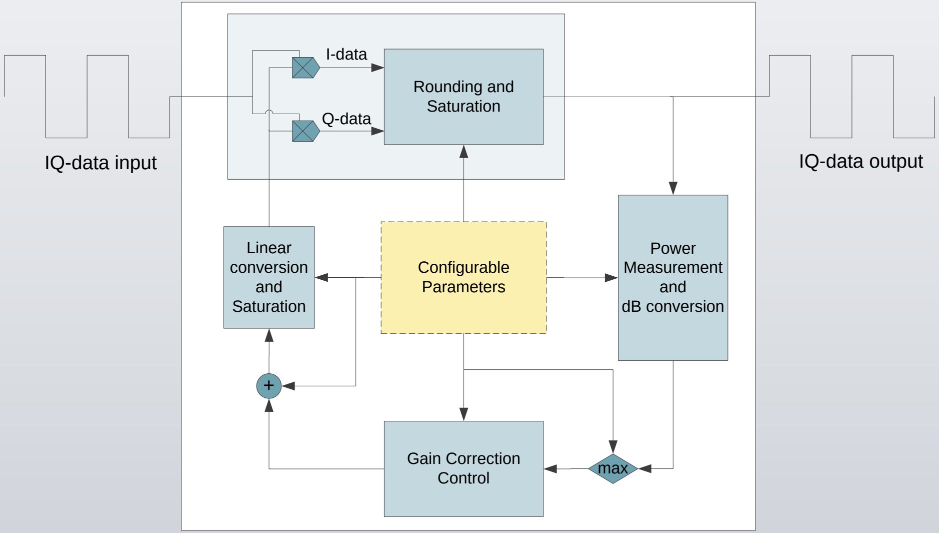
```
Startpoint: u_data_branching/data_I_reg_reg[11]
            (rising edge-triggered flip-flop clocked by clk)
Endpoint: u_multiplier/round_I_reg[13]
          (rising edge-triggered flip-flop clocked by clk)
Path Group: clk
Path Type: max

Des/Clust/Port     Wire Load Model     Library
------------------------------------------------
d15scale           lei28_ro_wlm_1.0    lei28_ro_wlm

Point                                                   Incr       Path
--------------------------------------------------------------------------
clock clk (rise edge)                                   0.00       0.00
clock network delay (ideal)                             0.00       0.00
u_data_branching/data_I_reg_reg[11]/CP (SDFPRQX4MV03I35D)
                                                        0.00 #     0.00 r
u_data_branching/data_I_reg_reg[11]/Q (SDFPRQX4MV03I35D)
                                                        0.15       0.15 f
u_multiplier/U699/I (BUFX6BV03I35D)                     0.07       0.22 f
u_multiplier/U2070/I (XNOR2X4MV03I35D)                  0.06       0.29 r
u_multiplier/U1917/I (NAND2X4MV03I35D)                  0.05       0.34 f
u_multiplier/U3120/I (BUFX8BV03I35D)                    0.07       0.41 f
u_multiplier/U2219/I (OAI22X4MV03I35D)                  0.06       0.47 r
u_multiplier/U3214/CO (ADDFX2MV03I35D)                  0.13       0.60 r
u_multiplier/U1024/B (ADDFX1MV03I35D)                   0.18       0.78 f
u_multiplier/U2164/B (ADDFX2MV03I35D)                   0.21       0.99 r
u_multiplier/U2406/B (ADDFX2MV03I35D)                   0.20       1.19 r
u_multiplier/U1966/I (NOR2X6MV03I35D)                   0.06       1.24 f
u_multiplier/U1084/I (INVX6BV03I35D)                    0.04       1.28 r
u_multiplier/U1621/I (NAND3X4MV03I35D)                  0.04       1.32 f
u_multiplier/U811/I (NOR2X6MV03I35D)                    0.04       1.36 r
u_multiplier/U1541/I (OAI21X6MV03I35D)                  0.05       1.40 f
u_multiplier/U1501/I (NOR2X6MV03I35D)                   0.04       1.45 r
u_multiplier/U3396/I (OAI21X8MV03I35D)                  0.04       1.49 f
u_multiplier/U3397/I (NAND2X8MV03I35D)                  0.05       1.53 r
u_multiplier/U3403/I (AOI21X8MV03I35D)                  0.07       1.60 f
u_multiplier/U2048/I (BUFX8BV03I35D)                    0.08       1.68 f
u_multiplier/U694/I (OAI21X6MV03I35D)                   0.05       1.73 r
u_multiplier/U734/I (XNOR2X4MV03I35D)                   0.07       1.81 r
u_multiplier/U3464/I (OAI21CXB4MV03I35D)                0.04       1.85 f
u_multiplier/round_I_reg[13]/D (SDFPRQNX1MV03I35D)      0.00       1.85 f
data arrival time                                                  1.85

clock clk (rise edge)                                   2.00       2.00
clock network delay (ideal)                             0.00       2.00
clock uncertainty                                      -0.50       1.50
u_multiplier/round_I_reg[13]/CP (SDFPRQNX1MV03I35D)     0.00       1.50 r
library setup time                                     -0.12       1.38
data required time                                                 1.38
--------------------------------------------------------------------------
data required time                                                 1.38
data arrival time                                                 -1.85
--------------------------------------------------------------------------
slack (VIOLATED)                                                  -0.47
```

Co-simulation

FPGA-in-the-loop

Example Design for HDL Coder Flow

Scaling and Power Limitation Block

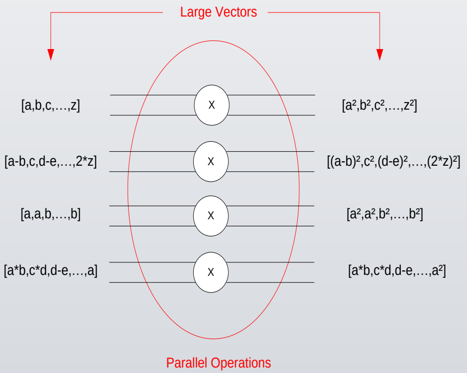


- Arithmetic logic (multipliers, adders etc.)
- Loop structures
- State-Machine
- Look-up tables for dB conversions
- Registers for state control and buffering
- Variable indexing
- Configurable parameters

Classic Division of Models

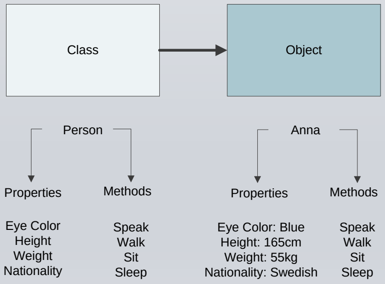
Algorithm and RTL

Algorithm Model



MATLAB operations optimized for maximized simulation performance

Object-Oriented Programming



RTL Model

- Hand-written based on algorithm model
- ASIC optimized performance
- Thorough verification required

Division in HDL Coder Workflow

Algorithm and RTL

Algorithm Model:

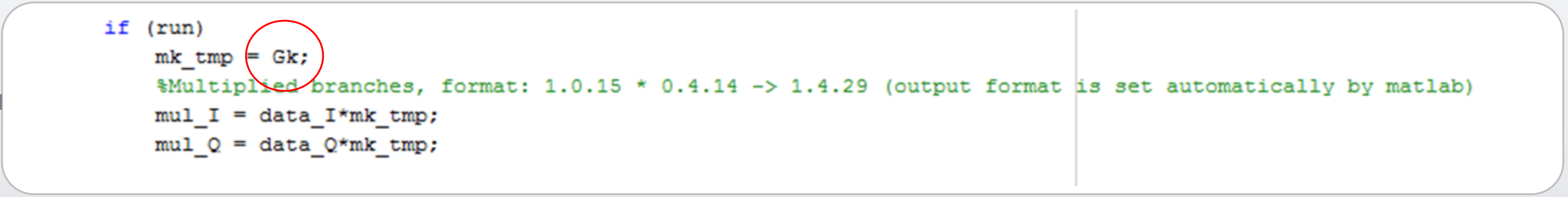
- Written in MATLAB function blocks/System Objects and Simulink library components
- Has to be written from HW perspective to generate feasible RTL

RTL Model:

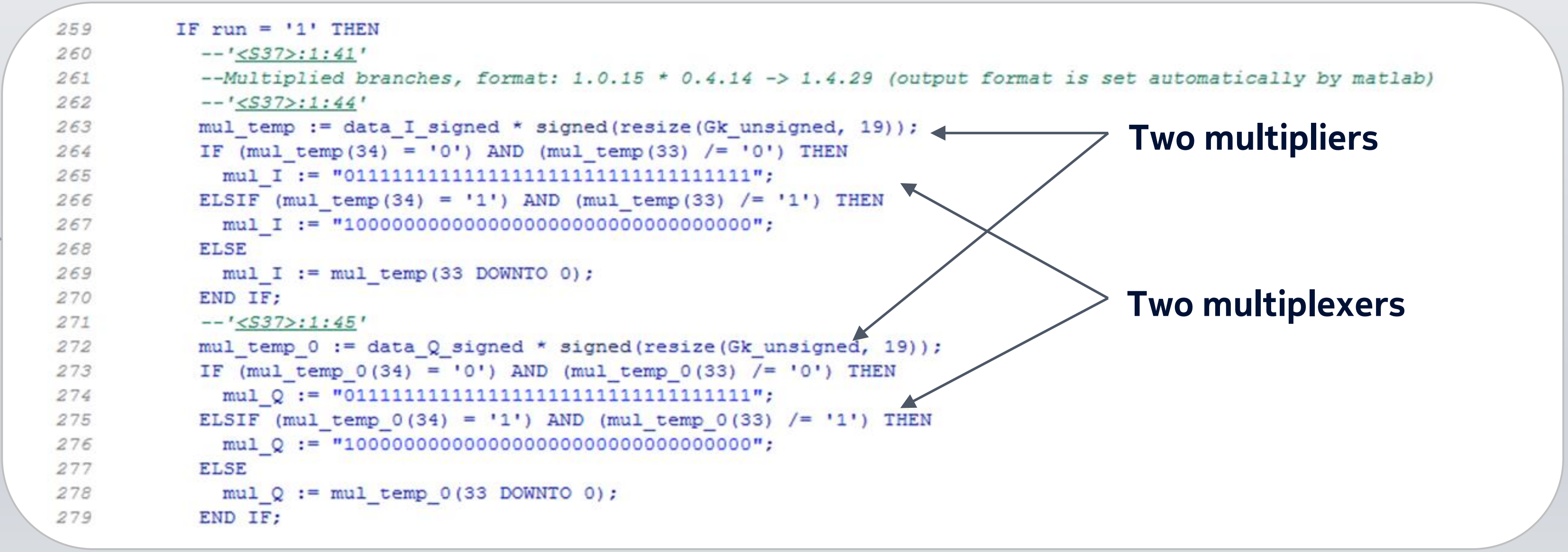
- Rapid generation from Simulink (or MATLAB) model
- Verification focus moves towards algorithm
- Cosimulation verificates RTL against algorithm model
- ”Is as good as the algorithm”

RTL Generation

Example 1: Algorithm without Data Type Definition



```
if (run)
    mk_tmp = Gk;
    %Multiplied branches, format: 1.0.15 * 0.4.14 -> 1.4.29 (output format is set automatically by matlab)
    mul_I = data_I*mk_tmp;
    mul_Q = data_Q*mk_tmp;
```



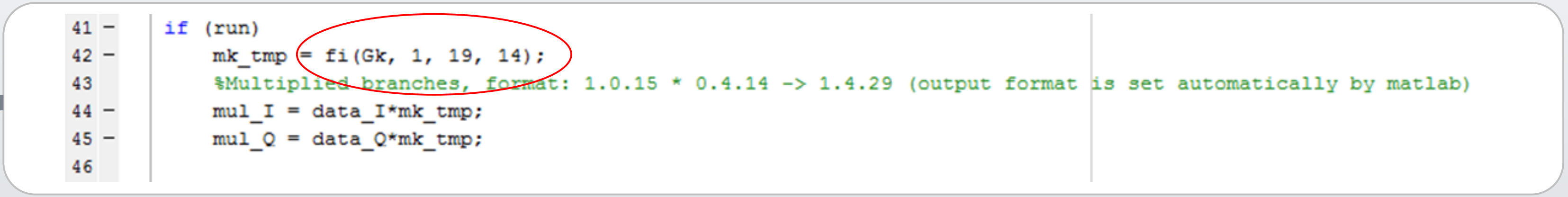
```
IF run = '1' THEN
  --'<S37>:1:41'
  --Multiplied branches, format: 1.0.15 * 0.4.14 -> 1.4.29 (output format is set automatically by matlab)
  --'<S37>:1:44'
  mul_temp := data_I_signed * signed(resize(Gk_unsigned, 19));
  IF (mul_temp(34) = '0') AND (mul_temp(33) /= '0') THEN
    mul_I := "011111111111111111111111111111111";
  ELSIF (mul_temp(34) = '1') AND (mul_temp(33) /= '1') THEN
    mul_I := "100000000000000000000000000000000";
  ELSE
    mul_I := mul_temp(33 DOWNTO 0);
  END IF;
  --'<S37>:1:45'
  mul_temp_0 := data_Q_signed * signed(resize(Gk_unsigned, 19));
  IF (mul_temp_0(34) = '0') AND (mul_temp_0(33) /= '0') THEN
    mul_Q := "011111111111111111111111111111111";
  ELSIF (mul_temp_0(34) = '1') AND (mul_temp_0(33) /= '1') THEN
    mul_Q := "100000000000000000000000000000000";
  ELSE
    mul_Q := mul_temp_0(33 DOWNTO 0);
  END IF;
```

Two multipliers

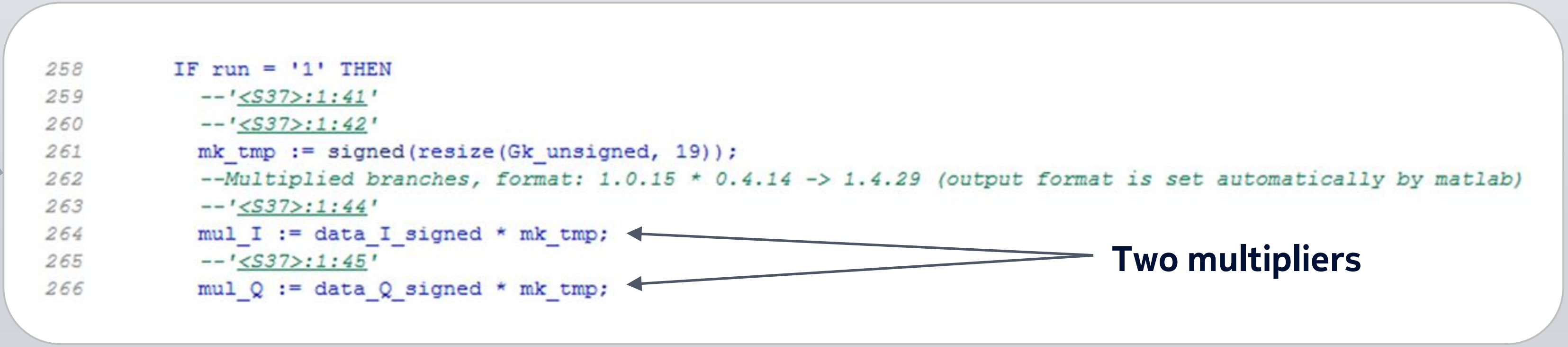
Two multiplexers

RTL Generation

Example 2: Algorithm with Data Type Definition



```
if (run)
    mk_tmp = fi(Gk, 1, 19, 14);
    %Multiplied branches, format: 1.0.15 * 0.4.14 -> 1.4.29 (output format is set automatically by matlab)
    mul_I = data_I*mk_tmp;
    mul_Q = data_Q*mk_tmp;
```

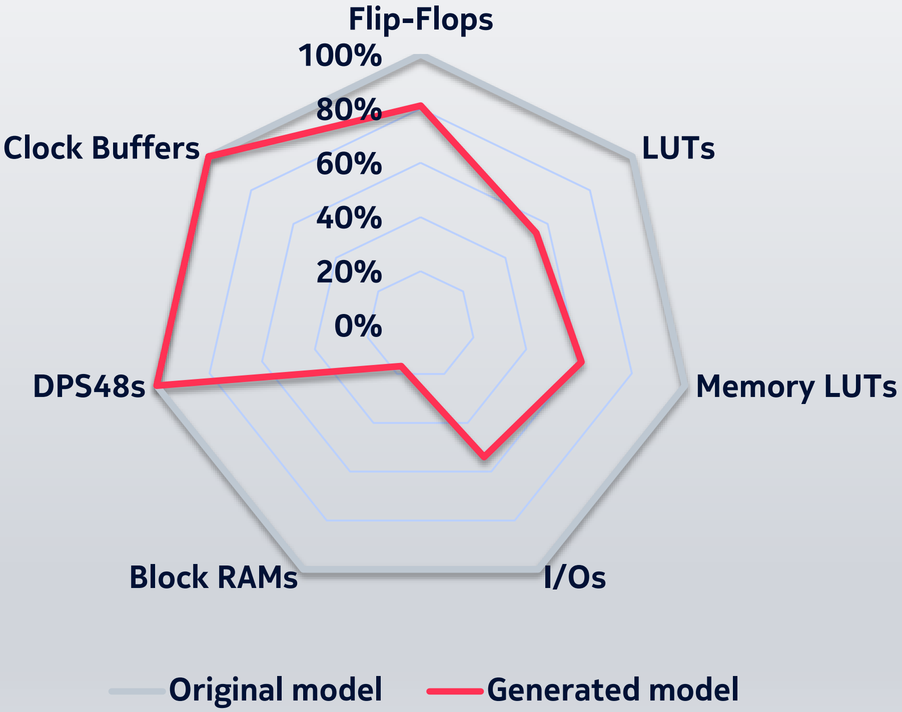


```
      IF run = '1' THEN
        --'<S37>:1:41'
        --'<S37>:1:42'
        mk_tmp := signed(resize(Gk_unsigned, 19));
        --Multiplied branches, format: 1.0.15 * 0.4.14 -> 1.4.29 (output format is set automatically by matlab)
        --'<S37>:1:44'
        mul_I := data_I_signed * mk_tmp;
        --'<S37>:1:45'
        mul_Q := data_Q_signed * mk_tmp;
```

Two multipliers

RTL Resource Utilization Comparison

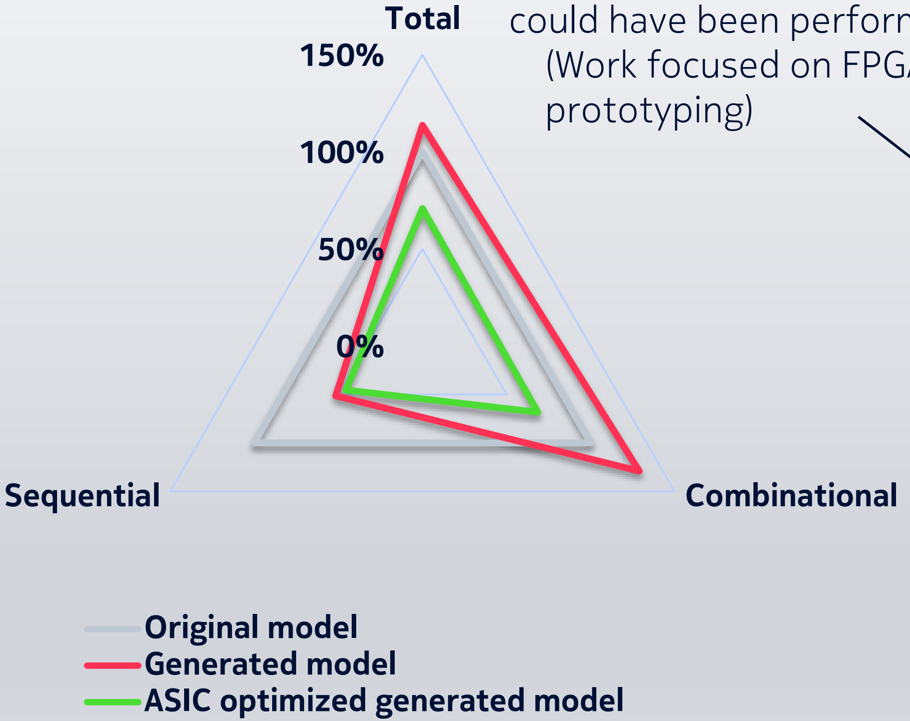
FPGA Prototype Vs. Original ASIC Targeted Model



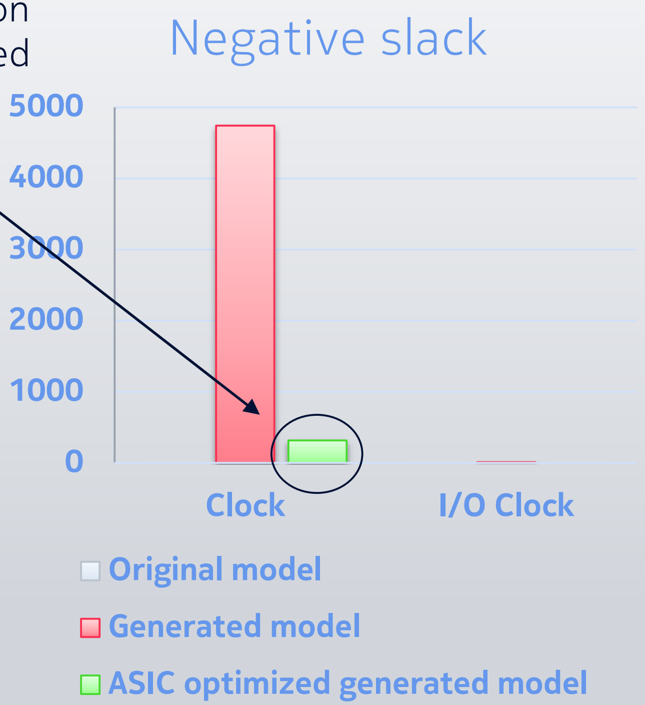
- Original hand-written model, targeted for ASIC, had slightly more signals and routing logic compared to generated model!
- Generated model tested succesfully in FPGA-in-the-loop configuration

ASIC Optimization

Area and Timing Results

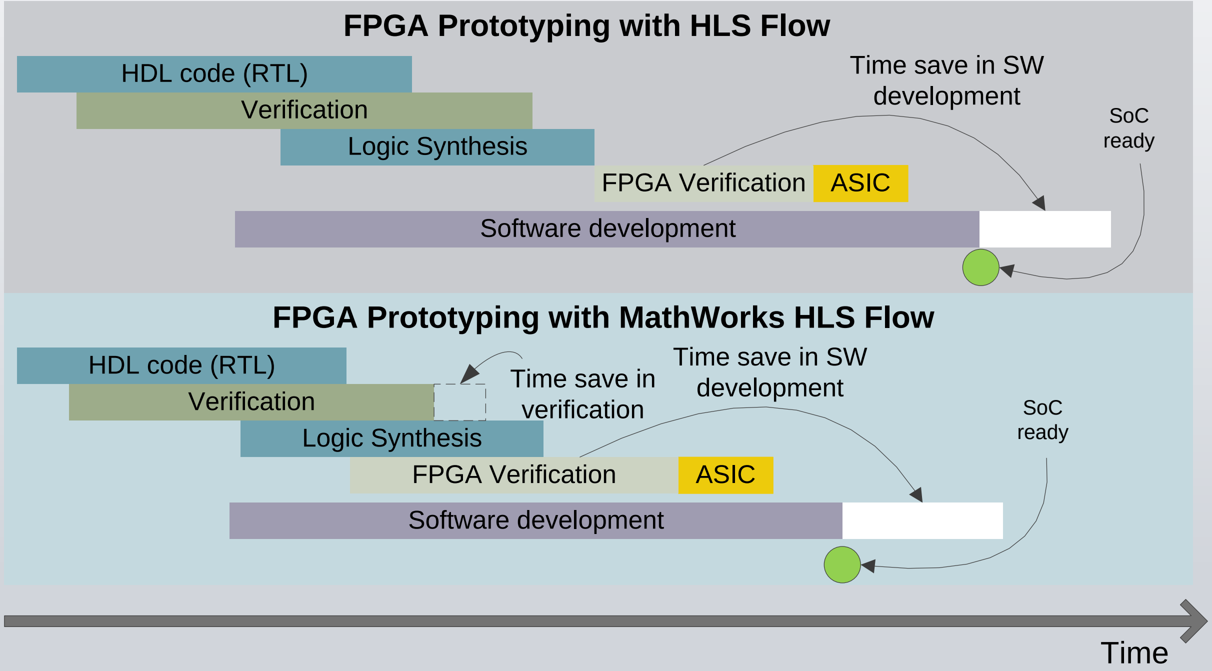


Further timing optimization could have been performed (Work focused on FPGA prototyping)



FPGA Prototyping Flow Timeline

Proportional Estimation in HDL Coder Flow



Conclusion

Benefits and Shortages

Benefits:

- Human readable HDL output
- Design work and verification focus moves on higher level
- Good synthesis results in both FPGA and ASIC cases
- Distinct GUI
- Support for 3rd party tools and FPGA boards

Shortages:

- For feasible HDL generation and FPGA prototyping, algorithms have to be written strictly from HW perspective
- No trivial way to generate generic variables to create scalable Ips (due to Model-Based Design flow)

Future Work

Algorithm design work change towards RTL design style required

- Close co-operation with algorithm and RTL designers is vital
- Algorithm simulation speed might be critical

IP generation with generic interfaces

- Was left out of scope in this study
- Needs to be verified

Projects ongoing

Q & A

NOKIA