Automatically exploiting regularity in applications to reduce reconfiguration memory requirements

Fatma Abouelella

Karel Bruneel and Dirk Stroobandt

Fatma.abouelella@ugent.be

Have you ever tried run-time reconfiguration ?

- How much time and effort have you spent to design for run-time reconfiguration system?

- Can you afford the reconfiguration overhead?

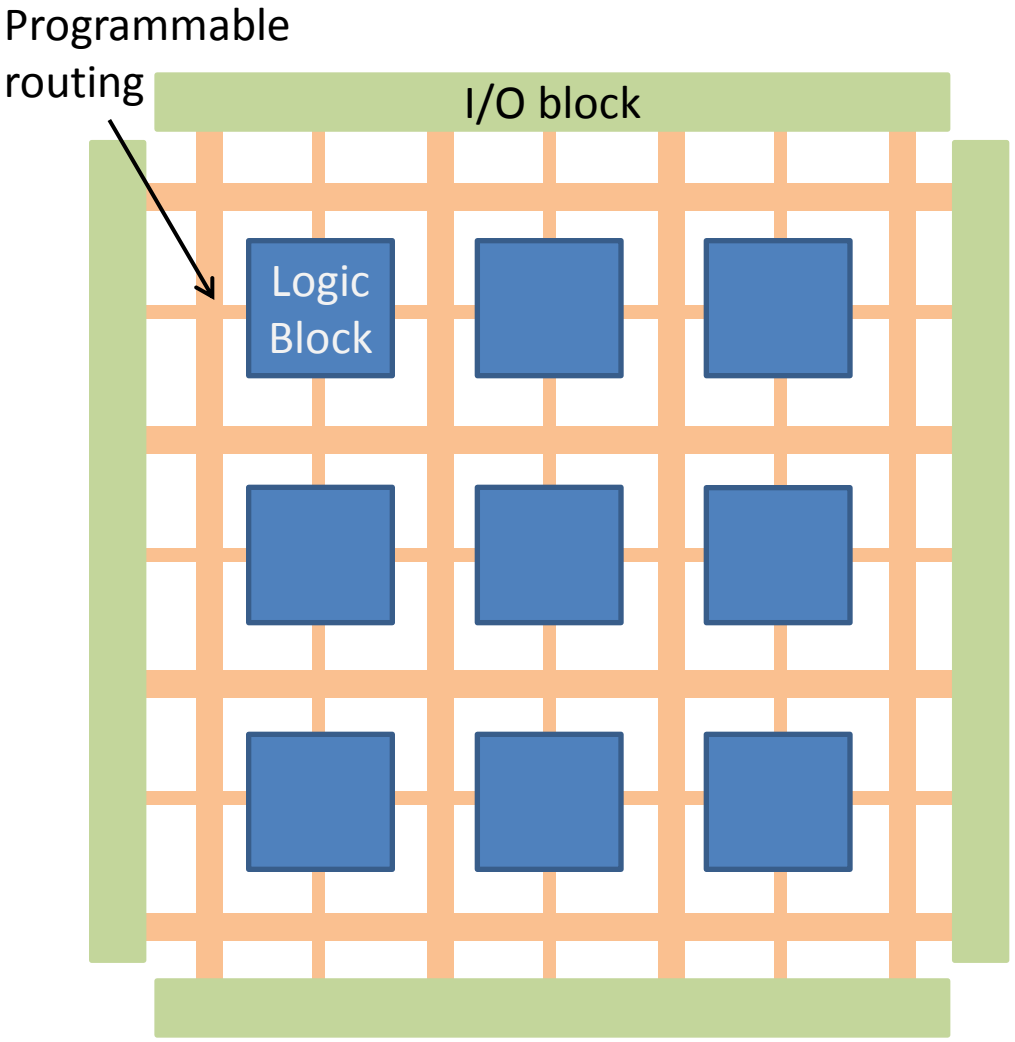
OUTLINE

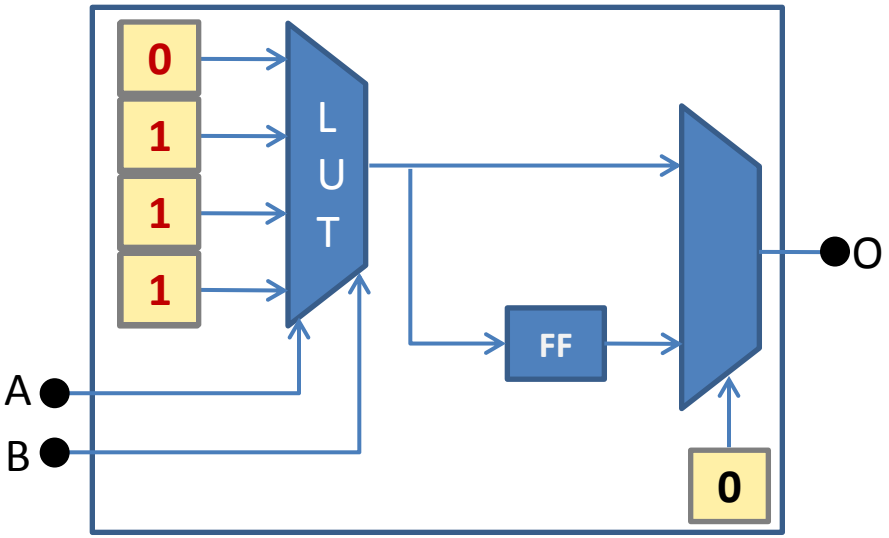
- FPGA configuration
- Dynamic Circuit Specialization (DCS)
- Exploitation of regularity in applications
- Experiments
- Conclusion

OUTLINE

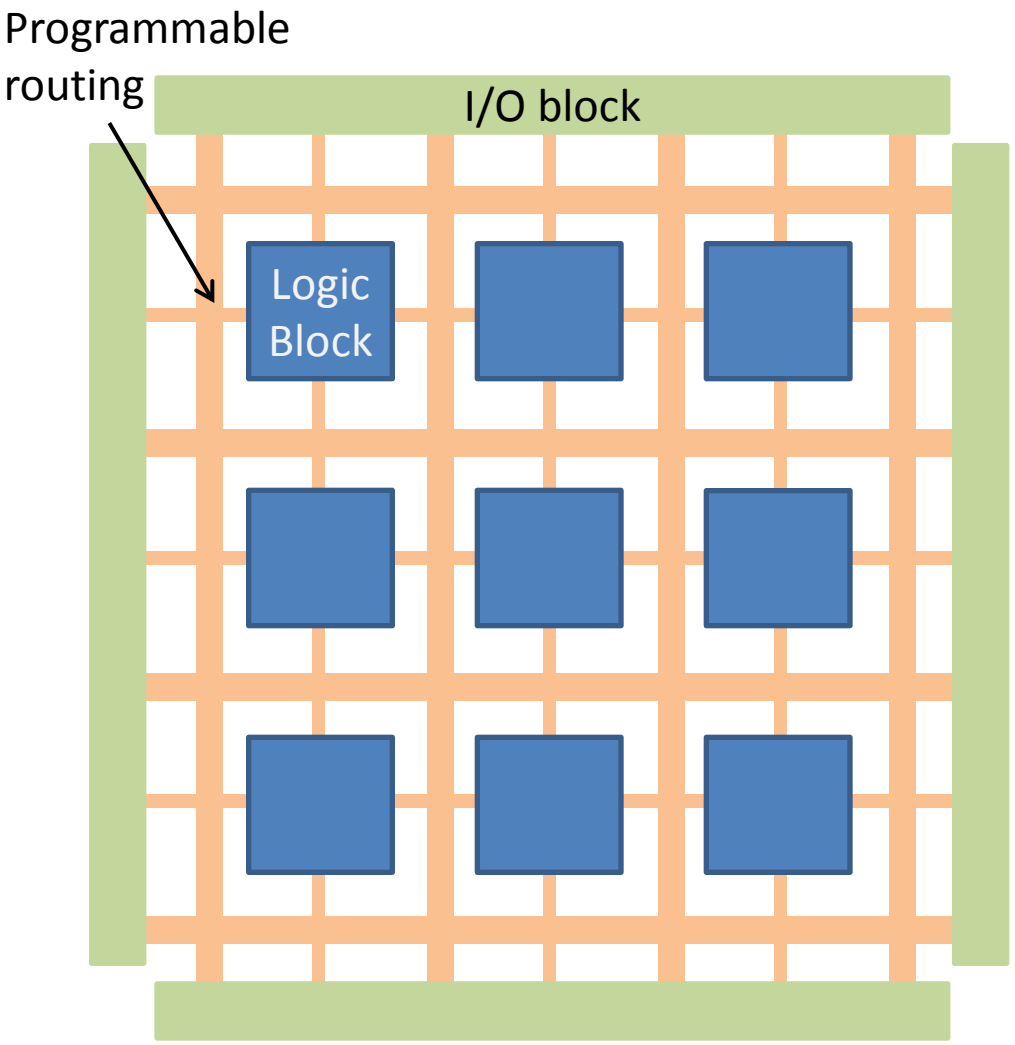
- FPGA configuration
- Dynamic Circuit Specialization (DCS)
- Exploitation of regularity in applications
- Experiments
- Conclusion

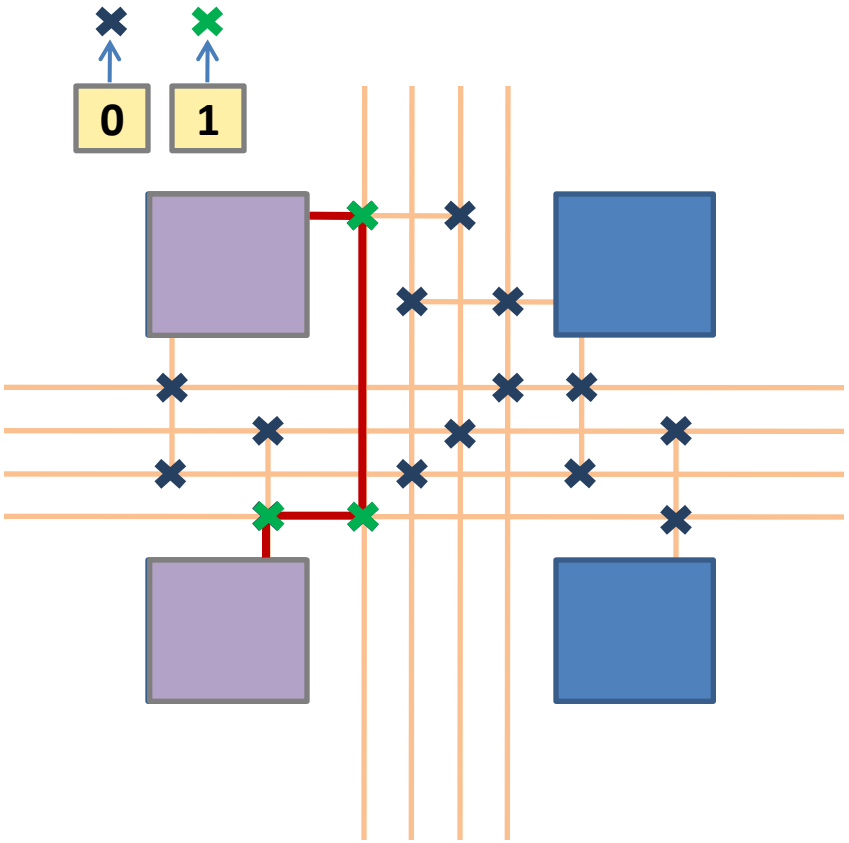
FPGA architecture



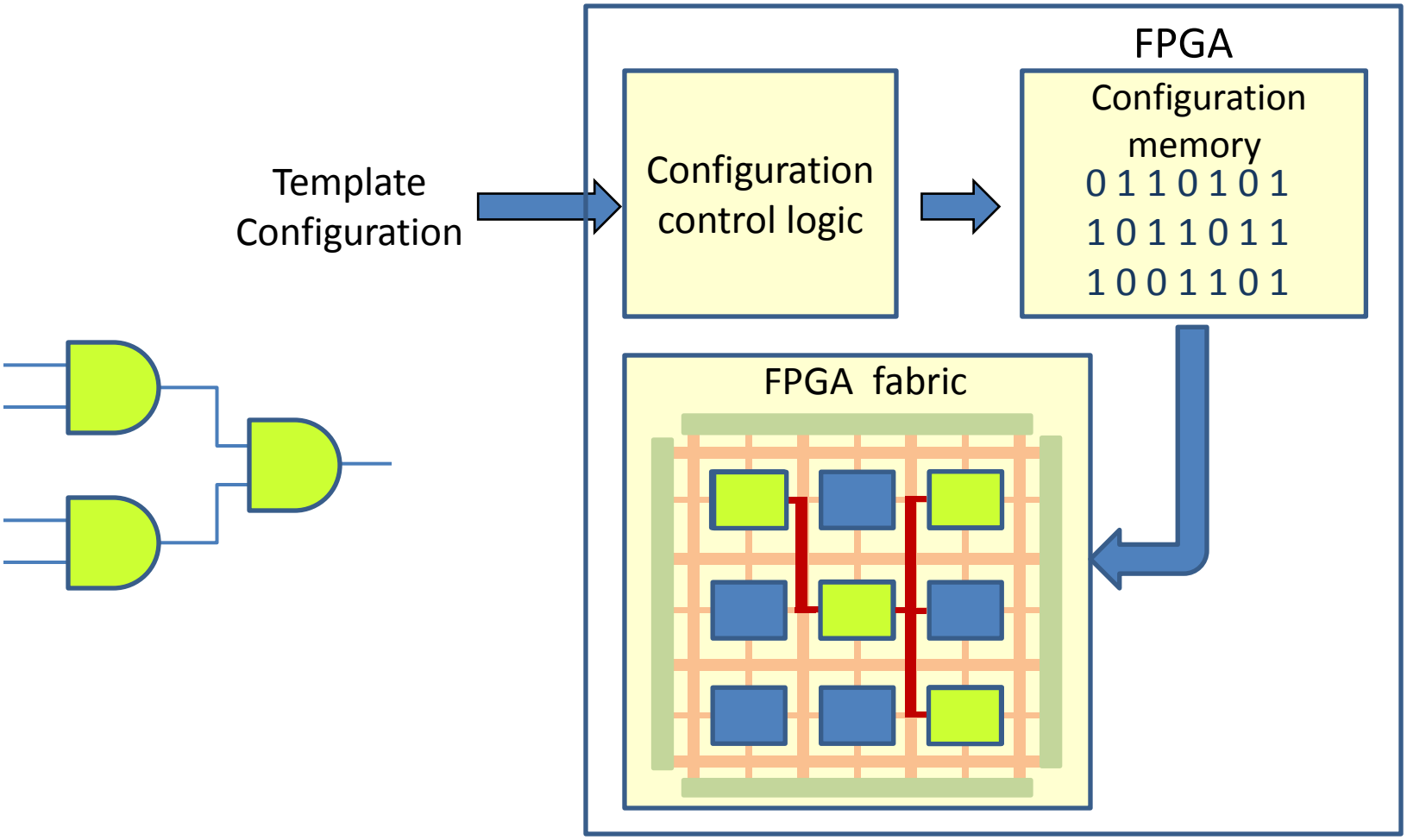


FPGA architecture

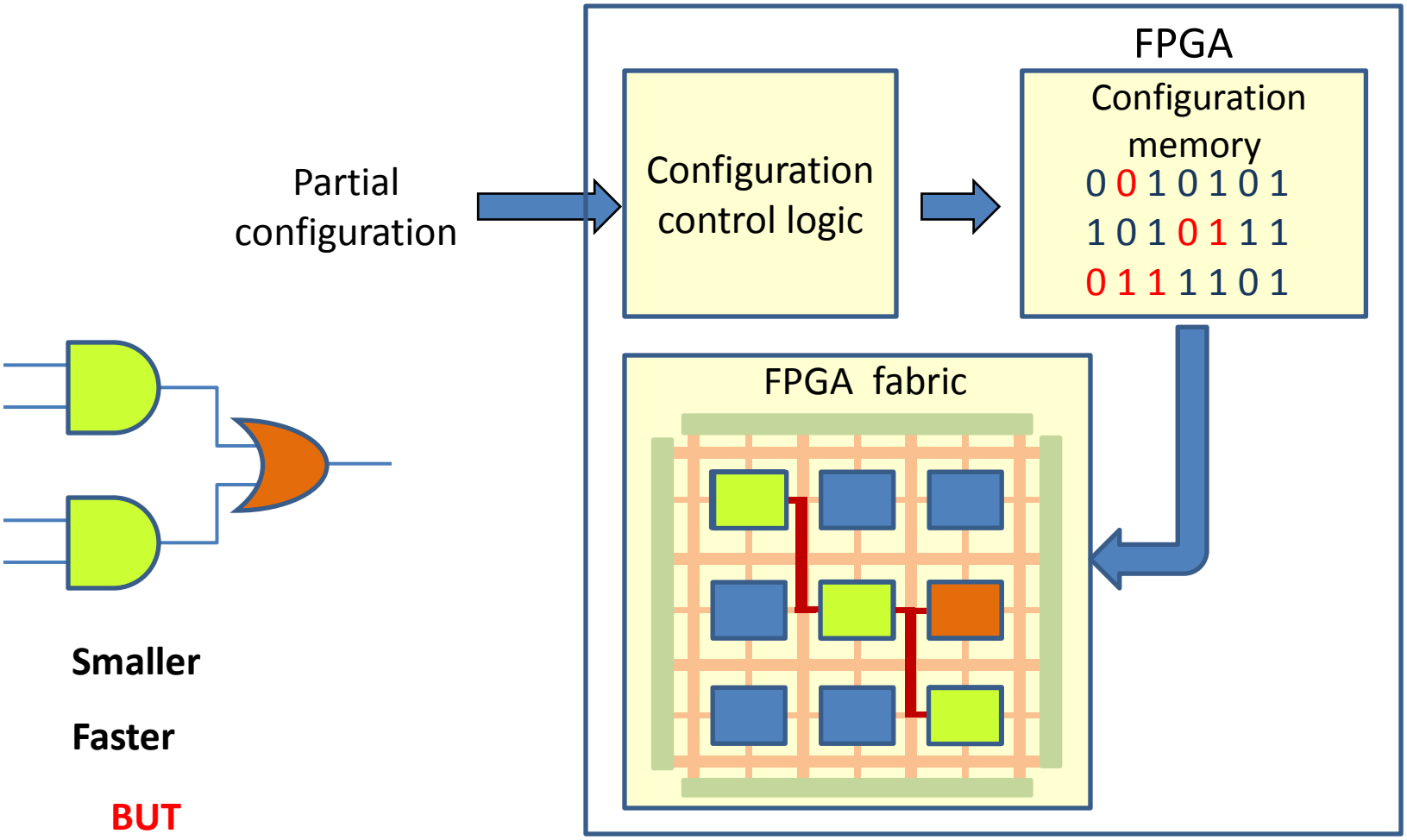




FPGA reconfiguration



FPGA reconfiguration



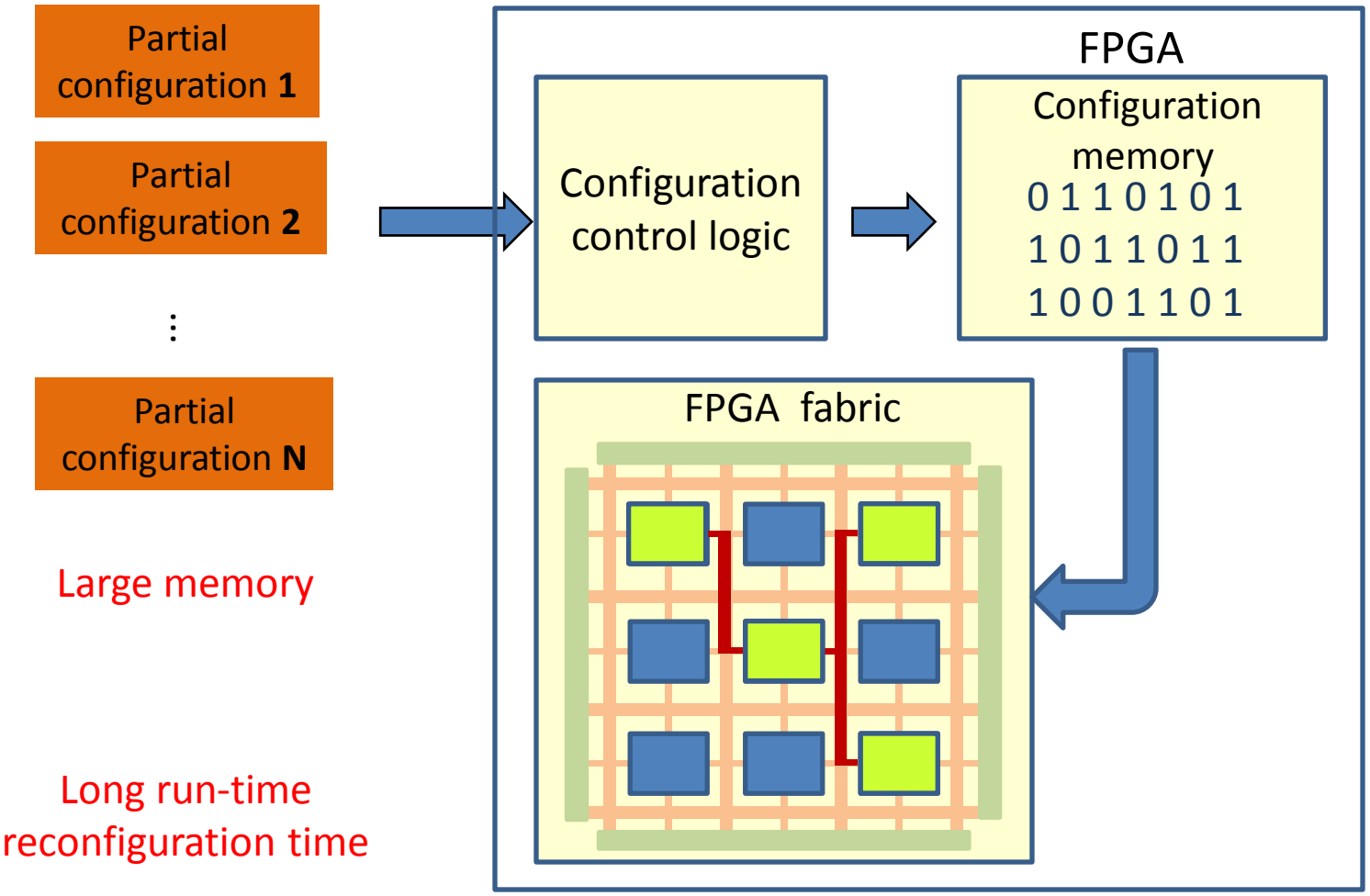
Smaller

Faster

BUT

How can we limit the reconfiguration overhead?

Reconfiguration overhead



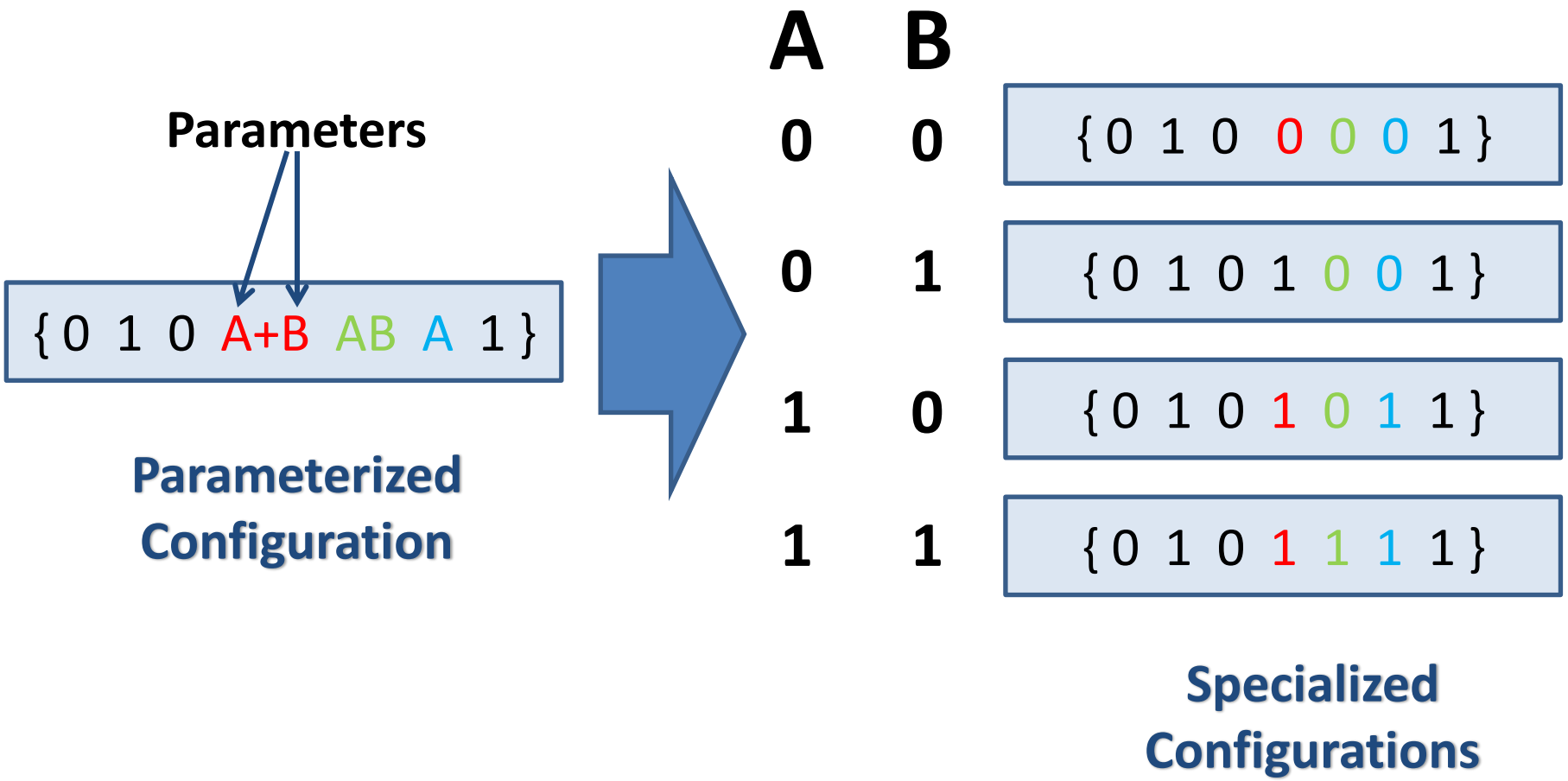
Large memory

Long run-time reconfiguration time

OUTLINE

- FPGA configuration
- Dynamic Circuit Specialization (DCS)
- Exploitation of regularity in applications
- Experiments
- Conclusion

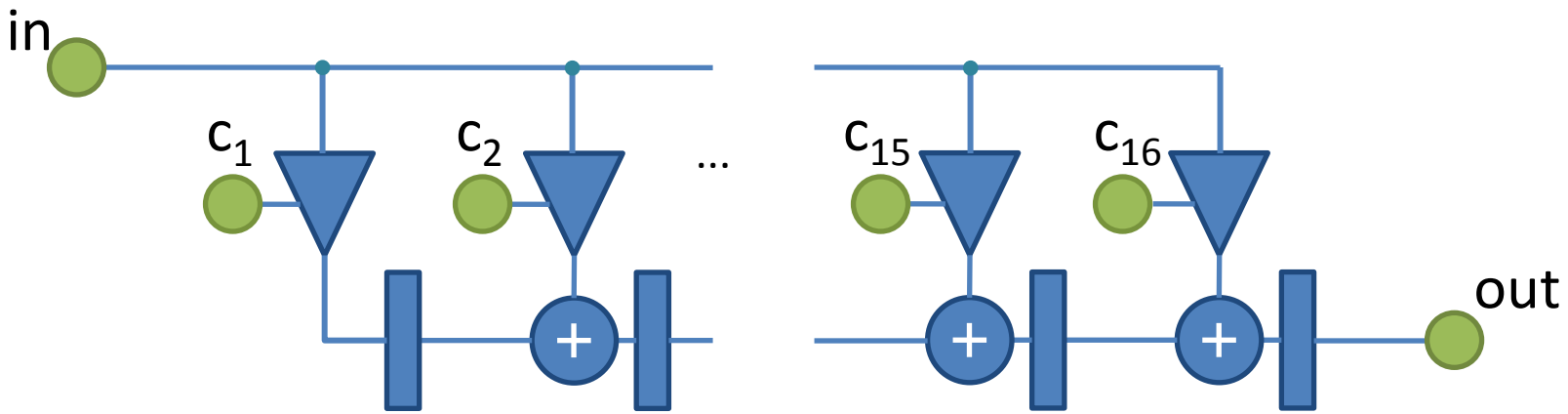
Parameterized Configuration (PC)



* K. Bruneel and D. Stroobandt, "Automatic Generation of Run-time Parameterizable Configurations," FPL 2008.

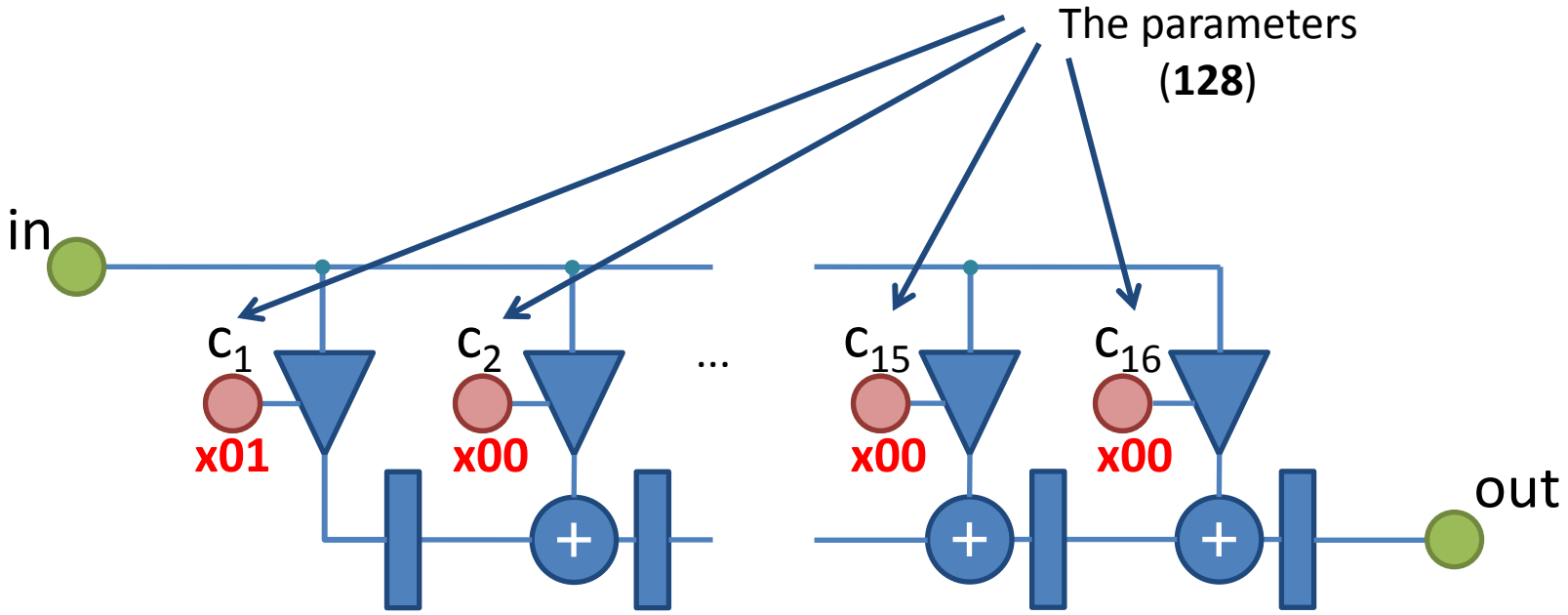
Example: Adaptive filtering

- FIR filter:
  - 16 taps
  - 8-bit input
  - 8-bit coefficients



The implementation of this generic FIR occupies large number of LUTs in the FPGA

Dynamic Circuit Specialization (DCS) technique



The implementation of a specialized FIR occupies less number of LUTs in the FPGA

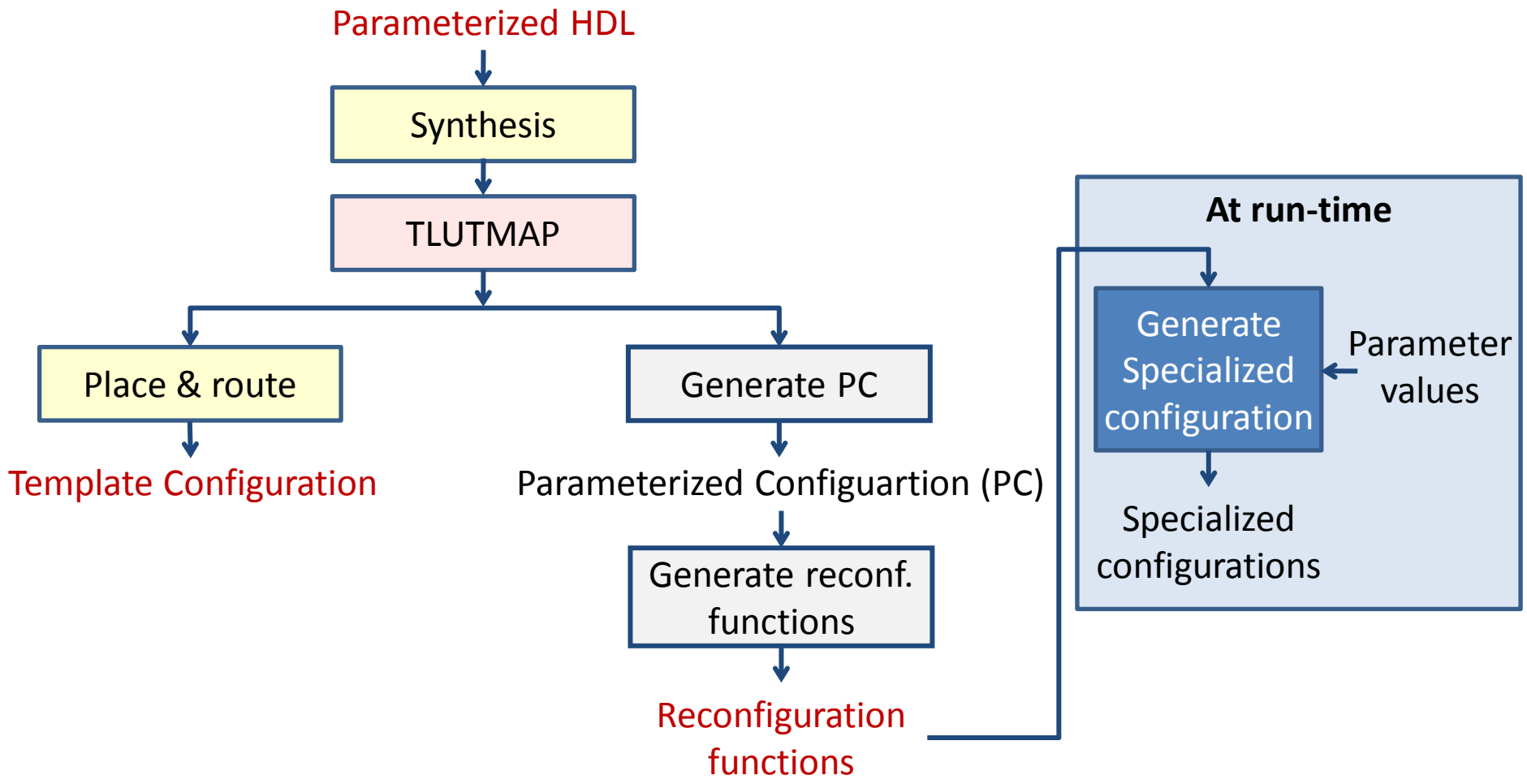
But

2^{128} configurations should be stored

Parameterized configuration and DCS

- By exploiting PC in DCS systems
  - ✚ One configuration needed to be stored
  - ✚ Configuration generation time is evaluation time

TLUT tool flow*



* K. Bruneel, F. Abouelella and D. Stroobandt, “Automatically Mapping Applications to a Self-reconfiguring Platform,” DATE 2009.

Current problem

- Exploiting parameterized configuratons in the TLUT tool flow

  ✚ Overhead of dynamically specializing circuits is significantly reduced

  (One configuration stored + configuration generation time)

  ▬ PC memory can be very large especially when applications scale

Proposed solution achievements

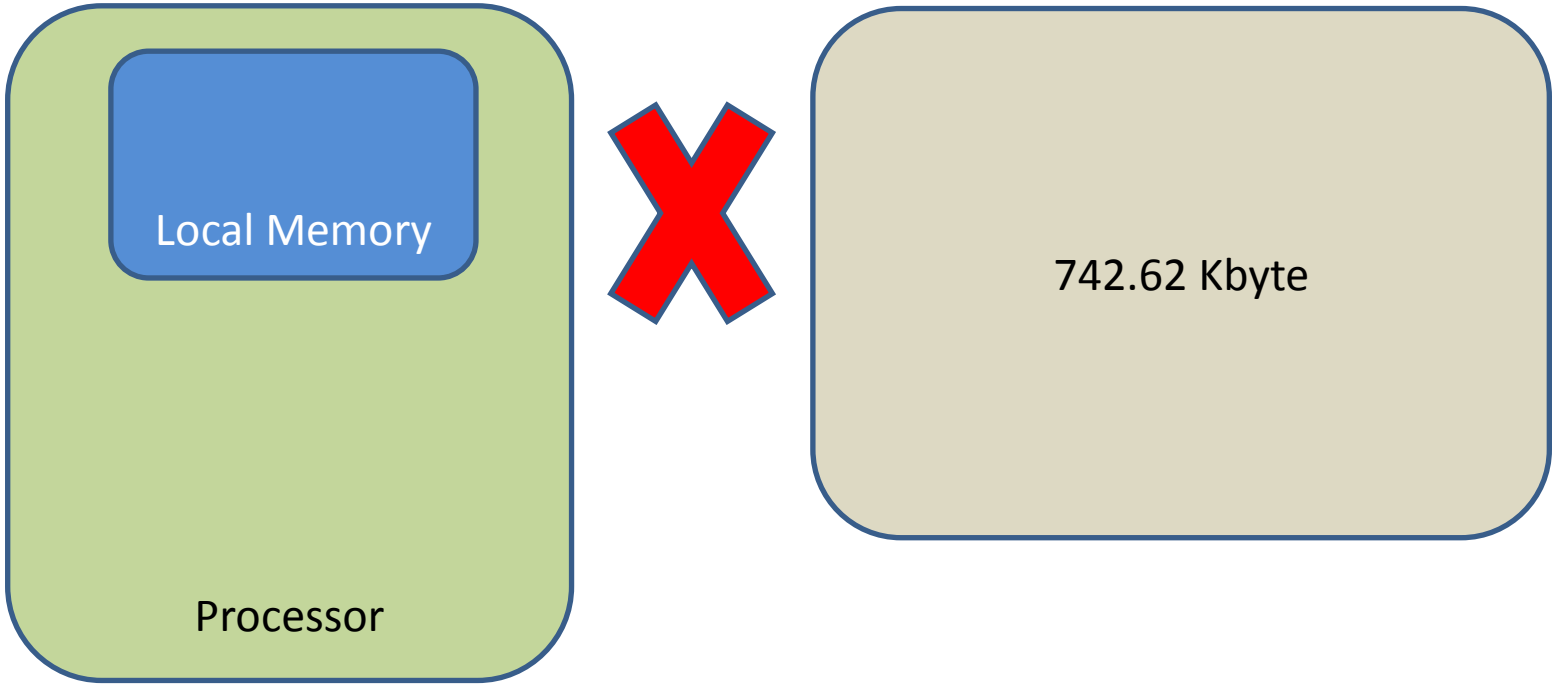
For 64-tap FIR filter

	Before	After	
PC resources (Kbyte)	742.67	9.85	76 times less memory

Proposed solution achievements

For 64-tap FIR filter

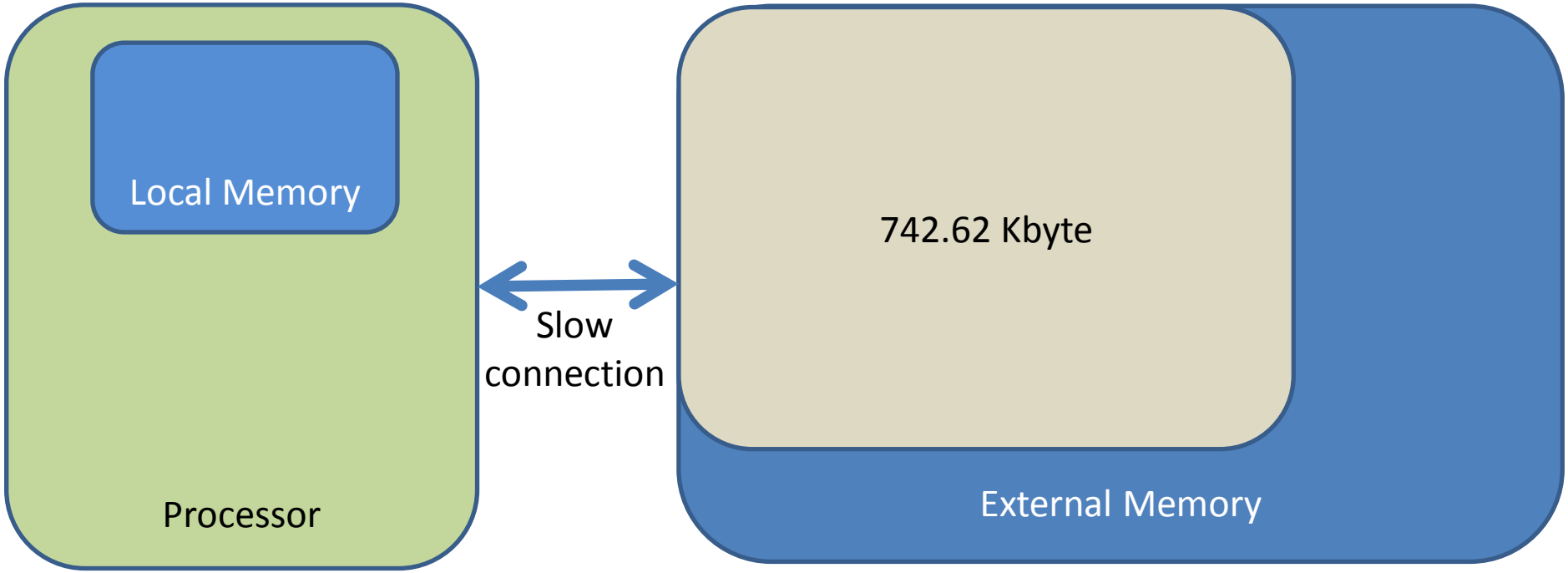
	Before	After	
PC resources (Kbyte)	742.67	9.85	76 times less memory



Proposed solution achievements

For 64-tap FIR filter

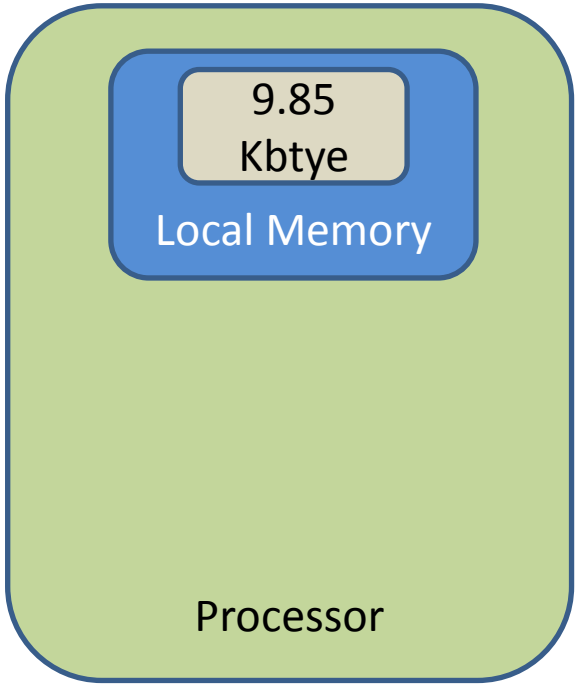
	Before	After	
PC resources (Kbyte)	742.67	9.85	76 times less memory



Proposed solution achievements

For 64-tap FIR filter

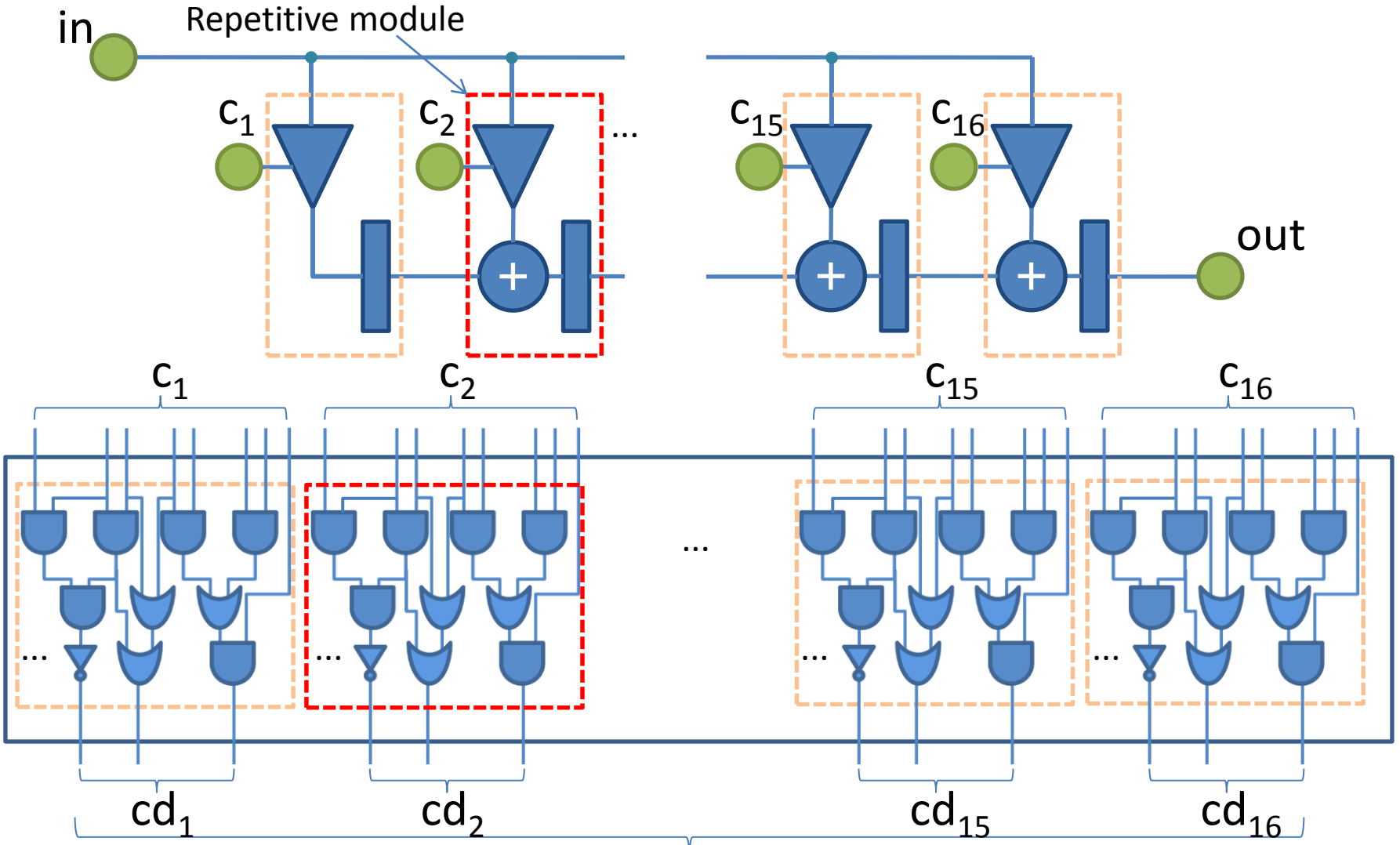
	Before	After	
PC resources (Kbyte)	742.67	9.85	76 times less memory



OUTLINE

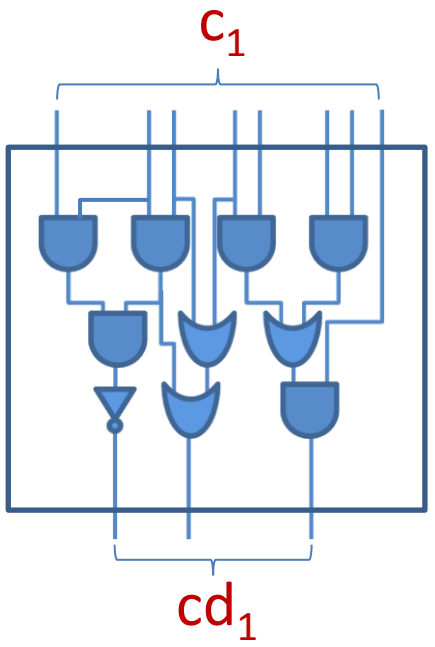
- FPGA configuration
- Dynamic Circuit Specialization (DCS)
- Exploitation of regularity in applications
- Experiments
- Conclusion

Main idea


in
Repetitive module
c1
c2
...
c15
c16
out
c1
c2
c15
c16
...
cd1
cd2
cd15
cd16
Configuration Data

Main idea

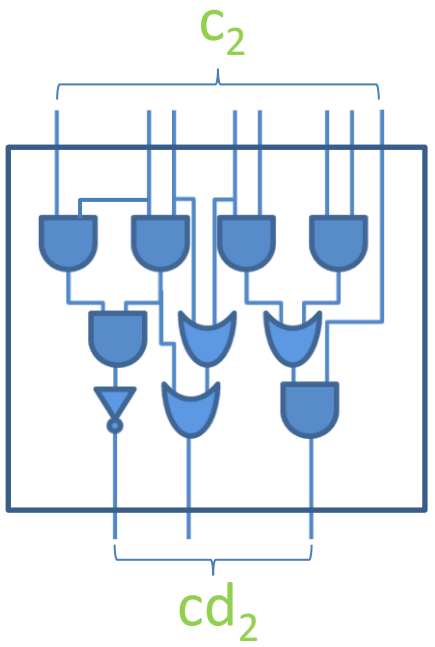
At run-time



Configuration Data

Main idea

At run-time

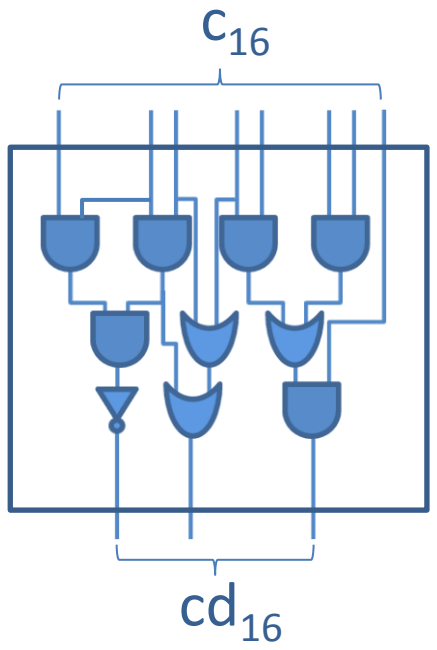


cd_1

Configuration Data

Main idea

At run-time

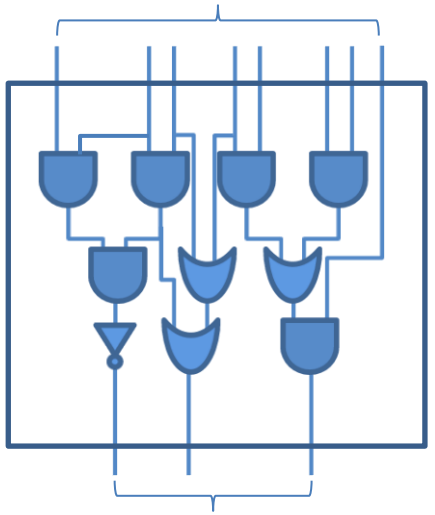


cd_1 cd_2 ...

Configuration Data

Main idea

At run-time



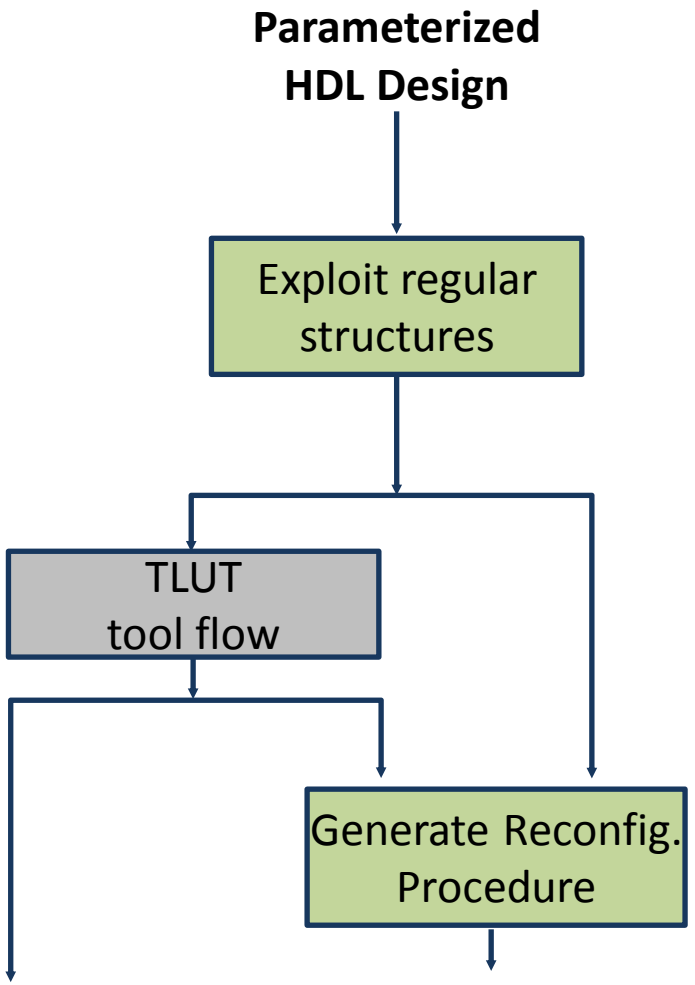
cd_1 cd_2 ... cd_{16}

Configuration Data

How to detect regularity in applications and transfer this regularity to the reconfiguration process?

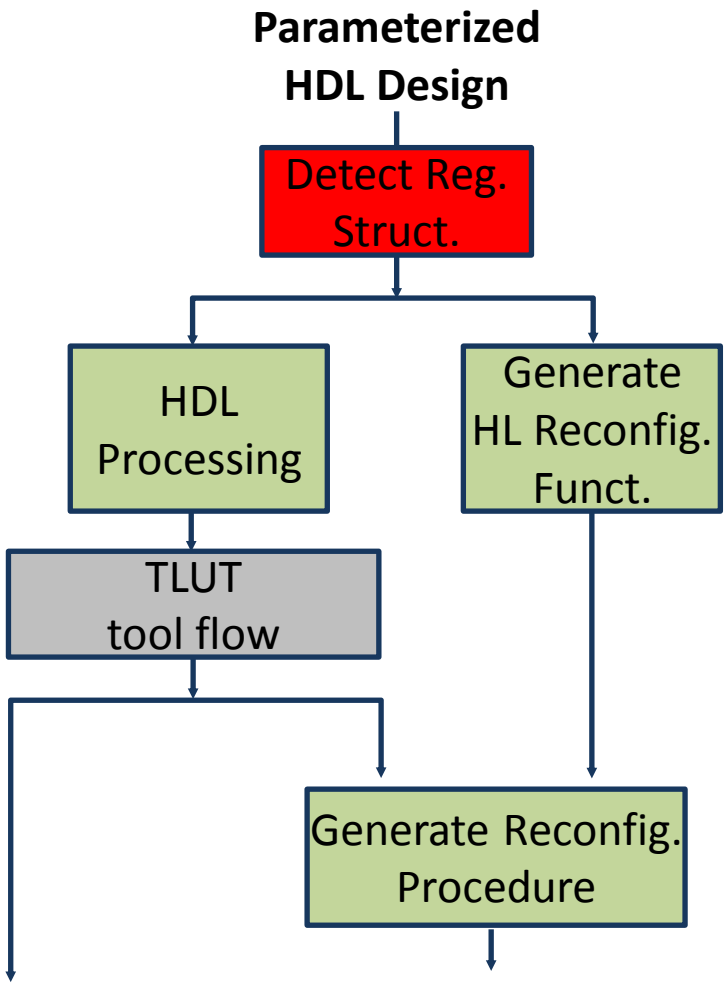
TLUT tool flow with front end

```
entity FIR is
port (clk : in std_logic;
      i : in std_logic_vector(DIW-1 downto 0);
   -- Start PARAM
   c : in arrayOfStdlogicVectors;
   -- End PARAM
      o : out std_logic_vector(DOW-1 downto 0));
end FIR;

architecture struct of FIR is
       -- declarations
begin
  GS_FOR : for T in 0 to N-1 generate

      GS_IF1 : if (T = 0) generate
            MULTIPLIER0 : entity work.multiplier(rtl)
                          port map (i, c(T), mult(T));
            LOAD : entity work.init(rtl)
                   port map (clk, mult(T), inter(T+1));
       end generate GS_IF1;
      GS_IF2 : if (0 < T < N) generate
            MULTIPLIER0 : entity work.multiplier(rtl)
                          port map (i, c(T), mult(T));
            ADDER0 : entity work.seqadder(rtl)
                     port map (clk, mult(T), inter(T), inter(T+1));
       end generate GS_IF2;
   end generate GS_FOR;

   o <= inter(N);
end struct;
```

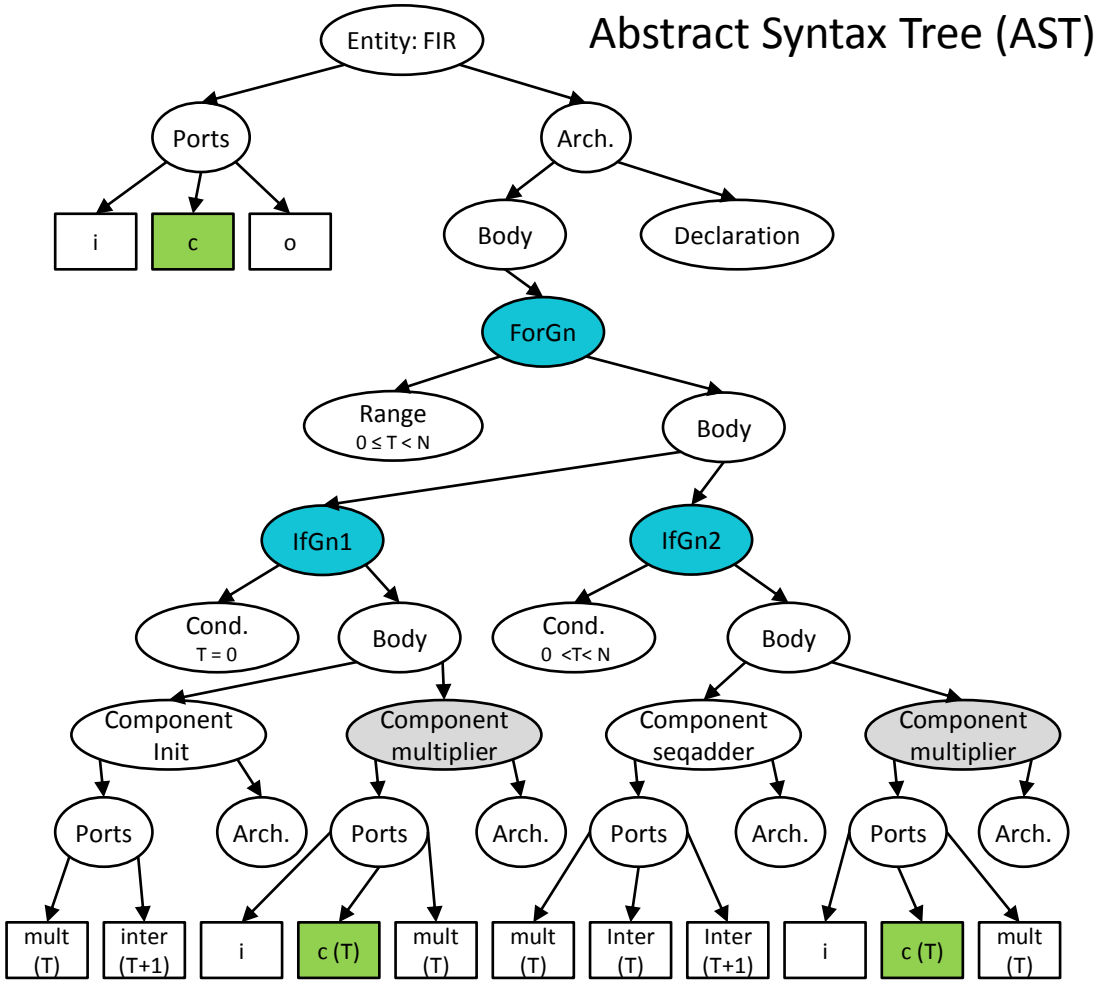


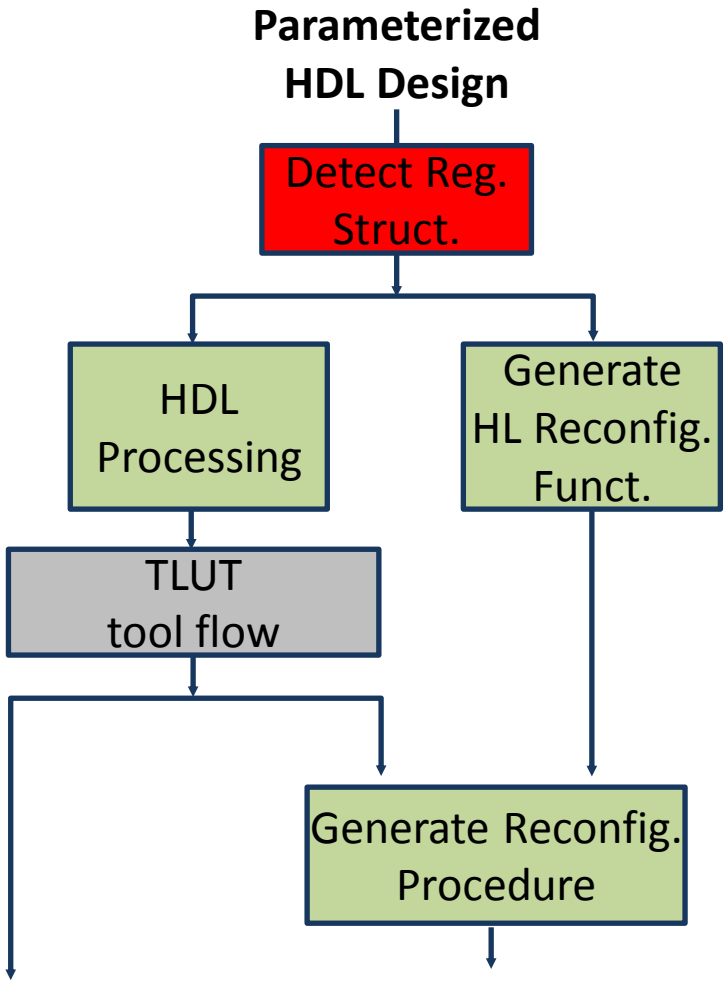
TLUT tool flow with front end

```
entity FIR is
port (clk : in std_logic;
      i : in std_logic_vector(DIW-1 downto 0);
   -- Start PARAM
   c : in arrayOfStdlogicVectors;
   -- End PARAM
      o : out std_logic_vector(DOW-1 downto 0));
end FIR;

architecture struct of FIR is
       -- declarations
begin
  GS_FOR : for T in 0 to N-1 generate

       GS_IF1 : if (T = 0) generate
            MULTIPLIER0 : entity work.multiplier(rtl)
                          port map (i, c(T), mult(T));
            LOAD : entity work.init(rtl)
                   port map (clk, mult(T), inter(T+1));
        end generate GS_IF1;
       GS_IF2 : if (0 < T < N) generate
            MULTIPLIER0 : entity work.multiplier(rtl)
                          port map (i, c(T), mult(T));
            ADDER0 : entity work.seqadder(rtl)
                     port map (clk, mult(T), inter(T), inter(T+1));
        end generate GS_IF2;
    end generate GS_FOR;

   o <= inter(N);
end struct;
```

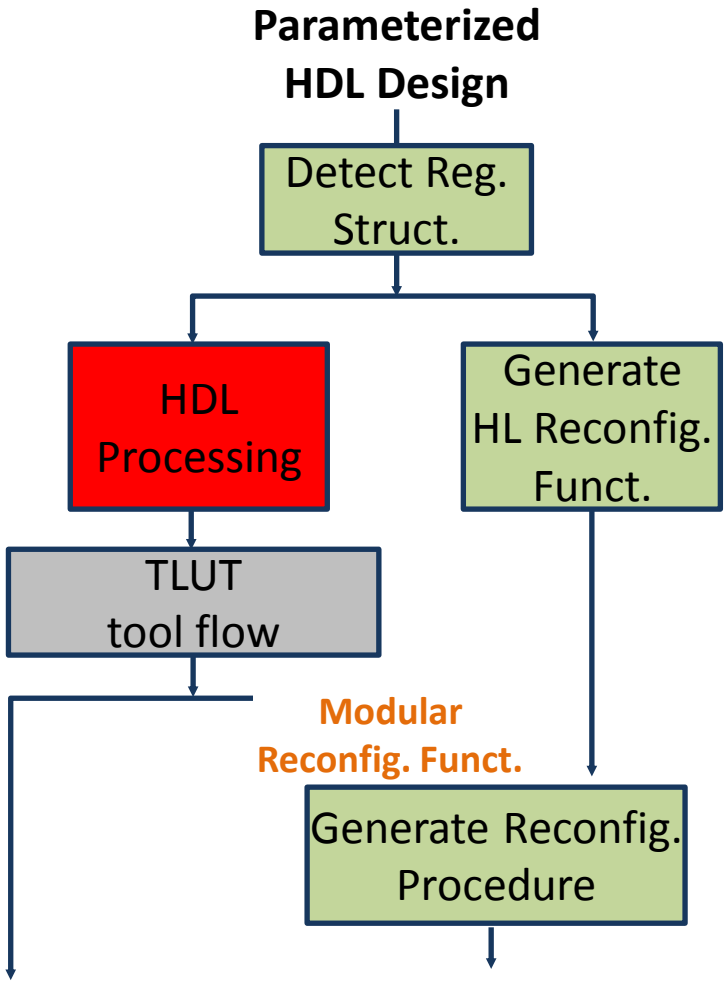


TLUT tool flow with front end

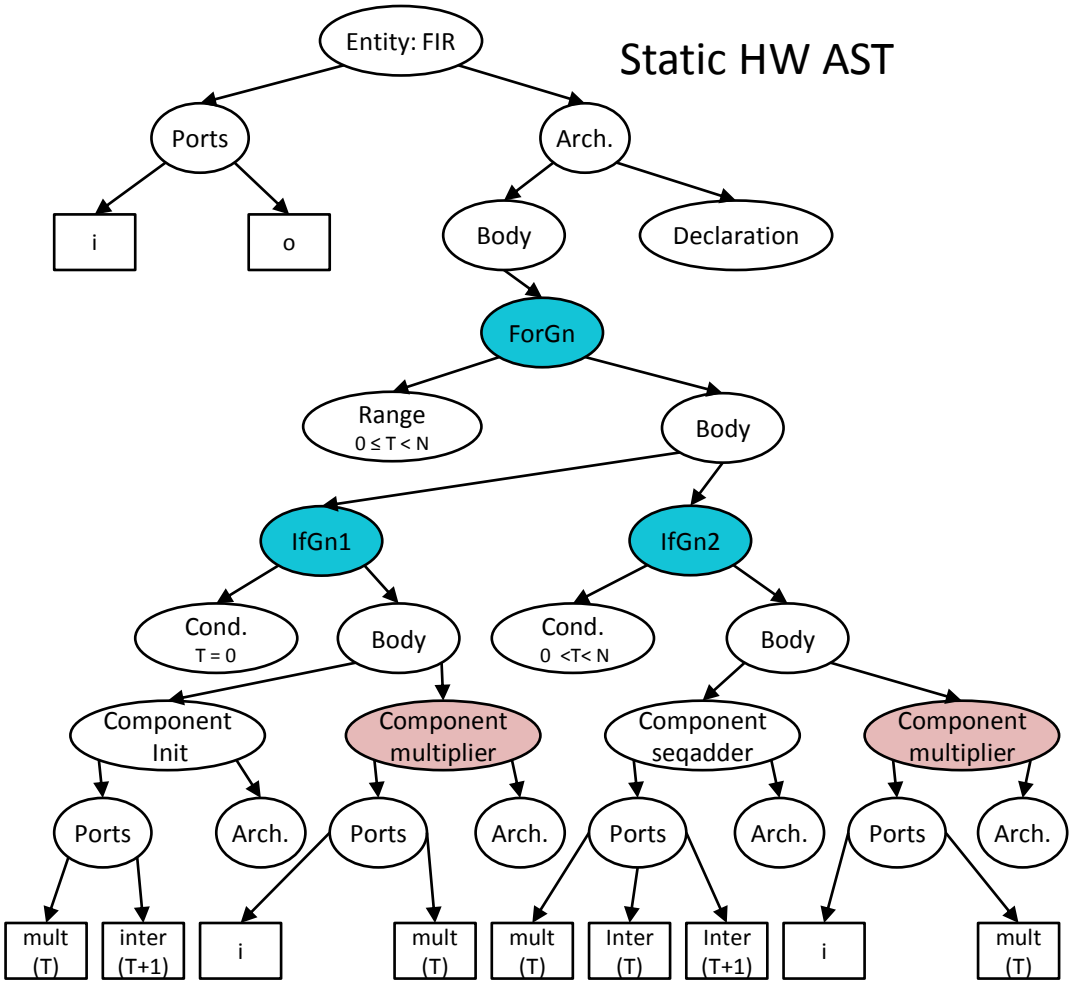


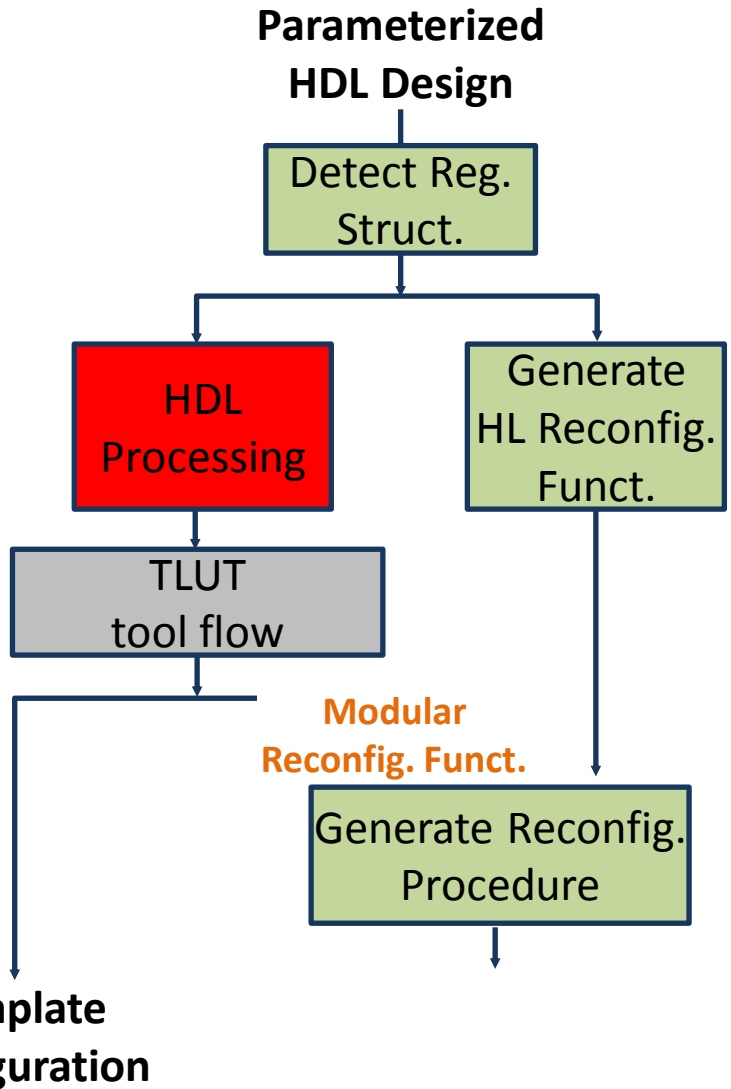


TLUT tool flow with front end

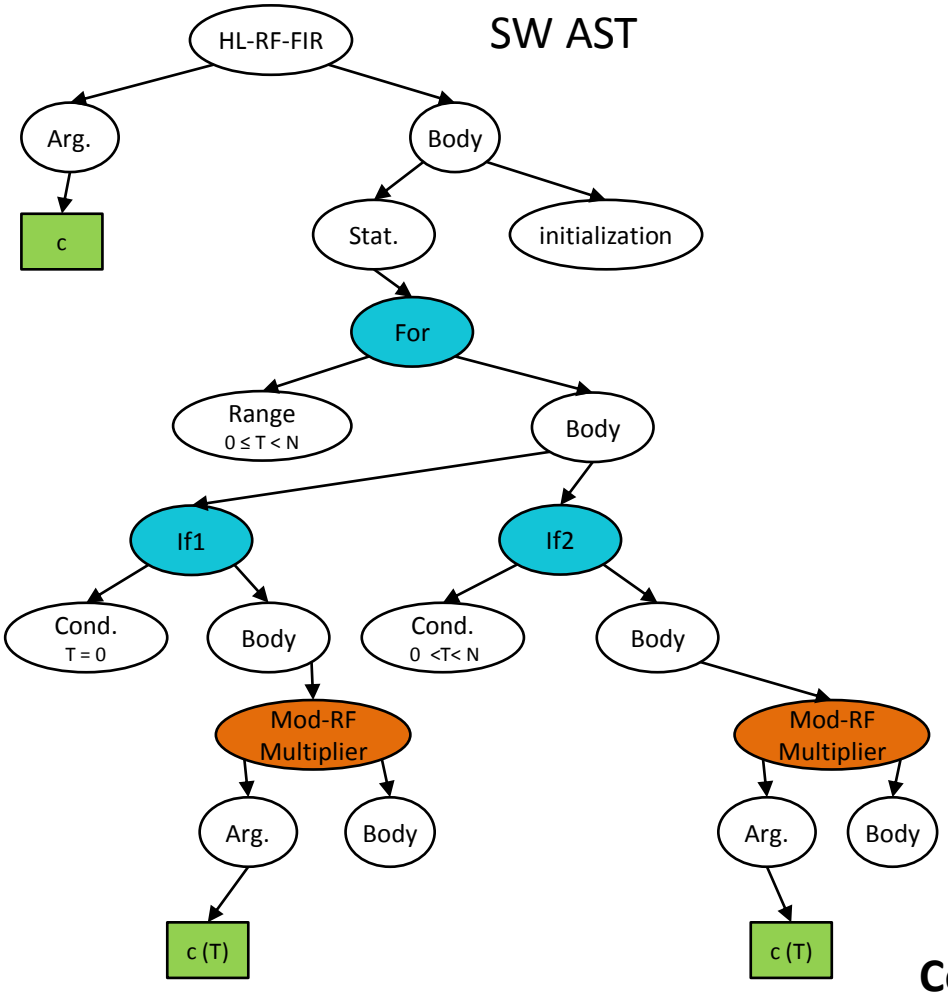


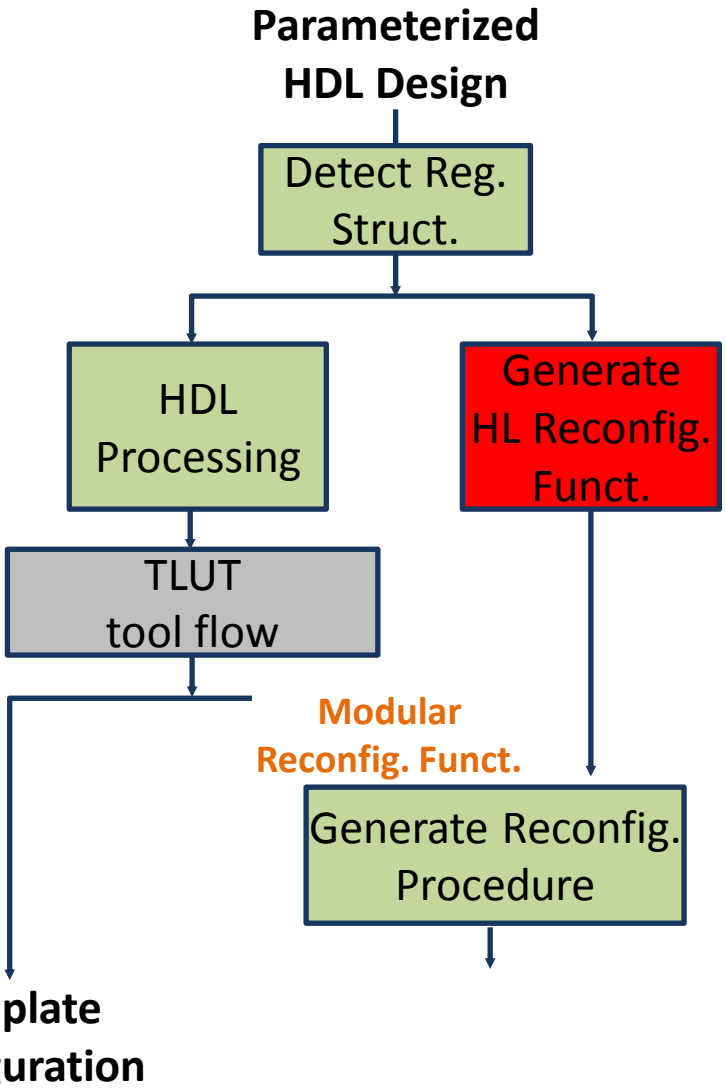
TLUT tool flow with front end





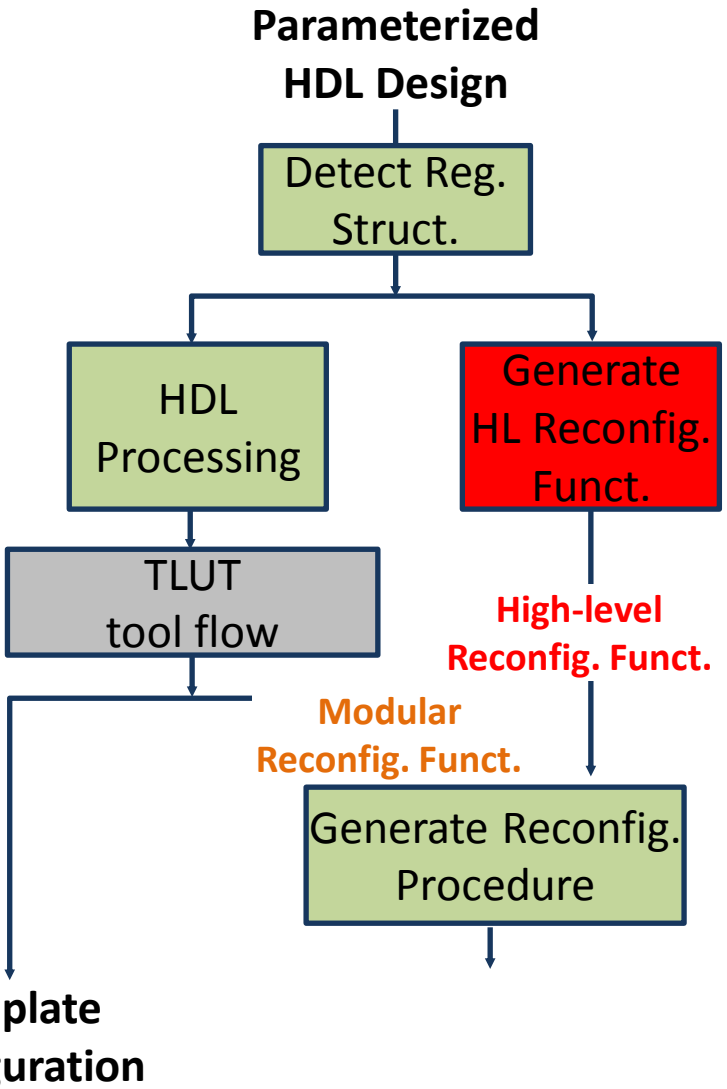
TLUT tool flow with front end



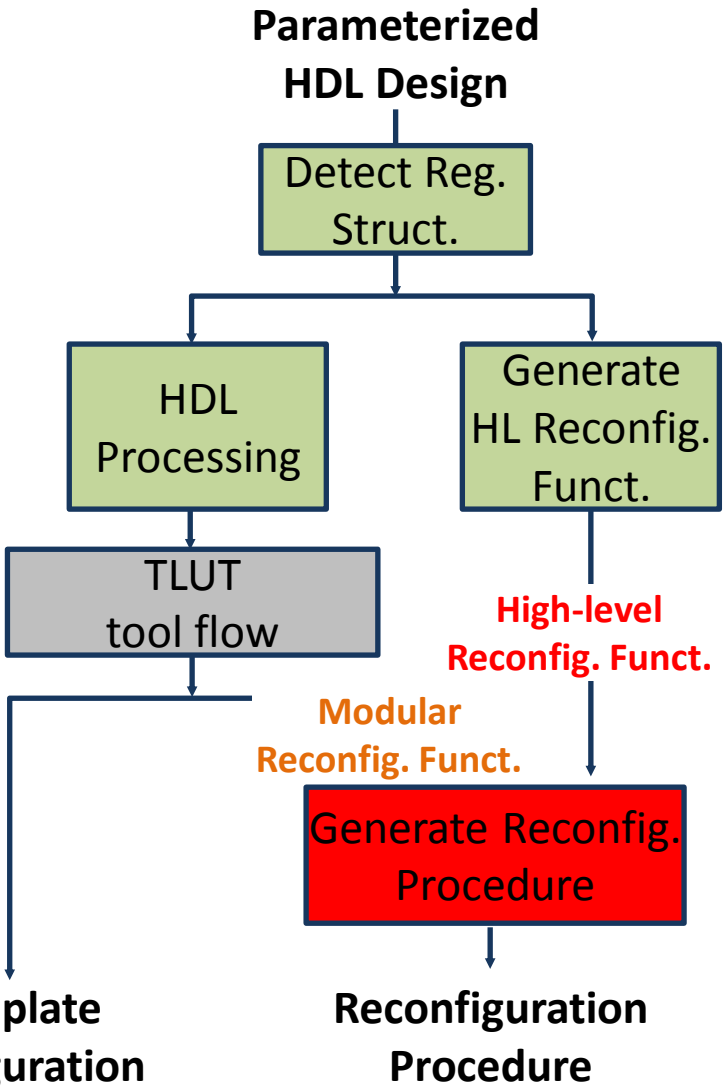


TLUT tool flow with front end

```
void HL-RF-FIR(int c[Ciel(N)][DIW]){
      for (T = 0; T< N; T++){
         if (T == 0)
            Mod-RF-multiplier (c[T]);
         if( T>0 && T<Ceil(N))
            Mod-RF-multiplier (c[T]);
      }
}
```



TLUT tool flow with front end



OUTLINE

- FPGA configuration
- Dynamic Circuit Specialization (DCS)
- Exploitation of regularity in applications
- Experiments
- Conclusion

Experiments

- Platform:
  - Xilinx Virtex-II Pro (XCVP30) FPGA
  - Reconfiguration procedure:
    - Set of C functions (HL and Modular RFs)
    - Executed on the embedded PowerPC of the Xilinx Virtex-II Pro

- Implementation methodologies:
  - Conventional
  - TLUT tool flow
  - TLUT tool flow with the front end (TLUT w.f.e.)

- measurements :
  - Design area (LUTs)
  - Maximum frequency
  - Memory resources needed to store reconfiguration procedure
  - Configuration generation time

FIR filter (64-tap)

	Implementation		Overhead	
	Design area (LUTs)	Frequency (Mhz)	Recon. Procedure (Kbyte)	Generation time (us)
Conventional	7578	85.44	-	-
TLUT	5080	195.72	742.67	21731.92*
TLUT w.f.e.	5080	185.61	9.85	982.22
	32%	~50%	76x ⬇	22x ⬇

*This generation time is associated to a reconfiguration procedure that is stored in an external memory.

RegExp matcher (13-GB)

	Implementation		Overhead	
	Design area (LUTs)	Frequency (Mhz)	Recon. Procedure (Kbyte)	Generation time (us)
Conventional	709	14.80	-	-
TLUT	392	78.55	37.29	95.44
TLUT w.f.e.	377	49.21	3.40	26.58
	44-46%	70-81%	10x ⬇	3x ⬇

Ternary Content Addressable Memory (TCAM) (256 entries)

	Implementation		Overhead	
	Design area (LUTs)	**Frequency (Mhz)**	**Recon. Procedure (Kbyte)**	**Generation time (us)**
Conventional	10871	65.47	-	-
TLUT	4036	67.38	543.39	15123.66*
TLUT w.f.e.	4036	78.98	3.25	480.44
	62%	**2-17%**	**167x ⬇**	**31x ⬇**

OUTLINE

- FPGA configuration
- Dynamic Circuit Specialization (DCS)
- Exploitation of regularity in applications
- Experiments
- Conclusion

Conclusion

- The PC memory was very large especially when applications scale.
- We introduce an automatic method that exploits regularity to reduce PC memory requirements.
- Exploiting the regularity achieves:
  - PC memory reduction factor of 76 and 176 for the FIR filter (64-tap) and TCAM (256 entries).
  - The reduction factor increases when applications scale.

Thank you