A Linear Feedback Shift Register for Crypto-Systems on Reconfigurable Computing Platforms

Amine Demidem (ECE)

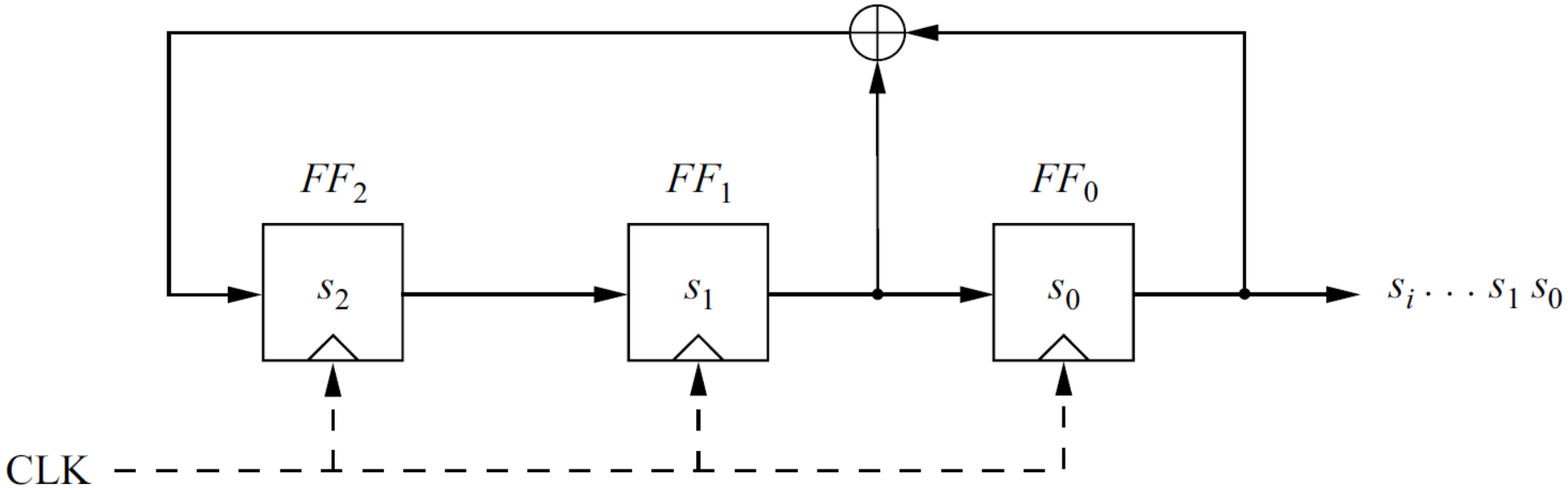
ELEC 569A, Fall 2015

demidem@uvic.ca

The Outline

- **Introduction to the LFSR**
- **Background about the FPGA**
- **The Brute Force Solution**
- **The Optimized Solutions**
- **Discussion**
- **Questions**

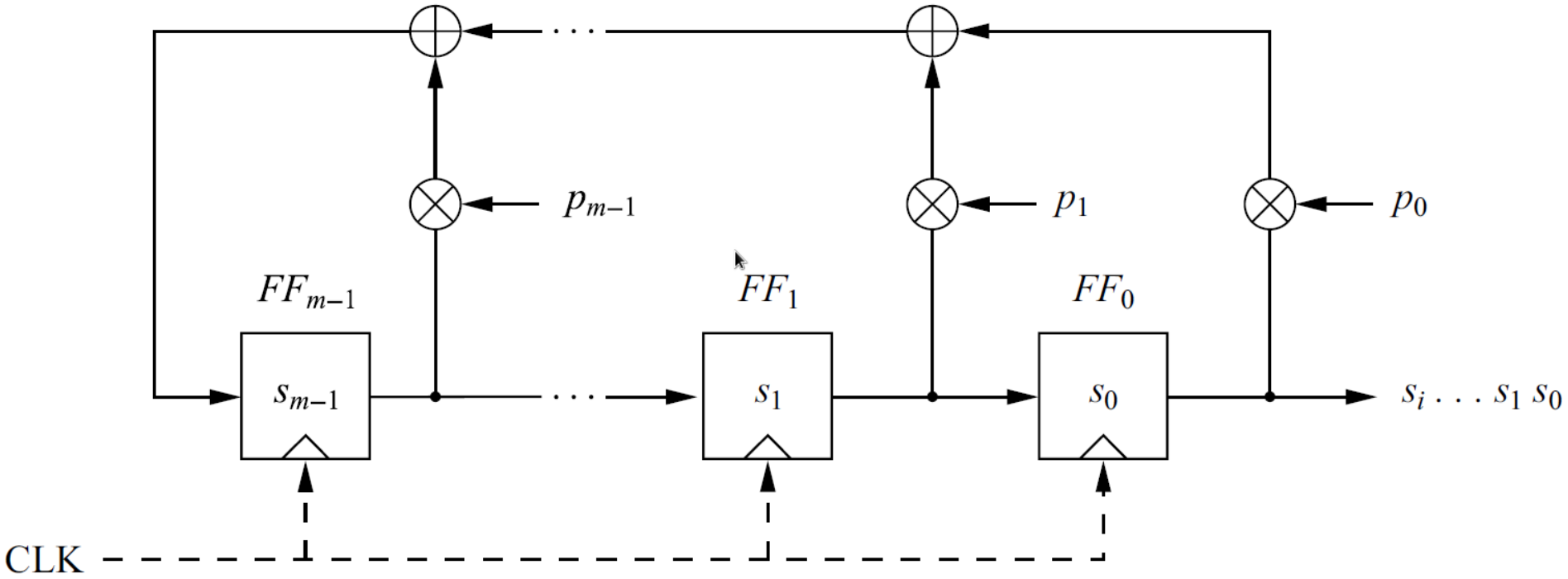
Introduction to the LFSR



Introduction to the LFSR

clk	FF_2	FF_1	$FF_0 = s_i$
0	1	0	0
1	0	1	0
2	1	0	1
3	1	1	0
4	1	1	1
5	0	1	1
6	0	0	1
7	1	0	0
8	0	1	0

Introduction to the LFSR



$$Sm = S_{m-1}P_{m-1} + \ldots + S_1P_1 + S_0P_0 \bmod 2$$

Introduction to the LFSR

- **M is called the degree or depth of the LFSR**

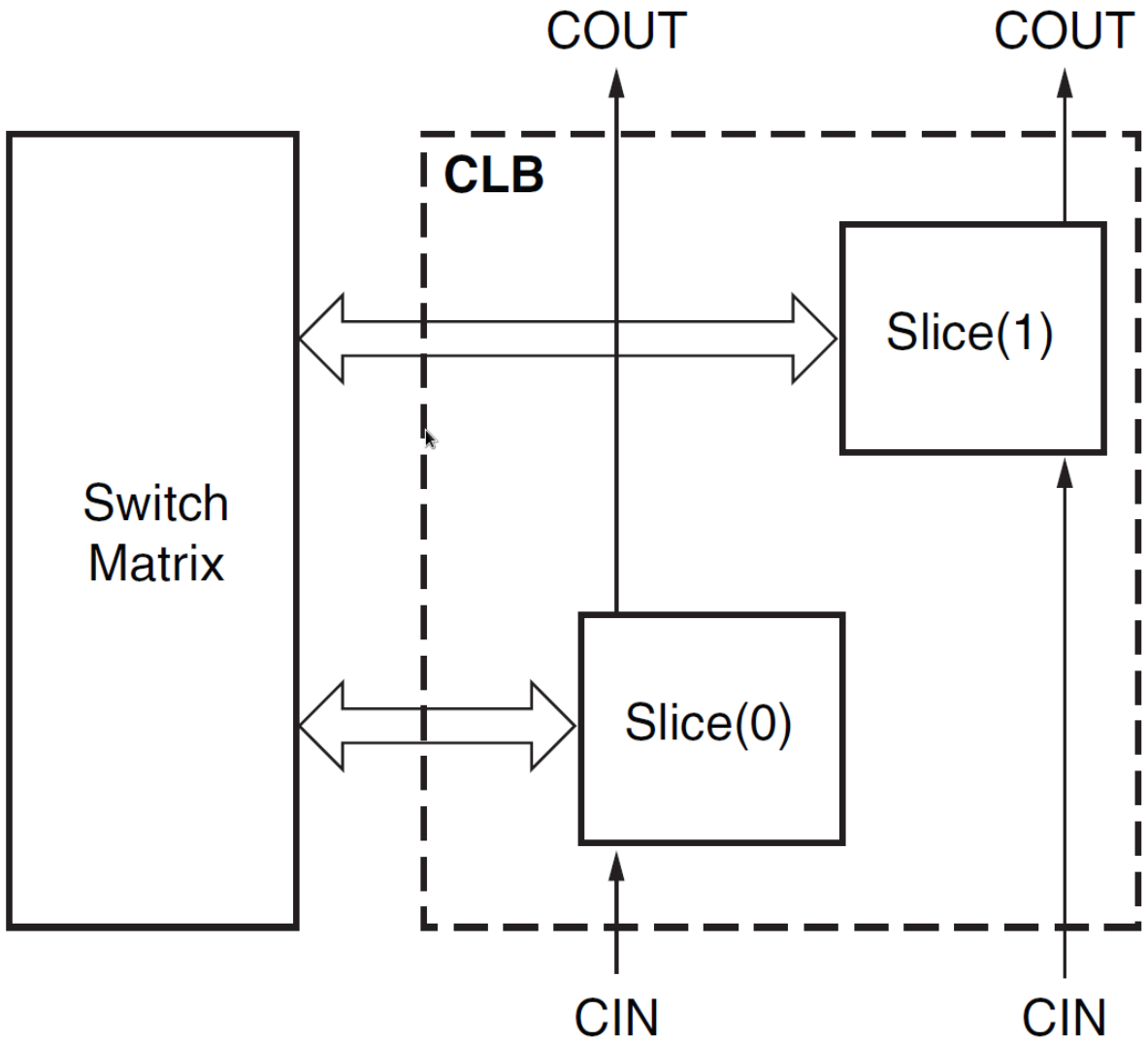
Introduction to the LFSR

- **M is called the degree or depth of the LFSR**
- **The maximum (attainable) sequence length generated by an LFSR of degree m is 2^M-1**

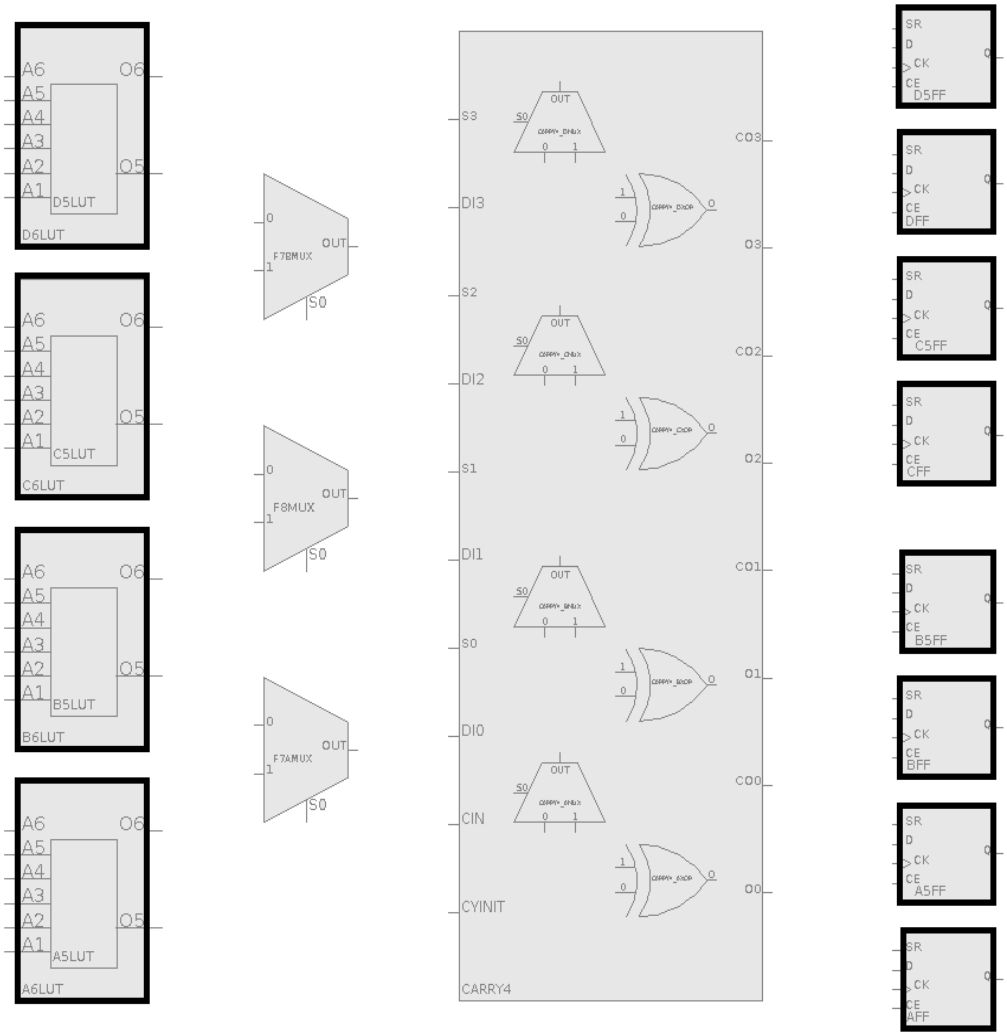
Introduction to the LFSR

- **M is called the degree or depth of the LFSR**
- **The maximum (attainable) sequence length generated by an LFSR of degree m is 2^M-1**
- **The maximum sequence length is only attained with some combinations of taps**

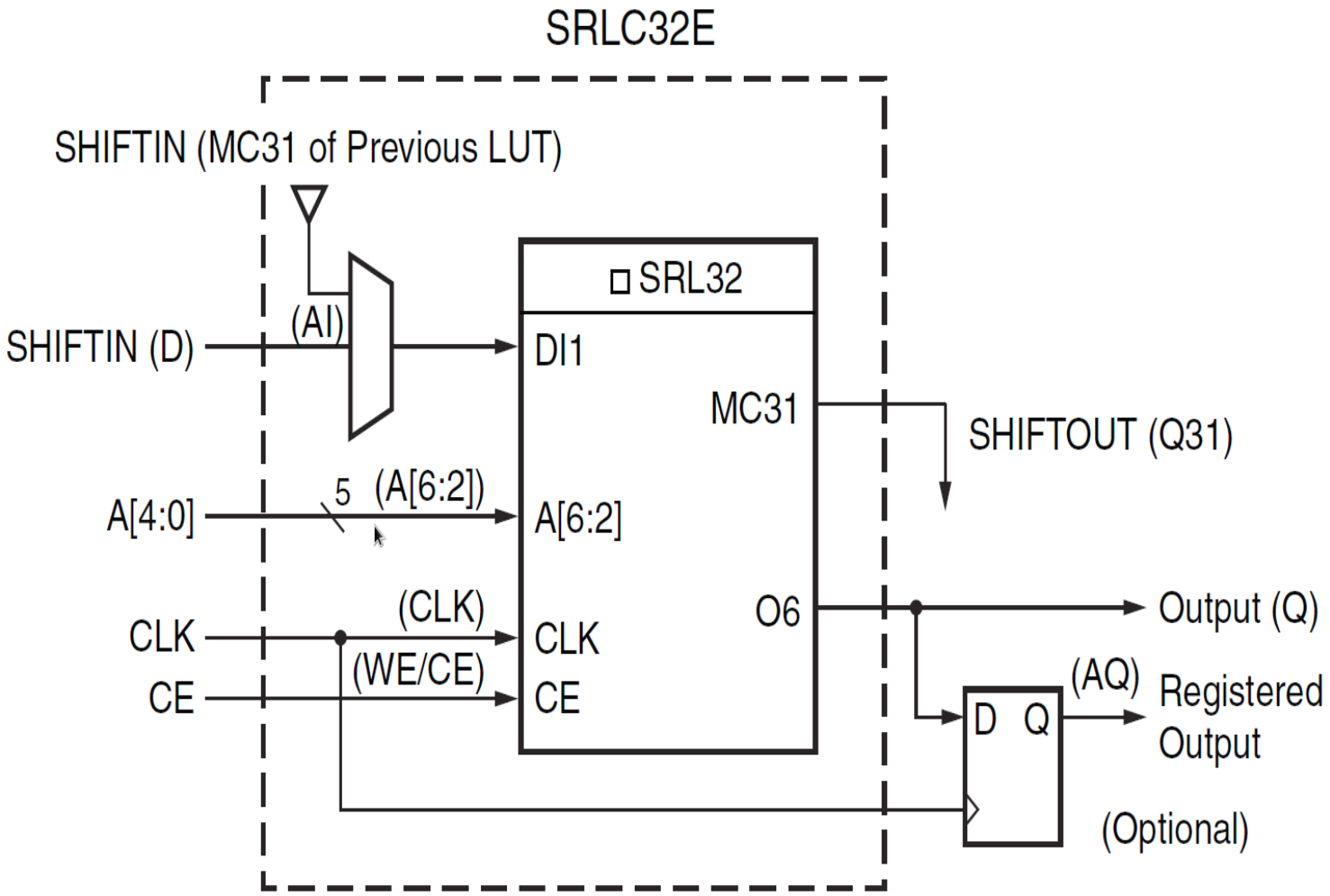
Background about the FPGA



Background about the FPGA



Background about the FPGA



Background about the FPGA

- **Flip Flops are virtually free with FPGA**

Background about the FPGA

- **Flip Flops are virtually free with FPGA**
- **Muxes, Logical Gates are not**

Background about the FPGA

- **Flip Flops are virtually free with FPGA**
- **Muxes, Logical Gates are not**
- **Wires are even less!**

Background about the FPGA

- **Flip Flops are virtually free with FPGA**
- **Mux's, Logical Gates are not**
- **Wires are even less!**
- **How do you compete with ASIC!?**

Performance Requirements

- **LUTs occupancy**

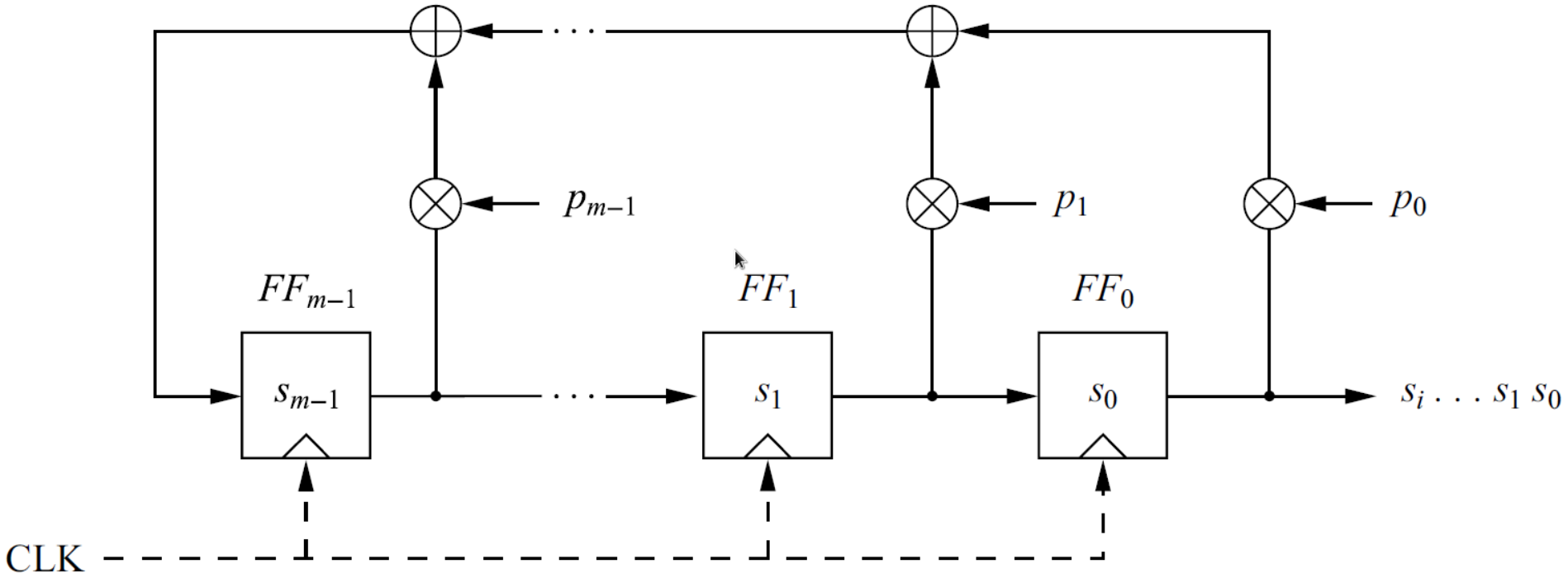
Performance Requirements

- **LUTs occupancy**
- **Critical Path delay**

Performance Requirements

- **LUTs occupancy**
- **Critical Path delay**
- **Wiring Requirements**

The Brute Force Solution



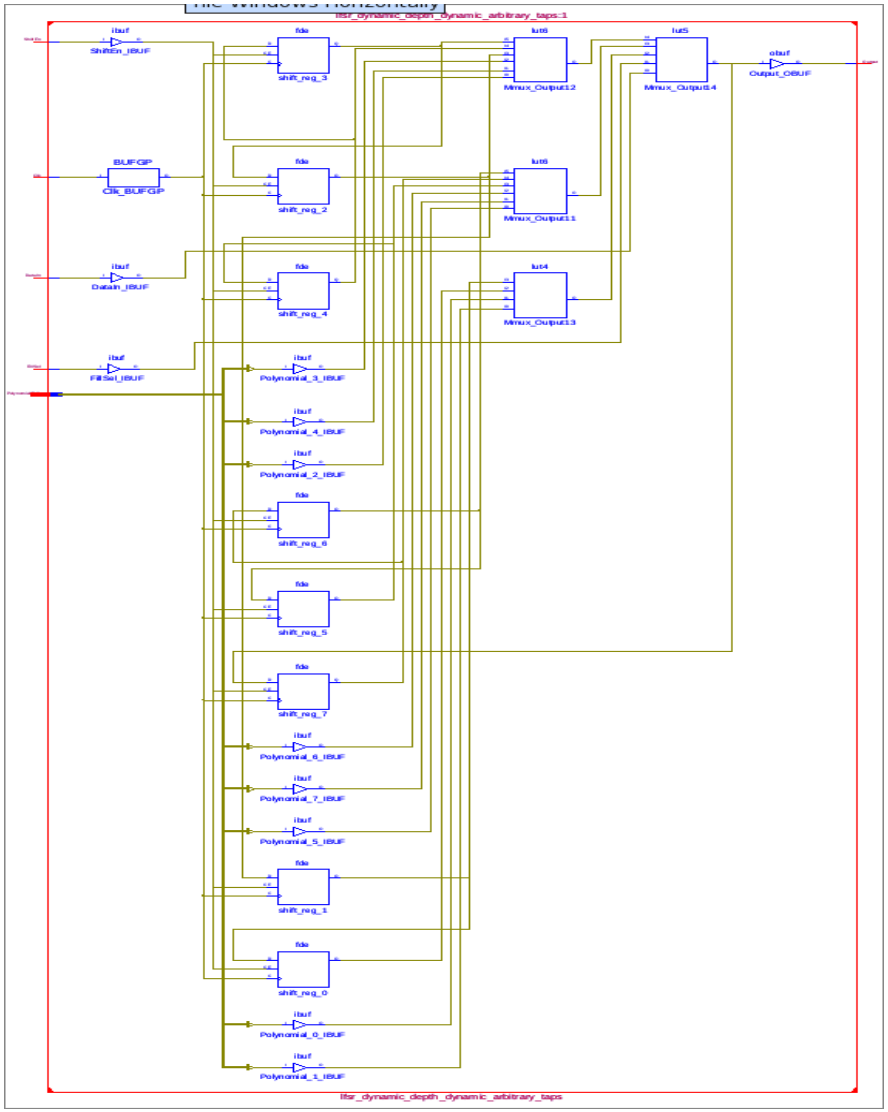
The Brute Force Solution

```
process (Clk)
begin
   if rising_edge(Clk) then
      if ShiftEn = '1' then
         shift_reg <= lfsr_in & shift_reg(shift_reg'high downto 1);
      end if;
   end if;
end process;

par_fdbk(0) <= '0';
fdbk_gen: for X in 0 to depth-1 generate
   bitxor: par_fdbk(X+1) <= par_fdbk(X) xor (shift_reg(X) and EnabledTaps(X));
end generate fdbk_gen;

lfsr_in <= DataIn when FillSel = '1' else par_fdbk(depth);
```

The Brute Force Solution



The Brute Force Solution

- **The Good:**
  - **Very Simple**

The Brute Force Solution

- **The Good:**
  - **Very Simple**

- **The Bad:**
  - **Resource wasteful**
  - **May lead to long delays**

The Brute Force Solution

Solution	Number of FF's	Number of LUTs	Critical Path (# of Hops)	Number of Input Wires
Brute Force	128	55	5	132

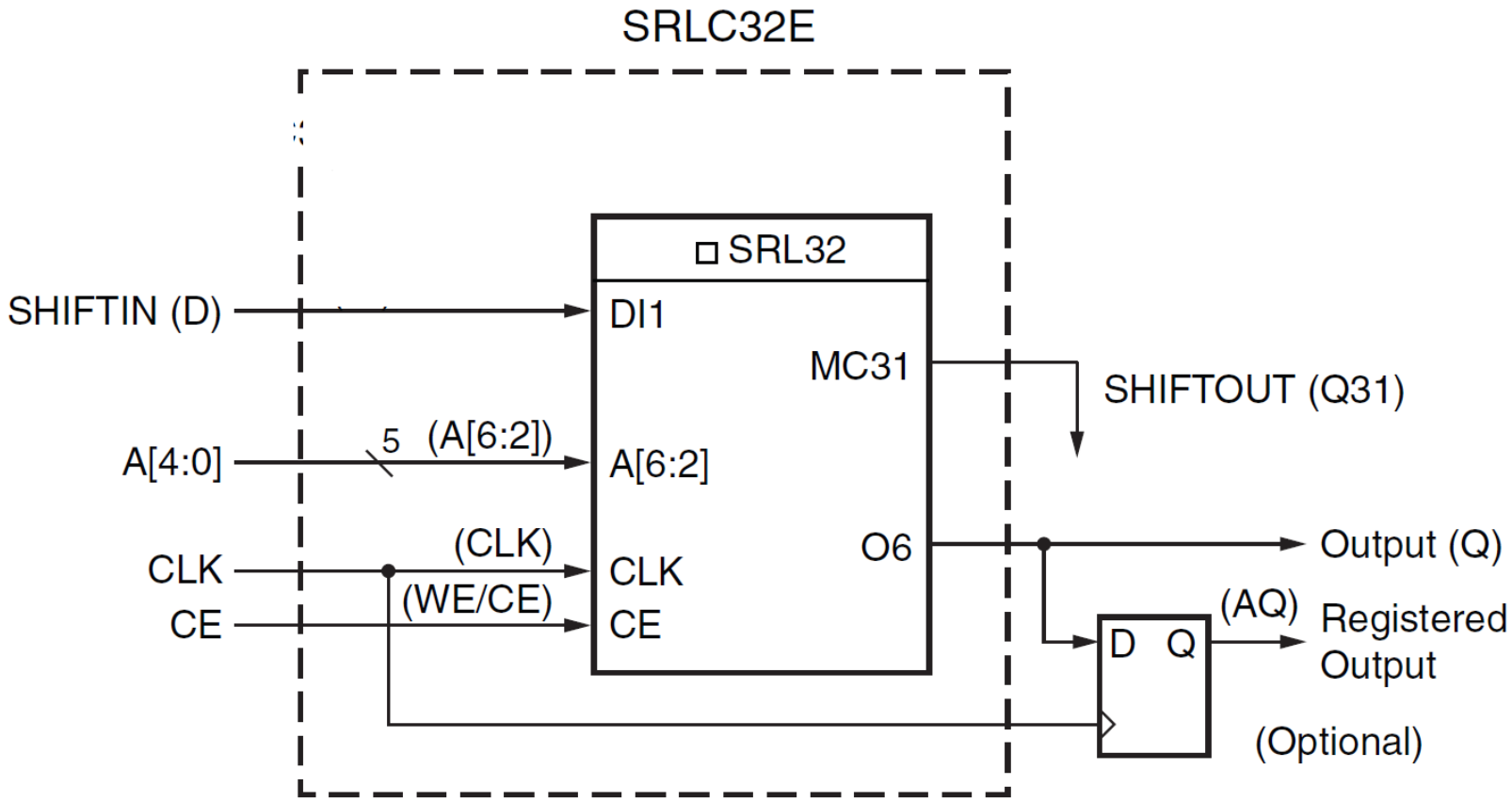
The Optimized Solutions

Dynamic Depth, Dynamic 4 Taps

- **Not all taps combinations are interesting**
- **At most, 4 taps are enough to achieve the longest sequence length**
- **For most LFSR depths, 4 taps can be used from the last 32 bits**
- **LUT's can be used as shift register**

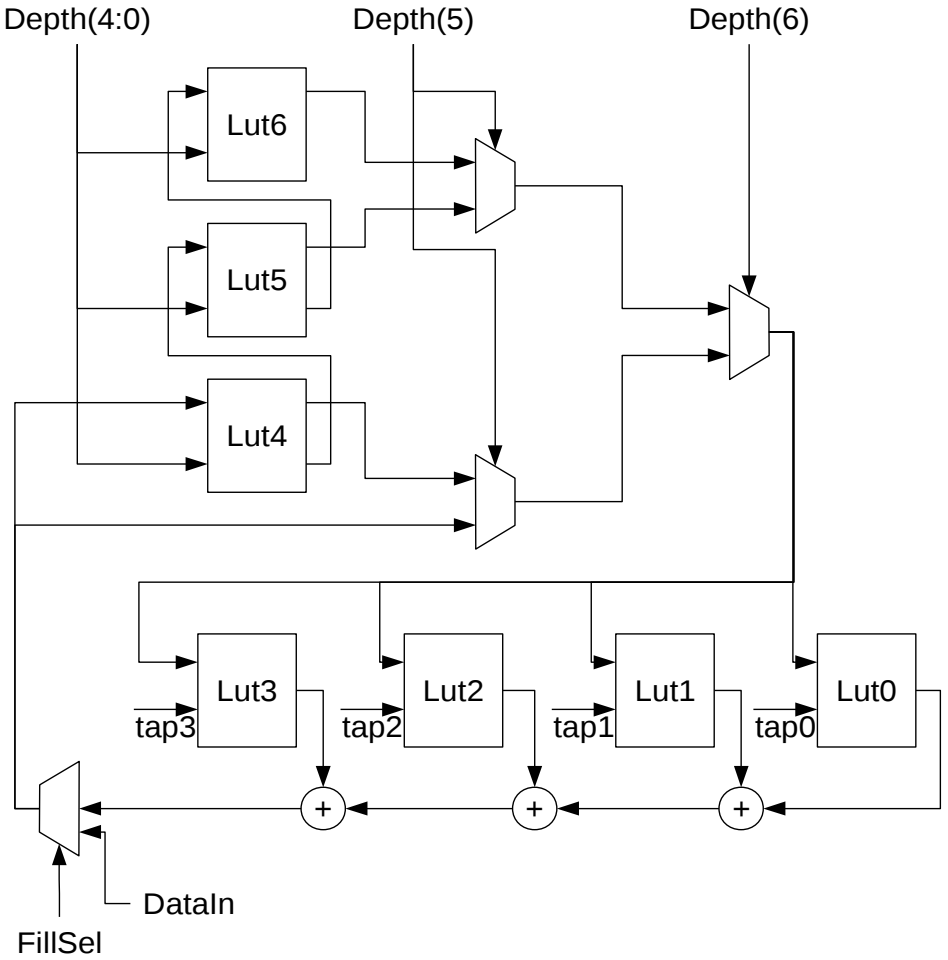
The Optimized Solutions

Dynamic Depth, Dynamic 4 Taps



The Optimized Solutions

Dynamic Depth, Dynamic 4 Taps



The Optimized Solutions

Dynamic Depth, Dynamic 4 Taps

- **The Good:**
  - **Dynamic Depth**
  - **Dynamic Taps**

The Optimized Solutions

Dynamic Depth, Dynamic 4 Taps

- **The Good:**
  - **Dynamic Depth**
  - **Dynamic Taps**

- **The Bad:**
  - **Still too many wires (29)**

The Optimized Solutions

Dynamic Depth, Dynamic 4 Taps

Solution	Number of FF's	Number of LUTs	Critical Path (# of Hops)	Number of Input Wires
Dynamic Depth with Dynamic 4 Taps	0	13	3	29

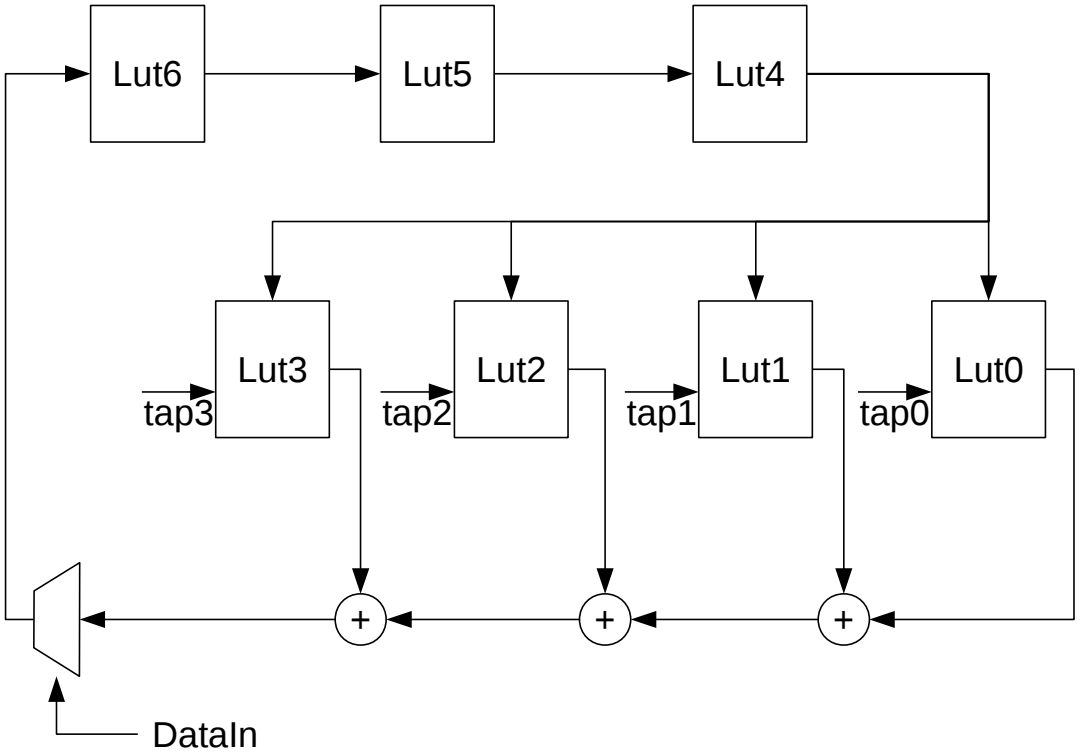
The Optimized Solutions

Fixed Depth, Dynamic 4 Taps

- **Do we really need a dynamic depth?**
- **Can we save if we fixed to 128?**

The Optimized Solutions

Fixed Depth, Dynamic 4 Taps



The Optimized Solutions

Fixed Depth, Dynamic 4 Taps

- **The Good:**
  - **Dynamic Taps**
  - **Less Wires**
  - **Less Resources**

The Optimized Solutions

Fixed Depth, Dynamic 4 Taps

- **The Good:**
  - **Dynamic Taps**
  - **Less Wires**
  - **Less Resources**
- **The Bad:**
  - **Too rigid**

The Optimized Solutions

Fixed Depth, Dynamic 4 Taps

Solution	Number of FF's	Number of LUTs	Critical Path (# of Hops)	Number of Input Wires
Fixed Depth with Dynamic 4 Taps	3	9	2	22

The Optimized Solutions

Static Variable Depth, Static 4 Taps

- **Hey, this is REconfigurable computing!**
- **Do we really need to write versatile code?**

The Optimized Solutions

Static Variable Depth, Static 4 Taps

```
constant Tap : TapPointArray := (7, 2, 1, 0);

process (Clk)
begin
    if rising_edge(Clk) then
        if ShiftEn = '1' then
            shift_reg <= lfsr_in & shift_reg(shift_reg'high downto 1);
        end if;
    end if;
end process;

par_fdbk(0) <= '0';
fdbk_gen: for X in 0 to Tap'high generate -- parity generator
    bitxor: par_fdbk(X+1) <= par_fdbk(X) xor shift_reg(Tap(X));
end generate fdbk_gen;

lfsr_in <= DataIn when FillSel = '1' else par_fdbk(par_fdbk'high);
```

The Optimized Solutions

Static Variable Depth, Static 4 Taps

- **The Good:**
  - **Back to simplicity**
  - **Efficient**

The Optimized Solutions

Static Variable Depth, Static 4 Taps

- **The Good:**
  - **Back to simplicity**
  - **Efficient**

- **The Bad:**
  - **Requires reconfiguration**

The Optimized Solutions

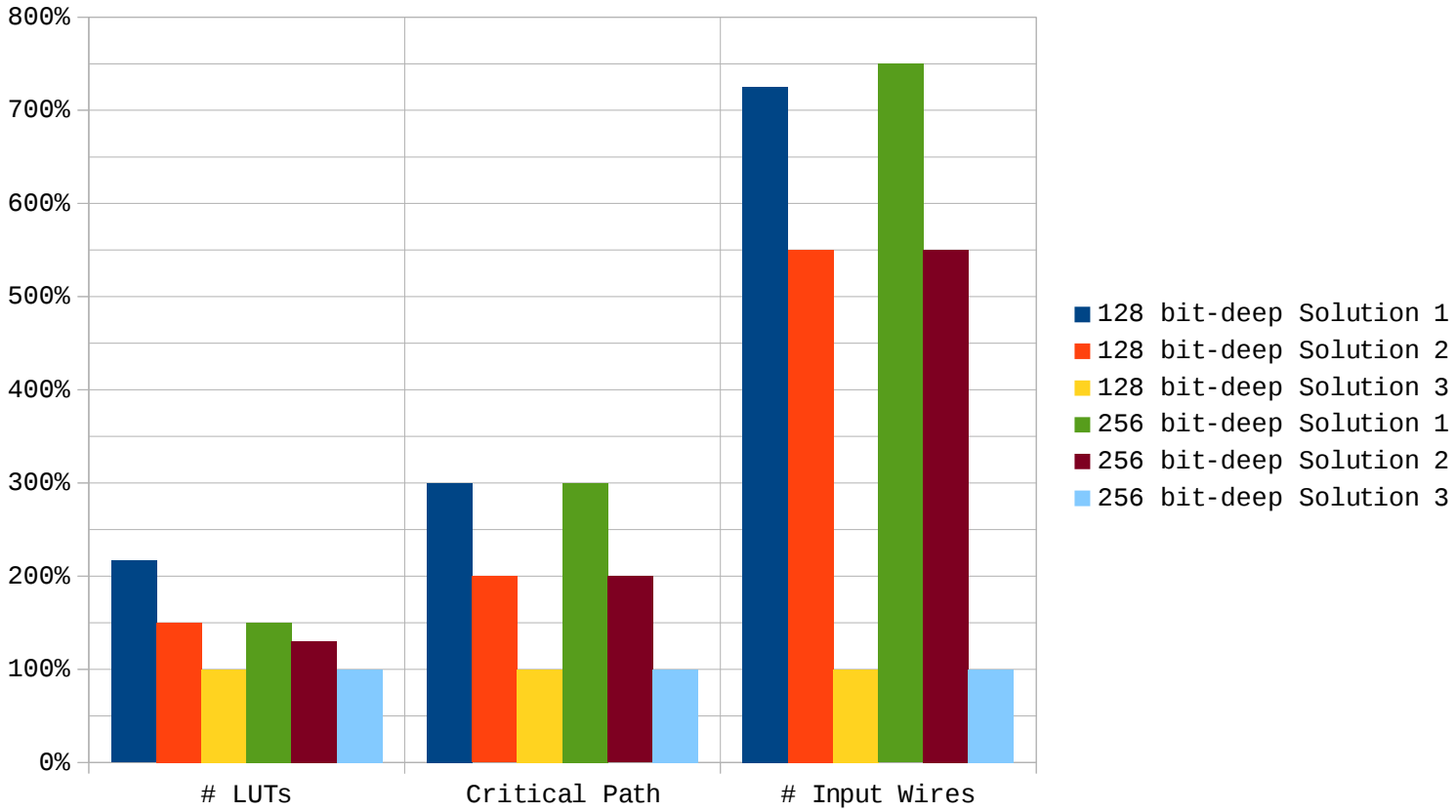
Static Variable Depth, Static 4 Taps

Solution	Number of FF's	Number of LUTs	Critical Path (# of Hops)	Number of Input Wires
Static Variable Depth with Static 4 Taps	5	6	1	4

Discussion

Solution	Number of FF's	Number of LUTs	Critical Path (# of Hops)	Number of Input Wires
Brute Force	128	55	5	132
Dynamic Depth with Dynamic 4 Taps	0	13	3	29
Fixed Depth with Dynamic 4 Taps	3	9	2	22
Static Variable Depth with Static 4 Taps	5	6	1	4

Discussion


800%
700%
600%
500%
400%
300%
200%
100%
0%
LUTs
Critical Path
Input Wires
128 bit-deep Solution 1
128 bit-deep Solution 2
128 bit-deep Solution 3
256 bit-deep Solution 1
256 bit-deep Solution 2
256 bit-deep Solution 3

Thank You

Questions? Comments?