NPTEL ONLINE CERTIFICATION COURSES

Course Name: Hardware Security

Faculty Name: Prof Debdeep Mukhopadhyay

Department : Computer Science and Engineering

Topic

Lecture 03: Algorithm to Hardware

CONCEPTS COVERED

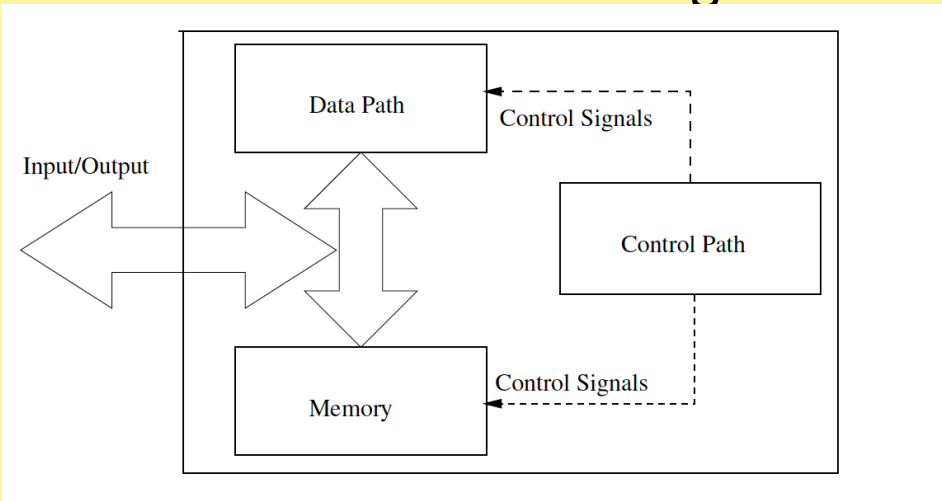
Concepts Covered:

- Data-Path & Control-Path
- Identification of Data-Path elements
- Control Path Design
- Data-Path and Control-Path design of gcd processor
- Performance modeling on FPGAs

Mapping an Algorithm to Hardware

- Conversion of an algorithm to an efficient hardware is a challenging task: Performance is a key-decider.
- For an efficient design, one needs to:
  - Understand the components of a hardware design
  - Understand the architecture of the design

Data-path elements are the computational units of the design



Control-path elements sequence the data flow through the data-path elements.

Case Study: Binary gcd processor

- Input: Integers u and v
- Output: Greatest Common Divisor of u and v, z=gcd(u,v)
- while (u!=v) do
  - If (u and v are even)
  - z=2gcd(u/2,v/2)
  - else if (u is even and v is odd)
  - z=gcd(u/2,v)
  - else if(u is odd and v is even)
  - z=gcd(u,v/2)

```
else
    if(u ≥ v)
        z=gcd((u-v)/2,v)
    else
        z=gcd(u,(v-u)/2)
```

We need to realize a co-processor on FPGA to compute gcd of two given numbers

References: Hardware Security: Design, Threats, and Safeguards, CRC Press.

Identification of the States of the Algorithm

- Input: Integers u and v
- Output: Greatest Common Divisor of u and v, z=gcd(u,v)

```
register u and v
XR=u, YR=v, count=0                          State 0
while (XR!=YR) do
    If (!XR[0] and !YR[0])                   State 1
        XR=RIGHT_SHIFT(XR)
        YR=RIGHT_SHIFT(YR)
        Count=Count+1
else if(XR[0] and !YR[0])                    State 2
    YR=RIGHT_SHIFT(YR)
else if(!XR[0] and YR[0])                    State 3
    XR=RIGHT_SHIFT(XR)
else                                         State 4
 if(XR ≥YR)
    XR=RIGHT_SHIFT(XR-YR)
else
    YR=RIGHT_SHIFT(YR-XR)
while(count > 0)                             State 5
    XR=LEFT_SHIFT(XR)
    count=count-1
```

References: Hardware Security: Design, Threats, and Safeguards, CRC Press.

Identification of the Data Path Elements

- Subtractor
- Complementer
- Right Shifter
- Left Shifter
- Counter
- Multiplexer:
  - Required in large numbers for the switching necessary for the computations done in the datapath.
  - Selection lines in the multiplexer are configured by the control circuitry, which is essentially a state machine.

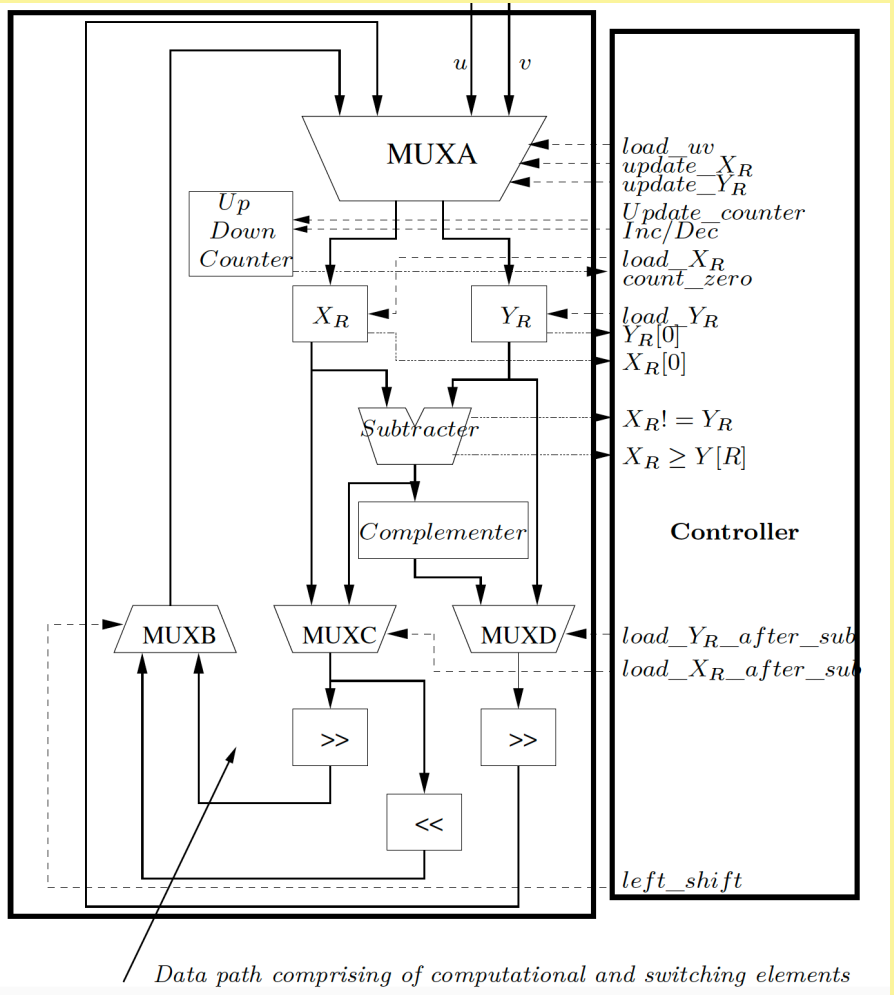
References: Hardware Security: Design, Threats, and Safeguards, CRC Press.

Identification of the State Machine of the Control Path

- Control Path is a sequential design.
- It can be represented by a state machine.
- In this example, there are 6 states.
- The controller receives inputs from the partial computations of the datapath.
- Based on the current state and input, it performs state transitions.
- It also produces control signals which configures the datapath elements or switches the multiplexers to sequence the dataflow.

References: Hardware Security: Design, Threats, and Safeguards, CRC Press.

Data Path Architecture

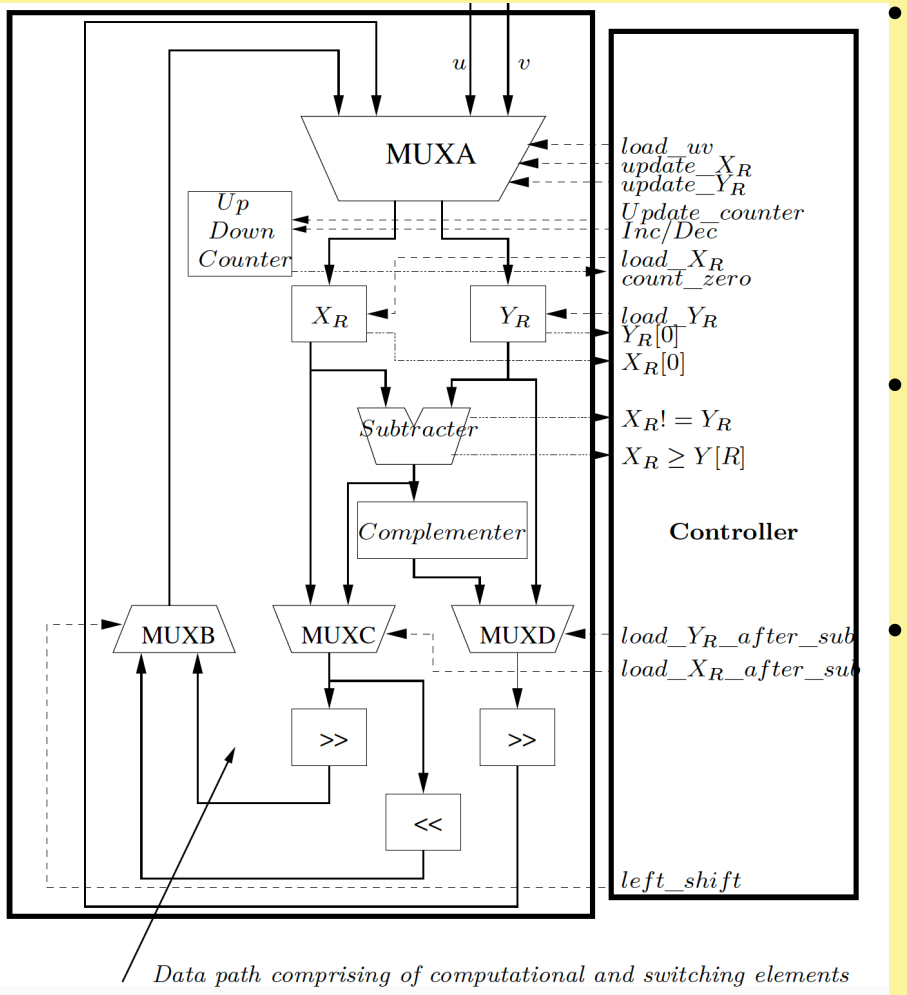


Data path comprising of computational and switching elements

References: Hardware Security: Design, Threats, and Safeguards, CRC Press.

- The data-path stores the value of XR and YR in two registers.
- Registers are loadable, ie. XR is updated when say *load_XR* is high.
- Values of u and v are initially loaded through the input multiplexer, using the control signal *load_uv*.
- Least bits of XR and YR are passed to the controller to indicate whether present values of XR and YR are even or not.
- Next iteration values of XR and YR are fed back after needed computations, and this is controlled by the signal *update_XR* and *update_YR*.

Data Path Architecture



Data path comprising of computational and switching elements

- Computations on XR and YR are:
  - Division by 2, which is done by two Right Shifters.
  - Subtraction, Equality Check (*XR!=YR*), Comparision (*XR≥YR*), all of which is done by a Subtractor.
  - In case, when *XR<YR*, subtraction YR-XR is to be performed, which is obtained by a Complementer.
- Next iteration values of XR and YR are loaded, either directly or after subtraction, which is controlled by the signal *load_XR_after_sub* and *load_YR_after_sub*
- Circuit also has an updown counter, which increments when both XR and YR are even.
  - Finally, when XR=YR, the result is obtained by computing 2^{count}(XR), which is done by a Left Left Shifter, until count becomes 0.

State Machine of the Controller

Present State	Next State					Output Signals										
	0____	100_	110_	101_	111_	*load uv*	*update* X_R	*update* Y_R	*load* X_R	*load* Y_R	*load*_X_R *after*_*sub*	*load*_Y_R *after*_*sub*	*Update counter*	*Inc /Dec*	*left shift*	*count zero*
S_0	S_5	S_1	S_2	S_3	S_4	1	0	0	1	1	0	0	0	_	_	_
S_1	S_5	S_1	S_2	S_3	S_4	0	1	1	1	1	0	0	1	1	_	_
S_2	S_5	S_1	S_2	S_3	S_4	0	0	1	0	1	0	0	0	_	_	_
S_3	S_5	S_1	S_2	S_3	S_4	0	1	0	1	0	0	0	0	_	_	_
S_4 $(X_R \geq Y_R)$	S_5	S_1	S_2	S_3	S_4	0	1	0	1	0	1	0	0	_	_	_
S_4 $(X_R < Y_R)$	S_5	S_1	S_2	S_3	S_4	0	0	1	0	1	0	1	0	_	_	_
S_5	S_5	S_5	S_5	S_5	S_5	0	0	0	0	0	0	0	1	0	1	0

There are 6 States of the Controller.

Controller receives 4 inputs from the data-path: [(XR!=YR), XR[0], YR[0], *XR≥YR*]

Example:

Present State: S0

load_uv=1, load_XR=load_YR=1.

Input=(0xxx)=>XR=YR=>Next State is S5.

Input=(100x)=>XR!=YR, both XR and YR are even=>Next State is S1.

State Machine of the Controller

Present State	Next State					Output Signals										
	0____	100_	110_	101_	111_	*load uv*	*update* X_R	*update* Y_R	*load* X_R	*load* Y_R	*load*_X_R *after*_*sub*	*load*_Y_R *after*_*sub*	*Update counter*	*Inc /Dec*	*left shift*	*count zero*
S_0	S_5	S_1	S_2	S_3	S_4	1	0	0	1	1	0	0	0	_	_	_
S_1	S_5	S_1	S_2	S_3	S_4	0	1	1	1	1	0	0	1	1	_	_
S_2	S_5	S_1	S_2	S_3	S_4	0	0	1	0	1	0	0	0	_	_	_
S_3	S_5	S_1	S_2	S_3	S_4	0	1	0	1	0	0	0	0	_	_	_
S_4 $(X_R \geq Y_R)$	S_5	S_1	S_2	S_3	S_4	0	1	0	1	0	1	0	0	_	_	_
S_4 $(X_R < Y_R)$	S_5	S_1	S_2	S_3	S_4	0	0	1	0	1	0	1	0	_	_	_
S_5	S_5	S_5	S_5	S_5	S_5	0	0	0	0	0	0	0	1	0	1	0

Present State: S1

load_uv=0, load_XR=load_YR=1, update_XR=update_YR=1, Update Counter=1, Inc/Dec=1.

Input=(0xxx)=>XR=YR=>Next State is S5.

Input=(100x)=>XR!=YR, both XR and YR are even=>Next State is S1.

Input=(110x)=>XR!=YR, XR is odd, YR is even=>Next State is S2.

Performance Evaluation of the Design

- One of the primary goals of developing a hardware design is performance.
- But performance is context sensitive.
- We revise some general definitions:
  - In a combinational circuit, the critical path delay is vital.
  - Synthesis tool assumes that all combinational paths in the design are to be performed in a single clock period.
  - Critical path is the maximum delay of a combinational path between two registers.
  - Critical path gives an upper bound to the clock frequency, f_{clk}, say f_{max}

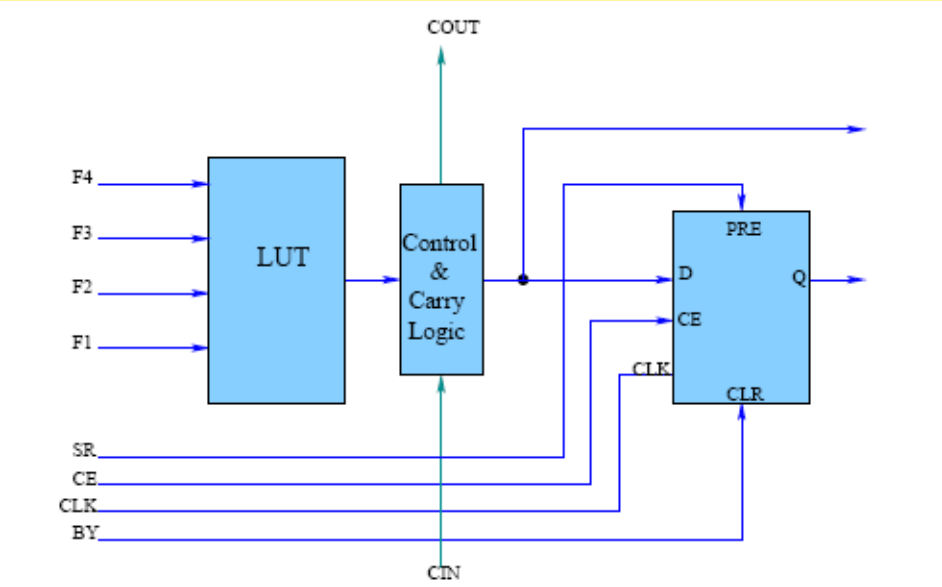
Performance Evaluation of the Design (Contd.)

- For a sequential circuit, like the gcd processor, number of clock cycles is also important.
- In the gcd processor, the number of clock cycles vary with the input, and is proportional to the bit length of the bigger argument.
- On an average, if the cycles required are cc_{avg}, then the total computation time is $t_c = cc_{avg}/f_{max}$
- If the number of bytes of data being simultaneously processed is Nb, then the throughput of the hardware, is $T = Nb/t_c = Nb(f_{max}/cc_{avg})$.

Resource Consumption

The other important aspect is resource consumed.

In context to FPGAs, the resources largely comprise of slices, which are made of LUTs, and flip-flops.



- **LUT Structure of Xilinx Virtex-4**
  - Four Input , One Output.
  - Can contain 16x1 SRAM.
  - Can implement any four input truth table.

LUT Utilization

Compact designs will be realized when the LUTs are utilized.

$y_1 = x_1 \oplus x_2 \oplus x_3 \oplus x_4$ requires 1 LUT

$y_2 = x_1 \oplus x_2$ requires 1 LUT.

Thus, $y_2$results in an under utilized LUT.

Design goal is to reduce the number of under utilized LUTs

LUT Under Utilization

Minimum number of LUTs required for a q-bit combinational circuit (for 4 input LUT)

$$\#LUT(q) = \begin{cases} 0, \text{ if } q = 1 \\ 1, \text{ if } 1 < q \le 4 \\ \lceil q/3 \rceil, \text{if q>4 and } q \bmod 3 = 2 \\ \lfloor q/3 \rfloor, \text{if q>4 and } q \bmod 3 \ne 2 \end{cases}$$

Delay of a q-bit combinational circuit.

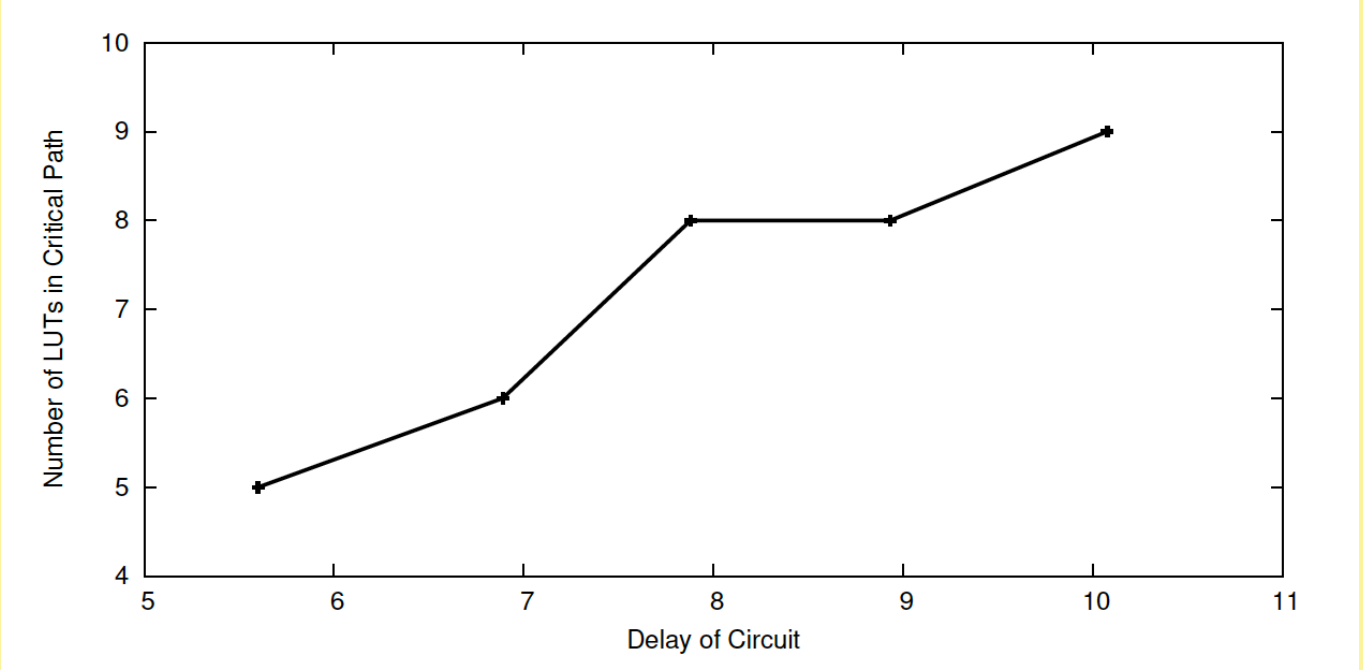
$$DELAY(q) = \lceil \log_4(q) \rceil D_{LUT}$$

LUT_k denotes that k inputs out of 4 are used by the design block realized by the LUT.

$$\%UnderUtilizedLUTs = \frac{LUT_2 + LUT_3}{LUT_2 + LUT_3 + LUT_4} * 100$$

LUTs in Critical Path to Measure Delay

Delay in FPGAs comprise of both LUT and routing delay, and thus is more complex to analyze.
Through experiments one can however see that for combinational circuits, the delay varies linearly with the number of LUTs in the critical path.



Linear relationship between number of LUTs in Critical Path and Delay of a Combinational Multiplier of increasing dimensions.

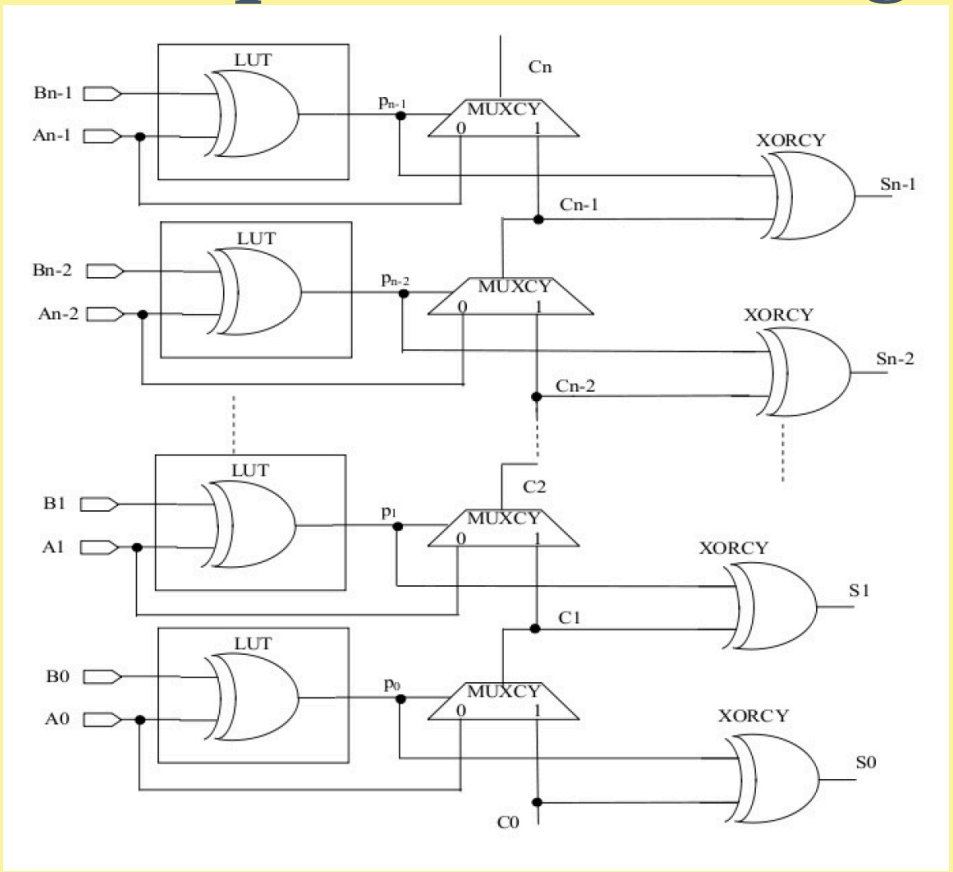
References: Hardware Security: Design, Threats, and Safeguards, CRC Press.

Modeling the Components of the gcd processor

Most important component in the data-path of the gcd processor is subtractor, which can be realized by an adder.

On FPGA platforms, carry chain based adder are specially optimized and are fast.

For Xilinx Virtex IV FPGAs, s is almost 17.



The carry chains use m-MUXCY for m-bit adder.
They are much faster compared to the LUTs which are used for other parts of the circuit.
So, in order to compare we scale the delay, and state:

$$D_{add}(or\ D_{sub}) = \lceil m/s \rceil$$

References: Hardware Security: Design, Threats, and Safeguards, CRC Press.

Modeling the Multiplexer

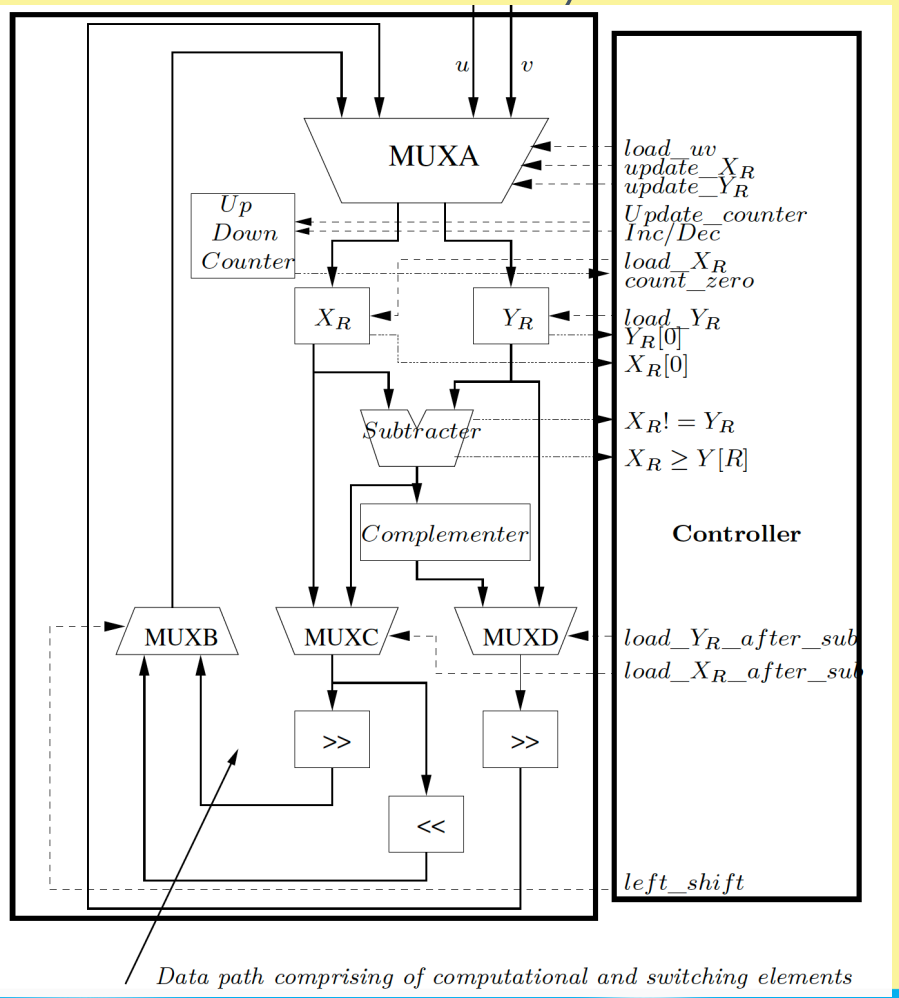
- 2^t:1 Multiplexer: Output is a Boolean function of (2^t+t) variables.
- So, for each bit output, it requires lut(2^t+t) LUTs.
- For, an m-bit gcd multiplier, hence no. of LUTs is m* lut(2^t+t)
- Delay: DMUX= $\lceil \log_4(2^t + t) \rceil$

Total LUT Estimate of the gcd processor

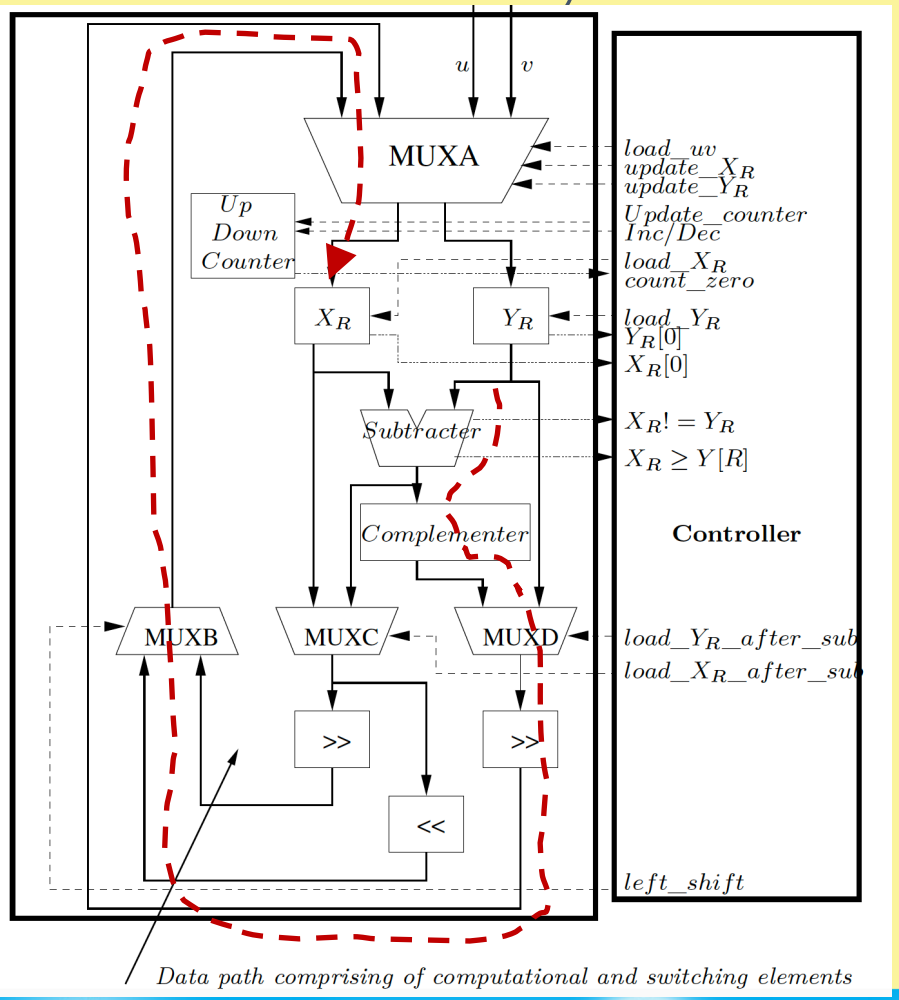
- Sum of the LUTs for MUXA, MUXB, MUXC, MUXD, and the subtractor along with the complementer (which is another subtractor).
- The state machine is assumed to consume very less LUTs and is ignored.
- Total number of 4-LUTs in the entire circuit:

$$\#\,LUT_{\text{gcd}} = 2\,LUT_{subtractor} + LUT_{MUXA} + LUT_{MUXB} + LUT_{MUXC} + LUT_{MUXD}$$

Total Delay Estimate of the gcd processor



Total Delay Estimate of the gcd processor



- Critical path of the design:

subtractor→complementer→MUXD→MUXB→ MUXA.

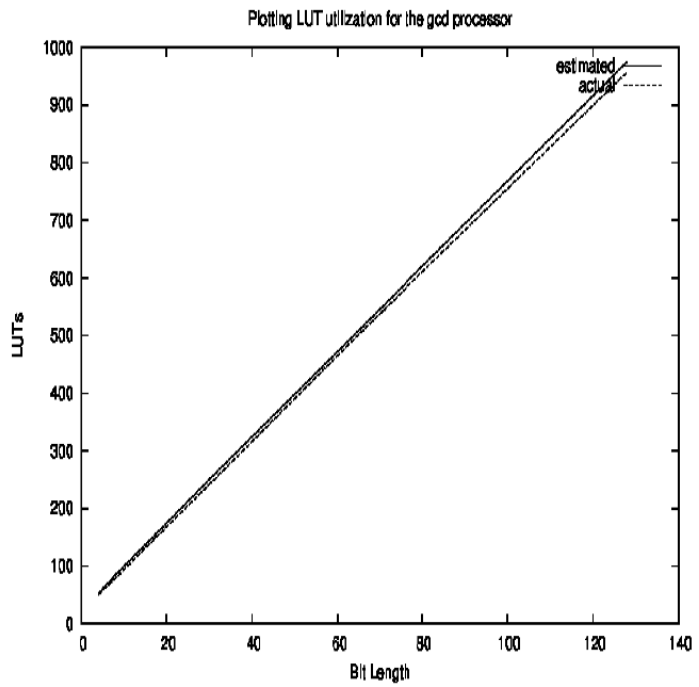
$$D_{\text{gcd}} = 2D_{sub} + D_{MUXD} + D_{MUXB} + D_{MUXA}$$
$$= 2\lceil m/s \rceil + 1 + 1 + 1 = 3 + 2\lceil m/s \rceil$$

Note that the delay of MUXA comes from the fact that the multiplexer is made of 2 smaller 2-input multiplexers in parallel: one input writing to XR, while the other writing to YR.

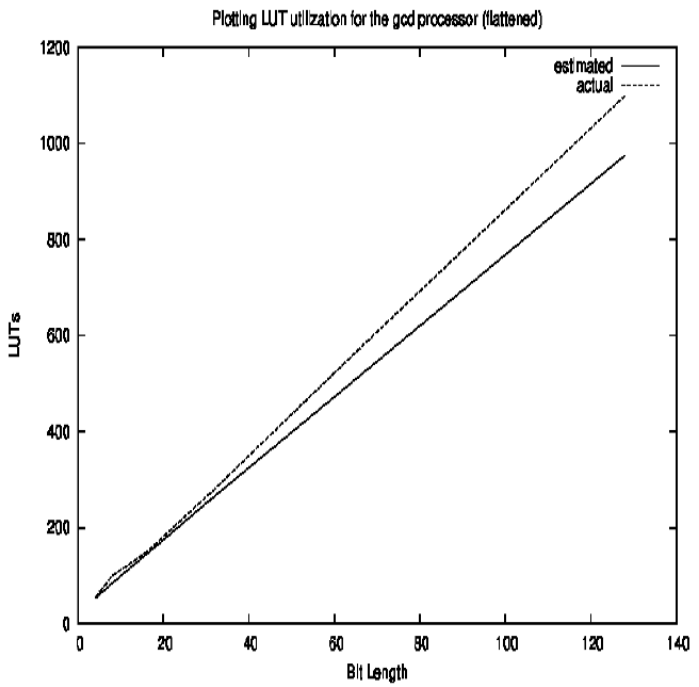
Experimental Exploration

- The above design was synthesized using Xilinx ISE tools and targeted on a Virtex-4 FPGA.
- Objective of the experiment was to study the above dependence on the LUT utilization and to estimate the critical path delay for the circuit.
- The objective of the exploration is to see the trend, and not the exact figures.
- We vary the bit length of the gcd processor, and repeat the experiments to study the scalability of the design.
- This helps in design exploration, as we are able to understand the dependence and tweak the design early in the design cycle.

LUT utilization

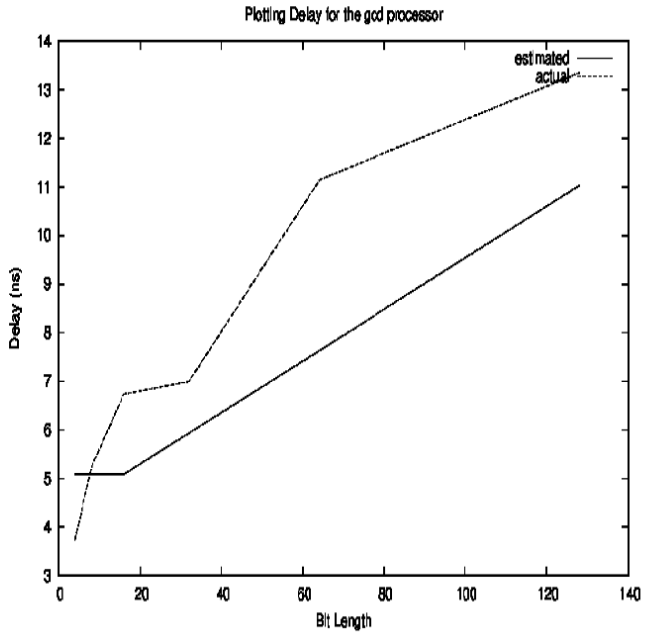


(a) LUT utilization of the gcd processor (hierarchy on)

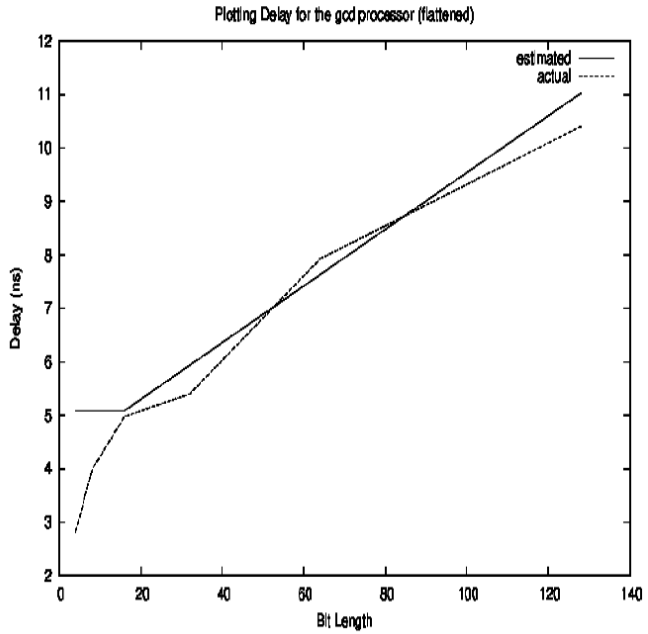


(b) LUT utilization of the gcd processor (hierarchy flattened)

Delay Estimation



(a) Critical Path Delay of the gcd processor (hierarchy on)

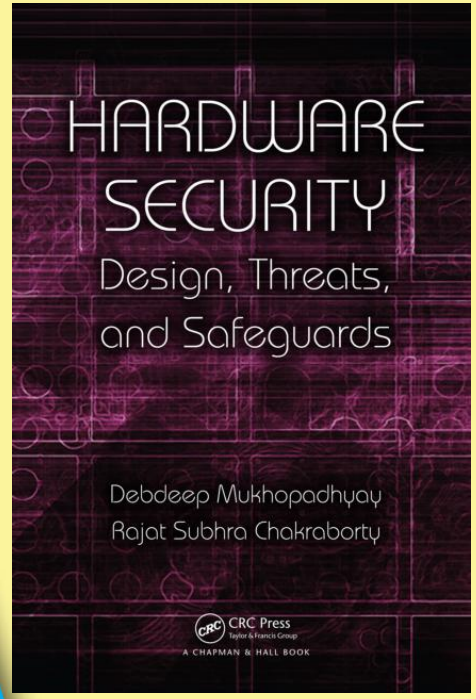


(b) Critical Path Delay of the gcd processor (hierarchy flattened)

References

References:

❑ Debdeep Mukhopadhyay and Rajat Subhra Chakraborty, Hardware Security: Design, Threats and Safeguards, CRC Press



Conclusion

Conclusion:

Proper division of data and control paths, is a key to design a hardware architecture for an algorithm.

Always start design with a nice architecture diagram: that is the 0th step of VLSI Design!

Performance modeling can be a useful tool to guide the exploration.

More than accurate estimations one can observe trends in the design which can be helpful to make early decisions

FREE ONLINE EDUCATION
swayam
शिक्षित भारत, उन्नत भारत

NPTEL ONLINE CERTIFICATION COURSES

Thank you!