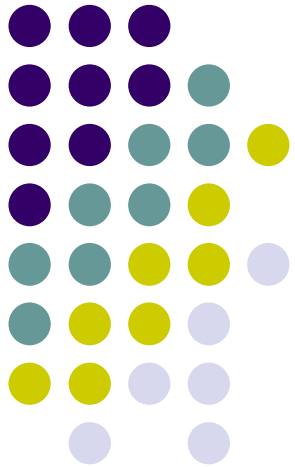
MLO:
A Multi-Level Optimizer For Burst Mode Asynchronous Controllers

Walter Dearing

Steven Nowick

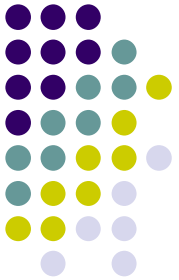
Computer Science Department

Columbia University



This work was supported by NSF ITR Award No. NSF-CCR-0086036 and
by an Initiatives in Science and Engineering (ISE) grant from
Columbia University (from the Office of the Executive Vice President for Research)

MLO: Access Information



- Accessible on the web from:

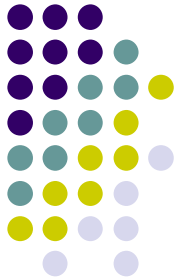
  http://www1.cs.columbia.edu/~nowick/asynctools

- Initial Release
  - One version – for Linux Distributions
- Includes
  - Complete Tutorial
  - Documentation
  - Examples
- Tool requires Python interpreter to run:

  http://www.python.org/download/

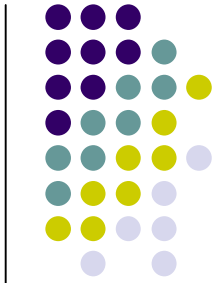
- Consult README for MLO installation information

Outline



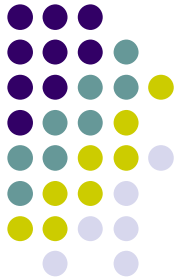
- Introduction to MLO (Multi-Level Optimizer)
- Overview of features
- Example of features
- Example of gate networks produced
- Built-in verifier
- Brief tutorial

Introduction to MLO



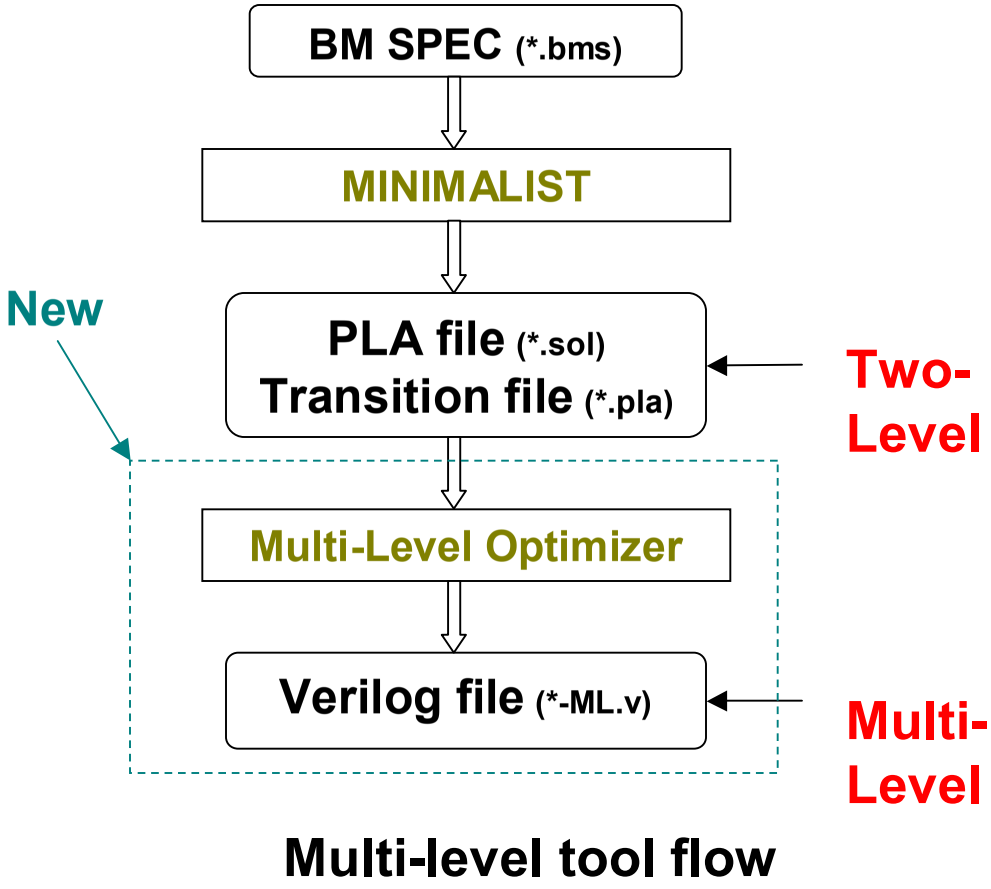
- **Introduction to MLO (Multi-Level Optimizer)** ←
- Overview of features
- Example of features
- Example of gate networks produced
- Built-in verifier
- Brief tutorial

Introduction to MLO



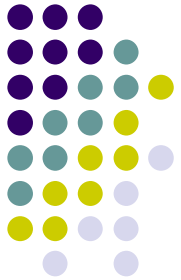
- MLO is an integrated **post-processing** (i.e. backend) tool for Minimalist.
- Targeted to **multi-level logic**.
  - In contrast, Minimalist currently is targeted to two-level logic.
- Designed to work on **combinational hazard-free logic** for Burst Mode controllers.
  - Uses “hazard-non-increasing” transforms.
- Output of MLO is Verilog.
- MLO is a standalone tool running from the Linux shell **outside of Minimalist.**

Introduction to MLO



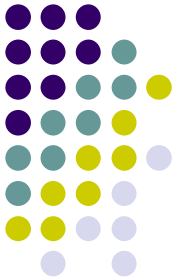
Multi-level tool flow

Overviews of features



- Introduction to MLO (Multi-Level Optimizer)
- **Overview of features** ←
- Example of features
- Example of gate networks produced
- Built-in verifier
- Brief tutorial

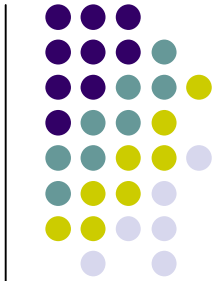
Feature Set – Overview



MLO has many features which are orthogonal and can be used together.

1. **Gate fan-in limitation**
2. **Negative-logic**
3. **Critical Event Optimizer (CEO)**
   - Two modes
     a) **Automated**
     b) **User-specified**

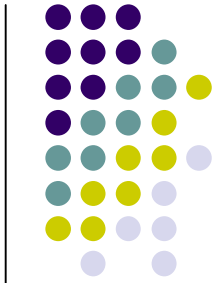
Feature Set – Goal of CEO



Goal of CEO is to reduce the **critical path** of **critical events**.

- **Critical events** are defined as a transition of a primary output in response to an input transition
- Done by moving critical inputs closer to the primary output.
- Intention is for critical inputs to travel through the circuit as fast a possible.

Feature Set – CEO Mode 1 - Automated Mode



CEO defaults to automated mode if no user input specified.

- Tool automatically optimizes **likely critical paths**.
- Focuses optimizations on **primary input-to-output paths** involved in **dynamic transitions**.
  - (i.e. 0→1 / 1→0)

Feature Set – CEO Mode 2 - User-Specified Critical Events

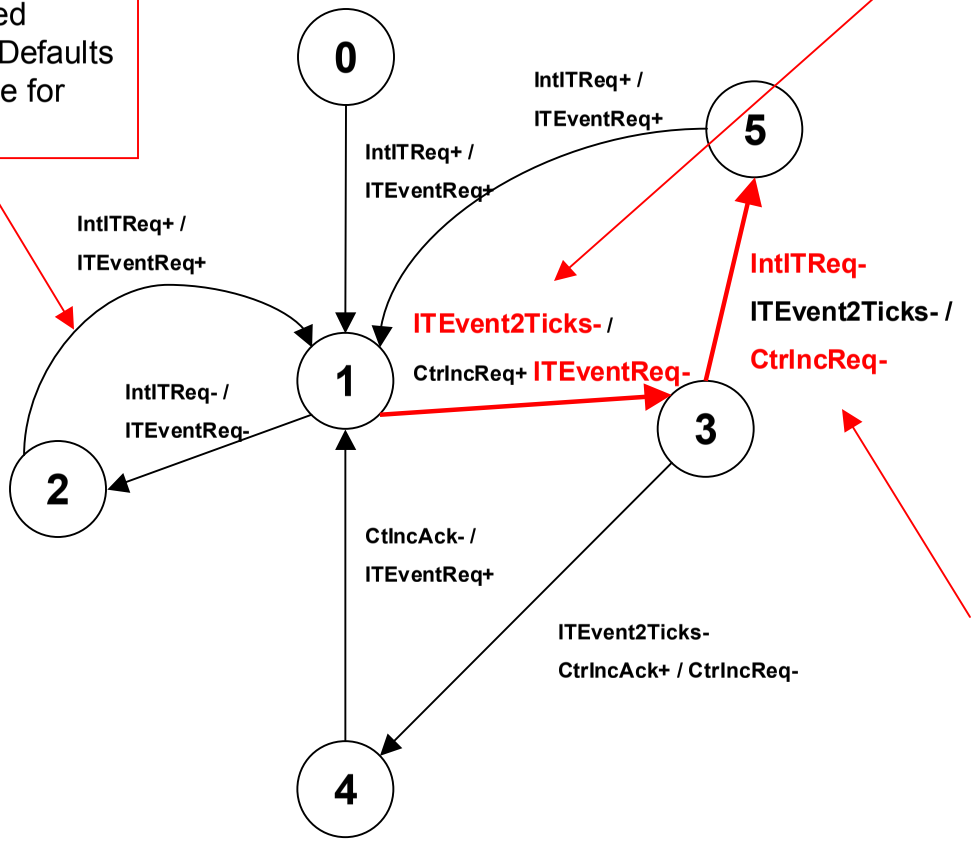
User can specify details for CEO to use.

- User may know which primary input(s) are important to critical primary output.
- Done by defining a **critical transition & input/output pair** at the Burst-Mode spec level.
- For particular outputs specified during transition, user **overrides** automated mode.
  - Consider the situation on the next slide where user specifies input & outputs off specific arcs.

Feature Set – CEO Mode 2 - User-Specified Critical Events

User-Specified Critical Arcs Highlighted in Red

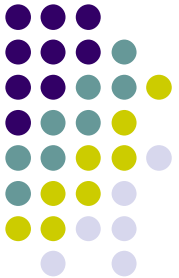
Case 1: Non colorized arc. User-Specified nothing is critical. Defaults to automated mode for every output.



Case 2: Some outputs colorized, some outputs not. Both user-specified data and automated approaches are used to determine criticality. **ITEventReq** will use user-specified data to determine criticality. **CtrincReq** will default to automated mode to determine criticality.

Case 3: Every output is colorized. Automated approach is never used. **IntITReq**- is critical with respect to **CtrIncReq-**, while **ITEvent2Ticks**- is NOT critical to **CtrIncReq-**.

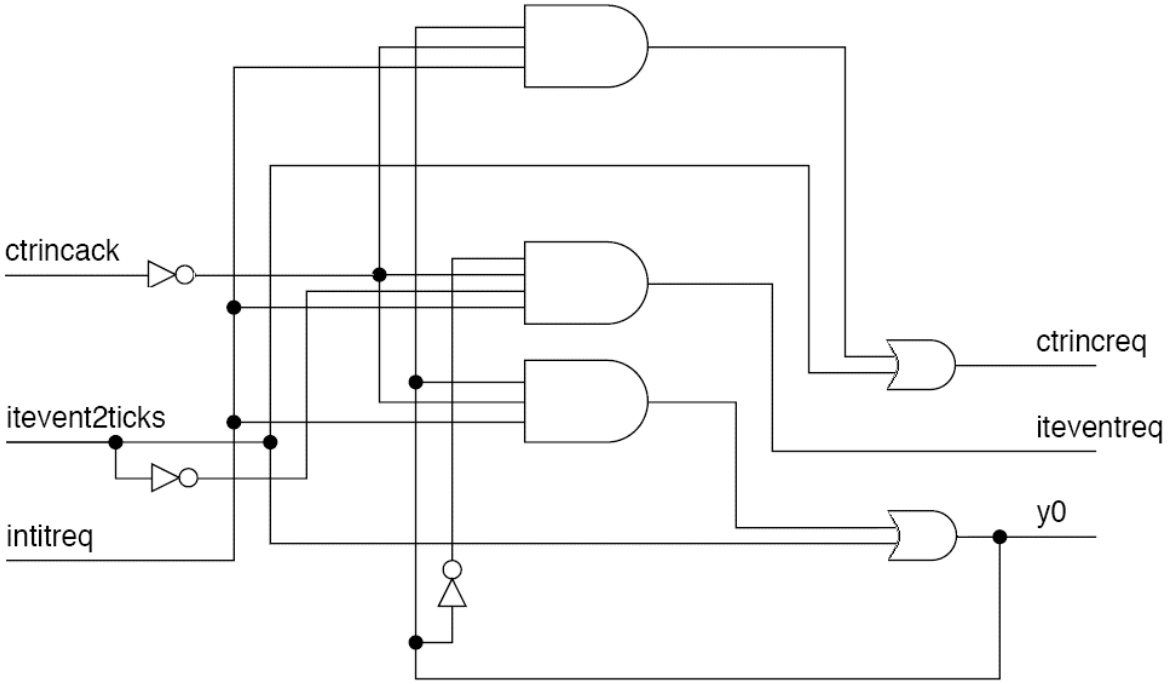
Example of features



- Introduction to MLO (Multi-Level Optimizer)
- Overview of features
- **Example of features**
- Example of gate networks produced
- Built-in verifier
- Brief tutorial

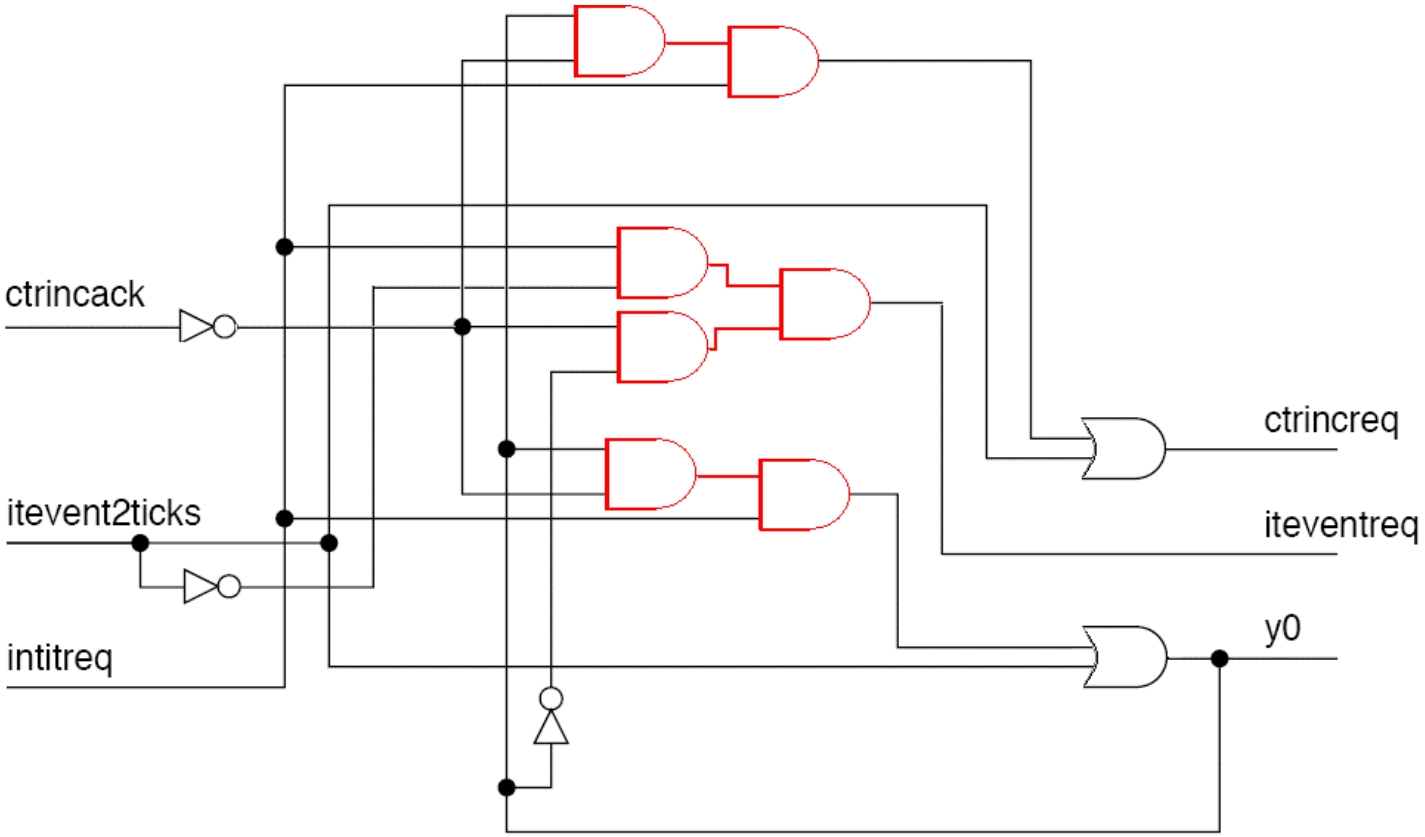
Feature Set - Initial Two-Level Implementation (before applying MLO)

The next four slides present different MLO output examples. For each example, the starting circuit (input to MLO) is this circuit



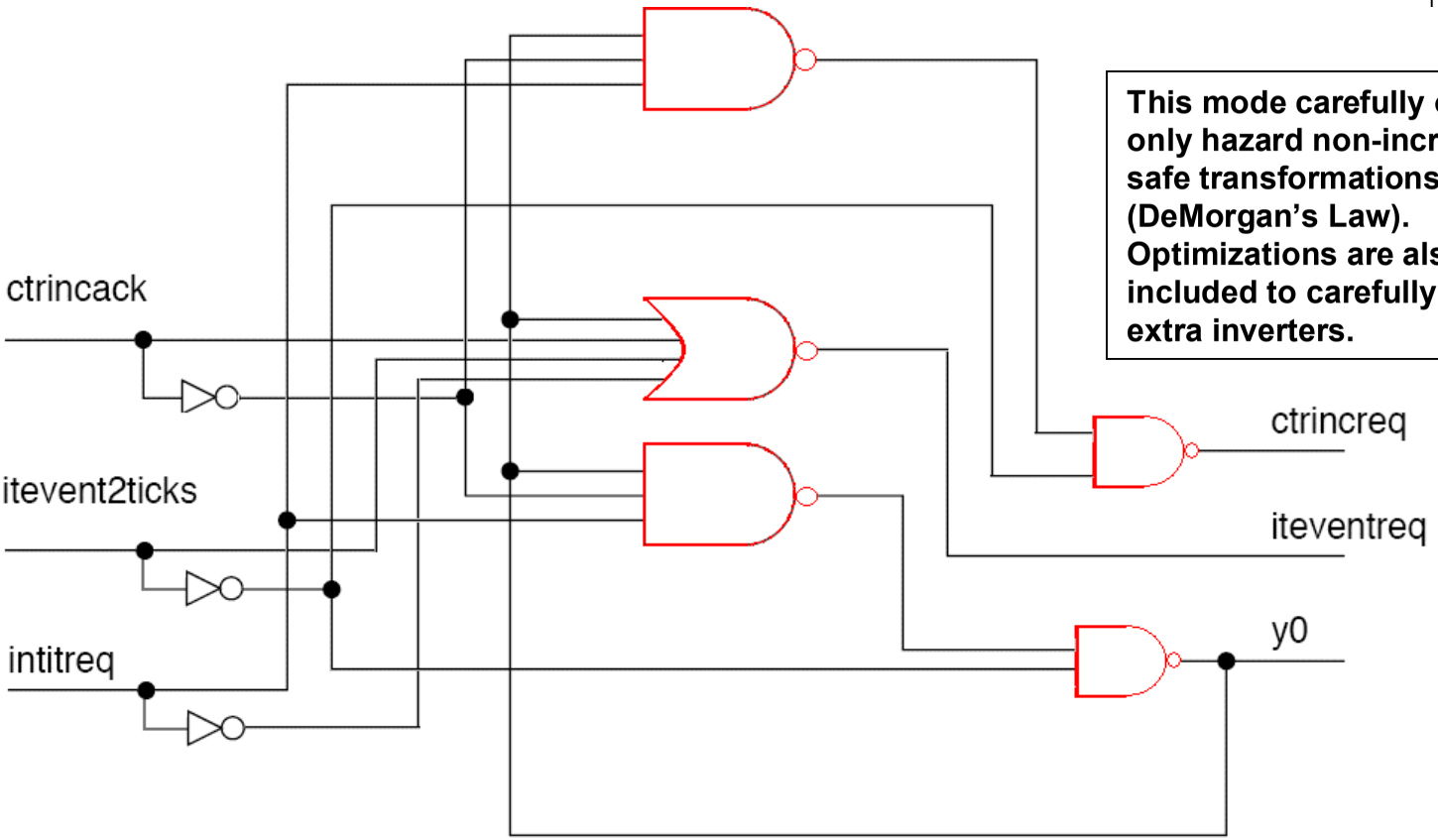
Two-level Structure from Minimalist Output

Feature Set Example 1 - Gate Fan-in Limitation



Result of MLO: Multi-Level circuit with AND gate fan-in limit of 2

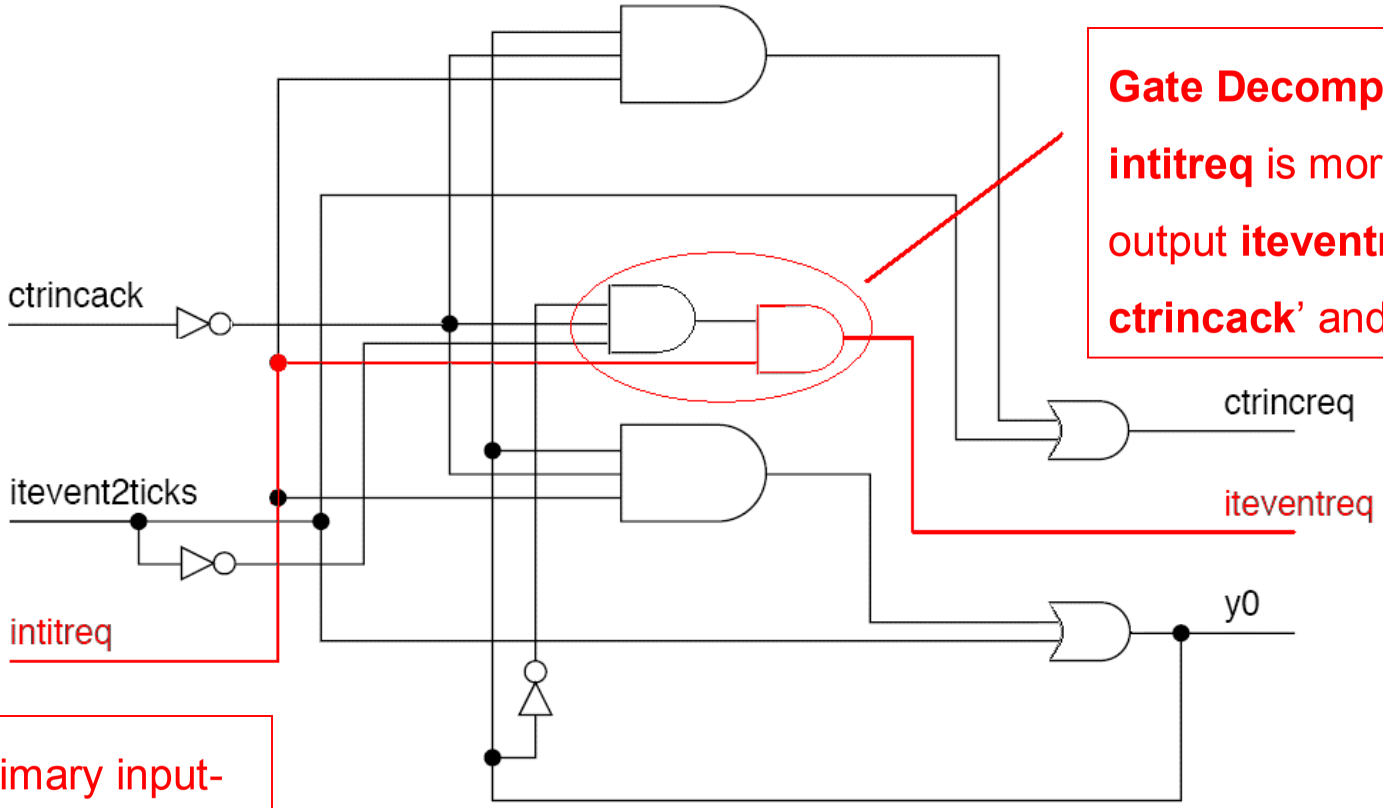
Feature Set Example 2 - Negative Logic



This mode carefully optimizes only hazard non-increasing safe transformations (DeMorgan's Law). Optimizations are also included to carefully eliminate extra inverters.

Result of MLO: Multi-Level Circuit using MLO Negative Logic

Feature Set Example 3 - CEO

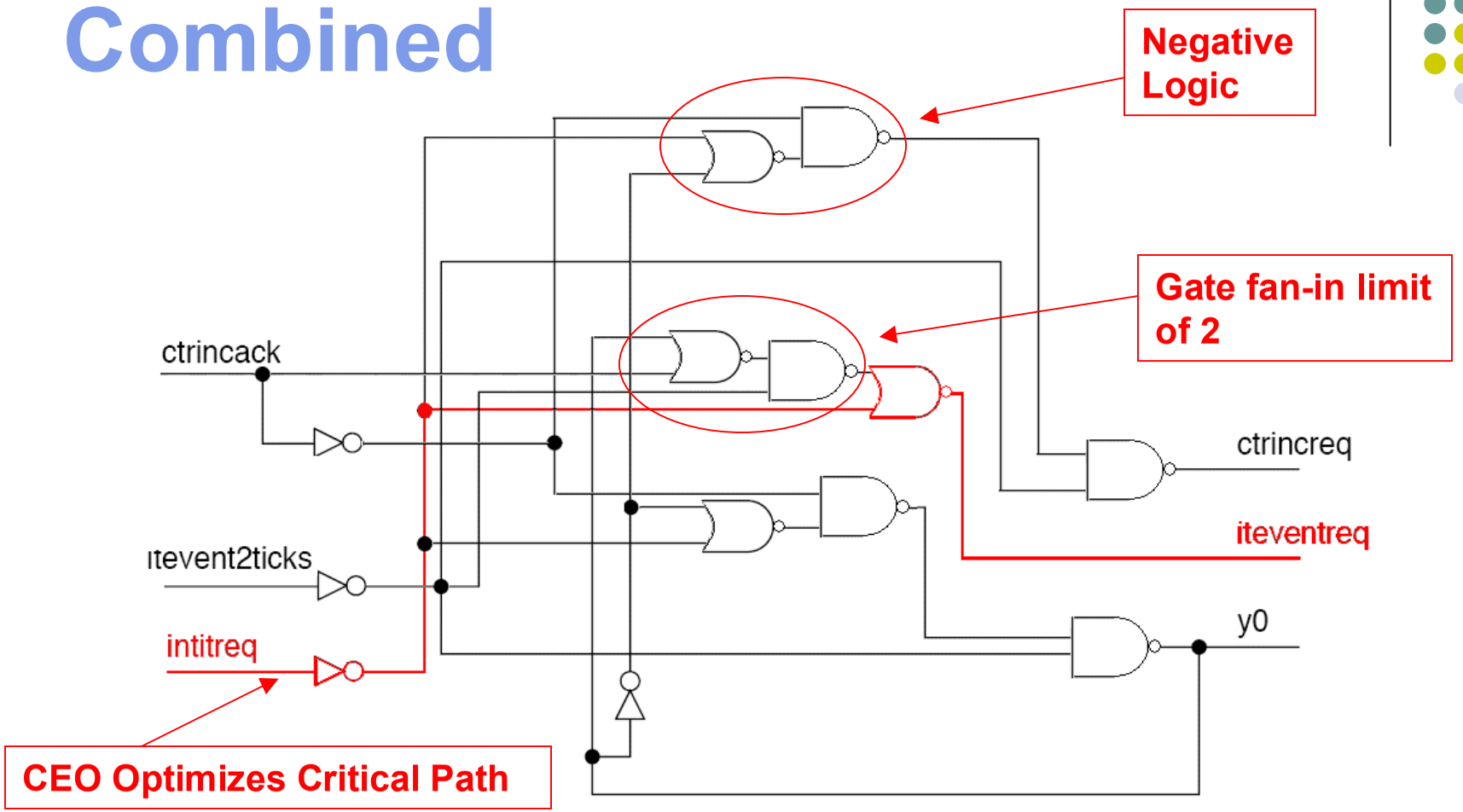


Gate Decomposed. Input **intitreq** is more critical to output **iteventreq** than **ctrincack**' and **y0**'

critical primary input-to-output path

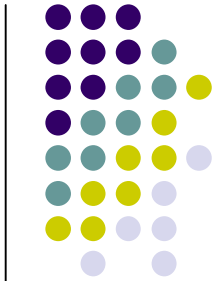
Result of MLO: Multi-Level Circuit after MLO CEO is used

Feature Set Example 4 - Combined



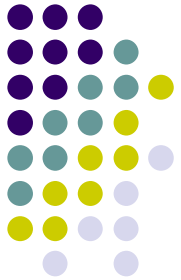
Result of MLO: Multi-Level Circuit with negative logic, AND gate fan-in limit of 2, and CEO.

Example of gate networks produced



- Introduction to MLO (Multi-Level Optimizer)
- Overview of features
- Example of features
- **Example of gate networks produced** ←
- Built-in verifier
- Brief tutorial

Gate Decomposition



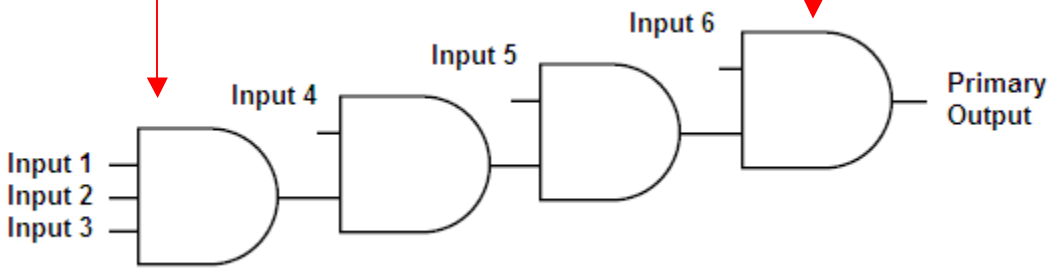
- Gates can be decomposed in two ways
  - CEO – resulting network will be cascaded
  - Fan-in limitations – resulting network will be balanced
- Network can be a mixture of two methods.
- Consider a 6 input AND gate….

Gate Decomposition - Cascaded

Gate is decomposed based on **priority difference of inputs**

Lowest Input Priority
(state variables only)

Highest Input Priority

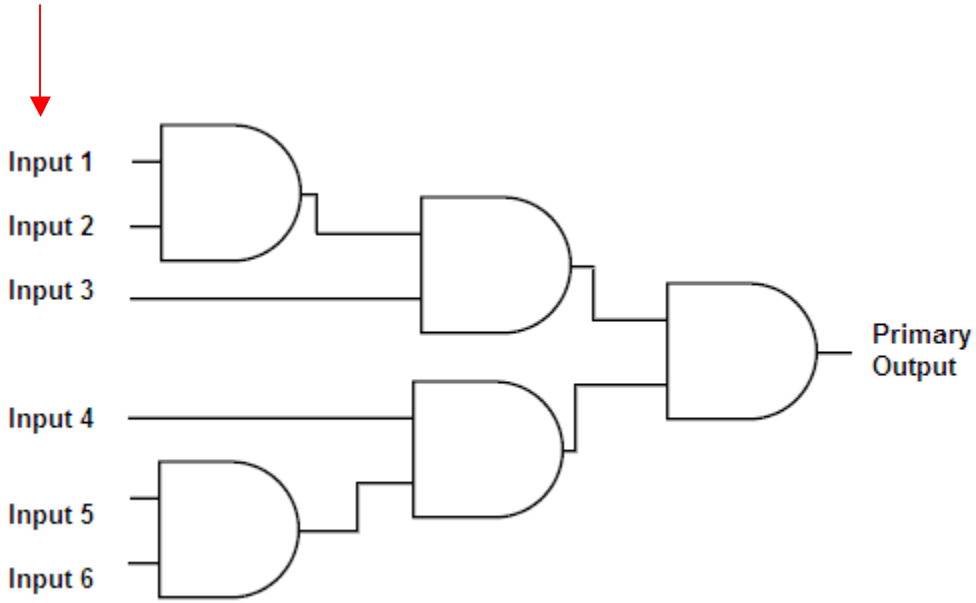


Only used for CEO

Gate Decomposition – Balanced

Gate is decomposed strictly based on **fan-in limitation**

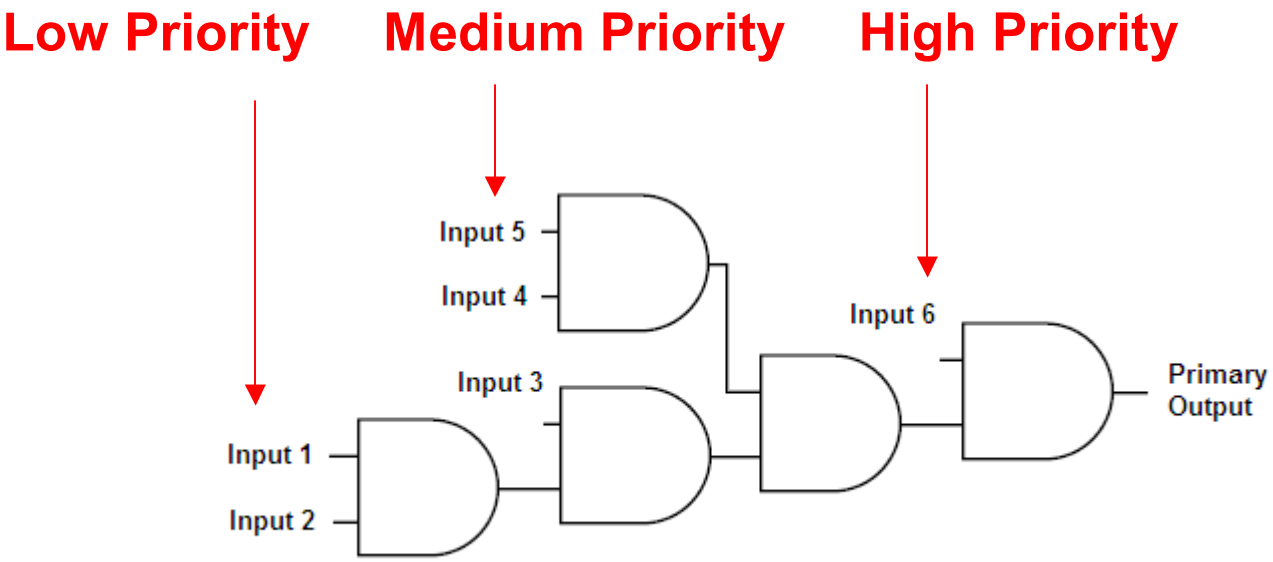
All Inputs have the same priority



Only used for gate fan-in limitation

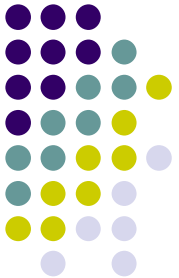
Gate Decomposition – Mix

Gate is decomposed based on **both** difference in priority of inputs and fan-in limits



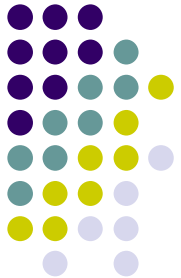
Combination of gate fan-in limitation and CEO

Brief tutorial



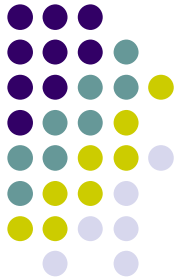
- Introduction to MLO (Multi-Level Optimizer)
- Overview of features
- Example of features
- Example of gate networks produced
- **Built-in verifier**
- Brief tutorial

Built-in Verifier



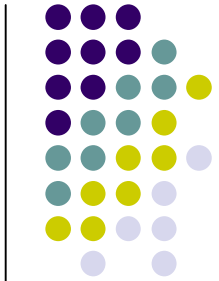
- A verifier is built into MLO.
- Verifies MLO output is **functionally correct** and **hazard free**
  - Compares two-level structure (Minimalist output) to multi-level structure (MLO output).
  - Verifies specific properties hold for each gate network in multi-level structure.
- Must be explicitly set using command line flag (not run by default)

Brief tutorial



- Introduction to MLO (Multi-Level Optimizer)
- Overview of features
- Example of features
- Example of gate networks produced
- Built-in verifier
- **Brief tutorial**

Tutorial



- The spec file hp-ir.bms will be used during this tutorial. This can be found in the examples folder.
- Every command used in the tutorial is also described in the **tool help menu.**
  - To display general help menu

    ```
    > MLO.py --help
    ```

  - To display user-specified critical event help menu

    ```
    > MLO.py --more_help
    ```

Tutorial - Setup

Create the files needed for MLO to process

Step 1 – Create working directory

```
> mkdir MLO_tutorial
```

Step 2 – Copy spec file

```
> cp examples/hp-ir.bms
MLO_tutorial/
```

Step 3 – Enter Directory

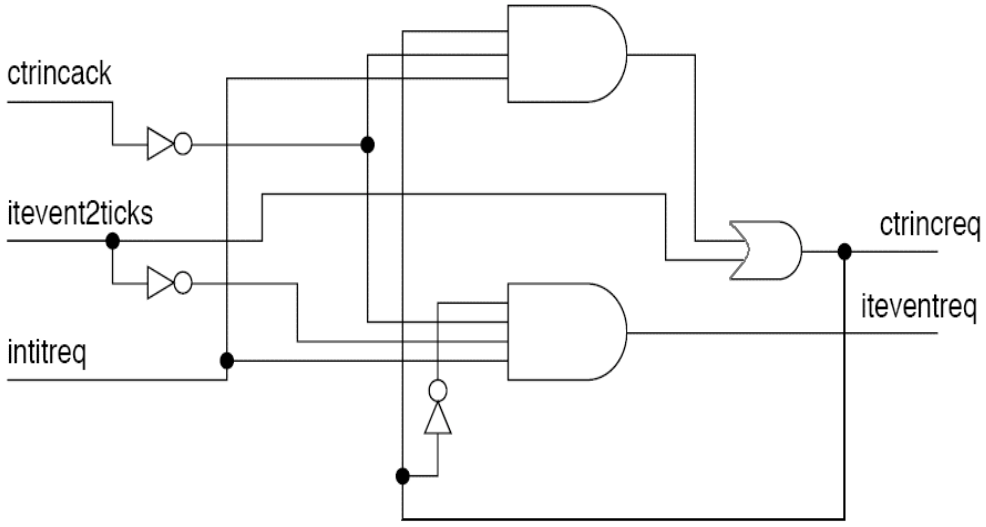
```
> cd MLO_tutorial
```

Step 4 – Create .pla & .sol files

```
>minimalist-speed hp-ir.bms
single-output fedback
```

Step 5 – Verify files created

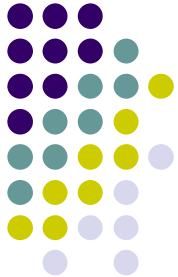
```
>Verify hp_IR-Fs.sol and hp_IR-
Fs.pla has been created
```



Two-Level circuit from Minimalist

Assumes Minimalist is loaded and paths are set correctly!

Tutorial – Gate Fan-In Limitation



Implement circuit with gate fan-in limit of 2 for AND gates. This can be done using two different methods (results will be the same)

Method 1: Independently
Specify fan-in limit for AND gates only

Step 1 – Run MLO

```
> MLO.py –A 2 –d hp_IR-Fs.sol
```

Step 2 – View Results

```
> less hp_IR-Fs-ML.v
```

Step 3 – Verify Results

> Compare output with circuit on next page

Method 2: Globally
Specify fan-in limit for every gate type

Step 1 – Run MLO

```
> MLO.py –M 2 –d hp_IR-Fs.sol
```

Step 2 – View Results

```
> less hp_IR-Fs-ML.v
```

Step 3 – Verify Results

> Compare output with circuit on next page

Tutorial – Gate Fan-In Limitation

```
module hp_IR_Fs (intitreq, itevent2ticks, ctrincack, iteventreq, ctrincreq);
  input intitreq, itevent2ticks, ctrincack;
  output iteventreq, ctrincreq;

  wire neg_ctrincack, neg_ctrincreq_i, neg_itevent2ticks, ctrincreq_i, ctrincreq, _i1;
  wire _i2, _i3;
  // Wires needed for multi-level
  wire _i4, _i5, _i6;
  // Feedback variables
  assign ctrincreq_i = ctrincreq;

  // Negative input literals
  not (neg_ctrincack, ctrincack);
  not (neg_ctrincreq_i, ctrincreq_i);
  not (neg_itevent2ticks, itevent2ticks);

  // First plane of logic
  // -- Network implementing gate _i3 --
  and (_i3, _i4, neg_ctrincack);
  and (_i4, intitreq, ctrincreq_i);
  // -- Network implementing gate _i2 --
  assign _i2 = itevent2ticks;
  // -- Network implementing gate _i1 --
  and (_i1, _i5, _i6);
  and (_i5, neg_ctrincack, neg_ctrinreq_i);
  and (_i6, intitreq, neg_itevent2ticks);

  // Second plane of logic
  // -- Network implementing gate CtrIncReq --
  or (ctrincreq, _i3, _i2);
  // -- Network implementing gate ITEventReq --
  assign iteventreq = _i1;

endmodule
```

MLO Output:

2 AND gates have been decomposed based on fan-in limit of 2.

Tutorial – Negative Logic

Implement circuit using negative logic.

Step 1 – Run MLO

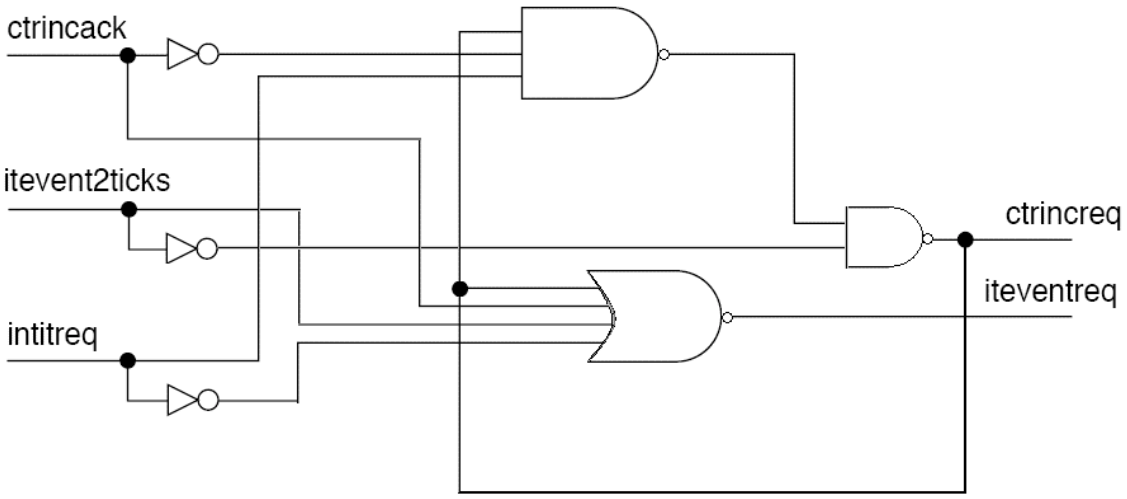
```
> MLO.py –n –d hp_IR-Fs.sol
```

Step 2 – View Results

```
> less hp_IR-Fs-ML.v
```

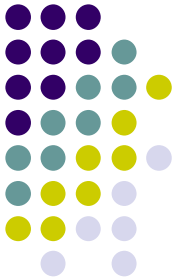
Step 3 – Verify Results

```
> Compare output with
circuit on next page
```



MLO Result: Multi-Level Circuit using negative logic only

Tutorial – Negative Logic



```
module hp_IR_Fs (intitreq, itevent2ticks, ctrincack, iteventreq, ctrincreq);
  input intitreq, itevent2ticks, ctrincack;
  output iteventreq, ctrincreq;

  wire neg__i2, neg_ctrincack, neg_intitreq, ctrincreq_i, ctrincreq, _i1;
  wire _i2, _i3;
  // Wires needed for multi-level
  wire neg__i3;
  // Feedback variables
  assign ctrincreq_i = ctrincreq;

  // Negative input literals
  not (neg__i2, _i2);
  not (neg_ctrincack, ctrincack);
  not (neg_intitreq, intitreq);

  // First plane of logic
  // -- Network implementing gate neg__i3 --
  nand (neg__i3, neg_ctrincack, intitreq, ctrincreq_i);
  // -- Network implementing gate _i2 --
  assign _i2 = itevent2ticks;
  // -- Network implementing gate _i1 --
  nor (_i1, ctrincack, ctrincreq_i, neg_intitreq, itevent2ticks);

  // Second plane of logic
  // -- Network implementing gate CtrIncReq --
  nand (ctrincreq, neg__i3, neg__i2);
  // -- Network implementing gate ITEventReq --
  assign iteventreq = _i1;
endmodule
```

MLO Output:

Only NOT/NOR/NAND gates used.

Tutorial – CEO Critical Event Optimizer (Automated Mode)

Run MLO using defaults (CEO on, no gate fan-in limits, and no negative logic).

No switch selected: default is CEO in auto mode

Step 1 – Run MLO

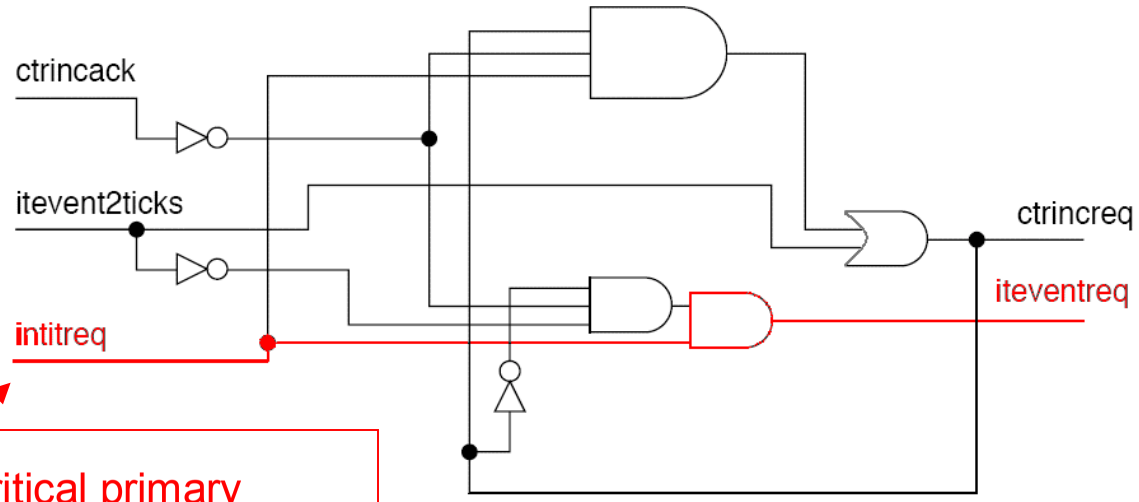
```
> MLO.py hp_IR-Fs.sol
```

Step 2 – View Results

```
> less hp_IR-Fs-ML.v
```

Step 3 – Verify Results

```
> Compare output with
circuit on next page
```



critical primary input-to-output path

Multi-Level circuit after CEO is used

Tutorial – CEO

```
module hp_IR_Fs (intitreq, itevent2ticks, ctrincack, iteventreq, ctrincreq);
  input intitreq, itevent2ticks, ctrincack;
  output iteventreq, ctrincreq;

  wire neg_ctrincack, neg_itevent2ticks, neg_ctrincreq_i, ctrincreq_i, ctrincreq, _i1;
  wire _i2, _i3;
  // Wires needed for multi-level
  wire _i4;
  // Feedback variables
  assign ctrincreq_i = ctrincreq;

  // Negative input literals
  not (neg_ctrincack, ctrincack);
  not (neg_itevent2ticks, itevent2ticks);
  not (neg_ctrincreq_i, ctrincreq_i);

  // First plane of logic
  // -- Network implementing gate _i3 --
  and (_i3, neg_ctrincack, intitreq, ctrincreq_i);
  // -- Network implementing gate _i2 --
  assign _i2 = itevent2ticks;
  // -- Network implementing gate _i1 --
  and (_i1, intitreq, _i4);
  and (_i4, neg_itevent2ticks, neg_ctrincack, neg_ctrincreq_i);

  // Second plane of logic
  // -- Network implementing gate CtrIncReq --
  or (ctrincreq, _i2, _i3);
  // -- Network implementing gate ITEventReq --
  assign iteventreq = _i1;
endmodule
```

MLO Output:

Gate Decomposed. Input intitreq is more critical to output iteventreq than neg_ctrincack and neg_ctrincreq_i

Tutorial – Features Mix

Features can be mixed and matched. This example does not utilize CEO, but it does target negative logic circuit and limits fan-in to 2.

Step 1 – Run MLO

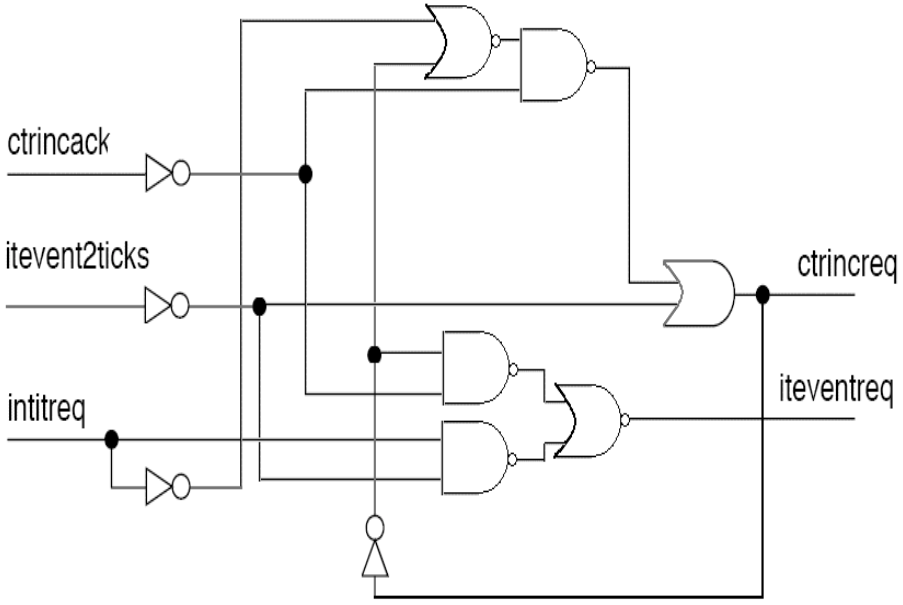
```
> MLO.py –d –n –R 2 –N 2 hp_IR-Fs.sol
```

Step 2 – View Results

```
> less hp_IR-Fs-ML.v
```

Step 3 – Verify Results

```
> Compare output with circuit on next page
```



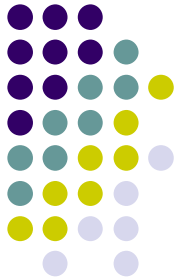
Tutorial – Features Mix

```
module hp_IR_Fs (intitreq, itevent2ticks, ctrincack, iteventreq, ctrincreq);
  input intitreq, itevent2ticks, ctrincack;
  output iteventreq, ctrincreq;

  wire neg__i2, neg_ctrincack, neg_intitreq, neg_ctrincreq_i, neg_itevent2ticks, ctrincreq_i;
  wire ctrincreq, _i1, _i2, _i3;
  // Wires needed for multi-level
  wire _i4, _i5, _i6, neg__i3;
  // Feedback variables
  assign ctrincreq_i = ctrincreq;
  // Negative input literals
  not (neg__i2, _i2);
  not (neg_ctrincack, ctrincack);
  not (neg_intitreq, intitreq);
  not (neg_ctrincreq_i, ctrincreq_i);
  not (neg_itevent2ticks, itevent2ticks);

  // First plane of logic
  // -- Network implementing gate neg__i3 --
  nand (neg__i3, _i4, neg_ctrincack);
  nor (_i4, neg_intitreq, neg_ctrincreq_i);
  // -- Network implementing gate _i2 --
  assign _i2 = itevent2ticks;
  // -- Network implementing gate _i1 --
  nor (_i1, _i5, _i6);
  nand (_i5, neg_ctrincack, neg_ctrincreq_i);
  nand (_i6, intitreq, neg_itevent2ticks);

  // Second plane of logic
  // -- Network implementing gate CtrIncReq --
  nand (ctrincreq, neg__i3, neg__i2);
  // -- Network implementing gate ITEventReq --
  assign iteventreq = _i1;
endmodule
```

MLO Output:

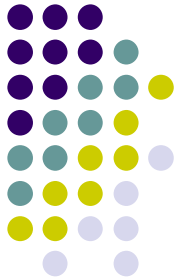
Only NOT/NOR/NAND gates used and fan-in limited to 2

Tutorial – User-Specified Critical Events - Overview



- MLO originally needed only two files to run (*.pla and *.sol)
- To utilize User-Specified Critical Events, MLO now needs two additional files:
  1. state_info.txt
  2. *.bms spec-file

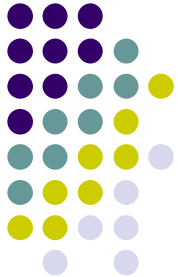
Tutorial – User-Specified Critical Events – state_info.txt



State minimization data is needed to utilize User-Specified Critical Events

- This information is only found on standard output during run of Minimalist.
- Therefore new program **caputure_min_data** is needed to redirect Minimalist output to **state_info.txt**
  - Runs from linux shell
  - Wrapper around Minimalist call
  - Important to note that **capture_min_data** **only** works with single-run Minimalist scripts. In other words, it doesn't work for script suites

Tutorial – User-Specified Critical Events – state_info.txt



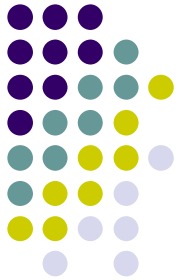
Syntax Rules:

1. caputure_min_data simply precedes Minimalist call.
   - For example, if user wants to capture data of minimalist-area script using multi-output run type, the syntax would be

```
> capture_min_data minimalist-area hp-ir.bms multi-output
```

2. Syntax for Minimalist command line options are same as if capture_min_data was **not** being used.

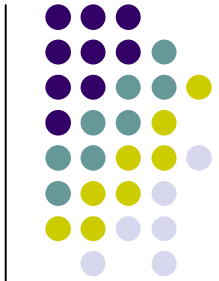
Tutorial – User-Specified Critical Events – *.bms spec-file



Burst Mode Spec file has to be edited to specify critical events.

- Critical events will be specified by providing critical input/output pairs for specified transitions in the burst-mode specification.
- If all inputs are critical for output during transition, wildcard("*") can be used.
- No blank lines can surround any inserted user-specified critical comments("; CRITICAL" statements) in the *.bms file.
  - Comment must immediately precede the transition that it is describing.

Tutorial – User-Specified Critical Events - *.bms spec-file



Syntax Rules (red comment):

1. To specify that the input CtrIncAck+ is critical to CtrIncReq- during transition from current state 3 to next state 4 (note that this is also saying that ITevent2Ticks is NOT critical):

```
;CRITICAL: CtrIncAck+/CtrIncReq-
3   4  ITevent2Ticks- CtrIncAck+   |     CtrIncReq-
```

2. To specify all inputs are critical to CtrIncReq- from current state 3 to next state 5:

```
;CRITICAL: */CtrIncReq-
3   5  ITevent2Ticks- CtrIncAck+   |     CtrIncReq-
```

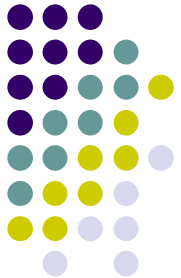
3. Another way to specify all inputs are critical (same as Rule #2) to CtrIncReq- during a transition from current state 3 to next state 5:

```
;CRITICAL: ITevent2Ticks-/CtrIncReq-, CtrIncAck+/CtrIncReq-
3   5  ITevent2Ticks- CtrIncAck+   |     CtrIncReq-
```

4. If no input/outputs are marked critical, automated CEO is used. Automated CEO is used to rate criticality of ITevent2Ticks and CtrIncAct with respect to CtrIncReq:

```
3   5  ITevent2Ticks- CtrIncAck+   |     CtrIncReq-
```

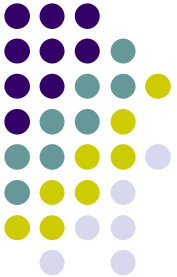
Tutorial – User-Specified Critical Events



Therefore, there are three steps to use User-Specified Critical Events

1. Capture **output of Minimalist**.
   - Run Minimalist with capture_min_data to create state_info.txt
2. **Specify critical events** in spec file (*.bms).
   - Add `;CRITICAL...` statements
3. **Run MLO** with proper command line options.
   - Must use –U [*.bms spec-file] option.
   - Only have to specify *.bms and *.sol file, MLO assumes state_info.txt and *.pla are in same directory as *.sol.

Tutorial – User-Specified Critical Events: Complete Run



Specify User-Specified Critical Events

Step 1 – Capture Minimalist Output

```
> capture_min_data minimalist-speed hp-ir.bms single-output
```

Step 2 – Edit spec file

```
>emacs (or vi) hp-ir.bms
```

Step 3 – Add critical events to spec file (default auto mode will be used for other events)

```
> add line in red to file:
;CRITICAL: CtrIncAck+/CtrIncReq-
  3      4        ITEvent2Ticks-    CtrIncAck+        |      CtrIncReq-
```

Step 4 – Run MLO

```
> MLO.py –U hp-ir.bms hp_IR-s.sol
```

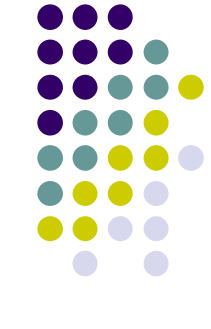
Step 5 – View Results

```
> less hp_IR-s-ML.v
```

Step 6 – Verify Results

```
> Compare output with circuit on next page
```

Tutorial – User Defined Critical Events



```
module hp_IR_s (intitreq, itevent2ticks, ctrincack, iteventreq, ctrincreq);
  input intitreq, itevent2ticks, ctrincack;
  output iteventreq, ctrincreq;
  wire neg_ctrincack, neg_itevent2ticks, neg_y0_i, y0_i, y0, _i1;
  wire _i2, _i3, _i4, _i5;
  // Wires needed for multi-level
  wire _i6, _i7;
  // Feedback variables
  assign y0_i = y0;
  // Negative input literals
  not (neg_ctrincack, ctrincack);
  not (neg_itevent2ticks, itevent2ticks);
  not (neg_y0_i, y0_i);
  // First plane of logic
  // -- Network implementing gate _i5 --
  and (_i5, neg_ctrincack, _i6);
  and (_i6, intitreq, y0_i);
  // -- Network implementing gate _i4 --
  assign _i4 = itevent2ticks;
  // -- Network implementing gate _i3 --
  and (_i3, intitreq, _i7);
  and (_i7, neg_itevent2ticks, neg_ctrincack, neg_y0_i);
  // -- Network implementing gate _i2 --
  and (_i2, neg_ctrincack, intitreq, y0_i);
  // -- Network implementing gate _i1 --
  assign _i1 = itevent2ticks;
  // Second plane of logic
  // -- Network implementing gate CtrIncReq --
  or (ctrincreq, _i5, _i4);
  // -- Network implementing gate y0 --
  or (y0, _i2, _i1);
  // -- Network implementing gate ITEventReq --
  assign iteventreq = _i3;
endmodule
```

Gate decomposed using fact that user-specified neg_ctrincack is critical to ctrincreq

Notice that Automated CEO is still used on outputs not specified by user

Tutorial – Verifier

Run MLO using with CEO and use the verifier to ensure output is functionally correct and hazard free

Step 1 – Run MLO

```
> MLO.py -V hp_IR-s.sol
```

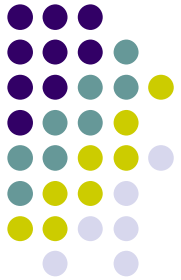
Step 2 – Verify MLO output passes verifier (verify success message is printed)

```
> check standard output
```

If output passes built-in verifier, a success message will be printed to standard output.

```
---------------------------------
Verification Results
Output has been verified - Everything OK.
---------------------------------
```

Recap



- MLO uses Minimalist two-level logic output and produces an optimized multi-level circuit specified in Verilog.
- Tool has multiple features including negative logic implementations, critical-event optimizer, and gate fan-in limitations.