

Hw3a Lab Report Ongoing

Ryan Oates

[Notation Key]

GateSymbolNotation = [List of logic gate symbols used in this design]

; e.g., AND (\wedge), OR (\vee), NOT (\neg), etc., to represent gate operations in Boolean expressions.

WireConnectionSymbols = [Notation for wires and connections in circuit diagrams]

; e.g., lines for connections, a dot for junction, labels/arrows for inputs and outputs.

LogicLevelsRepresentation = [Definition of logic high and low levels]

; e.g., use '1' (HIGH/True) and '0' (LOW/False) to denote binary logic levels.

BooleanExpressionNotation = [Format for writing Boolean formulas]

; e.g., use '+' for OR, '.' or concatenation for AND, overline or '!' for NOT ($A + B$ means A OR B, AB or $A \cdot B$ means A AND B, \overline{A} means NOT A).

; Note: These notations are used throughout (in the truth table, K-map, expressions, and diagrams).

See Glossary for definitions of terms and symbols.

; \checkmark/\times Verification:

verify_symbols_defined = (\checkmark/\times) ; All necessary gate symbols and wire notations are defined above.

(Comment: ---)

verify_logic_levels = (\checkmark/\times) ; Logic 0/1 representation is clear and used consistently. (Comment: ---)

verify_notation_consistency = (\checkmark/\times) ; Boolean expression notation is consistent across all sections. (Comment: ---)

[Possible Inputs]

InputVariables = [List of input variable names]

; e.g., A, B, C (representing all independent inputs to the logic circuit).

InputDomain = [All possible values or conditions for each input]

; e.g., each input $\in \{0,1\}$ for binary logic (enumerate any constraints or don't-care conditions if applicable).

; Note: These inputs define the domain for the Truth Table.

... (about 599 lines omitted) ...

| Time | A | B | Sum | Carry | Analysis |
|----------|---|---|-----|-------|-------------------------------|
| 20-40ns | 1 | 0 | 1 | 0 | A transition |
| 40-60ns | 1 | 1 | 0 | 1 | B transition, carry generated |
| 60-80ns | 0 | 1 | 1 | 0 | A transition |
| 80-100ns | 0 | 0 | 0 | 0 | Return to initial state |

[DESIGN_HISTORY]

iterations=""""

v1: Initial design (redundant inputs)

v2: Simplified inputs

v3: Corrected carry generation

v4: Final verified design

"""""

[VISUALIZATION_GUIDELINES]

setup_steps=""""

1. Right-click signal names for color options

2. Tools->Settings->Waveforms for trace width
 3. Tools->Color preferences for background
 4. Maximize RGB values (255) for clarity
- ”””

[VERIFICATION_CHECKLIST]

✓ truth_table_verified

✓ timing_requirements_met

✓ carry_generation_correct

✓ waveform_visualization_optimized

KMapLayout = [K-Map cell arrangement]

; e.g., 2x2, 4x4 grid with Gray code ordering of inputs on rows/columns, as appropriate for the number of variables.

KMapFilling = [Assignment of output values to K-map cells]

; e.g., place 1s and 0s in K-map cells corresponding to truth table outputs (1 for minterms where $F=1$).

KMapGrouping = [Identify groups of adjacent 1s (or 0s for POS) in the K-map]

; e.g., list groups by cells covered (binary indices) and the common variables for each group.

KMapSimplificationSteps = [Step-by-step simplification]

; e.g., "Group1 (cells 0101, 0100, 0000, 0001) yields term $\neg A \cdot \neg C$ ", etc.

KMapResultExpression = [Simplified Boolean expression from K-map]

; e.g., $F = \neg A \cdot \neg C + B \cdot \neg C$ (result of grouping). This is the minimized Sum-of-Products form derived via the K-map.

; Note: K-map simplification is based on the Truth Table and yields a minimal expression. The result will be used to inform the Gate Operations and is cross-checked by Boolean algebra in [Boolean Expression Reduction].

; ✓/× Verification:

verify_grouping_complete = (✓/×) ; All 1-cells are covered by groups (including any don't-cares if used), with groups sized in powers of 2. (Comment: ---)

verify_minimal_expression = (✓/×) ; The K-map result is a minimal expression (no further reduction possible, no redundant groups). (Comment: ---)

verify_expression_correct = (✓/×) ; The simplified expression from K-map produces the same output as the original truth table (validated by checking key input cases or re-deriving truth table from it). (Comment: ---)

[Gate Operations]

; Define the logic gates implementing the simplified expression:

Gate1 = [Gate type and inputs -> output]

; e.g., G1: AND gate with inputs A and B, output = X (implements $X = A \wedge B$).

Gate2 = [Gate type and inputs -> output]

; e.g., G2: NOT gate with input C, output = Y (implements $Y = \neg C$).

Gate3 = [Gate type and inputs -> output]

; e.g., G3: OR gate with inputs X and Y, output = F (implements $F = X + Y$).

GateOperationNotes = [Any additional info about gate behavior/timing]

; e.g., All gates are assumed ideal for logic levels (output transitions occur after a small propagation delay). If timing is critical, note propagation delays or if this is a synchronous circuit, note clocking.

; Note: The above gates realize the expression from [K-Map Reduction] ($F = X + Y$ in the example) using standard logic symbols from [Notation Key]. Each gate's logical function (AND, OR, NOT)

is defined in the Glossary (e.g., AND gate outputs 1 only if all inputs are 1).

; ✓/× Verification:

verify_gates_complete = (✓/×) ; All parts of the simplified Boolean expression are implemented by the listed gates (no missing term or signal). (Comment: ---)

verify_output_consistency = (✓/×) ; The output from the gate network matches the expected truth table output for each input combination (circuit logic is correct). (Comment: ---)

verify_notation_adherence = (✓/×) ; Gate symbols and wiring in the described implementation follow the Notation Key (correct symbols for each gate, proper label names). (Comment: ---)

verify_timing_addressed = (✓/×) ; Any necessary timing considerations (propagation delay, gate switching speed) are noted and acceptable for the design. (Comment: ---)

[Circuit Diagram Representation]

DiagramIllustration = [Reference or description of the circuit diagram]

; (In an actual document, this could be an embedded schematic image or ASCII diagram of the circuit. e.g., a drawn logic diagram with gates labeled G1, G2, G3 as above.)

DiagramNotation = [Explanation of diagram symbols/legends]

; e.g., AND/OR/NOT gate shapes per standard, dots on lines for junctions, distinctive symbols for inputs and outputs as defined in Notation Key.

DiagramLabels = [List of labeled signals in diagram]

; e.g., label inputs (A, B, C), outputs (F), and intermediate nodes (X, Y) corresponding to the Gate Operations.

; Note: The circuit diagram provides a visual confirmation of the [Gate Operations]. It should use the symbols from [Notation Key] and reflect the connections described (e.g., output of G1 feeds one input of G3, etc.).

; ✓/× Verification:

verify_diagram_accuracy = (✓/×) ; Diagram matches the gate-level implementation (all gates and connections correspond to those listed in Gate Operations). (Comment: ---)

verify_label_consistency = (✓/×) ; Every signal in the diagram (inputs, outputs, nodes) is clearly labeled and matches the naming in the text/tables. (Comment: ---)

verify_symbol_standard = (✓/×) ; All symbols in the diagram are standard and were defined in the Notation Key (no undefined symbols or ambiguous notations). (Comment: ---)

[Full Signal Analysis]

; Analyze signal propagation and node equations in the circuit:

NodeEquations = [List Boolean equations for intermediate nodes]

; e.g., $X = A \wedge B$, $Y = \neg C$ (from Gate1 and Gate2 outputs as defined in Gate Operations).

OutputEquation = [Boolean equation for output in terms of nodes or inputs]

; e.g., $F = X + Y$ (which, when expanded, matches the simplified Boolean expression).

PropagationSteps = [Describe the sequence of signal propagation through the circuit]

; e.g., "Inputs A,B feed AND gate -> X. Input C feeds NOT gate -> Y. Then X and Y feed OR gate -> F. Thus, a change in A or B propagates through G1 then G3 to F; a change in C propagates through G2 then G3."

TimingAnalysis = [If applicable, note the critical path or propagation delay]

; e.g., "Worst-case propagation: a change on A or B travels through two gates (AND then OR) to affect F, whereas a change on C travels through NOT then OR. Assuming each gate has similar delay, the critical path is two gates long. No hazards/glitches were observed as all inputs feed combinational gates with proper synchronization (see Propagation Delay in Glossary)."

; Note: This analysis verifies that each intermediate node's logic (node equations) and the timing of signal changes produce the correct final output as specified in the [Truth Table]. It ties the static logic design to dynamic behavior (propagation).

; ✓/× Verification:

verify_node_logic = (✓/×) ; Every intermediate node equation is consistent with the intended logic and the overall Boolean expression. (Comment: ---)

verify_propagation_correctness = (✓/×) ; For each input scenario, following the signal path through nodes yields the correct output (matches truth table – ensures no logical discrepancies at any stage). (Comment: ---)

verify_timing_consistency = (✓/×) ; Signal propagation times are acceptable and do not violate any design requirements (or timing not an issue for static combinational logic). Any potential glitches or race conditions checked. (Comment: ---)

[SOP Reduction]

InitialSOP = [Sum-of-Products expression derived from the truth table]

; e.g., list of minterms: $F = A \cdot B \cdot \neg C + A \cdot B \cdot C + \dots$ (each term corresponds to an input combination where output=1 in the Truth Table).

ReductionSteps = [Algebraic reduction steps applied to simplify the SOP]

; e.g., combine terms using Boolean algebra theorems:

; $F = A \cdot B \cdot \neg C + A \cdot B \cdot C$ -> factor $A \cdot B$: $F = A \cdot B(\neg C + C)$

; -> simplify ($\neg C + C = 1$): $F = A \cdot B$

; (show step-by-step simplification of the SOP expression).

ReducedSOP = [Final minimized SOP expression]

; e.g., $F = A \cdot B + \neg C \cdot B$ (the simplified sum-of-products form, which should match the K-Map result).

; Note: This section provides an algebraic simplification of the Sum-of-Products derived from the Truth Table. The final result should coincide with the expression from [K-Map Reduction], confirming the minimization is correct.

; ✓/× Verification:

verify_all_minterms_used = (✓/×) ; Initial SOP includes all required minterms for output=1 (covers every truth table case where F=1). (Comment: ---)

verify_algebraic_steps = (✓/×) ; Boolean algebra steps are correctly applied (each simplification step is valid). (Comment: ---)

verify_sop_matches_kmap = (✓/×) ; The reduced SOP expression matches the simplified result from K-Map Reduction (thus both methods agree on final expression). (Comment: ---)

[Boolean Expression Reduction]

InitialExpression = [Original Boolean expression before simplification]

; e.g., starting expression from requirements or the full SOP expression before reduction.

TargetExpression = [Target (simplified) Boolean expression]

; e.g., the expected simplified expression (from K-Map or known simplest form).

ReductionTechnique = [Method used for reduction]

; e.g., algebraic manipulation, applying identities (consensus theorem, De Morgan's law, etc.) to reach the simplified form.

ReductionProof = [Proof or validation of equivalence]

; e.g., a brief statement: "Verified equivalence by truth table comparison or Boolean algebra proof showing initial expression equals simplified expression."

FinalBooleanExpression = [The final simplified Boolean expression for the design]

; e.g., $F = A \cdot B + \neg C \cdot B$ (from our example), which is the form implemented by the circuit.
 ; Note: This section ensures the logical expression of the circuit is fully simplified and correct. It should confirm that the expression derived (via K-Map or SOP) is indeed the most reduced form and is logically equivalent to the original specification. This final expression is the one realized in [Gate Operations].

; \checkmark/\times Verification:

verify_equivalence_proven = (\checkmark/\times) ; Demonstrated that the initial and final expressions are equivalent (via truth table or symbolic proof). (Comment: ---)

verify_expression_minimal = (\checkmark/\times) ; The final Boolean expression is simplified to minimal terms/literals (no further simplification possible). (Comment: ---)

verify_consistency_final = (\checkmark/\times) ; The final expression aligns with the implemented circuit (the gates in the design realize this expression) and with the truth table outputs. (Comment: ---)

[Direct Links]

NotationKey_related = Used by all sections for consistent symbols (e.g., gate symbols in Gate Operations, logic levels in Truth Table).

PossibleInputs_related = Feeds into Truth Table (defines all combinations for which outputs are determined).

TruthTable_related = Basis for K-Map Reduction and SOP Reduction (provides the raw output combinations for simplification).

KMapReduction_related = Simplifies the Truth Table outputs to a minimal expression (used by Gate Operations, cross-checked in Boolean Expression Reduction).

GateOperations_related = Implements the simplified expression (from K-Map/SOP) with physical gates, referencing symbols from Notation Key.

CircuitDiagram_related = Visual representation of Gate Operations (wiring and gates), using the notations defined in Notation Key.

FullSignalAnalysis_related = Verifies signal flow and timing from inputs to output (ensures dynamic behavior matches static Truth Table logic).

SOPReduction_related = Alternative/parallel simplification method (derives simplified expression from truth table, should match K-Map result).

BooleanExprReduction_related = Final verification of expression simplification (confirms K-Map and SOP results, ensuring correct implementation).

Glossary_related = Provides definitions of terms and symbols (e.g., gates, K-Map, SOP) used throughout the document for quick reference.

[Glossary]

AND Gate = Logic gate that outputs 1 (true) only if **all** its inputs are 1. (Logical conjunction; symbol \wedge or $\&$)

OR Gate = Logic gate that outputs 1 if **at least one** input is 1. (Logical disjunction; symbol \vee or $+$)

NOT Gate = Logic gate that outputs the logical **negation** of its input (outputs 1 if input is 0, and vice versa). (Inversion; symbol \neg or $!$ or $'$)

Logic 1 (HIGH) = The higher logic level in binary (e.g., True, "1", high voltage level) representing boolean true.

Logic 0 (LOW) = The lower logic level (e.g., False, "0", low voltage level) representing boolean false.

Truth Table = A table listing all possible combinations of input values and the corresponding output for a logic function. Used to define the behavior of the logic circuit exhaustively.

Karnaugh Map (K-Map) = A graphical method for simplifying Boolean expressions by organizing truth table values into a grid, allowing common terms to be combined. It helps reduce logic expressions to minimal form.

Sum-of-Products (SOP) = A canonical Boolean form where the expression is a OR (sum) of multiple AND (product) terms. Each product term corresponds to a combination of inputs that yields output 1.

Boolean Expression = An algebraic expression composed of boolean variables and logic operators (AND, OR, NOT, etc.) representing a logic function. E.g., $F = A \cdot B + \neg C$.

Node (Intermediate) = A connection or intermediate signal in the circuit (output of a gate that is not the final output). Often labeled (e.g., X, Y) and has its own Boolean equation in the context of the circuit.

Node Equation = The Boolean expression for an intermediate node's value in terms of the circuit's input variables (and possibly other node values). E.g., $X = A \cdot B$ defines node X as the AND of A and B.

Propagation Delay = The time interval between an input change and the resulting output change in a logic circuit. Every logic gate has a finite switching speed, so signals take time to propagate through the circuit (see also Gate Delay).

Signal Propagation = The movement of logic level changes through the circuit's network of gates and connections, from inputs to intermediate nodes to outputs.

Verification Checklist = A list of steps or criteria used to verify the correctness of each part of the design. Each item is marked with a ✓ if passed or × if an issue is found, accompanied by comments explaining the result.

Cross-Referencing = The practice of linking related sections or terms in the document for easy navigation. For example, referring to the Truth Table section when discussing K-Map, or pointing to Glossary definitions for specific terms.

✓ (Check mark) = Indicates a successful verification or a condition met. In the template, replace this symbol in a checkbox once the verification step is confirmed.

× (Cross mark) = Indicates a failed verification or a condition not met. It denotes an issue that needs attention or correction, explained in the comments.

[SIMULATION_AND_RESULTS]

simulation_tool=Tina

test_duration=100ns

input_stimulus="""

A: 0-1 at 20ns, 1-0 at 60ns

B: 0-1 at 40ns, 1-0 at 80ns

"""

[WAVEFORM_SETTINGS]

display_configuration="""

Signal Colors:

- A: Blue (RGB: 0,0,255)

- B: Red (RGB: 255,0,0)

- Sum: Green (RGB: 0,255,0)

- Carry: Yellow (RGB: 255,255,0)

Trace Settings:

- Data Trace Width: 2px

- Background: White (RGB: 255,255,255)

- Grid: Light Gray (RGB: 200,200,200)
"""

[SIMULATION_RESULTS]

| | | | | | | |
|-----------------------|----------|---|---|-----|-------|-------------------------------|
| | Time | A | B | Sum | Carry | Analysis |
| | 0-20ns | 0 | 0 | 0 | 0 | Initial state |
| waveform_analysis=""" | 20-40ns | 1 | 0 | 1 | 0 | A transition |
| | 40-60ns | 1 | 1 | 0 | 1 | B transition, carry generated |
| | 60-80ns | 0 | 1 | 1 | 0 | A transition |
| """ | 80-100ns | 0 | 0 | 0 | 0 | Return to initial state |

[VISUALIZATION_TIPS]

waveform_optimization="""

1. Right-click signal name above graph to modify colors
 2. Access Tools->Settings->Waveforms for trace width
 3. Modify background via Tools->Color preferences->Waveform
 4. Set RGB values to 255 for maximum visibility
- """

[TROUBLESHOOTING_NOTES]

initial_attempts="""

1. First attempt: Four inputs (redundant)
 - Issue: Overcomplicated design
 - Resolution: Simplified to two inputs
 2. Carry implementation:
 - Initial mistake: Added C input for carry
 - Correction: Generated carry from A,B inputs
 - Learning: Document mistakes to prevent future repetition
- """

[DESIGN_ITERATIONS]

version_history="""

- v1: Basic design with redundant inputs
v2: Removed redundant inputs
v3: Corrected carry generation
v4: Final optimized design with simulation verification
- """

Let's combine and streamline these templates into a more focused structure for your half-adder design documentation:

[DOCUMENT_METADATA]

title=Half Adder Design Documentation

version=1.0

date=2024-02-XX

author=Student_Name

course=Digital_Design_Lab_Two

[NOTATION_AND_SYMBOLS]

gate_symbols="""

NAND = \overline{AB}

```

AND = .
OR = +
XOR = ⊕
NOT = ¬
"""

circuit_symbols="""
Wire = —
Junction = ⊢
Turn = ⌊ ⌈
Signal Flow = →
Inversion = ○
"""

logic_levels="""
HIGH = 1 (True)
LOW = 0 (False)
"""

[DESIGN_SPECIFICATIONS]
block_type=Half_Adder
inputs=A,B
outputs=Sum,Carry
constraints=carry_in=fixed_zero
implementation=NAND_Based

[IMPLEMENTATION_DETAILS]
circuit_diagram="""
A —⊢→ [NAND1] —⊢→ [NAND3] —→ [OR] —→ Sum
|
B —⊢→ [NAND2] —⌋
|
⌊→ [NAND4] —→ Carry
"""

truth_table="""


| A | B | Sum | Carry |
|---|---|-----|-------|
| 0 | 0 | 0   | 0     |
| 0 | 1 | 1   | 0     |
| 1 | 0 | 1   | 0     |
| 1 | 1 | 0   | 1     |


"""

[SIMULATION_SETUP]
tool=Tina
duration=100ns
input_stimulus="""
A: 0-1 at 20ns, 1-0 at 60ns
B: 0-1 at 40ns, 1-0 at 80ns
"""

[WAVEFORM_CONFIGURATION]
display_settings="""

```


Colors:
- A: Blue (0,0,255)
- B: Red (255,0,0)
- Sum: Green (0,255,0)
- Carry: Yellow (255,255,0)

Trace Width: 2px
Background: White
Grid: Light Gray
"""

[RESULTS_AND_ANALYSIS]

| | | | | | | | |
|---------------------|----------|---|---|-----|-------|-------------------------------|-----|
| | Time | A | B | Sum | Carry | Analysis | |
| | 0-20ns | 0 | 0 | 0 | 0 | Initial state | |
| timing_analysis=""" | 20-40ns | 1 | 0 | 1 | 0 | A transition | """ |
| | 40-60ns | 1 | 1 | 0 | 1 | B transition, carry generated | |
| | 60-80ns | 0 | 1 | 1 | 0 | A transition | |
| | 80-100ns | 0 | 0 | 0 | 0 | Return to initial state | |

[DESIGN_HISTORY]

iterations="""
v1: Initial design (redundant inputs)
v2: Simplified inputs
v3: Corrected carry generation
v4: Final verified design
"""

[VISUALIZATION_GUIDELINES]

setup_steps="""
1. Right-click signal names for color options
2. Tools->Settings->Waveforms for trace width
3. Tools->Color preferences for background
4. Maximize RGB values (255) for clarity
"""

[VERIFICATION_CHECKLIST]

✓ truth_table_verified
✓ timing_requirements_met
✓ carry_generation_correct
✓ waveform_visualization_optimized