Memo 2

Drexel University

To: Dr Christopher Peters
From: Mahad Faisal
Date: 02/02/2024
Re: Lab 2 -Sum of Minterms

Purpose

The aim of the lab assignments for these weeks was to apply the knowledge studied in the lectures which includes the *sum of minterms*, *Karnaugh maps*, and a deeper understanding of the Verilog HDL. This project was divided into 3 parts; the first being a written conversions of truth tables into sum of minterms and its simplification using Boolean algebra and with a Karnaugh map, the second part involved simulating the simplified equation using Verilog on EDA Playground and the final part involved the use of an Arduino MEGA 2560 kit and 75HC*XX* ICs to simulate the equation.

Methodology

At the start of this project a truth table [**Fig 1.1**] was given to be simplified using the sum of minterms. Both alphabetic and sigma representation were used when simplifying the truth table [**Fig 1.1**]. The alphabetic representation of the truth table was calculated to be the following: **F = A'B'CD' + A'B'CD + A'BCD' + A'BCD + AB'C'D' + AB'CD' + ABC'D' + ABCD'**. To represent this expression in the form of a diagram is required. Visualizing this expression in the form of a circuit diagram required 27 gates (16 AND gates, 7 OR gates and 4 NOT gates) which is very expensive, inefficient and complicated.

Meanwhile, the sigma representation was calculated to output the following: **F =∑ (2, 3, 6, 7, 8, 10, 12, 14)**. It was determined that the sigma representation was more time efficient as well as readable. Further simplifying the alphabetic representation gave the following result: **F = A'C + AD'**.

After utilizing the aforementioned method to simplify and represent the given truth table, a K-map was created with the given value, and with that information, it was easy to determine that the most simplified form was **F=A'C + AD'**. The sum of minterms approach was concluded to be a viable option but there was a consensus on the use of a K-map being substantially more time-efficient and accurate (less risk of errors).

A physical circuit was then drawn out with the derived expression which involved two (2) NOT logic gates, two (2) AND logic gates and one (1) OR logic gate. This was incredibly helpful in the following section of the project which was to simulate the expression on a digital platform, *Tinkercad*.

The final Tinkercad circuit can be seen in **Fig 1.2**. Developing this circuit was the most time-consuming section of the lab for me. In almost all the unsuccessful renditions [such as **Fig 1.3**] of this circuit developed, there was little to no readability and organization in terms of the wring and placement of components; mainly because of the lack of recognition of the fact that connection wires do not need to be inserted in the exact input pin(s) and that they could be placed in any pin in the same column. This was a detrimental mistake as attempting to debug the circuit in that state was practically impossible.

After the realization of the mistake(s), the process became tenfold easier. It was then concluded that the circuit failed to initialize due to a missing wire, a flaw that could not have been deciphered in the earlier unorganized environment. Simulating the circuit and observing the results of the different combinations of inputs verified that the components were connected correctly as the output followed the expression **F = A'C+AD'**.

Following the Tinkercad portion of the lab, came the Verilog simulation. The Verilog simulation was performed on a virtual platform called *EDA Playground*, with Icarus Verilog 0.9.7 as the tool and *EPWave* was used to visualize the output of the code as waveforms.

The code used can be seen in **Fig 2.1**. Initially the timescale directive is utilized to set the time unit (10ns) and the time precision (1ns) for the environment. In the second line, the module directive is used to build a container for the testbench file/hardware description. Inside the module, literals A, C and D are set as registers, which store logic values, and the output, F, is set as a wire. The subsequent line refers to the simplified Boolean expression found in the first section of the lab.

After all the variables etc. have been set inside the module of the testbench file, 'initial begin' is used to execute the code and create the 'dump file' for the output of the code as waveforms that last 90ns inside a VCD file. The final few lines simulate all the values for A, C and D ranging from HIGH to LOW every 40ns, 20ns, and 10ns respectively.

To ensure that the testbench file ran smoothly the design.sv file in the parallel side had to be cleared out and the settings for EDA Playground had to be adjusted so that the output in EP Wave opened in a new tab. Once the file was running and the tab with the waveform was open [**Fig 2.2**], the output of the waveforms was recorded at every 10ns up till the final time of 90ns to obtain all the possible outcomes of this code.

The final component of the lab project of Week 2- Sum of Minterms was to build a physical circuit with an ELEGOO Arduino Kit and to use the serial monitor in the Arduino IDE to generate truth tables for the obtained expression. To build this circuit, the instructions from the Lab Project book were used as reference and the equation for that circuit (**A'C+AD'**) is what was used in this part. The parts used in the making of this circuit were:

- (1) Long breadboard
- (1) 74HC08 Quad AND gate package
- (1) 74HC32 Quad OR gate package

- (1) 74HC04 Hex Inverter (NOT) gate package
- (1) Arduino Mega 2560 board
- Wires

This circuit board, pictured in **Fig 3.1**, like the Tinkercad simulation, is rather complicated to piece together perfectly. The logic gate packages were placed at the middle of the breadboard and wires were used to connect the power pins to the positive top rail and the ground pins to the negative bottom rail for grounding at for each logic gate package. Then the individual pins on the packages were connected as required.

Pin 1A on the 74HC04 Hex inverter package, which is essentially a NOT gate, was connected to the 74HC08 pin 2A as the common connection for signal A. The output of pin 1A on the Hex inverter, pin 1Y, was wired to pin 1A on the AND package for the term A'. For the term D', the 74HC04 pin 2Y was connected to 74HC08 pin 2B. With this arrangement all the literals were present and to get the two expressions 74HC04 pin 2Y was connected to 74HC08 pin 2B to make AC' and 74HC08 pin 2Y was wired to 74HC32 pin 1B for AD', as seen in **Fig 3.1**.

The Arduino MEGA 2560 board was then connected to the breadboard via a grounding wire in between the GND pin on the digital pin section of the Arduino and the negative bottom rail of the breadboard and the power was driven to the board via the connection in between the 5V power pin on the Arduin and the positive top rail on the breadboard.

After that, the digital pins on the Arduino were plugged in to the logic gate packages on the breadboard to control and relay the variables A, B, C, and D. 3. For A, Arduino digital pin 12 was carefully connected to 74HC04 pin 1A. Although B is not directly involved in the expression that is being tested, it is still necessary to have it in the code to avoid unlikely but possible errors among other reasons. Hence why there is no need to connect pin 11 to the circuit but its recognition is important. To represent C Arduino digital pin 10 was connected to 74HC08 pin 1B and finally, to represent D Arduino pin 9 to was attached to 74HC04 pin 2A.

It was harder to identify the pin types as unlike on Tinkercad, there are no markings to distinguish whether a pin on the logic gate ICs is an input or output or which number i.e. input **2**A it is. Upon connection to the power source, a laptop via USB, the board slowly powered off. It was hard to find the reason after multiple reviews. After dismantling the board and reconstructing it several times it was found that the 74HC32 OR gate was placed upside down which was determined by the placement of the semicircle divet at the end of the package. This was causing the circuit to short, but the failsafe system was preventing components from getting permanent damage.

After connecting the completed board to the laptop and verifying all connections the

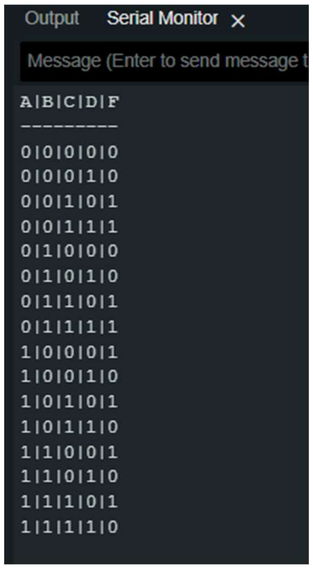
Arduino code was uploaded. The output of the program was transmitted to the serial monitor tool on the Arduino IDE. The output on the serial monitor had to be cleared and the red button on the Arduino board had to be pressed to reset the system and then the correct output was displayed, as expected, and that was the main goal of the project.

Results

Verilog result:

Time/ns	**A**	**C**	**D**	**F**
0	0	0	0	0
10	0	0	1	0
20	0	1	0	1
30	0	1	1	1
40	1	0	0	1
50	1	0	1	0
60	1	1	0	1
70	1	1	1	0
80	0	0	0	0
90	0	0	1	0

Arduino results



```
A|B|C|D|F
---------
0|0|0|0|0
0|0|0|1|0
0|0|1|0|1
0|0|1|1|1
0|1|0|0|0
0|1|0|1|0
0|1|1|0|1
0|1|1|1|1
1|0|0|0|1
1|0|0|1|0
1|0|1|0|1
1|0|1|1|0
1|1|0|0|1
1|1|0|1|0
1|1|1|0|1
1|1|1|1|0
```

As pictured above both follow the original expression which shows that both simulations were successful. However, the Verilog results seem to be missing a few combinations of inputs at first glance but that is not the case. This is because there are 4 inputs in the Arduino code hence why there must be 16 combinations (4^2) whereas the Verilog code does not account for B as it would be redundant, so it only has 3 inputs and therefore 9 (3^2) possible combinations. The Arduino code also produces repetitive outputs when ignoring B as it plays no role in the system whatsoever.

The Verilog results table shows values recorded at every 10ns as that is the time when the values for A, C and D were toggled. The output was only true (1 or HIGH) when the parameters set in the code, either A'C or AD' were met.

Similarly, the Arduino results, from the circuit built from the Lab Project book, only outputted '1' or 'True' when the parameters set in the Arduino Code were met. This shows that all the wires and pins were connected correctly and hence why there are no anomalous data points recorded.

An interesting phenomenon seen is seen on the Verilog results at times 80ns and 90ns. This can be explained by the code used which toggles A between HIGH and LOW every 40ns, B every 20ns, and C every 10ns. Due to this set up the waveform starts repeating at 80ns.

The fact that this lab produced results that match the ones given at the beginning show that there were no flaws substantial enough to affect the outcome in any way and it was a great lab as it helped me get comfortable with debugging and understand the importance of a neat and organized set up and how much time that saves in the long run.

Appendix

Figure 8.1

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Figure 1.2

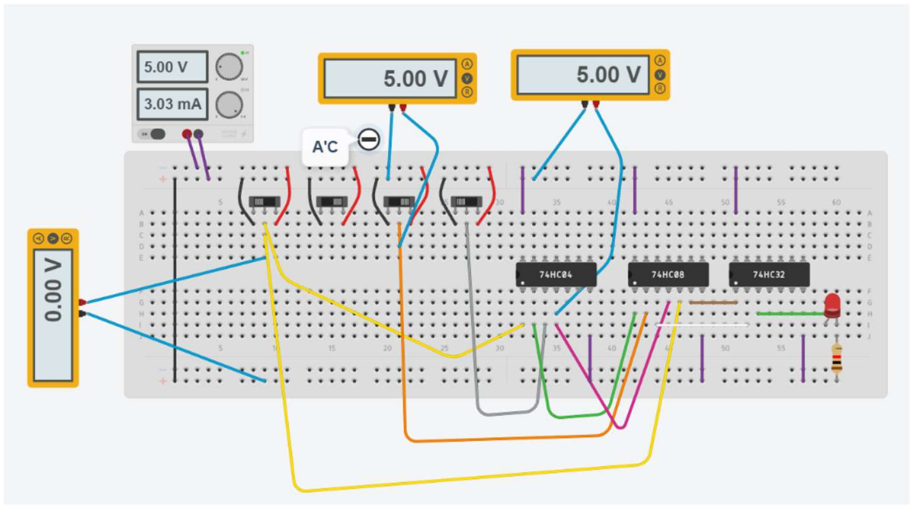


Figure 1.3

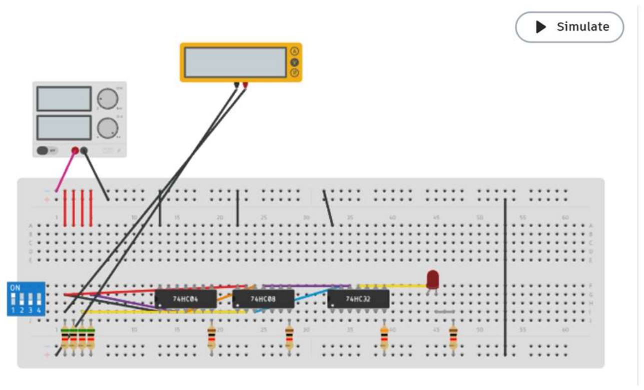
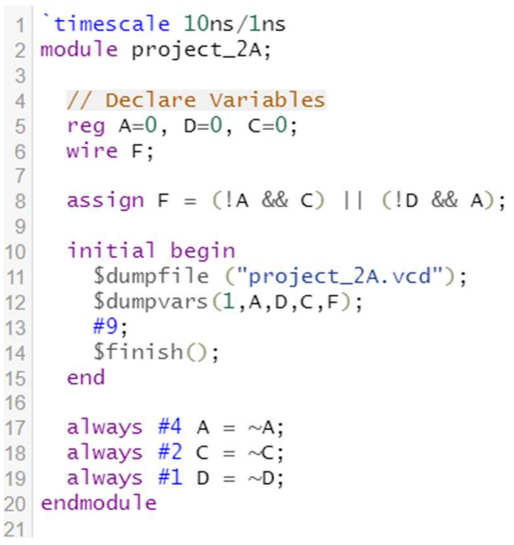


Figure 9.1



```
`timescale 10ns/1ns
module project_2A;

  // Declare Variables
  reg A=0, D=0, C=0;
  wire F;

  assign F = (!A && C) || (!D && A);

  initial begin
    $dumpfile ("project_2A.vcd");
    $dumpvars(1,A,D,C,F);
    #9;
    $finish();
  end

  always #4 A = ~A;
  always #2 C = ~C;
  always #1 D = ~D;
endmodule
```

Figure 2.2

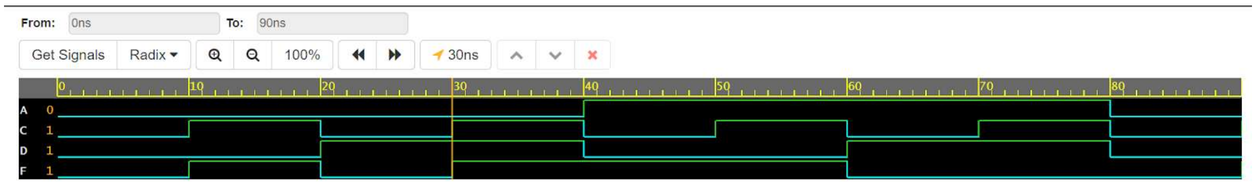


Figure 3.1

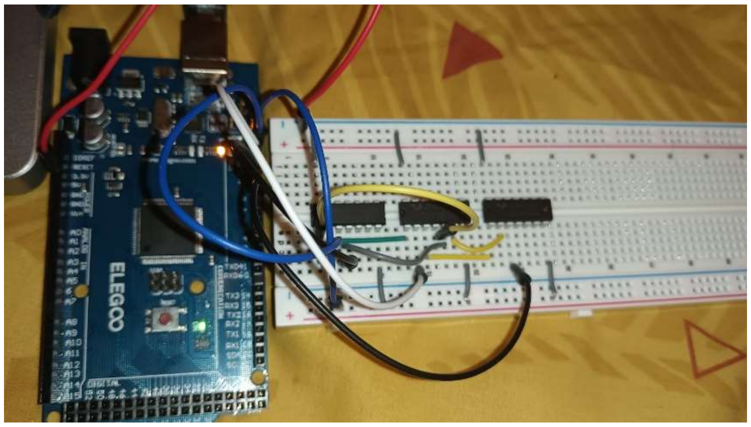
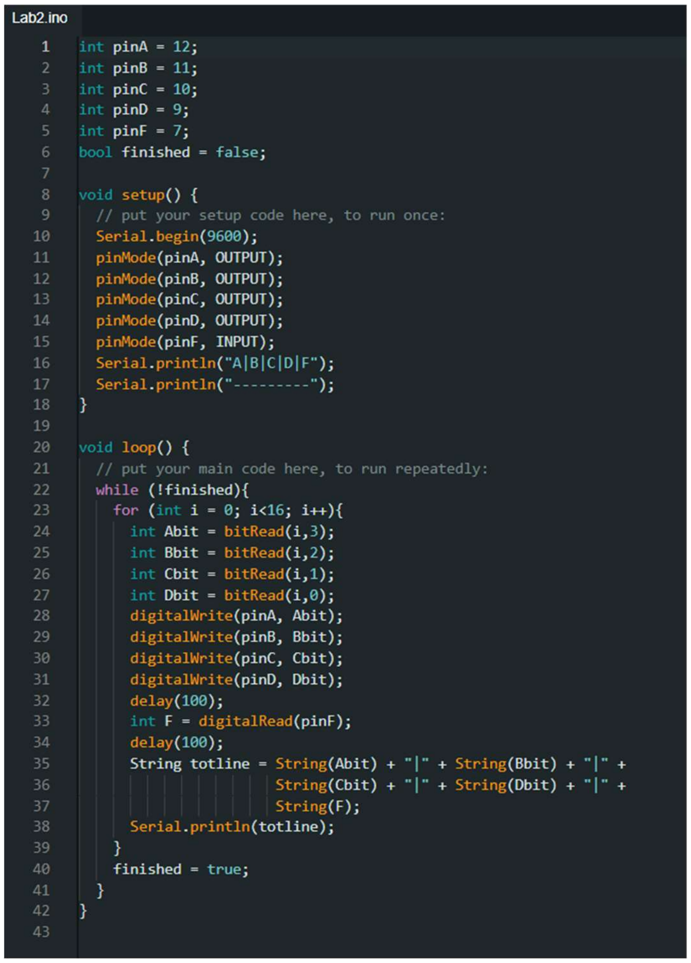


Figure 10.2



```
int pinA = 12;
int pinB = 11;
int pinC = 10;
int pinD = 9;
int pinF = 7;
bool finished = false;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(pinA, OUTPUT);
  pinMode(pinB, OUTPUT);
  pinMode(pinC, OUTPUT);
  pinMode(pinD, OUTPUT);
  pinMode(pinF, INPUT);
  Serial.println("A|B|C|D|F");
  Serial.println("---------");
}

void loop() {
  // put your main code here, to run repeatedly:
  while (!finished){
    for (int i = 0; i<16; i++){
      int Abit = bitRead(i,3);
      int Bbit = bitRead(i,2);
      int Cbit = bitRead(i,1);
      int Dbit = bitRead(i,0);
      digitalWrite(pinA, Abit);
      digitalWrite(pinB, Bbit);
      digitalWrite(pinC, Cbit);
      digitalWrite(pinD, Dbit);
      delay(100);
      int F = digitalRead(pinF);
      delay(100);
      String totline = String(Abit) + "|" + String(Bbit) + "|" +
                       String(Cbit) + "|" + String(Dbit) + "|" +
                       String(F);
      Serial.println(totline);
    }
    finished = true;
  }
}
```