CS 251 — Lecture 6

Bartosz Antczak *Instructor: Stephen Mann* *January 19, 2017*

6.1 Extending the Traffic Light Controller

Recall in the last lecture, we implemented a basic traffic light controller using a finite-state controller. Let's expand on this implementation by including a timer to change the lights and also a yellow light (rather than just green and red). The graphical representation looks like:

- Inputs: NScar, EWcar, TS
- Outputs: NSg, NSy, NSr, EWg, EWy, EWr, TR

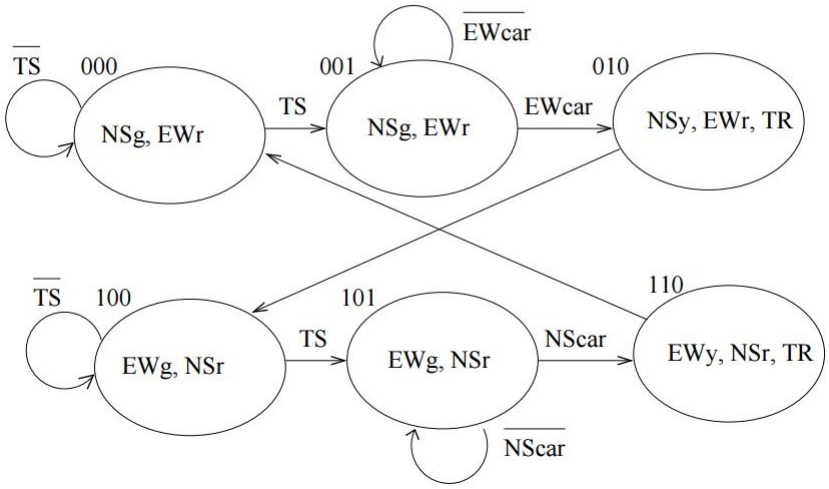

Figure 6.1: Courtesy of Prof. Mann's slides.

To analyse this diagram, we are given a particular input (which are listed in the diagram) and a current state (that's what the three binary string digits around each bubble represent). Based on the sequence of inputs, we can determine the state of each output (the outputs are listed in the diagram). From this diagram, we can construct the next-state table (on the next page):

current state	inputs			next state	current state	inputs			next state
	NS-car	EW-car	TS			NS-car	EW-car	TS	
$S_2S_1S_0$				$S_2'S_1'S_0'$	$S_2S_1S_0$				$S_2'S_1'S_0'$
0 0 0	X	X	0	0 0 0	1 0 0	X	X	0	1 0 0
0 0 0	X	X	1	0 0 1	1 0 0	X	X	1	1 0 1
0 0 1	X	0	X	0 0 1	1 0 1	0	X	X	1 0 1
0 0 1	X	1	X	0 1 0	1 0 1	1	X	X	1 1 0
0 1 0	X	X	X	1 0 0	1 1 0	X	X	X	0 0 0
0 1 1	X	X	X	X X X	1 1 1	X	X	X	X X X

Figure 6.2: Courtesy of Prof. Mann's slides.

From this table, we can construct the circuit diagram by analysing the unreduced sums of products for the output logic:

$$S_0' = \overline{S_1}\,\overline{S_0} \cdot TS + \overline{S_2}\,\overline{S_1}\,S_0 \cdot \overline{EWcar} + S_2\overline{S_1}\,S_0 \cdot \overline{NScar}$$

$$S_1' = \overline{S_2}\,\overline{S_1}\,S_0 \cdot EWcar + S_2\overline{S_1}\,S_0 \cdot NScar$$

$$S_2' = \overline{S_2}\,S_1\overline{S_0} + S_2\overline{S_1}$$

$$NSg = \overline{S_2}\,\overline{S_1},\ EWg = S_2\overline{S_1}$$

$$NSy = \overline{S_2}\,S_1\overline{S_0},\ EWy = S_2S_1\overline{S_0}$$

$$NSr = S_2,\ EWr = \overline{S_2}$$

$$TR = S_1\overline{S_0}$$

Figure 6.3: The output logic written as an unreduced sums of products. Courtesy of Prof. Mann's slides.

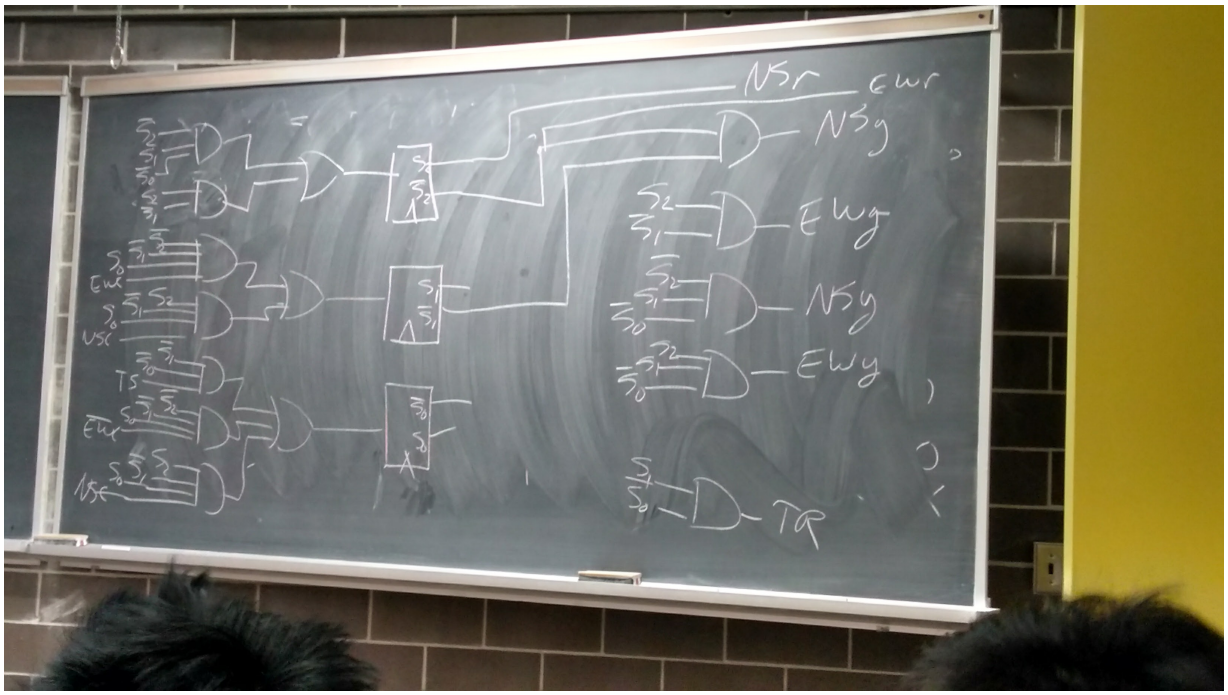

Figure 6.4: Source: Prof. Mann's drawing during today's lecture. It represents the circuit diagram constructed based on the output logic

6.2 Data Representation and Manipulation

Similar notes are found in my CS 241 course notes ;)

6.2.1 Recall MIPS

MIPS is an assembly language, which is what high-level language gets compiled into. MIPS then gets compiled to machine code. In CS 251, we're focusing on the MIPS 32-bit architecture, which means that every MIPS word is 32 bits (or 4 bytes) long (i.e., every MIPS instruction represents a binary string that is 32 digits long).

6.2.2 Character Representation

ASCII (American Standard Code for Information Interchange) is a standard that assigns numerical values to characters. For instance, the number 32 (in base-10) represents a space. 97 represents a lower-case 'a". ASCII only represents American characters, so to account for the other accents and letter from languages all over the world, *unicode* was standardized.

We'll continue more on this topic in the next lecture