Experiment 12
Digital Flip-Flop Circuits and Applications

1 Motivation

Beyond basic logic gates, there are several digital circuits that serve as building blocks for many useful applications. You will investigate several commonly used circuits related to counting and binary data management.

2 Background

A foundational logic circuit for a number of digital applications is the “flip-flop”. There are several versions, the most basic being the RS flip-flop. You will construct this circuit using basic NAND gates and then explore the integrated circuit version of the JK flip-flop to make a counter and a shift register. You will also use a universal asynchronous receiver-transmitter (UART) circuit to process 8-bit (byte) data. For some of the procedure you will input an external clock signal using the function generator. Please use the **trigger output** of the function generator, ***not*** the function output that you have many times in other experiments. Note that the grounds of the two circuits (logic board and generator) must be connected. The function generator’s trigger output is a TTL-like signal that is directly compatible with the logic board circuitry. **Do not input voltages into the logic board that are in excess of** 5.0 V!! You should monitor the function generator signal using the oscilloscope to verify that you have proper voltages. Doing this will help avoid permanent damage to the circuitry internal to the logic boards. In many applications, “external” signals are input via interface gates that condition the waveforms to create TTL-compatible outputs. Input the function generator via a NAND gate on the logic board (or two NAND gates connected in series, if the input should not be inverted) to “clean up” the signal and help protect the board’s circuitry.

3 Equipment

For this experiment, you will use:

- One “logic board” chassis for testing logic circuits
- One Tektronix MSO 2014B Digital Storage Oscilloscope
- One Tektronix P3010 10X scope probe
- One AFG2021 Arbitrary Function Generator

4 Procedure

All of the digital circuits needed for this experiment are mounted on a circuit board inside the “logic board” chassis. The $V_{CC} = +5$ V power supply is also located inside the chassis. Do not attempt to open the chassis on your own. If you are interested to see the board and circuity, ask your lab instructor to open one, if available.

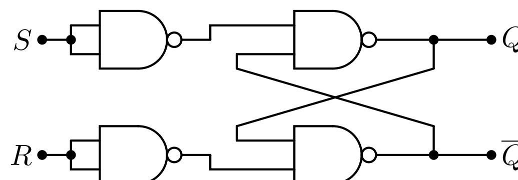
- **Gray colored connectors are inputs**: Use banana-connector patch cables to connect logic signals from various locations on the board (but not external sources). The input to a gate can be the output of a different gate. This is how more complicated digital circuits are

constructed. Inputs can also come from pull-up and pull-down circuitry associated with the switches described below.

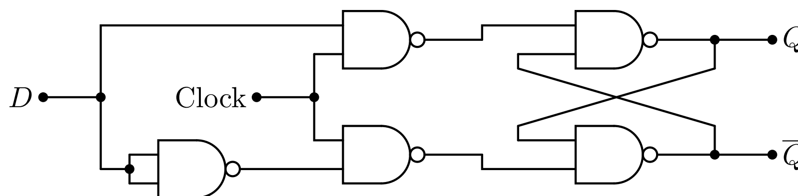
- **Green colors connectors are outputs**: The outputs of the digital circuits are made available as banana-connector terminals. You can monitor the logic level of an output using the LED indicators described below. Outputs from one gate can be used as inputs to other gates to build more complex logic circuits. Each output can be connected to multiple inputs The number of inputs that can be be connected to one single output is called the *fanout* capacity. To prevent damage by accidental shorting, a 47 Ω resistor is wired in series with each output. This reduces the fanout capacity to 4.
- **Light emitting diodes** (LEDs) can be used to indicate the digital level at selected points in a circuit: LED "on" is logical ONE, while LED "off" is logical ZERO.
- **Push-button and toggle switches** can be used to source logical ONE or logical ZERO inputs for other circuits on the board.
- **Pull-up resistors are installed internal to the chassis** that connect unused inputs to HIGH (logical ONE). This means you do not have to worry about floating inputs.

Do the following exercises:

1. Use NAND gates to construct the RS flip-flop circuit shown in Fig. 1(a). Experimentally determine how it operates. There is no single standard for describing flip-flop operation in a truth table, so you can create your own description.
2. Try applying simultaneous low-to-high transition inputs for R and S using two push-button switch sources. Are Q and $\overline{Q}$ always the same when you repeat this?
3. Construct the D flip-flop in Fig. 1(b) using a toggle switch source for the data input, D, and a push-button switch source for CLOCK. Experimentally determine the circuits function. How does the D flip-flop resolve the ambiguity of $R = 1$ and $S = 1$?



(a) Set-reset RS (or SR) flip-flop



(b) Clocked D flip-flop

Figure 1: (a) The RS flip-flop (b) The level-triggered or "transparent" D flip-flop. The D flip-flop is also available in an "edge-triggered" version.

4. Investigate the behavior of the JK flip-flops located on the logic boards. First determine how the flip-flop is triggered, i.e., does the output switch (a) any time the clock pulse is positive, (b) on the falling (negative) edge of the clock pulse, or (c) on the rising (positive) edge of the clock pulse? Experimentally determine the truth table for the flip-flop. Be sure to observe

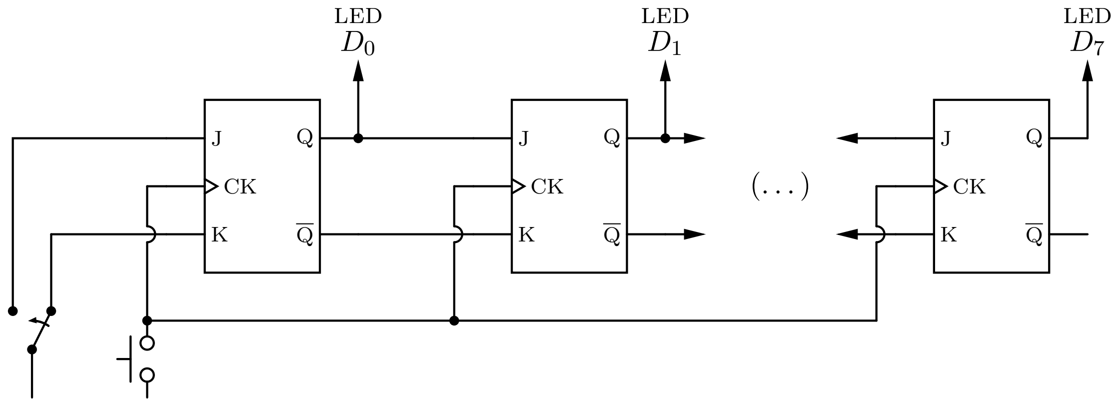


Figure 2: A circuit for an n-bit shift register. The case shown is 8-bit, with only the first two and last JK flip-flops shown. The arrows represent connections to the intermediate flip-flops. Note that unconnected inputs are wired HIGH (logical 1) internally to the logic boards.

what final state is obtained for all possible inputs $(J, K) = (0, 0)$, $(0, 1)$, $(1, 0)$, and $(1, 1)$.

5. Construct a 4-bit shift register based on the circuit shown in Fig. 2. Use a toggle switch to vary the input to the first stage and a push-button switch to provide the clock, CK. Write a brief description of the circuit's behavior in your lab notebook.
6. Create a 4-bit "circular" shift register as follows. First, load an arbitrary 4-bit word into the register using the two switches. Disconnect the toggle switch and loop Q and $\overline{Q}$ of the last stage over to J and K of the first stage. Use push-button CK to advance the register. Do you see why this is "circular"? Now create a continuous CK using the trigger (TTL) output of the function generator. Connect the generator to the logic board through a NAND gate. Set the generator's frequency to 20 Hz and monitor both CK and Q of the output stage on the scope. (Use a slow horizontal sweep rate.) What you see is a serial representation of the 4-bit word stored in the register. Try a different 4-bit word by repeating the steps above.
7. Construct a 4-bit (0–15) ripple counter based on the circuit shown in Fig. 3. Use the function generator for CK. Use the scope to measure the Q output of the first three flip flops (D_0, D_1, and D_2) along with the clock signal. Make a sketch of your measurements or print out a scope screen capture. Do not disassemble this circuit (see next step).

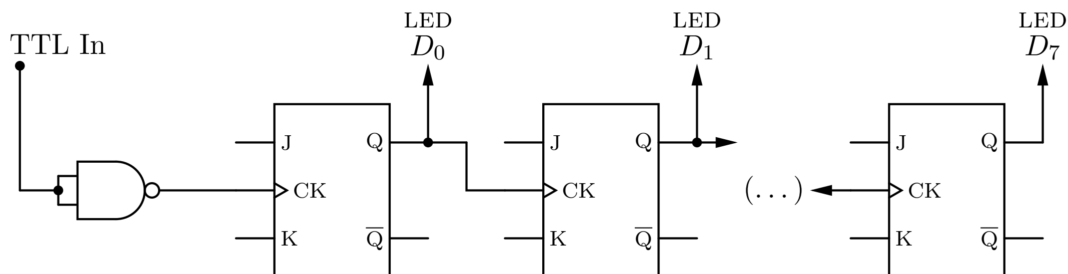
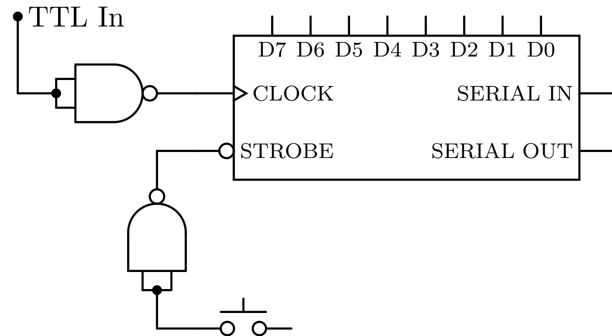
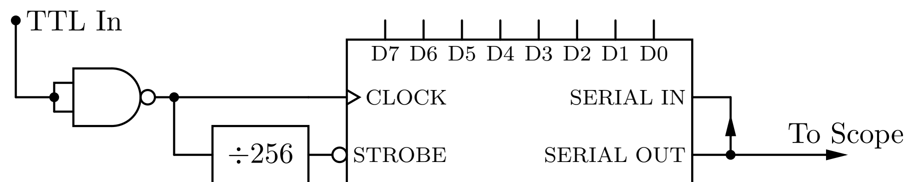


Figure 3: A circuit for an n-bit ripple counter. The case shown is 8-bit, with only the first two and last JK flip-flops shown. The arrows represent connections to the intermediate flip-flops. Note that unconnected inputs are wired HIGH (logical 1) internally to the logic boards.

8. A universal asynchronous receiver-transmitter (UART) allows sharing binary data between computing systems using a serial data format. A UART is bi-directional, meaning it is capable of both sending and receiving information. One byte representing a number or letter is entered at the UART's parallel inputs $D0$-$D7$. To transmit this information, the STROBE input is set LOW, which causes the UART to transmit the binary data one bit at a time through its SERIAL OUTput. The waveform for each transmitted bit has a duration of 16 clock cycles. The logic board's UART is set up such that the 8-bit byte word is prefaced with one 0 and appended with two 1's (11 bits total). In receive mode, data arriving at the SERIAL INput is decoded in a similar manner. The data received is represented by the eight LEDs on the logic board.



(a) Push-button–triggered UART configuration.



(b) Ripple counter–driven UART configuration. The "÷256" block represents an 8-bit counter circuit.

Figure 4: UART circuits

(a) Start by extending your ripple counter circuit in Step 7 to 8-bit (0-255).

(b) Set up the UART circuit shown in Fig. 4(a). Use toggle switches to set the input bits (D_0 to D_7) to either"0" or "1". Set the clock rate to about 100 Hz and use the push-button to strobe the UART. This causes the UART to transmit the byte data back to itself, displayed on the LEDs.

(c) Now use your 8-bit ripple counter to cyclically strobe the byte input as shown in Fig. 4(b). The counter produces one strobe every 256 clock pulses, and the UART will transmit the same byte data back to itself repeatedly. Use the oscilloscope to measure the clock and the serial output. Observe what happens as you toggle the input data, D_n, between "0" and "1". Record several traces of the clock and serial output and identify where the information contained in the input bits appears in the output.

(d) Connect SERIAL OUT $\rightarrow$ SERIAL IN and SERIAL IN $\rightarrow$ SERIAL OUT of the UART in a *separate* logic board. (Ask your lab instructor for advice on how to do this.) You need to input the respective signals using two NAND gates in series. You also need to connect the grounds of the two boards. You can use a parallel signal from the generator as the clock for the second board. Send a message from one board to the other using ASCII encoded data (see Table 1).

Table 1: Printable Non-Whitespace ASCII Characters

Char		Hex	Binary	Char		Hex	Binary	Char		Hex	Binary
33	!	0x21	0010 0001	65	A	0x41	0100 0001	97	a	0x61	0110 0001
34	"	0x22	0010 0010	66	B	0x42	0100 0010	98	b	0x62	0110 0010
35	#	0x23	0010 0011	67	C	0x43	0100 0011	99	c	0x63	0110 0011
36	$	0x24	0010 0100	68	D	0x44	0100 0100	100	d	0x64	0110 0100
37	%	0x25	0010 0101	69	E	0x45	0100 0101	101	e	0x65	0110 0101
38	&	0x26	0010 0110	70	F	0x46	0100 0110	102	f	0x66	0110 0110
39	'	0x27	0010 0111	71	G	0x47	0100 0111	103	g	0x67	0110 0111
40	(	0x28	0010 1000	72	H	0x48	0100 1000	104	h	0x68	0110 1000
41	)	0x29	0010 1001	73	I	0x49	0100 1001	105	i	0x69	0110 1001
42	*	0x2A	0010 1010	74	J	0x4A	0100 1010	106	j	0x6A	0110 1010
43	+	0x2B	0010 1011	75	K	0x4B	0100 1011	107	k	0x6B	0110 1011
44	,	0x2C	0010 1100	76	L	0x4C	0100 1100	108	l	0x6C	0110 1100
45	-	0x2D	0010 1101	77	M	0x4D	0100 1101	109	m	0x6D	0110 1101
46	.	0x2E	0010 1110	78	N	0x4E	0100 1110	110	n	0x6E	0110 1110
47	/	0x2F	0010 1111	79	O	0x4F	0100 1111	111	o	0x6F	0110 1111
48	0	0x30	0011 0000	80	P	0x50	0101 0000	112	p	0x70	0111 0000
49	1	0x31	0011 0001	81	Q	0x51	0101 0001	113	q	0x71	0111 0001
50	2	0x32	0011 0010	82	R	0x52	0101 0010	114	r	0x72	0111 0010
51	3	0x33	0011 0011	83	S	0x53	0101 0011	115	s	0x73	0111 0011
52	4	0x34	0011 0100	84	T	0x54	0101 0100	116	t	0x74	0111 0100
53	5	0x35	0011 0101	85	U	0x55	0101 0101	117	u	0x75	0111 0101
54	6	0x36	0011 0110	86	V	0x56	0101 0110	118	v	0x76	0111 0110
55	7	0x37	0011 0111	87	W	0x57	0101 0111	119	w	0x77	0111 0111
56	8	0x38	0011 1000	88	X	0x58	0101 1000	120	x	0x78	0111 1000
57	9	0x39	0011 1001	89	Y	0x59	0101 1001	121	y	0x79	0111 1001
58	:	0x3A	0011 1010	90	Z	0x5A	0101 1010	122	z	0x7A	0111 1010
59	;	0x3B	0011 1011	91	[	0x5B	0101 1011	123	{	0x7B	0111 1011
60	<	0x3C	0011 1100	92	\	0x5C	0101 1100	124	|	0x7C	0111 1100
61	=	0x3D	0011 1101	93	]	0x5D	0101 1101	125	}	0x7D	0111 1101
62	>	0x3E	0011 1110	94	^	0x5E	0101 1110	126	~	0x7E	0111 1110
63	?	0x3F	0011 1111	95	_	0x5F	0101 1111				
64	@	0x40	0100 0000	96	`	0x60	0110 0000				

Other ASCII characters include control codes (`0x00` through `0x1F` plus `0x7F`) and space (`0x20`). If the high bit of a byte is set (`0x80` through `0xFF`), its interpretation depends on which code page is being used; the most commonly used code pages in English-speaking countries are ISO 8859-1 (International standard encoding, Western European languages), CP1252 (Microsoft Windows, Western European languages; superset of ISO 8859-1), and UTF-8 (Unicode Consortium, all languages, 8-bit encoding). These are all supersets of ASCII.