Lattice™
Semiconductor
Corporation

Bar Code Reader

Introduction

The Universal Product Code was first implemented by the grocery industry in 1973 as a method for improving inventory control and checkout times. As its benefits were realized, it soon spread throughout the retail industry. UPC Version A is a simple numeric only code encoding a 12-digit number into a continuous, fixed length symbol. UPC Version E is similar to Version A, except that it only encodes six digits.

Figure 1. UPC Version A, 12-Digit Code

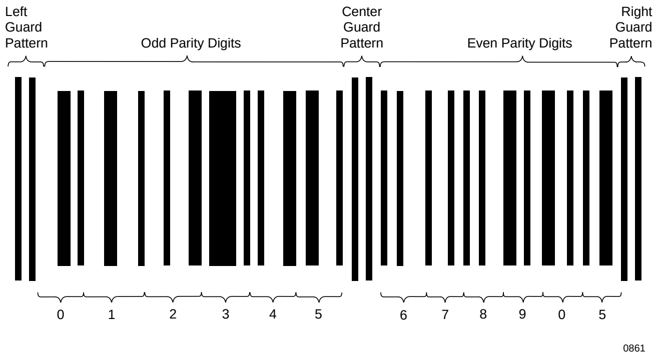
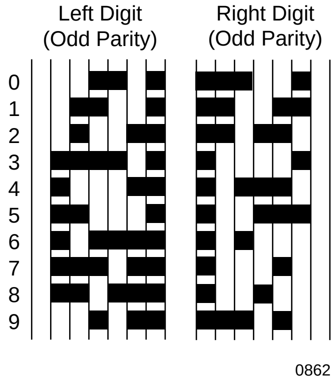


Figure 1 shows an example of the Version A pattern which encodes the number sequence 01234567890. The Version A pattern is divided into two halves. Each half consists of a guard pattern and six digits, with a guard pattern separating them. The patterns used to encode the digits on the left half have an odd number of bits (odd parity), and the patterns used to encode the digits on the right half have an even number of bits (even parity). This allows error checking to be performed, and a symbol which has been scanned backwards can be detected by detecting even parity codes being received before odd parity codes.

The basic width of a bar or space is determined by the width of the guard bars. This is defined to be in the range of 10.4 to 26 mils wide. Bar and space widths can be anywhere from one to four guard bar widths wide. The guard bars are usually printed with a slightly longer length than the data bars to allow a greater scanning tilt angle.

As mentioned before, the code which is used to represent numbers on the left side of the code is different from the code used on the right half. The data is encoded as a two-of-seven code using two bars and two spaces to describe 20 unique patterns. These 20 patterns encode the ten numbers with both odd and even parity. The patterns are shown in Figure 2.

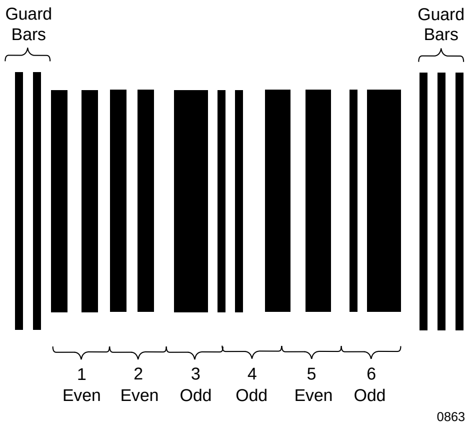
Figure 2. UPC Encodation Patterns



The first digit of the 12 is called the number system digit and is used to indicate the type of product the code is identifying. The next five digits are the manufacturer's ID number as assigned by the Uniform Code Council, and the following five digits are the item ID number assigned by the manufacturer. The last digit is a check digit. The value of this digit is based on the weighted sum of all of the other digits in the number. Using a weighted sum allows checking for transposition errors to be performed if the number is manually entered.

Figure 3 shows an example UPC Version E symbol which encodes the digits 123456. This code was specified to label small items. Because many of the digits in the manufacturer's ID number and item ID number are frequently zeros, by suppressing these zeros using a standard compression process the number of digits can be reduced from 12 to five. The last digit in the symbol indicates the type of suppression used in defining the symbol. This pattern uses intermixed digits of odd and even parity using the same encoding patterns shown in Figure 1.

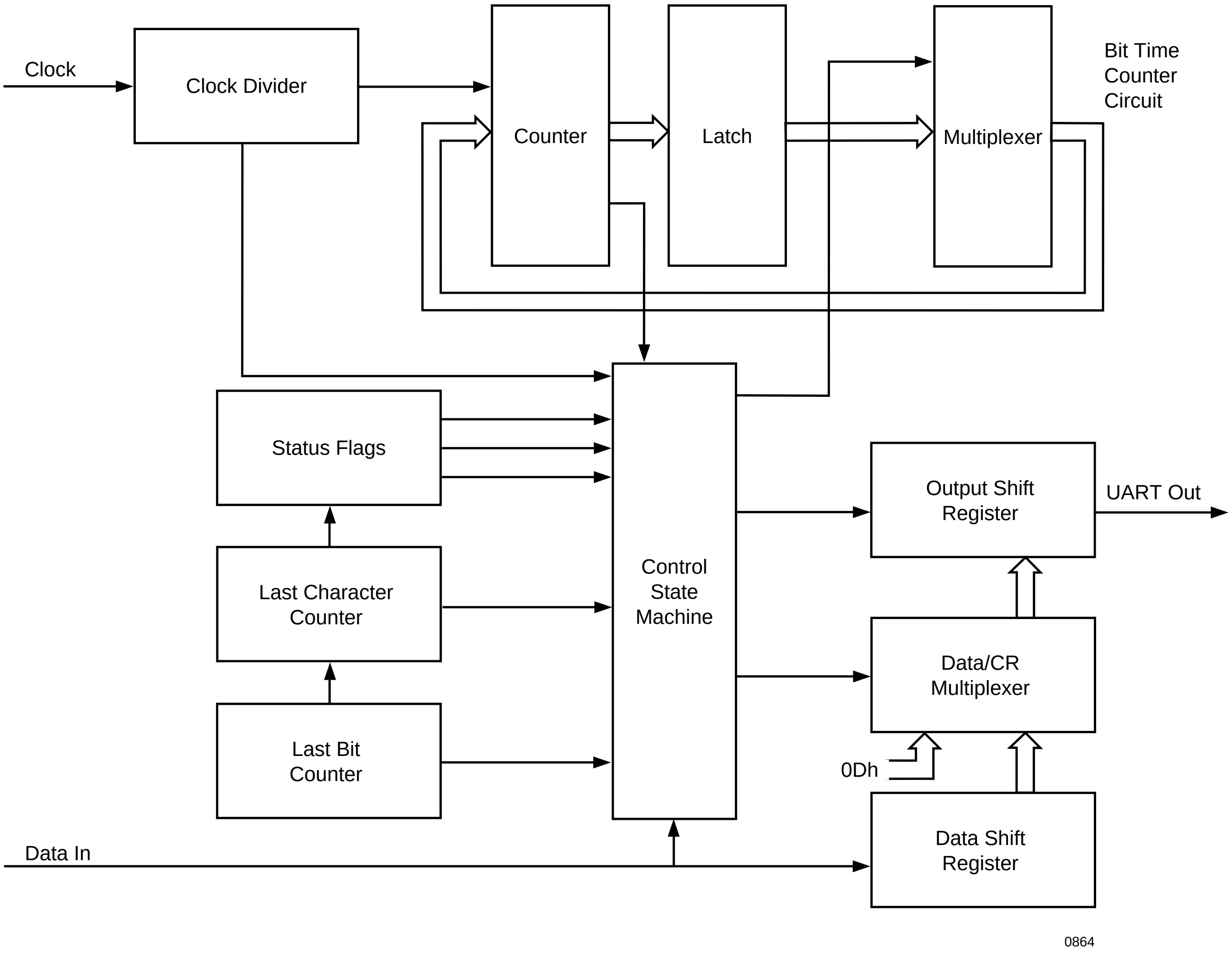
Figure 3. UPC Version E, Six-Digit Code



This example design shows how to use a pLSI device to implement a standard Universal Product Code (UPC), or Bar Code Reader in a single chip. The chip will receive digital signals from a bar code wand, determine the timing of those signals, separate the data from the wavetrain and transmit that data asynchronously to a PC RS232 serial port. Both UPC Version A (12-digit) and UPC version E (six-digit) types will be decoded.

The main components of the block diagram in Figure 4 are described in the following section.

Figure 4. UPC (Bar Code) Reader Block Diagram



Theory of Operation

The UPC (or Bar Code) Reader consists of six major parts: the Clock Divider, Bit Time Counter, Data Shift Register, Control State Machine, Status Bits and UART. A complete description of each component follows.

Clock Divider

The Clock Divider Circuit (Figure 8) can be divided into three parts. The divider takes the 1.8432 MHz reference clock and first divides it by 16 using a four-bit counter CBU44 to generate 115.2 KHz. This is used as the main reader frequency, and was chosen primarily for two reasons. First, a frequency was needed that was fast enough so the clock skew at the edges of the received wand data is minimal in relation to Terminal Count (when the data is valid). Clock skew is caused by the data edge not being aligned with the rising edge of the clock. They can be, in the worse case, a complete clock period apart. If, for example, the data period is very small and the clock frequency is very slow, the width of the first guard bar will correspond to only a few counts. Again under worse case conditions, the clock skew at the starting edge of data added to the clock skew at the end of data can cause the data to be either sampled twice, or not at all. This condition is worsened if the data periods are varying. Secondly, the frequency had to be slow enough so that a nine-bit counter would be sufficient to determine the width of the guard bar. Thus 115.2 KHz was selected as the reader frequency.

The 115.2 KHz is further divided using a two-bit counter CBU42. By preloading one and counting up, the end result is 38.4 KHz. In the final stage, 38.4 KHz is divided by four using CBU22 to generate 9.6 KHz, which is used in the transfer of data over the UART.

Bit Time Counter

The Bit Time Counter Circuit (Figure 9) consists of two counters, storage flip-flops and logic gates. The two counters namely CBUD8 and CBUD1 form the nine-bit counter which is used in determining the width of the first guard bar in the code. Once the leading edge of the first guard bar is received, the counters collectively start counting down. At the end of the first guard bar, the counters will contain a value relative to the width of the guard bar. For the most reliable reading, the data should be sampled in the middle of a bar or space. To achieve this, the value corresponding to the relative width of the first guard bar is divided by two and stored in the three sets of flip-flops namely two FD24s and one FD21. Thus, for each subsequent bar or space, the shifted value is preloaded into the counters and the counters are allowed to count up to terminal count. Terminal count is then used to strobe the input data into the data shift register. The terminal count circuitry which determines when the data becomes valid consists of two parts, namely the edge detector and the CNTTC detector. The edge detector generates a pulse when the data makes a transition from either low to high or high to low and reloads the nine-bit counter with the stored width value. The duration of the pulse is one clock period. The edge pulse also resets the toggle flip-flop which is part of the CNTTC detector circuitry. Thus when the nine-bit counter reaches zero and the sample signal goes high, the CNTTC flip-flop is set, which enables the data shift register, and data is latched. If an edge is not detected, the toggle flip-flop lets the counter count the stored guard bar width twice before CNTTC flip-flop is set and data is taken. The actual pDS code for this portion of the Bar Code Reader can be found in Figure 17.

The reason behind having the edge detector circuitry is to align the CNTTC pulses with the center of the data (bars and spaces). Since it is extremely hard to move the wand at a constant rate, the data pulses tend to have varying periods. If absolutely no alignment is done, the CNTTC pulses become invalid once the accumulated change in the data pulses is greater than half the width of the guard bar. Thus, by starting the nine-bit counter at the edges of the data, the abovementioned problem is greatly reduced. However, the problem still remains if the data periods reduce or enlarge more than half the width of the first guard bar. Therefore, it is recommended to move the wand as steadily as possible to keep the data pulses within a small margin, roughly half the width of the first guard bar.

Data Shift Register

The Data Shift Register (Figure 12) consists of two four-bit shift registers (SRR24). The Data Shift Register is used to store the incoming data until a complete character has been received. As discussed in the description of the Universal Product Code section, the bar code is two-of-seven code. Thus, a complete character is received after every seven bits. Moreover, Version E and Version A codes are six and 12 characters long respectively. Thus, two counters are needed to keep track of the bit and the character counts. Although not directly part of the Data Shift Register, two CBU34 counters keep track of the number of bits and characters received. Since CBU34s are four-bit counters, they are preloaded with nine for the bit count and ten for the character count respectively. The character counter, when reading the Version A case, is reset after the sixth character to accommodate the 12 characters in the bar code. After counting the seven bits, the bit counter generates LASTBIT signal which tells the rest of the logic a complete character has been received. The character counter, on the other hand, after receiving six characters asserts the LASTCHAR signal. The LASTCHAR signal is used by the Control State Machine to define various machine states.

Control State Machine

The Control State Machine (Figures 14 and 15) is the heart of the design. It is based on three state determining variables, namely SB0, SB1 and SB2. The job of the state machine is to detect and generate the basic width timing signals, generate the control signals for the Data Shift register and detect errors. The state diagram is shown in Figure 5. Figure 6 shows a typical waveform highlighting the various machine states. A complete description of each state follows.

Figure 5. State Diagram

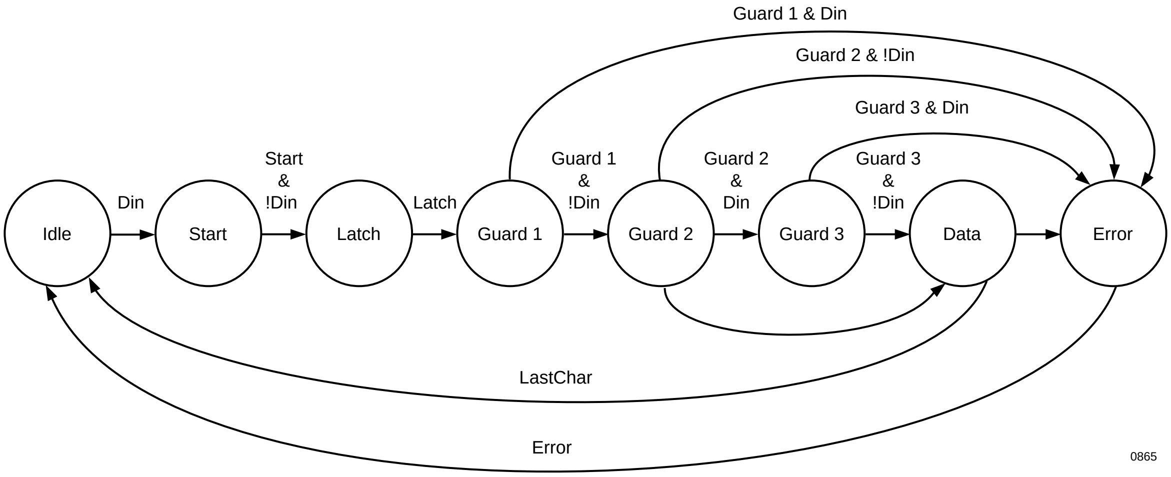
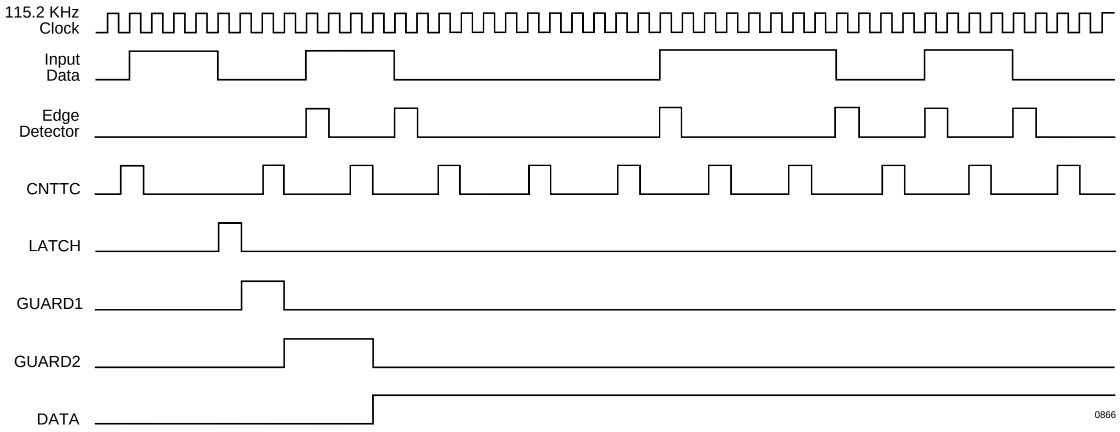


Figure 6. Data Waveforms Showing Machine States



The first state is the IDLE state. In this state, the Reader is waiting for a low-to-high data transition. This corresponds to the wand and seeing the first guard bar signifying the start of the bar code. IDLE is reached after six consecutive zeros are seen in the input data stream signifying the end of the bar code. IDLE is also reached immediately after an error or after the end of the first half of a Version A pattern.

Once a low-to-high transition is detected, the Reader goes into the START state. As Figure 5 shows, this is initiated by a DIN value which means the data received was a logic high. Another point to note is START can only be reached from the IDLE state. Once in the START state, the Bit Time Counter continuously decrements from zero. This state continues until the end of the first guard bar.

The end of the first guard bar is detected when a high-to-low data transition takes place. This places the Reader in the LATCH state, which is only a single 115.2 KHz bit wide. LATCH disables the Bit Time Counter and loads the counted value into the storage flip-flops. Remember that the value stored is actually one-half the counted value. This is done to shift the sampling point to the middle of the bit period. LATCH also loads the stored value in the flip-flops into the Bit Time Counter.

Once the value is loaded in the Bit Time Counter, the Reader enters the GUARD1 state. In this state, the Bit Time Counter is again enabled but instead of counting down, the counter counts up. The Edge Detector circuit is also enabled. Once the Bit Time Counter reaches zero, the sample signal goes high. This sets the CNTTC flip-flop which enables the shift register and data is read, and should be logic low. If it is, the Reader goes into the GUARD2 state. Otherwise, IDLE state is reached following the ERROR state. Before entering the GUARD2 state but after CNTTC, the stored bit width value is again loaded into the counter.

As Figure 5 shows, the GUARD2 state is reached when the Reader is in the GUARD1 state and a !DIN is received. In the GUARD2 state, the Bit Time Counter is again enabled and allowed to count up. However, the CNTTC flip-flop is disabled so the next Sample does not force the Shift Register to take data. This is achieved through the toggle flip-flop which is part of the CNTTC circuitry. Thus when the Bit Time Counter counts up to zero, the bit width value is again loaded into the counter and is allowed to count up. However, the Sample signal does reset the toggle flip-flop so the next Sample signal sets the CNTTC flip-flop which in turn enables the Shift Register and data is taken in. Remember, the Edge Detector circuitry was enabled in the GUARD1 state. Thus, what really happens is after the first Sample is detected in the GUARD2 state, the Edge Detector circuitry also notices the low-to-high transition of data and loads the stored bit width into the counter and resets the CNTTC flip-flop. Thus, the GUARD2 state is the first time data alignment takes place. Once the Bit Time counter counts up to zero and Sample is enabled, data is taken in. If the data is logic high, the control circuitry checks if LASTHALF status bit is high. If it is, then the Reader goes into the GUARD3 state. Otherwise, the Reader enters the DATA state. If instead of a logic high, a logic low is received, the Reader goes to the IDLE state following the ERROR state.

In the GUARD3 state, the same sequence of steps are repeated as in the GUARD2 state except that !DIN is the correct data type used to bring the Reader in the DATA state. Also, the Edge Detector circuitry is in effect and influences the CNTTC flip-flop. One important point to note is the GUARD3 state is entered only if the LASTHALF status bit is high. The GUARD3 is the extra guard bit found in the center guard pattern of a Version A code. The GUARD3 state looks for that bit to be low at the proper time to enter the DATA state and goes to the ERROR state if it is not.

In the DATA state, the Reader is ready to receive data bits, assemble them into seven-bit data words and transfer them out over the UART. In the DATA state, Edge Detector circuitry is again enabled and, depending on the edge transitions, Bit Time Counter alignment with input data takes place. If an edge is not detected, the toggle flip-flop ensures proper operation of the Reader. Once six characters are read, the Reader goes into the IDLE state. The IDLE state can also be reached if six consecutive zeros are read as discussed in the IDLE state description above.

The ERROR state is reached if in either of the GUARD bits the correct data type is not read in. Thus the ERROR state indicates an abnormal condition has been detected. When this happens, the Reader sets the ERROR flag and reverts to the IDLE state and awaits the start of a new character.

Table 1 provides a list of all the states, including a short description of each state and a list of conditions which cause that particular state to occur.

Table 1. State Table

S 2	S 1	S 0	Name	Equation	Description
0	0	0	IDLE	DEFAULT	Waiting for the first guard bit to appear
0	0	1	START	DIN&IDLE	Bit width counter is decremented
0	1	0	LATCH	START&!DIN	Counted width stored and loaded back into the bit width counter
0	1	1	GUARD1	LATCH	Verifies a space was seen
1	0	0	GUARD2	GUARD1 & !DIN	Verifies a bar was seen
1	0	1	GUARD3	GUARD2 & LASTHALF & DIN	Verifies a space was seen
1	1	0	DATA	GUARD2 & DIN & !LASTHALF # GUARD3 & !DIN & LASTHALF	Device ready to receive data
1	1	1	ERROR	GUARD1 & DIN & CNTTC # GUARD2 & !DIN & CNTTC # GUARD3 & DIN & CNTTC	Abnormal condition has been detected

Status Bits

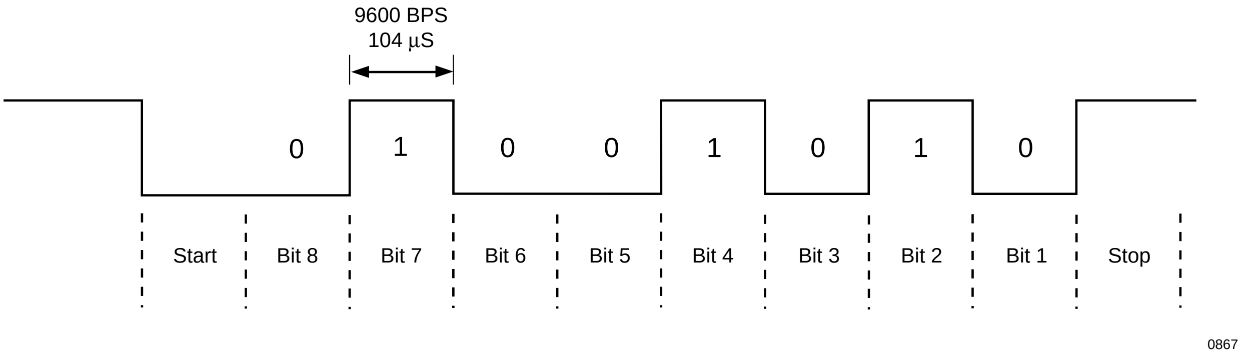
Four Status Bits are used (shown in Figures 12 and 13) to keep track of the progress of the read. LASTCHAR is used to store the fact that six characters have been read. This flip-flop is reset when the Reader is in the IDLE state. ERRORL stores the fact that an error occurred while reading the code. As mentioned before, the ERROR state occurs when in either of the GUARD bits, the correct data type is not read. Since a seven-bit data word is read and an eight-bit data transfer word is used, the ERRORL bit is transmitted as the most significant bit of each data word transferred out. Thus, the data-receiving device on the other end of the UART can check the eighth bit and determine if any errors had occurred while reading the Bar Code. The ERRORL flip-flop is also reset when the Reader is in the IDLE state. The LASTHALF bit keeps track of which part of the Version A code is being scanned. Since the two halves of the Version A code are decoded independently (the reader goes into the IDLE state after the first six characters are read), the logic needs to keep track of which half is being processed. When the character counter counts the first six characters, the LCHAR signal goes high which in turn sets the LASTHALF flip-flop. The LASTHALF bit is used in decoding the extra GUARD bit present in the center guard pattern which is the beginning of the second half of the code. The logic knows if the LASTHALF bit is low, the DATA state follows the GUARD2 state. Otherwise, the GUARD3 state follows the GUARD2 state. The DATA state comes after the GUARD3 state. Reset of the LASTHALF flip-flop is either caused by reading six consecutive zeros or when complete 12 characters of the Version A code are read. Finally, DATAREADY is used to gate the transfer of a data word from the Data Shift Register to the Output Shift Register. From the Output Shift Register, data is transferred over the UART. DATAREADY flip-flop is set every time seven bits are read. Reset is caused by the DATTAKEN signal.

Note: LASTHALF sets and resets another flip-flop called SECOND_HALF shown in Figure 12. SECOND_HALF is only used in decoding the last bit of the twelfth character in Version A code. The reason for SECOND_HALF is because the last bit of even parity digits is always logic low. When the last bit is read, the Reader immediately goes into the IDLE state without transferring the correct byte out because as soon as LCHAR is enabled, the LASTHALF flip-flop is reset. This takes the state machine out of the DATA state and into the IDLE state. By having the state machine depend on SECOND_HALF rather than LASTHALF for the twelfth character, the problem is resolved.

UART

The UART circuitry (shown in Figure 16) consists primarily of a shift register SRR31. Also shown on the schematic are two other shift registers and four sets of multiplexers. All these components are used in transferring the data properly. The multiplexers, namely MUX22, select between data from the Data Shift Register and a hard coded 0D Hex (ASCII Carriage Return). The carriage return is sent whenever six consecutive ones are received. It informs the receiver on the other end of the UART that the bar code has been read. The shift register SRR38 is the

Figure 7. Serial Data Word Diagram



Output Shift Register. The data which accumulated in the Data Shift Register is transferred to the Output Shift Register once the DATTAKEN signal goes high which is driven by the DATAREADY signal. In the Output Shift Register, using the second shift register SRR31 for clocking purposes, data is converted into the standard RS232 format and shifted into the UART Shift Register. The UART supports TTL signals. Thus a level shifter is needed at the output of the UART shift register. Figure 7 shows the data format out of the UART Shift Register.

Hints on Translating Bar Code Data to ASCII

The pLSI device receives data from the wand and tranfers it out through the UART Register. The UART Register supports TTL logic levels. An RS232 level shifter is needed before the data can be properly processed by the PC. The data format is: eight data bits, no parity, one stop bit. The data rate is 9600 BAUD. The PC's serial port has to configured accordingly. As mentioned in the Universal Product Code description, the data words either have even parity or odd parity. Thus a look-up table is needed so that the received data words can be converted to their ASCII equivalent. Each type of parity has ten different codes for the ten digits. Since the wand can be scanned either from left to right or from right to left, the look-up table has to have 40 entries. Once the look-up table is implemented, the received data can be compared and the proper ASCII code printed out. The look-up table is as follows:

Left-to-Right Scanning:

Number	Odd Parity (Hex)	Even Parity (Hex)
0	0D	72
1	19	66
2	13	6C
3	3D	42
4	23	5C
5	31	4E
6	2F	50
7	3B	44
8	37	48
9	0B	74

Right-to-Left Scanning:

Number	Odd Parity (Hex)	Even Parity (Hex)
0	58	27
1	4C	33
2	64	1B
3	5E	21
4	62	1D
5	46	39
6	7A	05
7	6E	11
8	76	09
9	68	17

Figure 8. Clock Divider

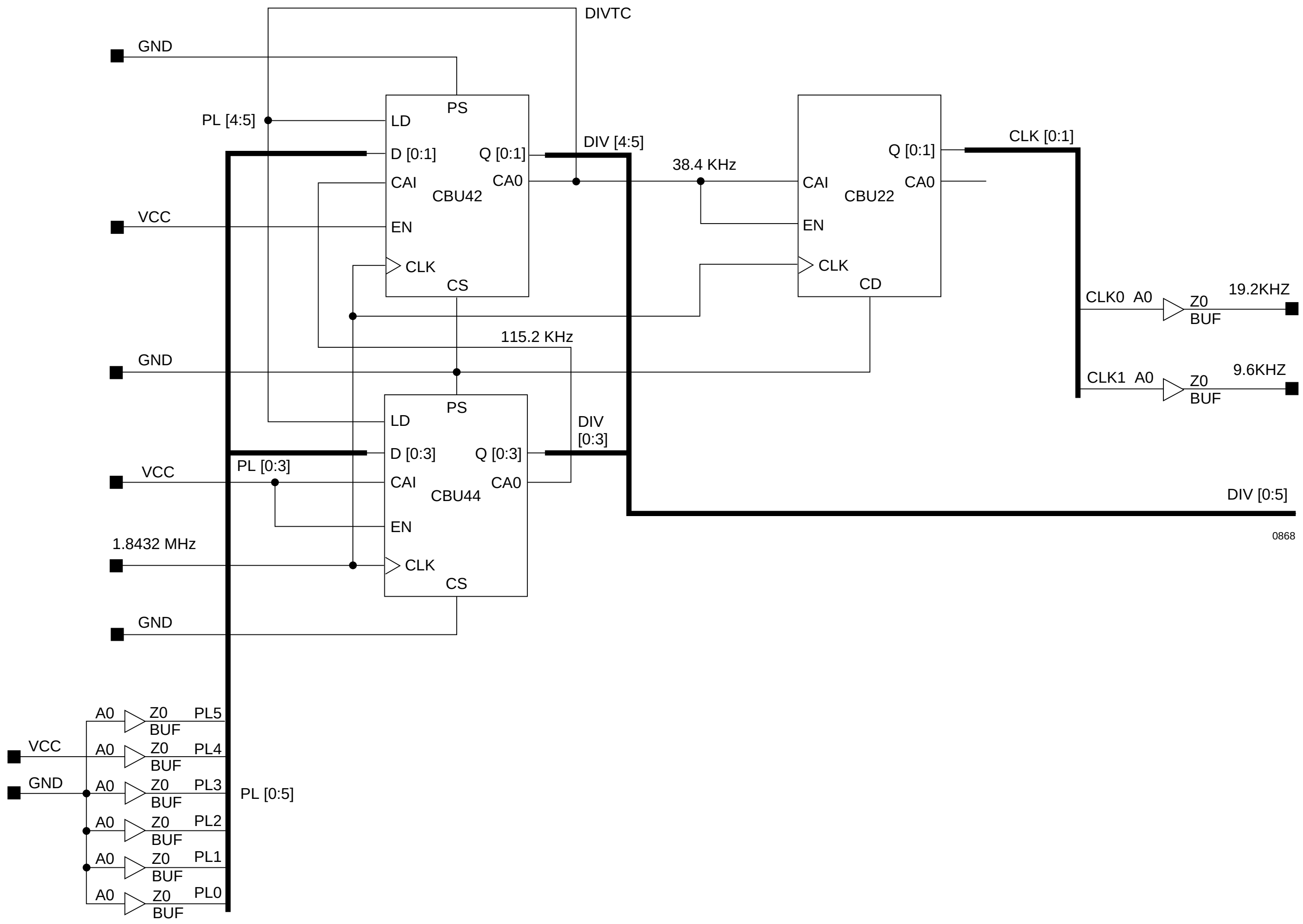


Figure 9. Bar Width Clock Circuitry

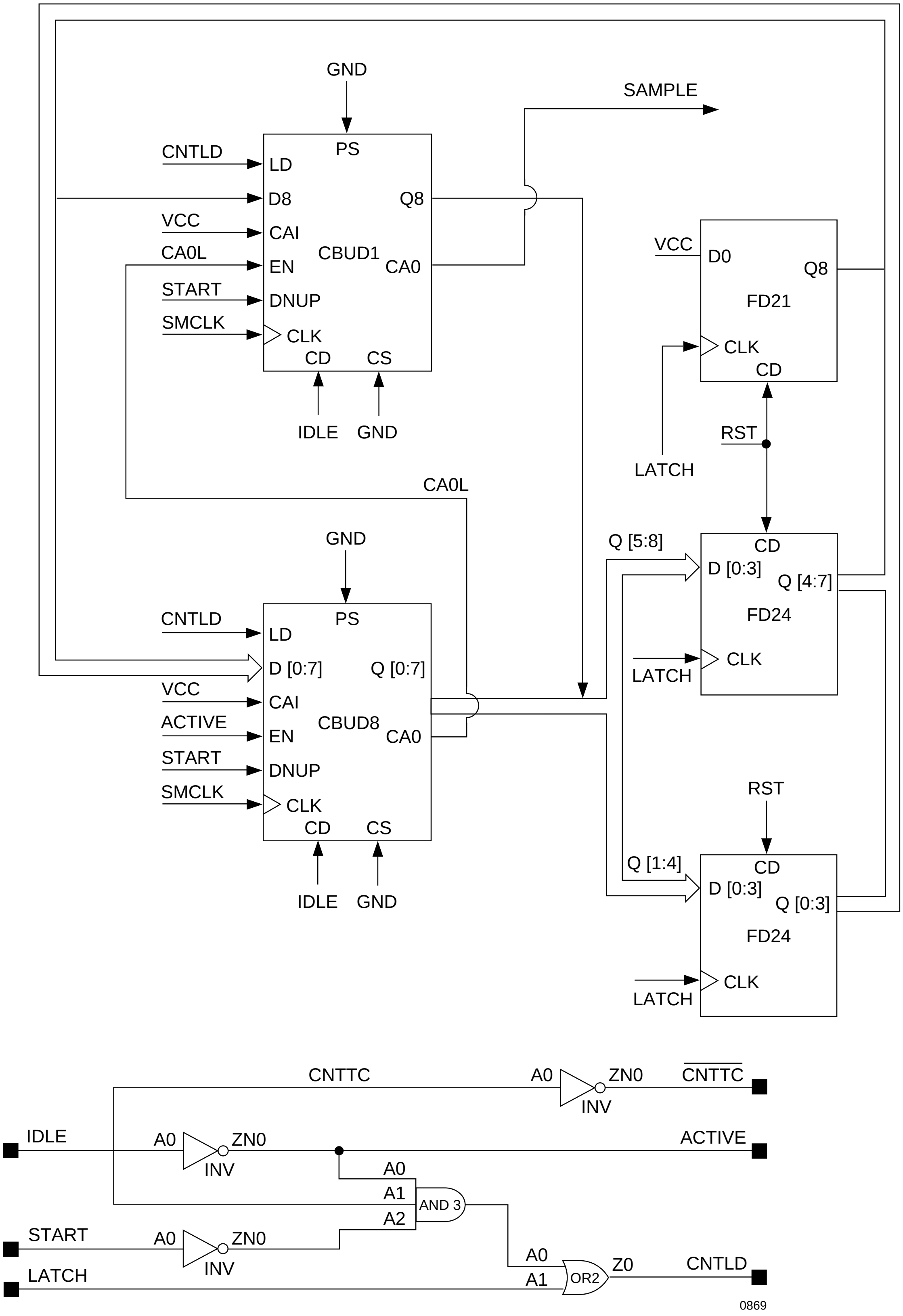
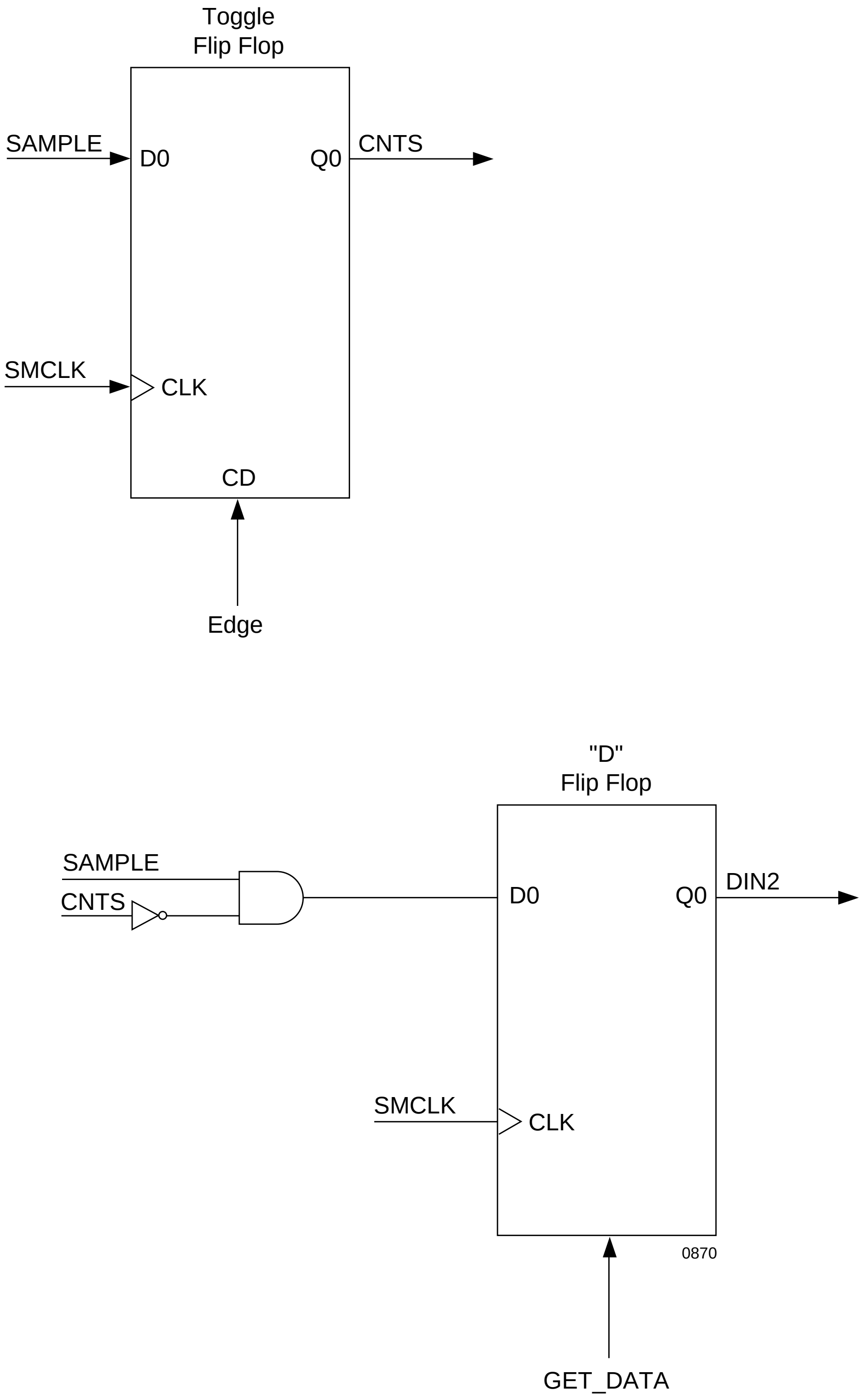


Figure 10. CNTTC Detector Circuitry



GET_DATA = RST # (EDGE & GUARD2) #
(EDGE & GUARD3) #
(EDGE & DATA)

Figure 11. Edge Detector Circuitry

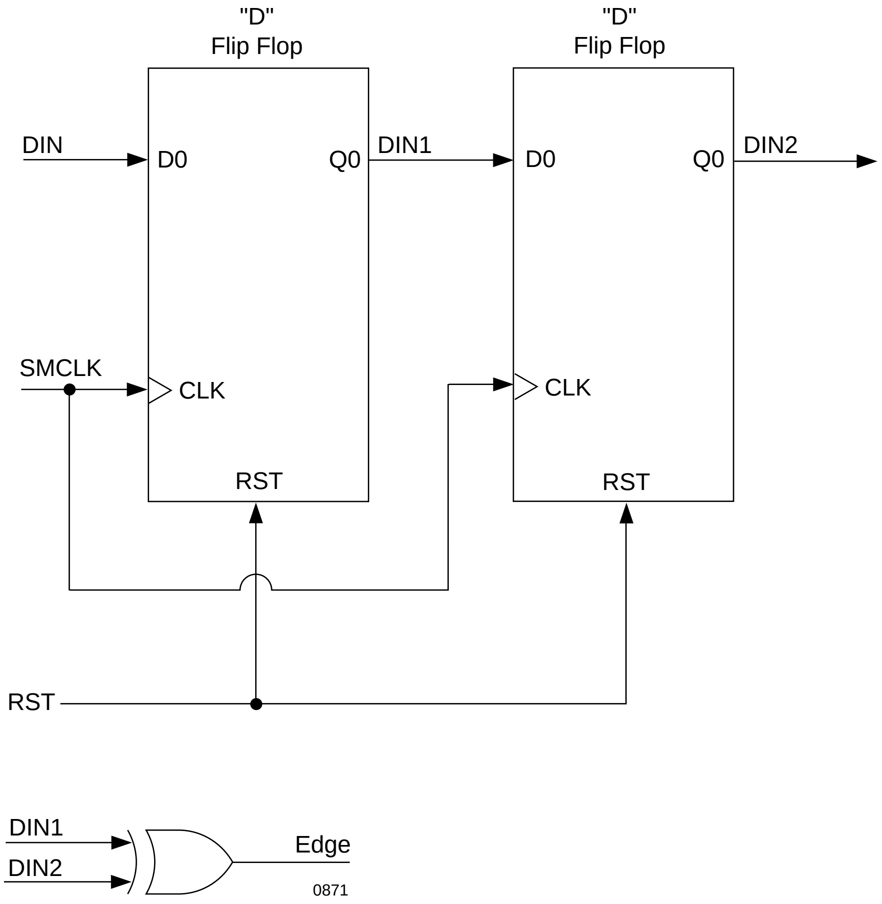


Figure 12. Data Shift Register and Status Bits

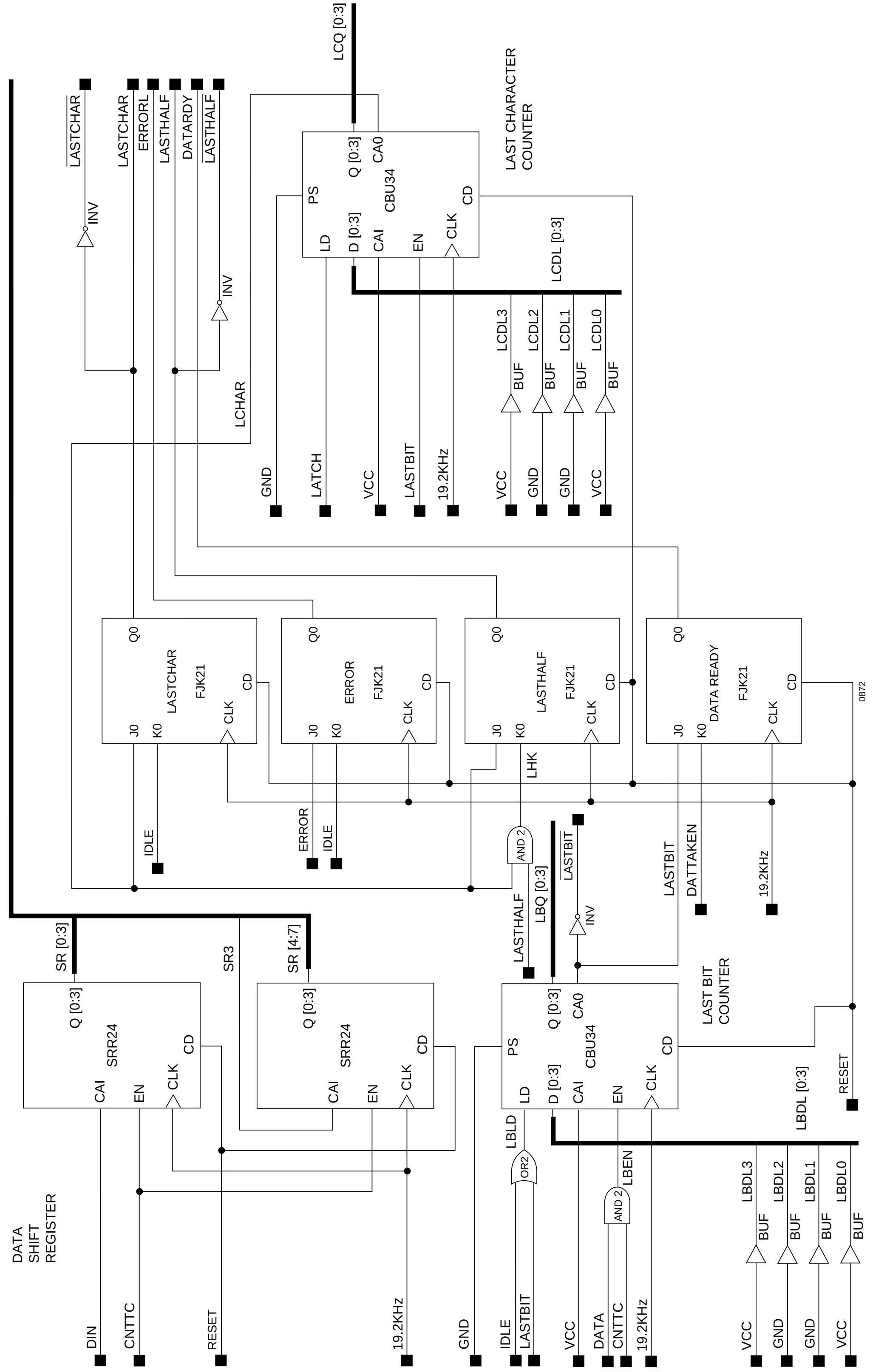


Figure 13. Second Half Circuit

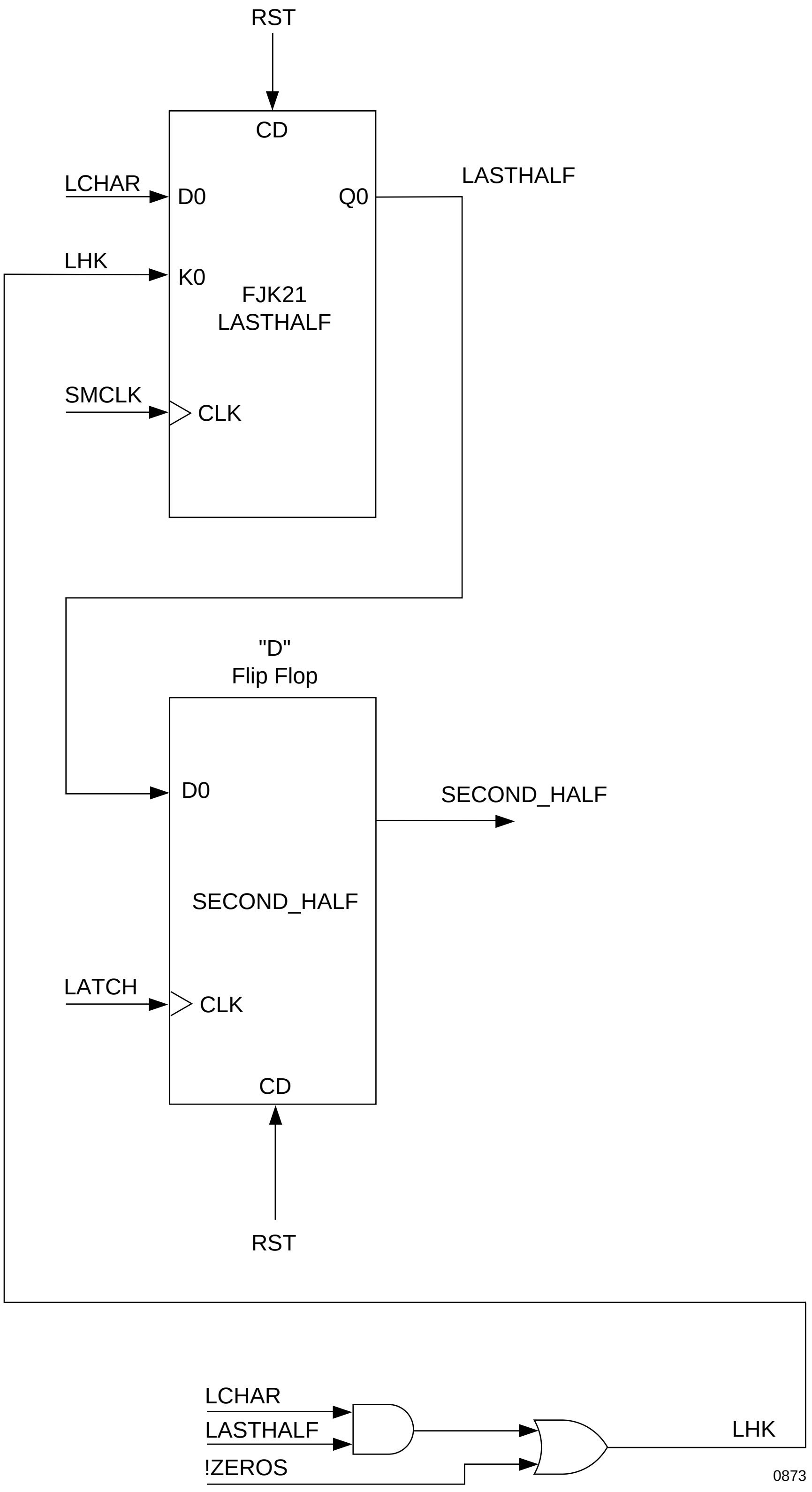


Figure 14. State Machine Registers

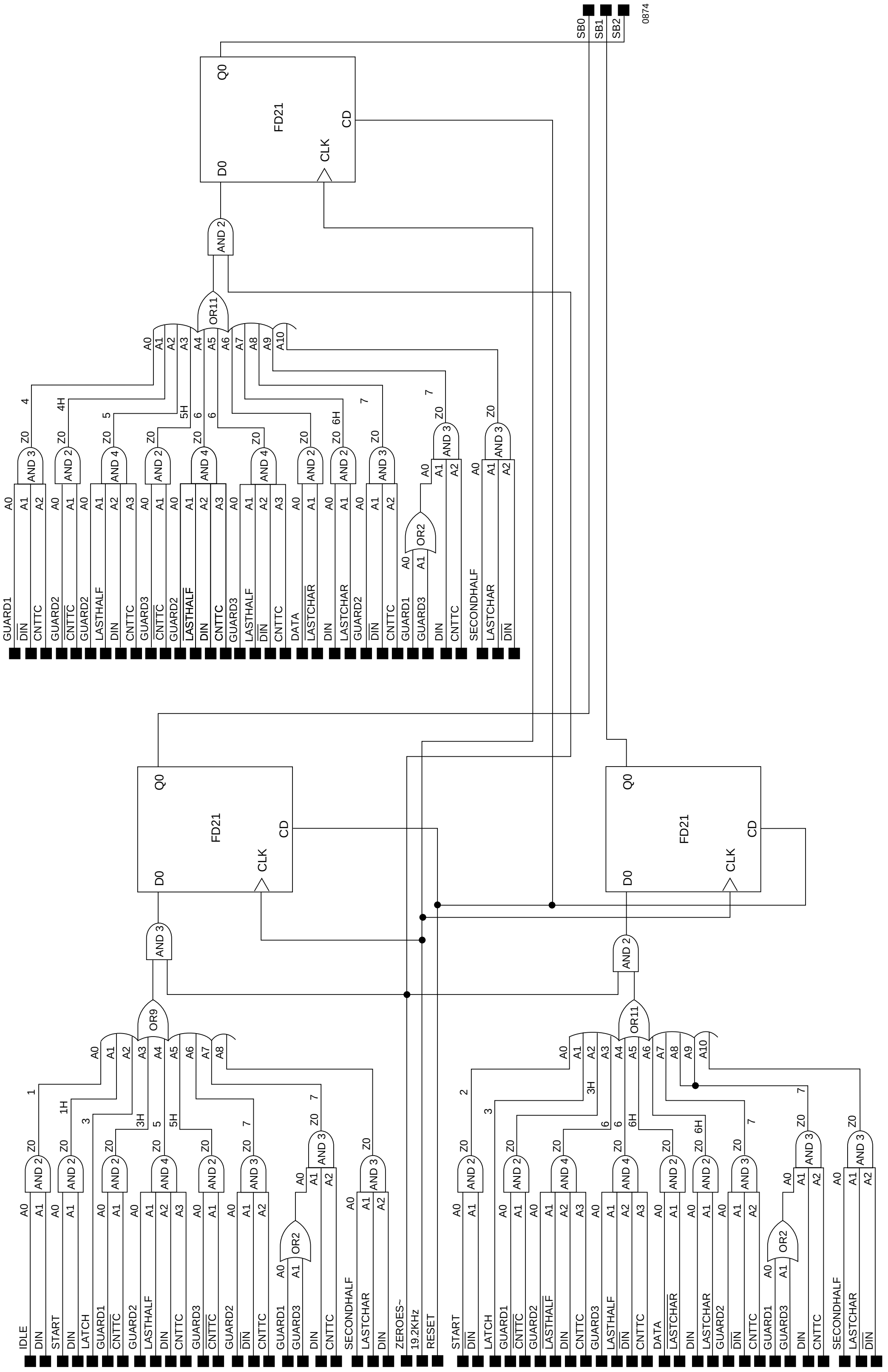


Figure 15. State Machine Decode

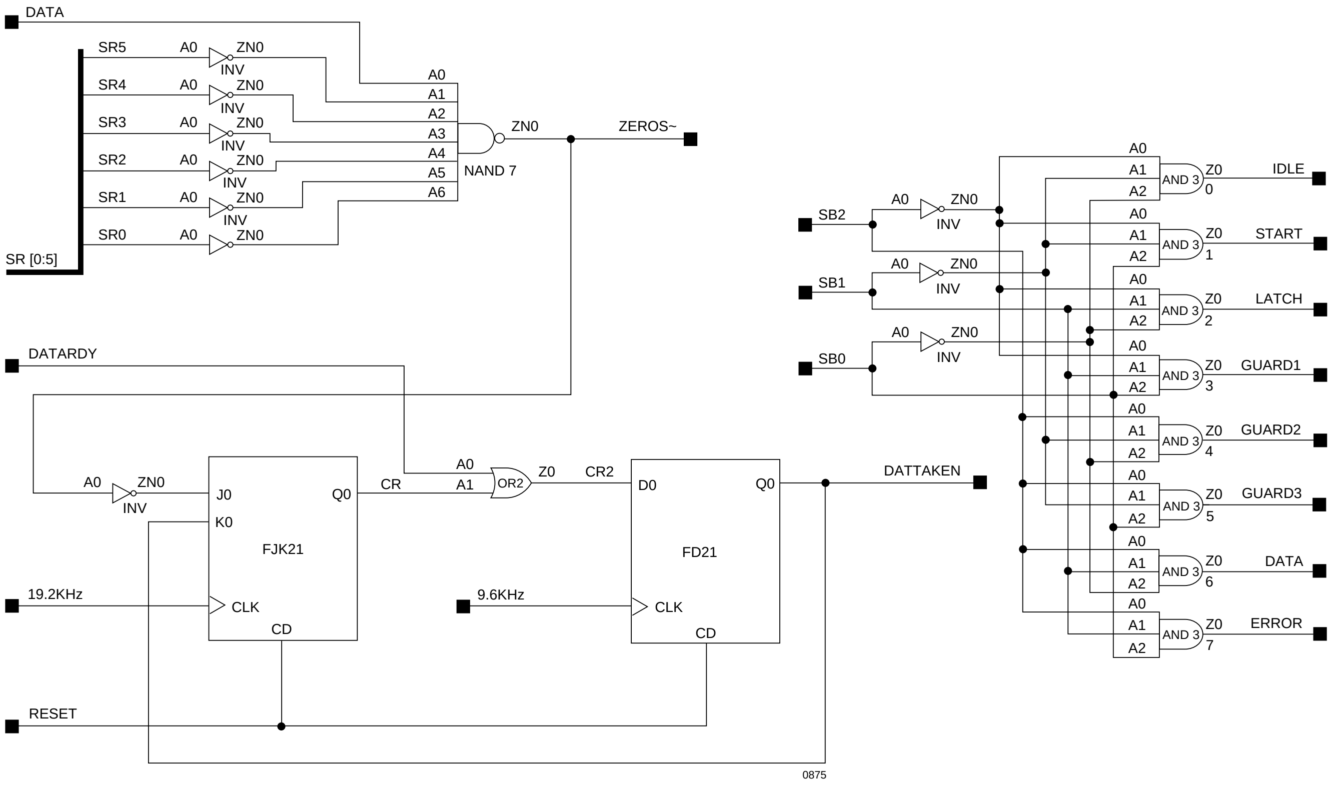


Figure 16. Output Shift Register

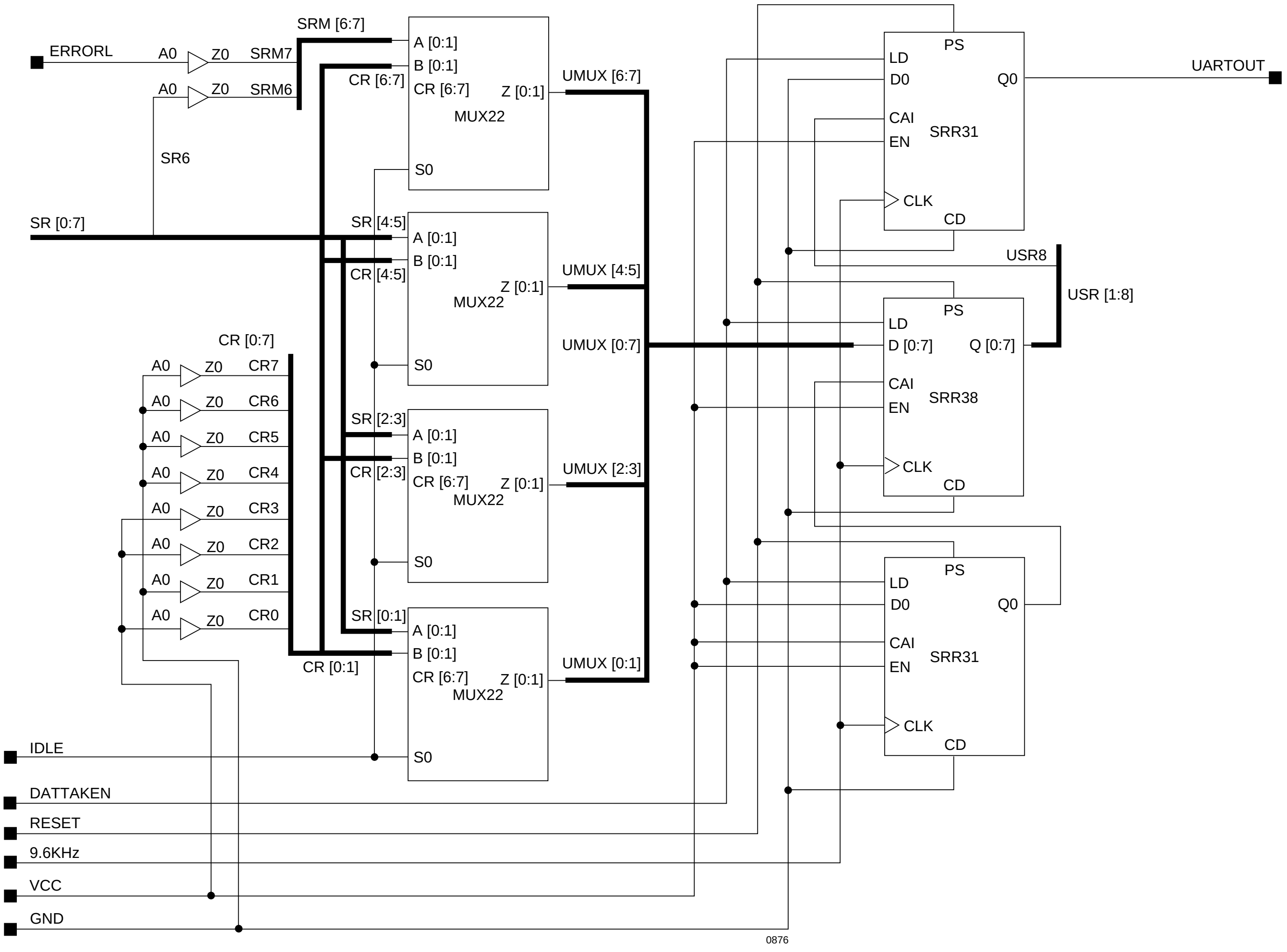


Figure 17. Portion of .LDF file

```
SYM GLB A2 1;
//    8-bit up/down counter wih Async preset, parallel load,
//    enable, up/dn, Async and Sync clear.  Uses 3 GLBs
//    Used as part of 9-bit counter to store the first bar’s width value.
//    Bit 9 is located in GLB A5.
CBUD8 ([C0..C7], CNTTC, [L0..L7], VCC, SMCLK, GND, CNTLD, ACTIVE, START, IDLE, GND)];
END;
SYM GLB A3 1;
//    This is bit 9 of the 9-bit value storing the first bar’s width
CBUD1 (8, SAMPLE, L8, VCC, SMCLK, GND, CNTLD, CNTTC, START, IDLE, GND)];
END;
SYM GLB A4 1;
//    This is the Flip Flop to store the lower 4 bits of the bar’s width
FD24  ([C0..C3]), [L0..L3], LATCH, RST);
END;
SYM GLB A5 1;
//    This is the Flip Flop to store the upper 4 bits of the bar’s width
FD24  ([C4..C7]), [L4..L7], LATCH, RST);
END;
SYM GLB A6 1;
//    This is the Flip Flop to store the MSB of the bar’s width
FD21  (C8, VCC, LATCH, RST);
END;
```

Lattice Semiconductor Corporation (LSC) products are made under one or more of the following U.S. and international patents: 4,761,768 US, 4,766,569 US, 4,833,646 US, 4,852,044 US, 4,855,954 US, 4,879,688 US, 4,887,239 US, 4,896,296 US, 5,130,574 US, 5,138,198 US, 5,162,679 US, 5,191,243 US, 5,204,556 US, 5,231,315 US, 5,231,316 US, 5,237,218 US, 5,245,226 US, 5,251,169 US, 5,272,666 US, 5,281,906 US, 5,295,095 US, 5,329,179 US, 5,331,590 US, 5,336,951 US, 5,353,246 US, 5,357,156 US, 5,359,573 US, 5,394,033 US, 5,394,037 US, 5,404,055 US, 5,418,390 US, 5,493,205 US, 0194091 EP, 0196771B1 EP, 0267271 EP, 0196771 UK, 0194091 GB, 0196771 WG, P3686070.0-08 WG. LSC does not represent that products described herein are free from patent infringement or from any third-party right.

LATTICE SEMICONDUCTOR CORPORATION
5555 Northeast Moore Court
Hillsboro, Oregon 97124 U.S.A.
Tel.: (503) 681-0118
FAX: (503) 681-3037
http://www.latticesemi.com

November 1996