AN-1136 LED Matrix Sequential Character Display

In this application note, we will explain how to control an 8x8 LED matrix with a Silego GreenPAK™ Configurable Mixed-signal IC. An 8x8 LED matrix and an Arduino program were used in this application note.

This project contains 2 stages:

1. Identification of the letters that are going to be displayed on the 8x8 LED Matrix in Arduino.

2. Receiving and processing data with the Silego CMIC.

System View

In this project we will write "SILEGO" onto the 8x8 LED matrix through Silego's GreenPAK. Using the GreenPAK this way leaves microcontroller GPIO's available for other tasks. First, let's talk about the working logic of the 8x8 LED matrix. It consists of 64 LEDs in 8 columns and 8 rows.

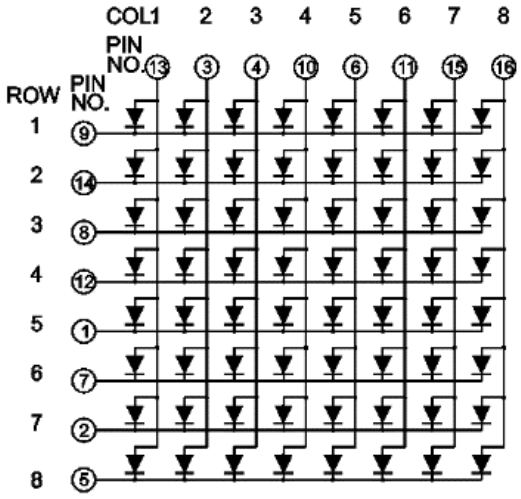

Figure 1. 8x8 LED Matrix schematic

As shown, each column is connected to the rows with LEDs. The columns must be connected to Power and the rows must be connected to Ground to illuminate the LEDs. When the circuit is completed, current flowing through the LEDs allows the LEDs to light up. As shown in Figure 1, the LED matrix to which cathodes are connected commonly is called LED matrix with common cathodes. Positive voltage (HIGH) is applied to the selected column, while the LOW level is applied to the selected row to light up a specific LEDs in this matrix. Let's take a look at how we can obtain the letters on the 8x8 LED matrix before we pass it to the Arduino program.

You see letter "S" formed on the 8x8 LED matrix below as an example;

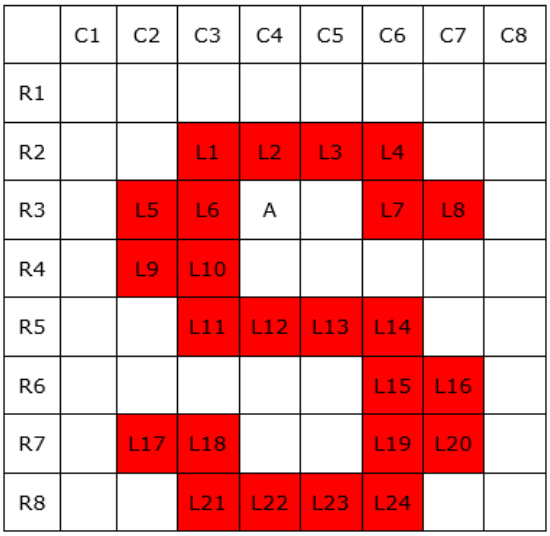

Figure 2. Display of "S" letter on LED Matrix

We need to illuminate 24 LED's to create the letter "S" as shown in Figure 2. We can't light them up at the same time because of the common cathode structure of the matrix. For instance, we must set C4 Column as HIGH and R2 Row as LOW to light up the L2 LED. Because we set up C4 Column for L2 LED as HIGH and R3 Row for L6 as LOW, the LED A located in C4 Column will light up too. Therefore, we have to show it row by row instead of showing the letter "S" at one time. The 8-bit data (1 byte) is enough for 8 columns. After each 8-bit data, Silego CMIC should complete the letter "S" by passing to the next row. It must appear as a complete character as we drive row by row. For this, transitions between the rows must take place very quickly. 25fps (Frame Per Second) is enough for people to easily discern the character, but we need more than 25fps to smooth the LED flicker. This is easily achievable with the SPI unit within the Silego CMIC. We learned the structure of LED Matrix and how to drive it. We can now proceed with the Arduino Program.

Arduino Program

We learned why we need to send images row by row in the previous part. We will separate the letters row by row and we are going to send it to the Silego CMIC using SPI. We are going to define each letter as an 8-byte sequence because each row is of 1 byte. Let's make the letter "S" again.

R1 Column goes on as 10000000, R2 Column as 00111100 and R3 Column as 101100110 as shown in Figure 3. We are going to define the letter "S" as an 8-byte assay. Unlike the previous figure, you can clearly see the alterations in the first column in this figure. The reason for this is to set the first column for row switching functionality. In this way, we aim for all the rows to be scanned when new row of data passes to the next row.

So we imagined the first column as a clock signal that controls the row changes. So the first column should be like: R1 as 1, R2 as 0, R3 as 1, R4 as 0, R5 as 1, R6 as 0, R7 as 1 and R8 as 0.

Note: From here onwards, the first column will be named as rClock and it's not to be confused with other columns.

And according to this, our new codes are shown in Table 1.

	Binary		Hexadecimal
	C8...C2	rClock	
R1	0000000	1	01
R2	0011110	0	3c
R3	0110011	1	67
R4	0000011	0	06
R5	0011110	1	3d
R6	0110000	0	60
R7	0110011	1	67
R8	0011110	0	3c

Table 1. Binary and Hexadecimal Representation of row data

Warning: Pay attention to the order of the columns in Figure 2, because it was made according to the diagram shown in Figure 1. However, in Figure 3, the order of bits is opposite to the order that is shown in Figure 2. This is because C8 has been taken as MSB and rClock as LSB.

	C1	C2	C3	C4	C5	C6	C7	C8
R1	1	0	0	0	0	0	0	0
R2	0	0	1	1	1	1	0	0
R3	1	1	1	0	0	1	1	0
R4	0	1	1	0	0	0	0	0
R5	1	0	1	1	1	1	0	0
R6	0	0	0	0	0	1	1	0
R7	1	1	1	0	0	1	1	0
R8	0	0	1	1	1	1	0	0

Figure 3. Display of "S" letter with row clock

We keep 8x8 led matrix fonts in original form and are going to add rClock signal before sending row information to the Silego CMIC.

Arduino variable for letter "S";

```
{0x3c,0x66,0x60,0x3c,0x06,0x66,0x3c,0x00},// S
```

Warning: Order of data continues like R8, R7, R6. 0x00 represents R1.

If other letters and markings are recorded as two-dimensional matrix to comply with the ASCII table, a variable which is similar to the Figure to the upper right reveals the following data.

This matrix is defined as constant at the beginning of the program because its value will not change. We arranged the settings of the SPI communication in the "setup" stage and we arranged a PIN as an output to activate the SPI unit of Silego CMIC.

```
void setup() {

  pinMode(ss, OUTPUT);
  digitalWrite(ss,LOW);
  SPI.begin();
  SPI.beginTransaction(SPISettings(100000, MSBFIRST, SPI_MODE1));


}
```

```
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},// Space
{0x18,0x00,0x18,0x18,0x3c,0x3c,0x18,0x00},// !
{0x00,0x00,0x00,0x00,0x28,0x6c,0x6c,0x00},// "
{0x6c,0x6c,0xfe,0x6c,0xfe,0x6c,0x6c,0x00},// #
{0x10,0x3c,0x40,0x38,0x04,0x78,0x10,0x00},// $
{0x60,0x66,0x0c,0x18,0x30,0x66,0x06,0x00},// %
{0xfc,0x66,0xa6,0x14,0x3c,0x66,0x3c,0x00},// &
{0x00,0x00,0x00,0x0c,0x18,0x18,0x18,0x00},// '
{0x60,0x30,0x18,0x18,0x18,0x30,0x60,0x00},// (
{0x06,0x0c,0x18,0x18,0x18,0x0c,0x06,0x00},// )
{0x00,0x6c,0x38,0xfe,0x38,0x6c,0x00,0x00},// *
{0x00,0x10,0x10,0x7c,0x10,0x10,0x00,0x00},// +
{0x06,0x0c,0x0c,0x0c,0x00,0x00,0x00,0x00},// ,
{0x00,0x00,0x00,0x3c,0x00,0x00,0x00,0x00},// -
{0x06,0x06,0x00,0x00,0x00,0x00,0x00,0x00},// .
{0x00,0x06,0x0c,0x18,0x30,0x60,0x00,0x00},// /
{0x3c,0x66,0x66,0x6e,0x76,0x66,0x3c,0x00},// 0
{0x7e,0x18,0x18,0x18,0x1c,0x18,0x18,0x00},// 1
{0x7e,0x06,0x0c,0x30,0x60,0x66,0x3c,0x00},// 2
{0x3c,0x66,0x60,0x38,0x60,0x66,0x3c,0x00},// 3
{0x30,0x30,0x7e,0x32,0x34,0x38,0x30,0x00},// 4
{0x3c,0x66,0x60,0x60,0x3e,0x06,0x7e,0x00},// 5
{0x3c,0x66,0x66,0x3e,0x06,0x66,0x3c,0x00},// 6
{0x18,0x18,0x18,0x30,0x30,0x66,0x7e,0x00},// 7
```

We wrote a function after we made some arrangements in the setup function.

```
void silego8x8(char sentence[],unsigned long appearTime)
{
  int i,y;
  byte row;

  for(i=0;i<strlen(sentence);i++)
  {

    enterTime=millis();
    while(millis()-enterTime<appearTime)
    {
      for (y = 7; y >= 0; y--) {

        if(y%2==1) row=font[(int)sentence[i]][y]|0x01;
        else row=font[(int)sentence[i]][y]&0xfe;
        SPI.transfer(row);
        delay(1);

      }
    }
  }
}
```

This function takes 2 arguments as its input. These are: 1. A sentence that is going to be shown (char sentence []) and 2. The changing time of letter/mark (unsigned long appearTime). First, variables that are going to be used in the "For loop" (int i, y) and a variable (byte row) that keeps the row information to be sent to Silego CMIC, have been identified.

Then we created the "For cycle" using strlen function to scan all the characters of the sentence variable that is received as input.

```
for(i=0;i<strlen(sentence);i++)
```

Then, we got the current time information by using millis function and saved it to the variable that is named as "enterTime". We obtained "while loop" by using this variable and the millis function.

```
enterTime=millis();
while(millis()-enterTime<appearTime)
```

This loop continuously sends the same character to Silego CMIC for a period equal to the value of the appearTime variable received as input.

The equation in the while loop subtracts the current time with the time that we have saved (enterTime) and compares the difference with the time that the user had entered (appearTime). As

a result of this, it keeps sending the same character even if a shorter time passes than the amount requested by the user. If more time passes than what the user entered, it breaks the cycle and the next letter is picked.

```
for (y = 7; y >= 0; y--) {

  if(y%2==1) row=font[(int)sentence[i]][y]|0x01;

  else row=font[(int)sentence[i]][y]&0xfe;

  SPI.transfer(row);
  delay(1);

}
```

These rows need to be sent one by one because each character is comprised of 8 rows. For this, the "For cycle" and row data are sent one by one.

Note: The reason for decreasing the variable y from 7 to 0 is because the first row data (R1) is located in the 8th column of the two-dimensional matrix.

Then we checked out the row order to add rClock signal. We set the first column of data as HIGH in the odd-numbered indexes and set the first column of data as LOW in the even-numbered indexes by using the indexes of the two-dimensional matrix.

The index numbers of column in the two-dimensional matrix and corresponding row numbers:

Index	0	1	2	3	4	5	6	7
Row	R8	R7	R6	R5	R4	R3	R2	R1

Table 2. Order of row data in font matrix

We benefited from the ASCII table to find out the corresponding character in the sentence that the user entered on the 8x8 led matrix font. Each character, letter and mark has a numerical value according to the ASCII table. We created an 8x8 led matrix font by using this numerical correspondence. For instance:

ASCII value	Character	Control character	ASCII value	Character	ASCII value	Character	ASCII value	Character
000	(null)	NUL	032	(space)	064	@	096	
001	☺	SOH	033	!	065	A	097	a
002	☻	STX	034	"	066	B	098	b
003	♥	ETX	035	#	067	C	099	c
004	♦	EOT	036	$	068	D	100	d
005	♣	ENQ	037	%	069	E	101	e
006	♠	ACK	038	&	070	F	102	f
007	(beep)	BEL	039	'	071	G	103	g
008	◘	BS	040	(	072	H	104	h
009	(tab)	HT	041	)	073	I	105	i
010	(line feed)	LF	042	*	074	J	106	j
011	(home)	VT	043	+	075	K	107	k
012	(form feed)	FF	044	,	076	L	108	l
013	(carriage return)	CR	045	-	077	M	109	m
014	♫	SO	046	.	078	N	110	n
015	☼	SI	047	/	079	O	111	o
016	►	DLE	048	0	080	P	112	p
017	◄	DC1	049	1	081	Q	113	q
018	↕	DC2	050	2	082	R	114	r
019	‼	DC3	051	3	083	S	115	s
020	¶	DC4	052	4	084	T	116	t
021	§	NAK	053	5	085	U	117	u
022	▬	SYN	054	6	086	V	118	v
023	↨	ETB	055	7	087	W	119	w
024	↑	CAN	056	8	088	X	120	x
025	↓	EM	057	9	089	Y	121	y
026	→	SUB	058	:	090	Z	122	z
027	←	ESC	059	;	091	[	123	{
028	(cursor right)	FS	060	<	092	\	124	|
029	(cursor left)	GS	061	=	093	]	125	}
030	(cursor up)	RS	062	>	094	^	126	~
031	(cursor down)	US	063	?	095	_	127	⌂

Table 3. ASCII Table

The value of "L" in the ASCII table is 76. We obtain the data that has been created for "L" if we type this number to the row index in the two-dimensional matrix that we have created. You can inspect the ASCII table below.

```
font[(int)sentence[i]][y]
```

The code above "(int)sentence(i)" represents the corresponding ASCII code of the related character. After this, we obtained the row information of the related character from the 8x8 LED Matrix Font that we have created. We processed the rows one by one with Index[y].

```
if(y%2==1) row=font[(int)sentence[i]][y]|0x01;

else row=font[(int)sentence[i]][y]&0xfe;
```

If we need to set first column HIGH for rClock signal, then we apply "Logical or" to that row with 0x01. The first column goes HIGH while the other columns stay unaffected because 0 is an ineffective element in the "Logic or" process.

If we need to set the first column to LOW, we apply "Logical AND" to that row with 0xfe. Because of the ineffective element in "Logical AND", the first column goes LOW while other columns stay unaffected.

```
SPI.transfer(row);
delay(1);
```

At last, we send this row information to the Silego CMIC by using the SPI interface.

You can call the function in the loop section as it is shown below.

```
void loop() {

 silego8x8("SILEGO",1000);

}
```

Design of the Silego CMIC

First, we must adjust the settings of the SPI unit to get the data that comes from the Arduino.

Settings should be as shown in Fig. 4 below:

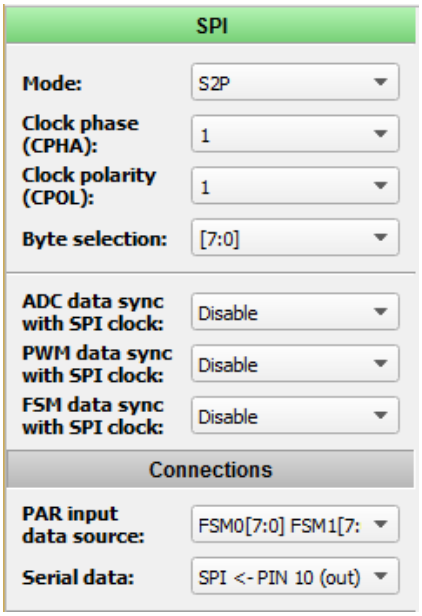

Figure 4. Settings of SPI

The SPI unit converts the data that comes from the Arduino to parallel data and allows us to process the bits separately. Data that comes from the Arduino is column data and this allows us to connect this data directly to the column pins. When programming Arduino, we have arranged the first column as 1-0-1-0…. to ensure that it keeps changing constantly.

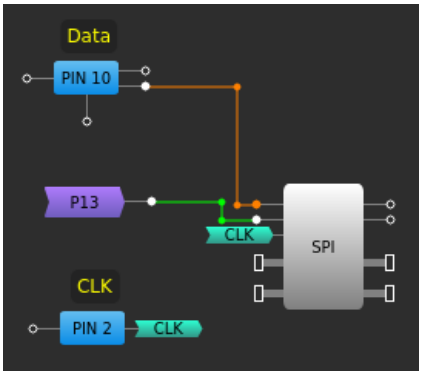

Figure 5. Connections of SPI

In the S2P mode of the SPI unit, we will use rClock as a clock to understand that the serial to parallel data conversion has finished. The first column that changes after every new row will report to us that the data is ready. We connected the first column to P DLY1 unit to detect any change in the first column. The settings of DLY1 units are shown in Fig. 6 below:

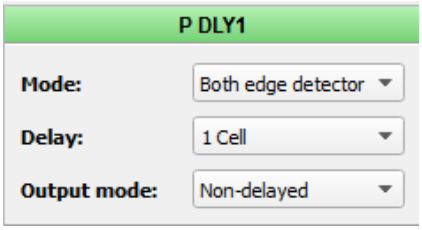

Figure 6. Settings of P DLY1

In this way, every copy of incoming data triggers the P DLY unit and declares that we should pass to the next row. We used a simple counter to activate the rows in a sequential manner. This counter increases one by one and counts up to 8 whenever a new row arrives.

The first state is set as idle and the next 8 states are set as row states and that is why the counter counts up to 9. You can see the counter that was used in Fig.7 below:

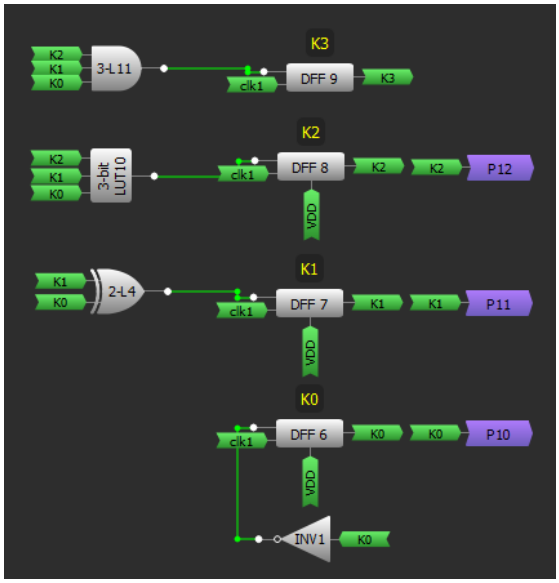

Figure 7. Row counter

You can see active row signals based on this counter in Table 4.

K3	K2	K1	K0	R8	R7	R6	R5	R4	R3	R2	R1
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	1	0
0	0	1	1	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	1	0	0	0
0	1	0	1	0	0	0	1	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0	0
0	1	1	1	0	1	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0

Table 4. Active line signals based on row counter

We either energized each row or stopped energizing according the situation of K3-K2-K1-K0. You can see the LUTs and pins that we have used in Figure 8.

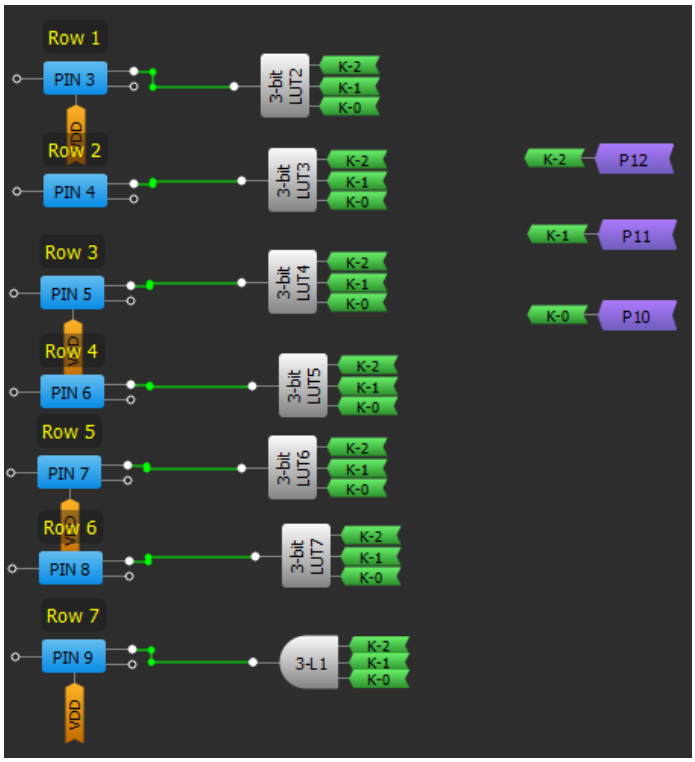

Figure 8. Connection of row pins

Instead of using the INV1 cell we can set the Q output polarity of DFF6 as inverted. In this way we can eliminate the INV1 cell. You should make the settings of DFF6 as shown in Figure 9.

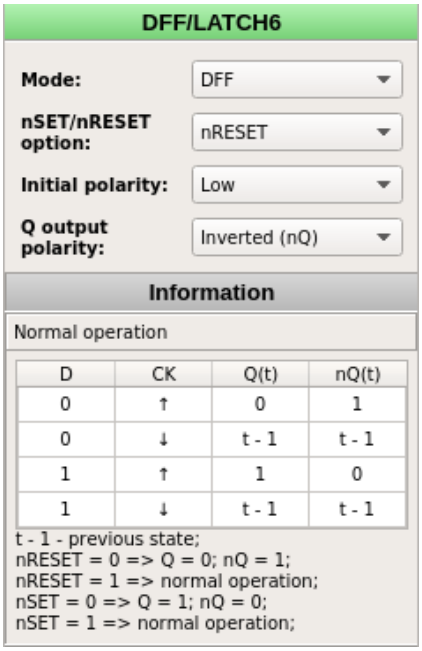

Figure 9. Settings of DFF 6

However, each equation that contains K0 is needed to be edited again. For example, in the first situation, 3 bits LUT1 is HIGH when K2, K1, K0 = 111, while in the new situation it must be HIGH in K2, K1, K0 = 110.

Alternative Design

Unlike the first design, we can use DFFs to shift the signal for the row switching function. In this way we can eliminate LUTs that are connected with row pins and we can connect the DFF to row pins directly.

This design is shown in Figure 10.

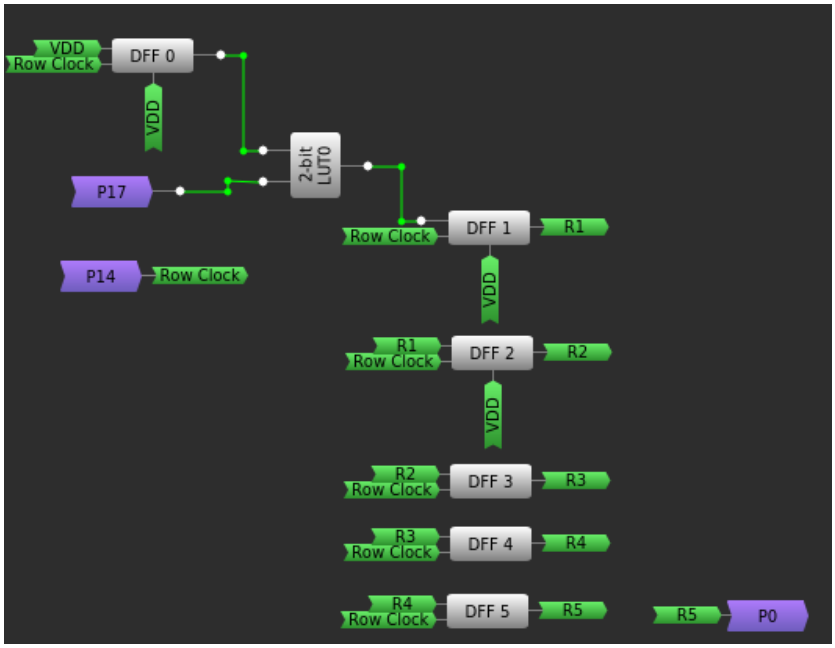

Figure 10. Chaining DFFs R1 to R5 in Matrix0

DFF0 located in Matrix 0 sees the idle state task. At each change of rClock , DFFs go to HIGH in order. The first R1 goes HIGH followed by R2. R8 is connected to R1 and then this cycle is completed. The first design uses less DFF's while the second design uses less LUT's.

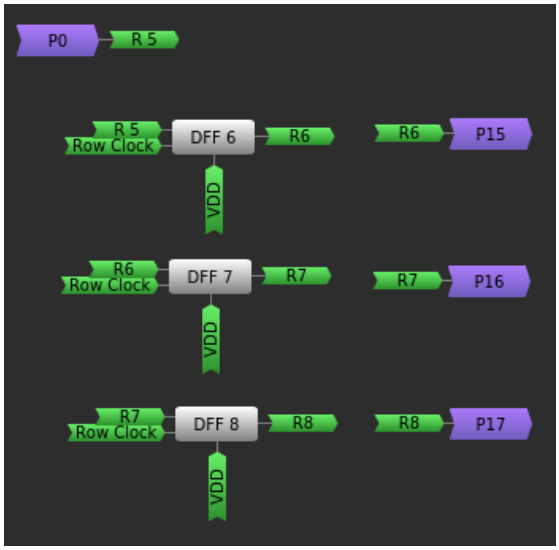

Figure 11. Chaining DFFs R6 to R8 in Matrix1

Connections

You can change the brightness of the LEDs by changing the current. Using an external power supply and by controlling this current with transistors, we will have a better solution instead of just fully driving the LEDs through the Silego CMIC. You can adjust the brightness of the LED's with a potentiometer that you connect to an external power source or drive the transistors with PWM.

Arduino Pins	SLG46620V Pins
43	13
52	2
51	10

Table 5. Top-Level Connections 1

SLG46620V Pins	NPN transistor base
3	R1 base
4	R2 base
5	R3 base
6	R4 base
7	R5 base
8	R6 base
9	R7 base
12	R8 base

Table 6. Top-Level Connections 2

SLG46620V Pins	8x8 LED Matrix pins
14	3
15	4
16	10
17	6
18	11
19	15
20	16

Table 7. Top-Level Connections 3

8x8 LED Matrix pins	NPN transistor base
9	R1
14	R2
8	R3
12	R4
1	R5
7	R6
2	R7
5	R8

Table 8. Top-Level Connections 5

NPN transistor base	Potentiometer pin
R1	3
R2	3
R3	3
R4	3
R5	3
R6	3
R7	3
R8	3

Table 9. Top-Level Connections 6

Potentiometer pin	Power Supply pin
2	GND

Table 10. Top-Level Connections 4

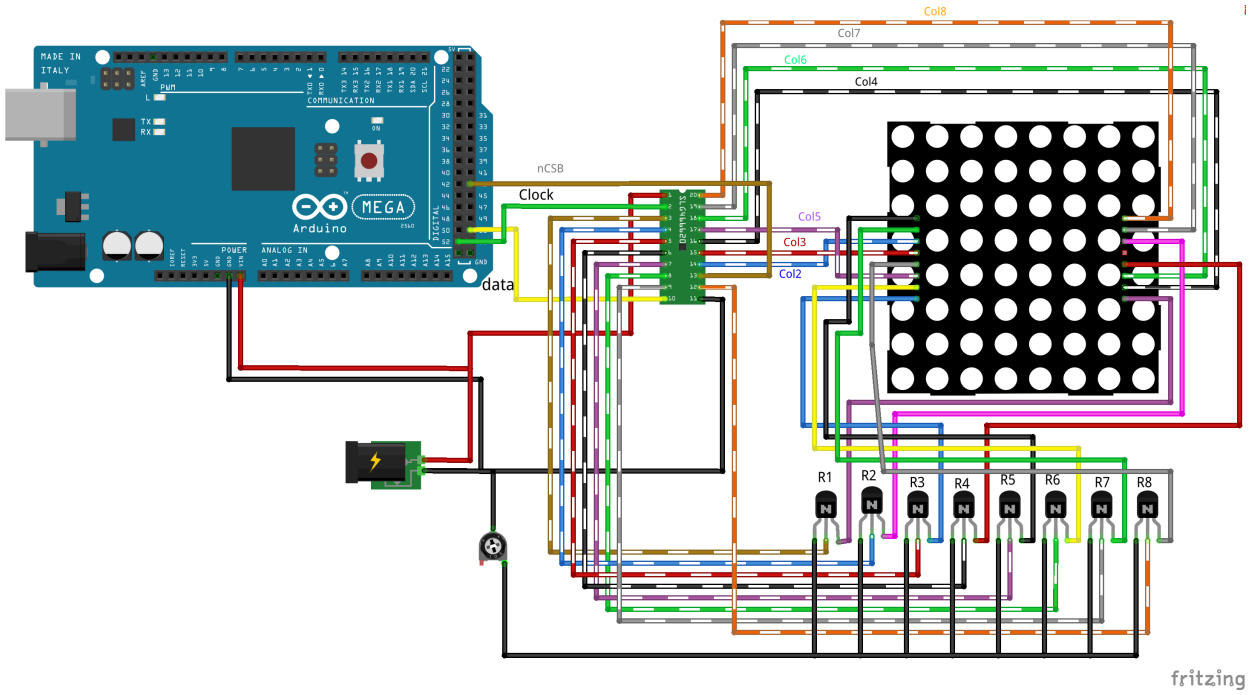

Figure 12. Top-Level Schematic

Conclusion

In this project, we learned to control the commonly used 8x8 LED Matrix with the Silego CMIC. It is quite easy to control the LED Matrix with the SPI unit located in the SLG46620V version and you can easily control the LED Matrix with this method. Alternatively, you can control the pins' output by using the I2C protocol which is available within GreenPAK5.

Design Limitations:

- Column 1 of SPI is reserved for rCLK
- Not enough GPIOs for Column 1 if design is changed

About the Author

Name: Mustafa Balcı & Tuğçe Çelt

Background: Students in the Electrical-Electronic engineering program (Eskisehir Anadolu University)

Contact: appnotes@silego.com

Files

- AN-1136 LED Matrix Sequential Character Display.gp4- (50 KB)
- AN-1136 LED Matrix Sequential Character Display.ino- (1 KB)
- AN-1136 LED Matrix Sequential Character Display.pdf- (871 KB)
- AN-1136.zip- (763 KB)

See full list of Application Notes