Charlieplexing library for Raspberry Pi RP2040 PIO Version 1.0.0

Pierre Molinaro

June 4, 2021

Contents

1 Releases

Version	Date	Comment
1.0.0	June 4, 2021	Initial release

2 Introduction

Charlieplexing is a technique for driving a multiplexed display in which relatively few I/O pins on a microcontroller are used e.g. to drive an array of LEDs[1]. Very often, an interrupt routine is used to periodically refresh the leds. The PIOs built into the RP2040 microcontroller allow this refresh to be performed automatically, without processor intervention.

This library uses the PIO (Programmable IO) module of RP2040 microcontroller for handling 5-leds and 6-leds charlieplexing:

- the 6-leds version requires 11 PIO instructions, and 3 consecutive GPIO ports;
- the 5-leds version requires 7 PIO instructions, and 3 consecutive GPIO ports.

3 6-leds version

The 6-leds version uses three consecutive pins (for example: GP0, GP1, GP2) for driving six leds.

3.1 Connections

The figure 1 shows how to connect the six leds to the RP2040. The value of the R resistors depends on the brightness you want to achieve; typically 150 Ω or 220 Ω is used.

3.2 Example: 6-leds-charlieplexing sketch

```
1 #include "charlieplexing.h"
2
3 static const PIO pio = pio0 ; // pio0 or pio1
4 static const uint stateMachine = 0 ; // 0, 1, 2 or 3
5
6 void setup () {
```

[1] From https://en.wikipedia.org/wiki/Charlieplexing

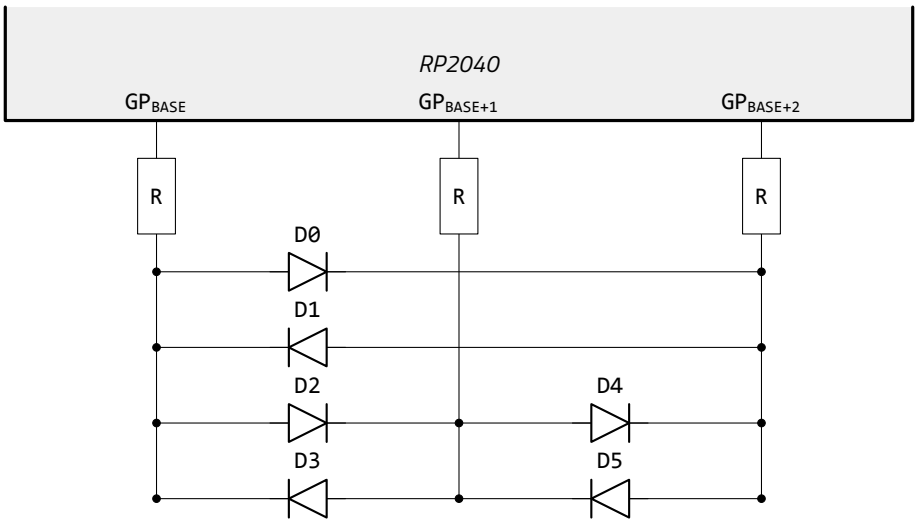

Figure 1 – 6-leds Charlieplexing, using 3 pins

```
7   pinMode (LED_BUILTIN, OUTPUT) ;
8   const uint prgmOffset = 0 ;
9   const uint outputBasePin = 0 ;
10  charlieplexing6_add_program (pio, prgmOffset) ;
11  charlieplexing6_program_init (pio, stateMachine, prgmOffset, outputBasePin) ;
12  pio_sm_set_enabled (pio, stateMachine, true) ;
13 }
14
15 static uint32_t gDeadline = 0 ;
16 static uint32_t gValue = 1 ;
17
18 void loop () {
19   if (gDeadline <= millis ()) {
20     gDeadline += 1000 ;
21     digitalWrite (LED_BUILTIN, !digitalRead (LED_BUILTIN)) ;
22     charlieplexing6_set_output (pio, stateMachine, gValue) ;
23     gValue <<= 1 ;
24     gValue &= 0x3F ;
25     if (gValue == 0) {
26       gValue = 1 ;
27     }
28   }
29 }
```

Line 1. This line includes the `RP2040-Charlieplexing` library.

Lines 3, 4. Select the PIO (`pio0` or `pio1`) and the state machine (`0`, `1`, 2 or 3) you use.

Line 7. This line configures the `LED_BUILTIN` as `OUTPUT`; optional, this sketch makes this led blink.

Line 8. Set the PIO program offset. As each PIO has a 32 instructions memory, and the code has 11 instructions, you can choose any value between 0 and 32 - 11 = 21 inclusive.

Line 9. Set the `GPIO` you use. Three consecutive pins are used, you set here the first one.

Line 10. The `charlieplexing6_add_program` function writes the program instructions (11) in the selected PIO instruction memory, from the specified offset.

Line 11. The `charlieplexing6_program_int` function performs all required initializations for configuring the state machine and the three output pins.

Line 12. The `pio_sm_set_enabled` function is defined by the SDK; it starts the state machine (last parameter is `true`), or it stops it (last parameter is `false`). From now on, the refresh is launched and is carried out without intervention of a processor. Initially, all the leds are off. To change, you have to call the `charlieplexing6_set_output` function, which is done in the `loop` function.

Lines 18 to 29. The `loop` function uses the `gDeadline` global variable for performing every 1000 ms:

- blinking the `LED_BUILTIN` led;
- updating the `gValue` global variable, the successive values are `0x1`, `0x2`, `0x4`, `0x8`, `0x10`, `0x20`, `0x1` again, ... So the active led is `D0`, `D1`, ..., `D5`, `D0` again...

3.3 Reference

Four functions are provided:

- the `charlieplexing6_add_program` function, section 3.3.1 page 4;
- the `charlieplexing6_can_add_program` function, section 3.3.2 page 4;
- the `charlieplexing6_program_init` function, section 3.3.3 page 5;
- the `charlieplexing6_set_output` function, section 3.3.4 page 5.

3.3.1 The `charlieplexing6_add_program` function

```
uint32_t charlieplexing6_add_program (const PIO pio,
                                      const uint32_t prgmOffset) ;
```

Attempt to load the program at the specified instruction memory offset, panicking if not possible. Call the `charlieplexing6_can_add_program` function if you need to check whether the program can be loaded.

`pio`: the PIO instance; either `pio0` or `pio1`.

`prgmOffset`: the instruction memory offset wanted for the start of the program, i.e. a value between 0 and 23 inclusive.

Returned value: the instruction memory offset available for loading an other program.

3.3.2 The `charlieplexing6_can_add_program` function

```
bool charlieplexing6_can_add_program (const PIO pio,
                                      const uint32_t prgmOffset) ;
```

Determine whether the given program can (at the time of the call) be loaded onto the PIO instance starting at a particular location.

pio: the PIO instance; either `pio0` or `pio1`.

prgmOffset: the instruction memory offset wanted for the start of the program, i.e. a value between 0 and 23 inclusive.

Returned value: `true` if the program can be loaded at that location (calling the `charlieplexing6_add_program` function succeeds); `false` if there is not space in the instruction memory.

3.3.3 The `charlieplexing6_program_init` function

```
void charlieplexing6_program_init (const PIO pio,
                                   const uint32_t stateMachine,
                                   const uint32_t prgmOffset,
                                   const uint32_t basePin) ;
```

Initialize the selected state machine.

pio: the PIO instance; either `pio0` or `pio1`.

stateMachine: the state machine index, `0` to `3`.

prgmOffset: the instruction memory offset used when calling the `charlieplexing6_add_program` function.

basePin: the first pin index to set as output. The two next pins are also set as output.

3.3.4 The `charlieplexing6_set_output` function

```
void charlieplexing6_set_output (const PIO pio,
                                 const uint32_t stateMachine,
                                 const uint32_t value) ;
```

Set the output configuration. As six leds are handled, a led is driven ON for one sixth of the period.

pio: the PIO instance; either `pio0` or `pio1`.

stateMachine: the state machine index, `0` to `3`.

value: a bit-field value that specifies the state of the six leds:

- bit 0: if `true`, `D0` is ON, if `false`, `D0` is OFF;
- bit 1: if `true`, `D1` is ON, if `false`, `D1` is OFF;
- bit 2: if `true`, `D2` is ON, if `false`, `D2` is OFF;

- bit 3: if `true`, D3 is ON, if `false`, D3 is OFF;
- bit 4: if `true`, D4 is ON, if `false`, D4 is OFF;
- bit 5: if `true`, D5 is ON, if `false`, D5 is OFF;
- bits 6 to 31: any value, are ignored by the function.

4 5-leds version

The 5-leds version uses three consecutive pins (for example: `GP0`, `GP1`, `GP2`) for driving six leds. It is very similar to the 6-leds version, the only advantage is that it requires 7 instructions, unlike the 6-leds version which requires 11.

4.1 Connections

The figure 2 shows how to connect the six leds to the `RP2040`. The value of the `R` resistors depends on the brightness you want to achieve; typically 150 Ω or 220 Ω is used.

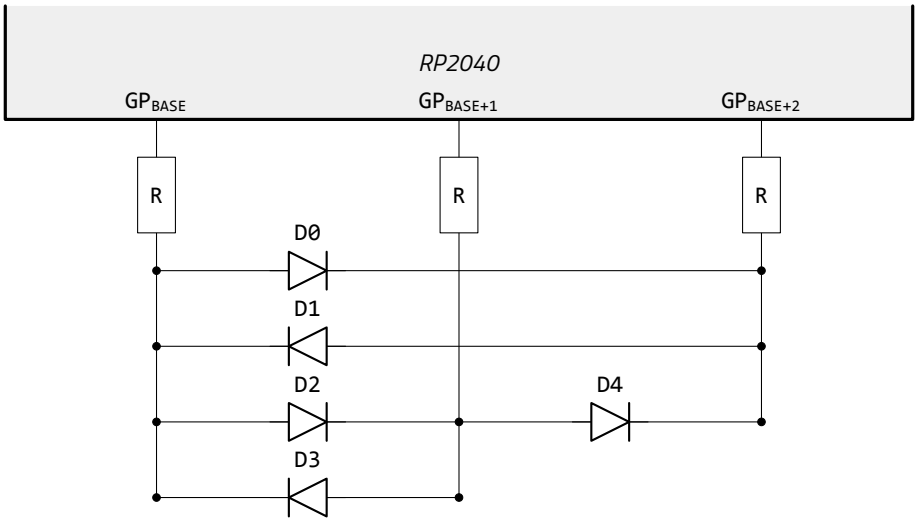

Figure 2 – 5-leds Charlieplexing, using 3 pins

4.2 Example: `5-leds-charlieplexing` sketch

```
#include "charlieplexing.h"

static const PIO pio = pio0 ; // pio0 or pio1
static const uint stateMachine = 0 ; // 0, 1, 2 or 3

void setup () {
  pinMode (LED_BUILTIN, OUTPUT) ;
```

```
 8   const uint prgmOffset = 0 ;
 9   const uint outputBasePin = 0 ;
10   charlieplexing5_add_program (pio, prgmOffset) ;
11   charlieplexing5_program_init (pio, stateMachine, prgmOffset, outputBasePin) ;
12   pio_sm_set_enabled (pio, stateMachine, true) ;
13 }
14
15 static uint32_t gDeadline = 0 ;
16 static uint32_t gValue = 1 ;
17
18 void loop () {
19   if (gDeadline <= millis ()) {
20     gDeadline += 1000 ;
21     digitalWrite (LED_BUILTIN, !digitalRead (LED_BUILTIN)) ;
22     charlieplexing5_set_output (pio, stateMachine, gValue) ;
23     gValue <<= 1 ;
24     gValue &= 0x1F ;
25     if (gValue == 0) {
26       gValue = 1 ;
27     }
28   }
29 }
```

Line 1. This line includes the `RP2040-Charlieplexing` library.

Lines 3, 4. Select the PIO (`pio0` or `pio1`) and the state machine (`0`, `1`, 2 or 3) you use.

Line 7. This line configures the `LED_BUILTIN` as `OUTPUT`; optional, this sketch makes this led blink.

Line 8. Set the PIO program offset. As each PIO has a 32 instructions memory, and the code has 7 instructions, you can choose any value between 0 and 32 - 7 = 25 inclusive.

Line 9. Set the `GPIO` you use. Three consecutive pins are used, you set here the first one.

Line 10. The `charlieplexing5_add_program` function writes the 11 instructions in the selected PIO instruction memory, from the specified offset.

Line 11. The `charlieplexing5_program_int` function performs all required initializations for configuring the state machine and the three output pins.

Line 12. The `pio_sm_set_enabled` function is defined by the SDK; it starts the state machine (last parameter is `true`), or it stops it (last parameter is `false`). From now on, the refresh is launched and is carried out without intervention of a processor. Initially, all the leds are off. To change, you have to call the `charlieplexing5_set_output` function, which is done in the `loop` function.

Lines 18 to 29. The `loop` function uses the `gDeadline` global variable for performing every 1000 ms:

- blinking the `LED_BUILTIN` led;
- updating the `gValue` global variable, the successive values are `0x1`, `0x2`, `0x4`, `0x8`, `0x10`, `0x1` again, ... So the active led is `D0`, `D1`, ..., `D4`, `D0` again...

4.3 Reference

Five functions are provided:

- the `charlieplexing5_add_program` function, section 4.3.1 page 8;
- the `charlieplexing5_can_add_program` function, section 4.3.2 page 8;
- the `charlieplexing5_program_init` function, section 4.3.3 page 8;
- the `charlieplexing5_set_output` function, section 4.3.4 page 9;
- the `charlieplexing5_set_output_2` function, section 4.3.5 page 9.

4.3.1 The `charlieplexing5_add_program` function

```
uint32_t charlieplexing5_add_program (const PIO pio,
                                      const uint32_t prgmOffset) ;
```

Attempt to load the program at the specified instruction memory offset, panicking if not possible. Call the `charlieplexing5_can_add_program` function if you need to check whether the program can be loaded.

`pio`: the PIO instance; either `pio0` or `pio1`.

`prgmOffset`: the instruction memory offset wanted for the start of the program, i.e. a value between 0 and 25 inclusive.

Returned value: the instruction memory offset available for loading an other program.

4.3.2 The `charlieplexing5_can_add_program` function

```
bool charlieplexing5_can_add_program (const PIO pio,
                                      const uint32_t prgmOffset) ;
```

Determine whether the given program can (at the time of the call) be loaded onto the PIO instance starting at a particular location.

`pio`: the PIO instance; either `pio0` or `pio1`.

`prgmOffset`: the instruction memory offset wanted for the start of the program, i.e. a value between 0 and 25 inclusive.

Returned value: `true` if the program can be loaded at that location (calling the `charlieplexing5_add_program` function succeeds); `false` if there is not space in the instruction memory.

4.3.3 The `charlieplexing5_program_init` function

```
void charlieplexing5_program_init (const PIO pio,
                                   const uint32_t stateMachine,
                                   const uint32_t prgmOffset,
                                   const uint32_t basePin) ;
```

Initialize the selected state machine.

pio: the PIO instance; either `pio0` or `pio1`.

stateMachine: the state machine index, `0` to 3.

prgmOffset: the instruction memory offset used when calling the `charlieplexing5_add_program` function.

basePin: the first pin index to set as output. The two next pins are also set as output.

4.3.4 The `charlieplexing5_set_output` function

```
void charlieplexing5_set_output (const PIO pio,
                                 const uint32_t stateMachine,
                                 const uint32_t value) ;
```

Set the output configuration. As five leds are handled, a led is driven ON for one fifth of the period.

pio: the PIO instance; either `pio0` or `pio1`.

stateMachine: the state machine index, `0` to 3.

value: a bit-field value that specifies the state of the five leds:

- bit 0: if `true`, `D0` is ON, if `false`, `D0` is OFF;
- bit 1: if `true`, `D1` is ON, if `false`, `D1` is OFF;
- bit 2: if `true`, `D2` is ON, if `false`, `D2` is OFF;
- bit 3: if `true`, `D3` is ON, if `false`, `D3` is OFF;
- bit 4: if `true`, `D4` is ON, if `false`, `D4` is OFF;
- bits 5 to 31: any value, are ignored by the function.

4.3.5 The `charlieplexing5_set_output_2` function

```
void charlieplexing5_set_output_2 (const PIO pio,
                                   const uint32_t stateMachine,
                                   const uint32_t value) ;
```

Set the output configuration. The only difference with the `charlieplexing5_set_output` function is that each led is driven for one sixth of the period.

5 Running several state machines

The **`6-leds-charlieplexing`** and **`5-leds-charlieplexing`** demo sketches show how to run one state machine. The PIO is versatile, allowing many combinations: several state machines can run the same program, and several programs can be loaded and executed in a PIO.

The **2-6-leds-charlieplexing** sketch shows how a same PIO program can be ran by two state machines. The reader can easily extend the demo code to run the same program on the 4 state machines of a single PIO, allowing to control 24 leds via 12 pins. Using the 4 state machines of the 2 PIOs, you can drive 48 leds with 24 GPIOs.

The **5-leds-6-leds-charlieplexing** sketch shows how a two state machines of a same PIO can run their own program. Note this demo sketch has no practical interest: it requires 18 PIO instructions and drives 11 leds, while the `2-6leds-charliepexing` demo sketch requires 11 PIO instructions and drives 12 leds. The only purpose is to illustrate the way to deploy and run different program in a PIO. The reader can easily adapt this demo code to its own purpose.

6 How it works

6.1 The `charlieplexing5_set_output` function

This function is the easiest to understand. It sends to the TX FIFO a 32-bit number with the following composition:

Bits	**31-30**	**29-27**	**26-24**	**23-21**	**20-18**	**17-15**	**14-12**	**11-9**	**8-6**	**5-3**	**2-0**
Value	00	d4	110	d3	011	d2	011	d1	101	d0	101

Where:

- `d0` is `000` when `D0` is OFF, and `001` if it is ON;
- `d1` is `000` when `D1` is OFF, and `100` if it is ON;
- `d2` is `000` when `D2` is OFF, and `001` if it is ON;
- `d3` is `000` when `D3` is OFF, and `010` if it is ON;
- `d4` is `000` when `D4` is OFF, and `010` if it is ON.

Each led requires 6 bits, 3 fixed bits for setting the GPIO direction, 3 d_i bits for the current output level.

So for 5 leds 30 bits are used. Bits 31 - 30 are unused and set to `0`.

The PIO program is in the `charlieplexing5.pioasm` file:

```
 .wrap_target
    pull noblock       ; Pull from FIFO to OSR if available, else copy X to OSR
    mov x, osr         ; Copy most-recently-pulled value back to scratch X
mainloop:
    out pindirs, 3     ; Set pin directions
    out pins, 3   [31] ; Set pin output values
    set y, 31     [31] ; Load Y register for wait loop
wait1:
    jmp y-- wait1 [31] ; Wait loop : 31 * 32 cycles
```

```
10      set pins, 0         ; Switch off any led
11      jmp !osre mainloop ; if OSRE not null, handle next led
12  .wrap                  ; Implicit jump to the first instruction
```

From rp2040 datasheet, section 3.6.8 page 384: *Often, leds activation can be left unchanged for thousands of cycles, rather than supplying a new configuration each time. The two first instructions highlight how a nonblocking PULL (Section 3.4.7) can achieve this: if the TX FIFO is empty, a nonblocking PULL will copy X to the OSR. After pulling, the program copies the OSR into X. The net effect is that, if a new configuration has not been supplied through the TX FIFO at the start of this period, the configuration from the previous period (which has been copied from X to OSR via the failed PULL, and then back to X via the MOV) is reused, for as many periods as necessary.*

Then, at every `mainloop` iteration, the `OUT` instructions right shift the `OSR` value, inserting `0` bits at MSB. At the end of the fifth iteration, `OSR` is null, and the execution wraps to the first instruction.

6.2 The `charlieplexing5_set_output_2` function

The value sent to the TX FIFO differs from the previous one by the two most significant bits :

Bits	**31-30**	**29-27**	**26-24**	**23-21**	**20-18**	**17-15**	**14-12**	**11-9**	**8-6**	**5-3**	**2-0**
Value	11	d4	110	d3	011	d2	011	d1	101	d0	101

So, at the end of the of the fifth iteration, `OSR` is not null, its value is `0b11`. A sixth iteration is performed, where no led is ON, as the output value set by the `out pins, 3` instruction is `0`.

6.3 The `charlieplexing6_set_output` function

The previous function uses 6 bits for every led. For 6 leds we would need 36 bits, but a TX FIFO input has 32 bits. As two consecutive leds require the same pin direction, we use 9 bits for 2 consecutive leds, so the 6 leds are handled with 27 bits; the five MSBs are set to `0`.

The `charlieplexing6_set_output` function sends to the TX FIFO a 32-bit number with the following composition:

Bits	**31-27**	**26-24**	**23-21**	**20-18**	**17-15**	**14-12**	**11-9**	**8-6**	**5-3**	**2-0**
Value	00000	d5	d4	110	d3	d2	011	d1	d0	101

Where:

- `d0` is `000` when `D0` is OFF, and `001` if it is ON;
- `d1` is `000` when `D1` is OFF, and `100` if it is ON;
- `d2` is `000` when `D2` is OFF, and `001` if it is ON;
- `d3` is `000` when `D3` is OFF, and `010` if it is ON;
- `d4` is `000` when `D4` is OFF, and `010` if it is ON;

- `d5` is `000` when `D5` is OFF, and `100` if it is ON.

The corresponding PIO program is in the `charlieplexing6.pioasm` file:

```
 .wrap_target
    pull noblock       ; Pull from FIFO to OSR if available, else copy X to OSR
    mov x, osr         ; Copy most-recently-pulled value back to scratch X
mainloop:
    out pindirs, 3     ; Set pin directions
    out pins, 3   [31] ; Set pin output values
    set y, 31     [31] ; Load Y register for first wait loop
wait1:
    jmp y-- wait1 [31] ; First wait loop : 31 * 32 cycles
    out pins, 3   [31] ; Set pin output values
    set y, 31     [31] ; Load Y register for second wait loop
wait2:
    jmp y-- wait2 [31] ; Second wait loop : 31 * 32 cycles
    set pins, 0        ; Switch off any led
    jmp !osre mainloop ; if OSRE not null, handle next led
 .wrap                 ; Implicit jump to the first instruction
```

Now, two pin output values setting and two wait loops are needed at each iteration, increasing the program by 4 instructions.