In February 2021 I bought four Raspberry Pi Pico microcontroller boards and various 'pico packs' from Pimoroni Ltd, U.K..

One of the Raspberry Pi Pico microcontroller boards I mounted on a Pimoroni 'pico explorer base'. On this base I also mounted a Pimoroni rv3028 RTC breakout board and a Pimoroni RGB Encoder Breakout .

I wanted to have the RTC and the RGB encoder working together with the Raspberry Pi Pico via the i2c bus. I know that the Raspberry Pi Pico contains a real time clock but it is quite basic and (as far as I remember) it doesn't keep time when disconnected from a power source. With the current project one can leave this hardware combination for weeks without electrical power. After re-applying electrical power the clock will continue to show the correct date and time (if initially set a correct date and time that is).

After quite some study and experimenting, I decided to do a 'clean start', I mean: I had edited the i2cdevice and rv2038 python modules quite a bit, but I ran into too much trouble.

The original i2cdevice files ('__init__.py' and 'adapter.py') I downloaded from the Pimoroni GitHub webpage: i2cdevice-python . From the webpage rv3028-python I downloaded from the /library/ subpage the file '__ini__.py' which contains the RV3028 class among other classes.

I already had flashed the Raspberry Pi Pico with the latest version of the Pimoroni-pico firmware from the GitHub webpage: latest Alfa .

On May 17, 2021, I successfully finished a project using the rv3028 RTC breakout and the Pimoroni pico explorer base running with a Raspberry Pi Pico.

With this experience I proceeded to integrate also the RGB Encoder Breakout. First I did a small project to test and gain experience with the Encoder. See PaulskPt/breakout_encoder on GitHub.

Finally I started this project to use both the rv3028 RTC and the RGB Encoder breakout boards with a Raspberry Pi Pico on the Pimoroni Pico explorer base.
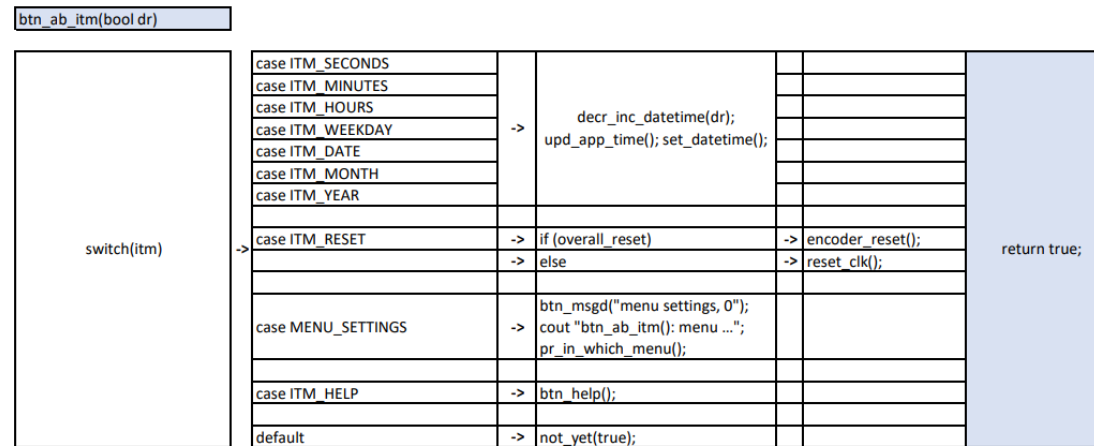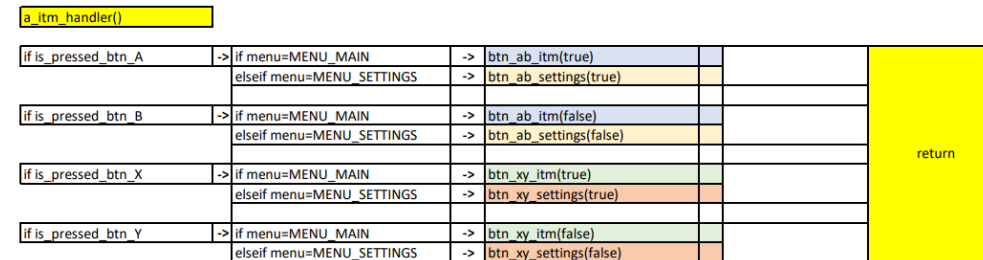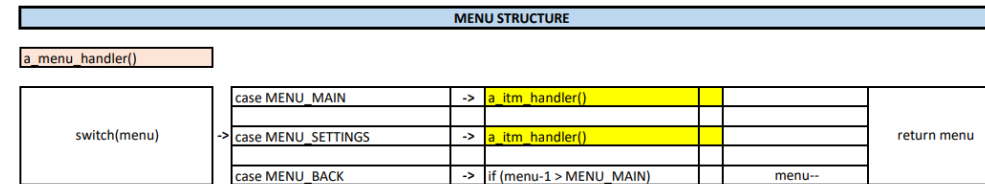
I gave the project the name 'rtc_encoder_combo'.

This project assumes that the reader is familiar with programming in C++; is also familiar with the Raspberry Pi Pico SDK; is familiar with tools like CMake. For readers with less experience, there is the binary file 'rtc_enc.uf2' (see page 11) that you can flash to the RPi Pico.

The image below shows the folder structure of the project:

Nome	Data de modificação	Tipo	Tamanho
.git	05/08/2021 15:52	Pasta de arquiv...	
build	05/08/2021 15:46	Pasta de arquiv...	
common	01/07/2021 16:38	Pasta de arquiv...	
documentation	05/08/2021 15:47	Pasta de arquiv...	
drivers	30/06/2021 10:11	Pasta de arquiv...	
libraries	01/07/2021 17:32	Pasta de arquiv...	
src	01/07/2021 13:19	Pasta de arquiv...	
.gitattributes	05/08/2021 14:26	Documento de ...	1 KB
CMakeLists.txt	30/06/2021 13:45	Documento de ...	1 KB
LICENSE	05/08/2021 14:26	Arquivo	2 KB
pico_sdk_import.cmake	23/02/2021 19:18	Arquivo CMAKE	3 KB
README.md	05/08/2021 14:26	MD Documento	1 KB

The following images show the menu structure and program execution points/parameters:

MENU STRUCTURE					

a_menu_handler()

switch(menu)	case MENU_MAIN	->	a_itm_handler()		return menu
	case MENU_SETTINGS	->	a_itm_handler()		
	case MENU_BACK	->	if (menu-1 > MENU_MAIN)	menu--	

a_itm_handler()

if is_pressed_btn_A	if menu=MENU_MAIN	->	btn_ab_itm(true)		
	elseif menu=MENU_SETTINGS	->	btn_ab_settings(true)		
if is_pressed_btn_B	if menu=MENU_MAIN	->	btn_ab_itm(false)		
	elseif menu=MENU_SETTINGS	->	btn_ab_settings(false)		
					return
if is_pressed_btn_X	if menu=MENU_MAIN	->	btn_xy_itm(true)		
	elseif menu=MENU_SETTINGS	->	btn_xy_settings(true)		
if is_pressed_btn_Y	if menu=MENU_MAIN	->	btn_xy_itm(false)		
	elseif menu=MENU_SETTINGS	->	btn_xy_settings(false)		

btn_ab_itm(bool dr)

switch(itm)	case ITM_SECONDS				
	case ITM_MINUTES				
	case ITM_HOURS				
	case ITM_WEEKDAY	->	decr_inc_datetime(dr); upd_app_time(); set_datetime();		
	case ITM_DATE				
	case ITM_MONTH				
	case ITM_YEAR				
	case ITM_RESET	->	if (overall_reset)	-> encoder_reset();	return true;
		->	else	-> reset_clk();	
	case MENU_SETTINGS	->	btn_msgd("menu settings, 0"); cout "btn_ab_itm(): menu ..."; pr_in_which_menu();		
	case ITM_HELP	->	btn_help();		
	default	->	not_yet(true);		

btn_ab_settings(bool disp)

switch(sett_itm)	->					
		case SETT_LED	->	success = backup_to_EP();		
		case SETT_12HR	->	success = backup_to_EP();		
		case SETT_ALRM	->	success = backup_to_EP();		
		case SETT_TIMR	->	success = backup_to_EP();		
		case SETT_BGND	->	set_background(); nsuccess = backp_to_EP();	->	return success;
		case SETT_FGND	->	set_foreground(); success = backp_to_EP();		
		case SETT_ENC	->	success = encoder_doit();		
		case SETT_BACK	->	menu = MENU_MAIN		
		default	->	not_yet(disp);		

btn_xy_itm(bool dr);

cout "btn_xy_itm(): "				
if (dr == false)	->	itm -= 1		
		if (itm < ITM_MINUTES)	->	itm = ITM_HELP
else if (dr == true)	->	sett_itm += 1		
		if (itm > SETT_HELP)	->	itm = ITM_MINUTES
pr_itmsel();				
UltItmChgdTime= rtc.getUNIX();				

btn_xy_settings(bool dr);

cout "btn_xy_settings(): "				
if (dr == false)		sett_itm -= 1		
	->	if (sett_itm < SETT_LED)	->	set_itm = SETT_BACK
else if (dr == true)		sett_itm += 1		
	->	if (sett_itm > SETT_BACK)	->	set_itm = SETT_LED
pr_itmsel();				
UltItmChgdTime= rtc.getUNIX();				

The menus

The idea of the menu structure is:

- There is a main menu that handles most of the basic realtime clock settings as: year, month, day, day-of-the-week, hour, minute, help. One item in this menu is to go to a second menu, called the 'settings menu';
- A 'second menu', the 'settings menu' is used to set: - 12/24 hour clock; led blink; timer; alarm (not yet implemented; rotary encoder; back to main menu.
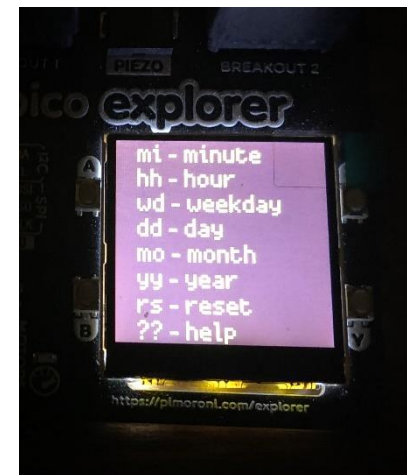
The main menu item selection is default set to the 'help' function.

There exist a 'main menu' and a 'settings menu'.

The following items are the main menu items:

- help;
- minute;
- hour;
- weekday;
- day;
- month;
- year;
- reset;
- menu;
-

The default menu-item is 'help'.

When in 'help' pressing either 'A' or 'B' button will display the help page.

When the user has chosen before another menu-item than 'help', after 30 seconds, the menu-item selection will revert to 'help'.

The main menu item 'menu' brings the program into the 'settings menu'.

To get into the settings menu the user has first to select the 'menu' item in the main menu, then press 'A' or 'B'. The program will then display for one second the text 'MENU SETTINGS'

The 'settings menu' has the following items:

- menu exit;
- encoder;
- f_gnd;   (to change the colour of the foreground (the text colour);
- b_gnd (to change the colour of the background);
- set timer;
- set alarm; (not implemented yet)
- 12/24 HRS;
- led onoff.
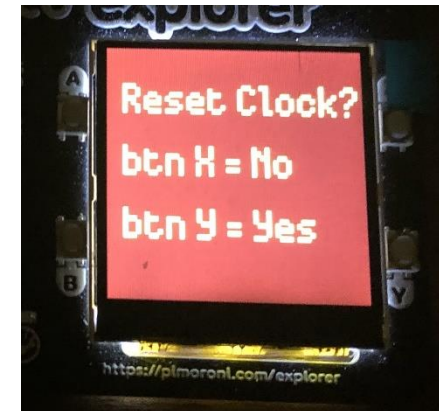
When switching forward or backward through the menu items, the current selection will be shown on display for a second or two. The user has to wait pressing any button until the notification text has disappeared.

Buttons

The project makes use of the four buttons ('A', 'B', 'X' and 'Y') that are on the pico explorer base, 2 x 2 on both sides of the lcd.

The buttons have been given the following functions:

Button	Function #1	Function #2 (answer to questions)
A	Increase (yy, mm, ..)	
B	Decrease (idem)	
X	Up (menu item)	No (e.g.: reset clock? Or blink led?)
Y	Dn (menu item)	Yes

For the following two settings the program will ask the user for confirmation:

'Reset clock?   'Blink LED?
BTN X = no   BTN X = no
BTN Y = yes'   BTN Y = yes'

The reset cycle takes about ten seconds. The reset cycle finishes with the message "DATETIME SAVED".

Rotary encoder

When the Rotary encoder function is activated it will search for the presence of a RGB encoder breakout board. If it does find such a breakout board, the function will display the address found (see image). If no RGB encoder breakout board is found, the function will show a message (see other image).
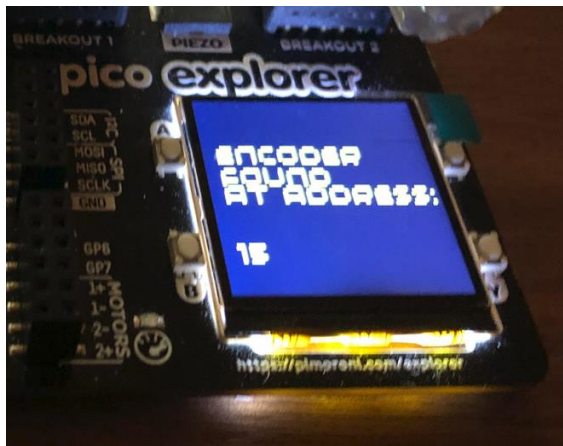
When the user turns the knob of the RGB encoder, the color of the led in the button will change. A corresponding resulting count value will be displayed (see the two images).
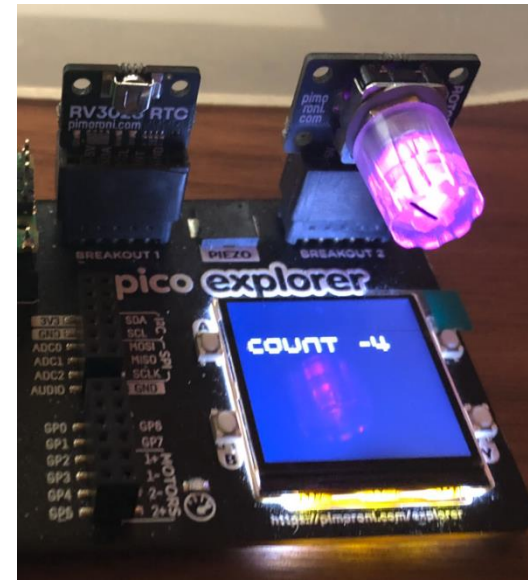
Because the realtime clock break module has a backup battery (no rechargeable !), the clock will keep the time, even when the raspberry pi is not powered via usb or another external electrical source, e.g.: a battery.

The software sets the external realtime clock module to detect when the RPi Pico is without of electrical power.

The reset clock function set the clock to: Saturday, May 1, 2021, 12:00:00.

The SET-TIMER function is shown in the image on the right:

At the end of the timer countdown there will appear a flashing screen as shown in the image below:

However the Pimoroni pico explorer base consists of a tiny speaker, my experience is that the volume of this speaker is too soft. That is why I did not add it to the timer alarm moment.

ToDo: add an audio output to one of the PWM pins of the Pico to drive a more audible speaker.

The flash'able .uf2 file (and other files) resulting from the build process is saved in the folder:

rtc_encoder_combo › build › src › main

Nome	Data de modificação	Tipo	Tamanho
rtc_enc.bin	02/07/2021 20:50	Arquivo BIN	385 KB
rtc_enc.dis	02/07/2021 20:50	Arquivo DIS	8 012 KB
rtc_enc.elf	02/07/2021 20:50	Arquivo ELF	8 176 KB
rtc_enc.hex	02/07/2021 20:50	Arquivo HEX	1 081 KB
rtc_enc.uf2	02/07/2021 20:50	Arquivo UF2	769 KB

Tools used to create and build the project:

- Microsoft Visual Studio Code (VSCode);
- Raspberry Pi Pico SDK (GitHub);
- Pimoroni libraries for various hardware used (GitHub).

This ends the report of my project rtc_encoder_combo.

Thank you for taking your time to read it.

Finally my thanks for the support of various people that I asked help for on my 'journey' into building C++ apps for the RPi Pico.

Feedback always appreciated.

Paulus Schulinck,
Lisbon, Portugal
June 11, 2021

@PaulskPt on GitHub;
@paulsk on Discord, CircuitPython (member of 'deepdivers');
@ct7agr on Twitter.