

FALL 2024

RCA 2020

I2C drivers for the RCA 2020

Design Document

RCA2020

An RCA CDP1802 COSMAC revival project

Designed by Richard van Harderwijk

I2C drivers Design Document

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Description/Changes** |
| 0.1 | March 2024 | Richard van Harderwijk | Initial Draft |
| 1.0 | Sep 2024 | Richard van Harderwijk | First Version |
|  |  |  |  |

Content

[Introduction 4](#_Toc178339781)

[High Level Design 5](#_Toc178339782)

[A brief description of the solution 5](#_Toc178339783)

[Requirements 5](#_Toc178339784)

[Potential use cases 5](#_Toc178339785)

[Data handling 6](#_Toc178339786)

[Routines 6](#_Toc178339787)

[Low Level Design 9](#_Toc178339788)

[System description – register selection 9](#_Toc178339789)

[Register use and variables 9](#_Toc178339790)

[Pseudo-code/flowcharts 10](#_Toc178339791)

[Return values and error handling 10](#_Toc178339792)

[Timing considerations 11](#_Toc178339793)

[Annex A – References and Additional Documentation 12](#_Toc178339794)

[Annex B – Flowcharts (application note AN10148) 13](#_Toc178339795)

[Initialization 13](#_Toc178339796)

[Controller Transmitter mode 14](#_Toc178339797)

[Controller Receiver mode 16](#_Toc178339798)

# Introduction

**About this document**

This document describes the design and implementation specifics of the drivers for the RCA2020 I2C interface [1].

The RCA2020 system is an SBC (Single Board Computer) based on the 1970’s RCA CDP1802 COSMAC CPU. While specific for this system, the interface should be easily portable to any other 1802 system, both the hardware and the software.

Substantial parts of the necessary low level implementations are already detailed by the I2C controller supplier ([3], [4]).

**Terminology**

The industry moved away from the old terms master and slave to *controller* and *peripheral*. These terms are used in this document, but in references you may still find the older terms.

# High Level Design

## A brief description of the solution

A part of the RCA2020 system [1] is an **interface from the 1802 CPU to the I2C bus**.

The RCA2020 I2C system consists of two hardware parts:

* The core: a controller chip (PCA9564) [3], [4] connected to the CPU, as described in the design, build and test manual [1]. This interfaces the CPU to the I2C bus.
* An on-board peripheral: a date/time and alarm chip (PCF8563) [1] connected via the I2C bus.

The four registers of the controller chip are mapped to INP/OUT 4..7 (not memory mapped).

This document describes the design of the drivers.

## Requirements

The initial requirements are basic:

* Act as the I2C controller
* Be able to send and receive information to peripherals

This in polled mode. While the RCA2020 facilitates interrupt modus, this is currently not a requirement.

Another requirement is to mimic the Arduino Wire library functionality. The drivers are written in assembler, but by synchronizing to this functionality, in the future they can be extended to an 1802 Wire.h library implementation. This opens a vast library of C-language drivers for all kind of peripherals.

The drivers have to offer error handling. Returning a ‘success’ value; or various error values.

In a later stage, using the send/receive routines:

* Be able to use the onboard RTC (date/time and alarm chip)

## Potential use cases

The I2C bus opens a world of peripherals for the 1802 CPU, think of:

* All kind of sensors and actuators
* Various input and output devices (ADC, DAC, IoT, LCD displays etc)
* Interfaces (ethernet, wifi, lora wifi, USB, etc)
* Mass storage (memory cards etc)
* All easily connectable via the two I2C wires and ground (apart from the necessary software/drivers specific to the peripheral)

Apart from these direct connectable peripherals, a future update may facilitate a connection to other computer systems, for example Arduinos, Adafruits, Raspberry Pi’s, realtime SBC’s and desktops. This can be directly via the I2C bus, or with an interface (for example ethernet, albeit a slow one). In these kinds of connections, the 1802 system can function as the controller or as a peripheral.

## Data handling

Data handling of the read/write driver routines is via a pointer.

* For sending, the main program is to assemble all bytes in a consecutive memory location and offer a pointer to the ‘write’ driver routine.
* For receiving, the main program is to free a consecutive memory location and offer a pointer to the ‘read’ driver routine.
* Both the read and write driver routines must be capable of handling one to multiple bytes in one call.

The (peripheral) on-board date/time and alarm chip (RTC) driver routine:

* Facilitate reading the values of the clock (date/time, alarms)
* Facilitate writing values to the clock (initial date/time, set alarms)
* In polled mode. While the RCA2020 clock implementation facilitates interrupt modus, this is currently not a requirement.

The drivers are to be called from a main program as (sub)routines.

## Routines

|  |  |  |
| --- | --- | --- |
| **Arduino wire.h** | **RCA2020 implementation** | **MVP** |
| begin() - Initialise the I2C bus | Routine ‘Initialization Sequence’ as described in [4] Flowcharts.  Modus to operate as peripheral not implemented in the MVP. | √ |
| end() - Close the I2C bus | Not in the MVP. By hard reset. | - |
| requestFrom()- Request bytes from a peripheral device | Your main program should allocate a consecutive memory location and offer a pointer to the Routine ‘Controller Receiver Mode‘ as described in [4] Flowcharts.  Your main program offers the peripheral address.  The routine reads the bytes from the peripheral and stores them in the allocated memory location. | √ |
| beginTransmission() - Begins queueing up a transmission | Your main program offers the peripheral address to the Routine ‘Controller Transmitter Mode‘ as described in [4] Flowcharts | n/a |
| endTransmission() - Transmit the bytes that have been queued and end the transmission | Routine ‘Controller Transmitter Mode‘ as described in [4] Flowcharts. Stop command is always sent in MVP. | √ |
| write()- Writes data from peripheral to controller or vice versa  No actual transmitting in this routine, writing to an internal buffer. | Your main program should assemble all bytes in a consecutive memory location and offer a pointer to the Routine ‘Controller Transmitter Mode‘ as described in [4] Flowcharts | n/a |
| available() - returns the number of bytes available for retrieval | Not in the MVP. | - |
| read() - Reads a byte that was transmitted from a peripheral to a controller. | Your main program can read all bytes in the reserved memory location | n/a |
| setClock() - Modify the clock frequency | In the MVP, a fixed clock freq is used. Can later be set via routine ‘Initialization Sequence’ as described in [4] Flowcharts. | - |
| onReceive() - Register a function to be called when a peripheral receives a transmission | No configuration of the RCA2020 as peripheral (yet). | - |
| onRequest() - Register a function to be called when a controller requests data | No configuration of the RCA2020 as peripheral (yet). | - |
| setWireTimeout() - Sets the timeout for transmissions in controller mode | Not in the MVP. Per 9564 controller default, the time-out function is enabled. | - |
| clearWireTimeoutFlag() - Clears the timeout flag | Not in the MVP. | - |
| getWireTimeoutFlag() - Checks whether a timeout has occurred since the last time the flag was cleared. | Not in the MVP. | - |

The I2C protocol, Arduino, other systems and bare metal I2C programming including lots of code examples are excellently described in [5].

MVP means Minimum Viable Product, the first version/basic implementation.

# Low Level Design

## System connection

The controller chip (PCA9564) is connected to the 1802 via the n-lines input/output.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A1** | **A0** | **Register Name** | **Register Function** | **Mapping to 1802** |
| 0 | 0 | I2CSTA | Status Register | INP 4 |
| 0 | 0 | I2CTO | Time-out Register | OUT 4 |
| 0 | 1 | I2CDAT | Data Register | INP/OUT 5 |
| 1 | 0 | I2CADR | Own Address Register | INP/OUT 6 |
| 1 | 1 | I2CCON | Control Register | INP/OUT 7 |

## Register use and variables

1802 Register usage and mapping :

### Initialization Sequence

|  |  |  |
| --- | --- | --- |
| **Register** | **High Order Byte – R(N).1** | **Low Order Byte – R(N).0** |
| R(7) | Future use (own peripheral address) | Clock frequency (bit [2..0]) |

|  |  |  |  |
| --- | --- | --- | --- |
| **R(7).0 bit2** | **R(7).0 bit1** | **R(7).0 bit0/LSB** | **Serial Clock Frequency (khz)** |
| 0 | 0 | 0 | 330 |
| 0 | 0 | 1 | 288 |
| 0 | 1 | 0 | 217 |
| 0 | 1 | 1 | 146 |
| 1 | 0 | 0 | 88 |
| 1 | 0 | 1 | 59 |
| 1 | 1 | 0 | 44 |
| 1 | 1 | 1 | 36 |

In the MVP a fixed clock frequency at 88 kHz (100kHz with tolerance).

In the MVP the own peripheral address is fixed at 0x64

### Controller Transmitter Mode

|  |  |  |
| --- | --- | --- |
| **Register** | **High Order Byte – R(N).1** | **Low Order Byte – R(N).0** |
| R(8) | Pointer to assembled byte(s) string to transmit | |
| R(7) | Peripheral address | Number of bytes to transmit |

The max number of bytes to be transmitted is 63, in bit [5..0], leaving the two MSB’s of R(7).0 available for future use.

### Controller Receiver Mode

|  |  |  |
| --- | --- | --- |
| **Register** | **High Order Byte – R(N).1** | **Low Order Byte – R(N).0** |
| R(8) | Pointer to memory allocated to store byte(s) received | |
| R(7) | Peripheral address | Number of bytes to receive |

The max number of bytes to be received is 63, in bit [5..0], leaving the two MSB’s of R(7).0 available for future use.

## Pseudo-code/flowcharts

In the RCA2020 implementation sending/receiving happens with the two controller routines (controller transmitter/receiver), after preparation by the calling main program. This in contrary to the multiple wire.h routines.

The flowcharts are described in application note AN10148 [4]. They are copied in this document in annex B.

The routines are to be called via the Standard Call and Return Technique (SCRT).

## Return values and error handling

On success:

R(7).1 and R(7).0 contain 0x00.

On error:

### Initialization Sequence

|  |  |  |
| --- | --- | --- |
| **Register** | **High Order Byte – R(N).1** | **Low Order Byte – R(N).0** |
| R(7) | Not implemented in MVP | Not implemented in MVP |

### Controller Transmitter Mode

|  |  |  |
| --- | --- | --- |
| **Register** | **High Order Byte – R(N).1** | **Low Order Byte – R(N).0** |
| R(7) | Expected I2CSTA value | Actual I2CSTA value |

If at the end the STO-bit is not 0; both R(7).1 and R(7).0 contain the general error value 0xFF

### Controller Receiver Mode

|  |  |  |
| --- | --- | --- |
| **Register** | **High Order Byte – R(N).1** | **Low Order Byte – R(N).0** |
| R(7) | Expected I2CSTA value | Actual I2CSTA value |

If at the end the STO-bit is not 0; both R(7).1 and R(7).0 contain the general error value 0xFF

## Timing considerations

The transmission of a byte via the I2C bus, generally takes 9 bits (8 bits for the byte and 1 acknowledge bit). Sending a byte @100 kHz takes 90 μs.

An 1802 takes 16 clock cycles for a two machine cycle instruction, that is 8 μs @2 MHz. This means the driver routines can at a maximum execute 11 instructions between each byte to/from the controller, before slowing down transmission at 100 kHz.

Transmitting 63 bytes @100 kHz takes about 6 ms (fully utilizing the CPU).

# Annex A – References and Additional Documentation

**References**

[1] RCA2020 system/github

<https://github.com/richardvanharderwijk/1802-SBC-the-RCA2020>

[2] MPM-201 User Manual for the CDP1802 COSMAC Microprocessor <http://www.bitsavers.org/pdf/rca/1802/MPM-201A_CDP1802_User_Manual_1976.pdf>

[3] Data sheet PCA9564

<https://www.nxp.com/docs/en/data-sheet/PCA9564.pdf>

[4] AN10148: PCA9564 – I2C-bus controller <https://www.nxp.com/docs/en/application-note/AN10148.pdf>

[5] The Book Of I2C – Randall Hyde; ISBN 9781718502468

**Additional Documentation**

Libraries for using various I2C devices with an 1802-Mini with the PIO and I2C expansion boards. – Various contributors from the COSMAC ELF Group [cosmacelf@groups.io](mailto:cosmacelf@groups.io) - <https://github.com/fourstix/Elfos-I2C-Libraries/tree/main?tab=readme-ov-file>

Documentation links for the RTC can be found in the RCA2020 documentation [1]

# Annex B – Flowcharts (application note AN10148)

## Initialization

From [4] Figure 5

Afbeelding met tekst, schermopname, diagram, document

Automatisch gegenereerde beschrijving

## Controller Transmitter mode

From [4] Figure 6

Afbeelding met tekst, diagram, ontvangst, schermopname

Automatisch gegenereerde beschrijving

Afbeelding met tekst, diagram, ontvangst, Parallel

Automatisch gegenereerde beschrijving

## Controller Receiver mode

From [4] Figure 7

Afbeelding met tekst, ontvangst, diagram, Parallel

Automatisch gegenereerde beschrijving

Afbeelding met tekst, diagram, ontvangst, Parallel

Automatisch gegenereerde beschrijving