TULIP4041 – The ULtimate Intelligent Peripheral for the 41

Introduction

In the mid 00's I had finished the MLDL2000 project, and started to consider a follow-up project that would be named the MLDL3000. Most important addition would be a microcontroller to perform more complex tasks, such as interfacing to a PC, printer emulation and saving and storing user programs and ROM images. This resulted in a working prototype, but real life and family interfered. And with the introduction of Diego's Clonix and NoVRAM modules and finally Monte's HP41CL I thought that the MLDL2000 or -3000 would simply be superfluous.

After starting to reduce my working hours I picked up some activities again in 2018, and late 2022 I decided to give it a go again, now with a slightly different platform. But still using an FPGA for the HP41 bus interfacing, and a microcontroller for the high-level stuff. In the spring of 2023 Andrew Menahue posted a message and video on the HP Museum forum, and that did it. A new approach using the RP2040 microcontroller that did all the work, including the low-level bus interfacing. Not entirely new, as Diego's modules also used a microcontroller to interface with the HP41 bus. The RP2040 however has so much more performance that it could run more complex tasks. After studying the datasheet I made a decision to go for it and in the summer of 2023 a first breadboard version was working. My initial goal was to use the PIO in the RP2040 for the HP41 bus interfacing, and to create emulation of the HP-IL module. After that steps were taken to create a product that is useable for the community.

During October 2024 the first units of the DevBoard were shipped and design of the module version started, and the design has been migrated to the RP2350 processor. After designing the final PCB and housing (by Andrew Menahue) and two design iterations In October 2025 the first final units were shipped.

This document describes the background and internal functioning of the TULIP. A separate document serves as the user manual.

Meindert Kuipers *Email: meindert@kuiprs.nl*

November 2025

1. Credits

Many thanks to Andrew Menahue for inspiring this project, his cooperation and the design of the TULIP housing, Thomas Fänge for his contributions, cooperation and testing. HP-IL emulation is based on V41 and EMU41 by Christoph Giesselink and Jean-Francois Garnier, PILBox emulation is based on the PILBox from Jean Francois Garnier.

2. Version info

The TULIP4041 is a project that is in constant change. This documentation may not match the current available version but rather serves as a specification for the TULIP4041, and this means that is may describe functionality that is not (yet) implemented. Refer to the TULIP help menu for the actual supported functions of your firmware version.

IMPORTANT: This document and firmware is migrated to the RP2350. The RP2040 is no longer supported by the TULIP firmware.

3. Conventions

`00FF`	Hexadecimal numbers are used to indicate addresses, leading zeroes are typically used to indicate the total possible range, e.g. 00FFFF. The 'x' character is used for a "don't care" situation
`0x040`	The 0x sequence is used to indicate hexadecimal values when context is not clear
`0b1010`	Binary values are preceded with the 0b sequence, the 'x' character is used for don't care bits
HP41	HP41 indicates all versions of the HP41 calculator, including HP41C, CV, CX, CL, DM41X
41CL	Is used when specifically referring to the HP41CL system, also 41CL is used
help	is a command to be typed in the TULIP4041 Command Line Interface (CLI)

4. Copyrights and Disclaimer

Information in this document and the function of the hard- and software has been carefully checked and is believed to be accurate as of the date of publication; however, no responsibility is assumed for inaccuracies. I will not be liable for any consequential or incidental damages arising from reliance on the accuracy of this document and the use of the hard- and software. The information contained herein is subject to change without notice.

Table of Contents

5. TULIP4041 overview

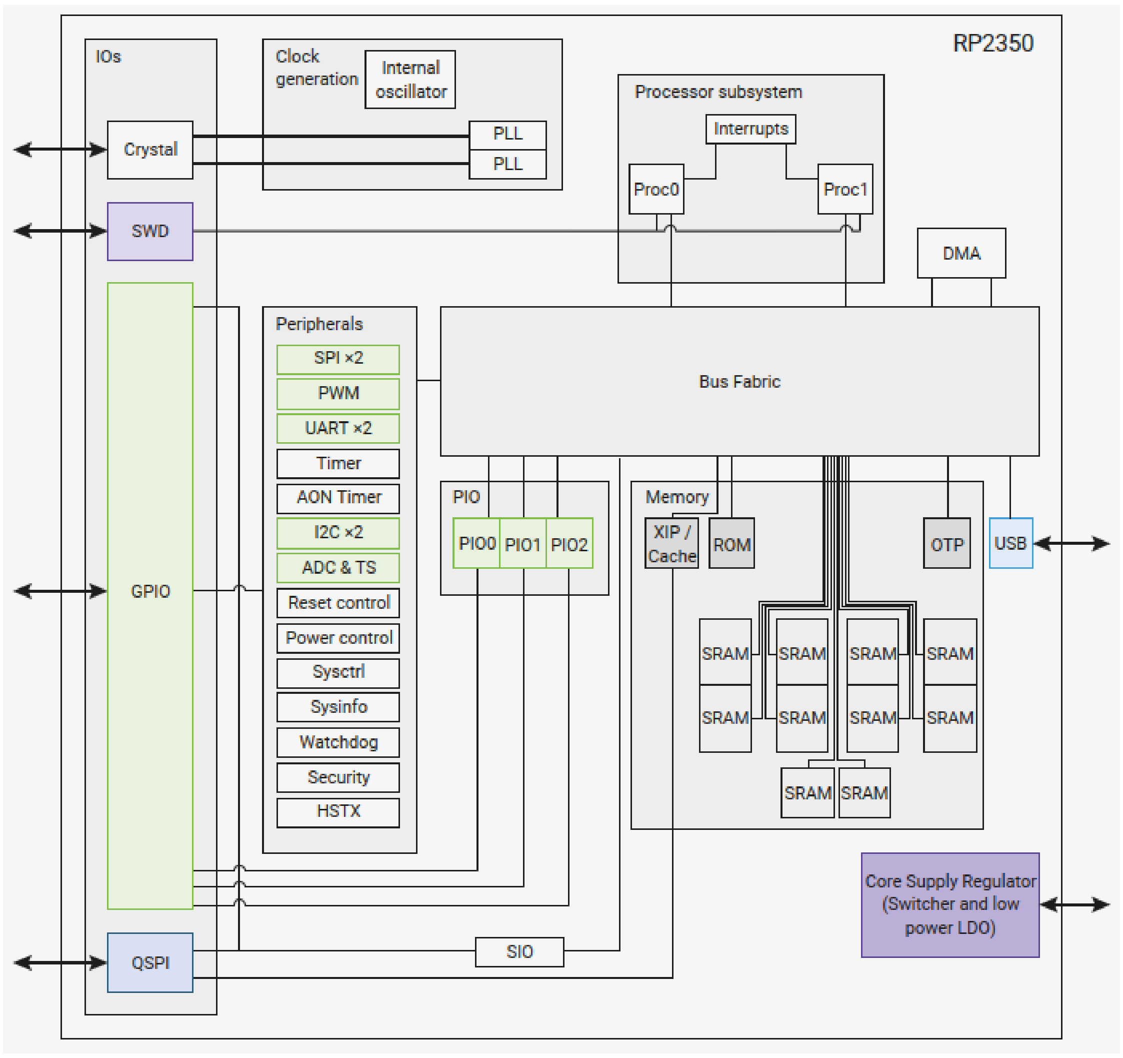
The TULIP4041 is a smart peripheral for the HP41 series of calculators. It is capable of emulating nearly all of the original memory and peripheral modules and offers a number of new features for the HP41. Some of the features it supports:

- Extended Functions and Extended Memory module emulation
- User Memory emulation
- HP-IL and HP-IL printer to virtual devices on a host computer
- HP82143A printer by printing to an IR port or virtual serial port to a printer simulator on a host computer
- Plugging virtual ROM and MOD images, including those with MLDL (QROM) functionality
- HEPAX emulation
- TIME module emulation
- USB-C interface
- Easy firmware upgrade
- Open Source hardware and firmware (MIT license)
- Micro SD card that appears as a USB thumb drive on the host computer for exchanging ROM and MOD images
- Multi-port virtual serial interface to the host computer for the following functions:
    - Command Line Interface for plugging/unplugging ROMs and control functions
    - HP41 mcode level bus tracer with disassembler
    - Virtual HP-IL serial interface according to the PILBox protocol for connection with virtual HP-IL devices on the host computer
    - HP-IL frame monitoring
    - Virtual printer connection for use with a HP82143 simulated printer of a host computer
    - Most functions are platform independent and work on Windows, Apple and/or Linux

Note: many of the above functions are implemented, some are not (yet) implemented as the firmware is in constant development.

The TULIP4041 firmware is originally designed for the RP2040 microcontroller and has been migrated to the RP2350A in September 2024. This controller is designed by Raspberry Pi. The device is very low cost (typically between €1 and €2 depending on quantities) and offers the following main features:

- Dual-core Arm Cortex-M33 processor, flexible clock running up to 150 MHz (overclocking possible)
- 520kByte on-chip SRAM
- Up to 16 MByte external QSPI flash
- 2 × UART, 2 × SPI controllers, 2 × I2C controllers, 24 × PWM channels
- 1 × USB 1.1 controller and PHY, with host and device support
- 12 × Programmable I/O (PIO) state machines (3 PIO blocks) for custom peripheral support
- Operating temperature -40°C to +85°C
- Drag-and-drop firmware programming using mass storage over USB
- Low-power sleep and dormant modes
- Temperature sensor
- Accelerated integer and floating-point libraries on-chip
- Excellent support for multi-platform development tools and many (open source) libraries



RP2350 BLOCK DIAGRAM (FROM RP2350 DATASHEET)

The TULIP4041 uses additional hardware:

- 4 or 16 MByte of FLASH memory, part of the RP2350 structure, code actually runs from the flash (XIP) and most of the ROM images are stored in flash and directly accessed there (4 MByte for the DevBoard with the standard Pico2 module and 16 MByte on the module version)
- 256 KByte FRAM memory to emulate QROM, user memory and for saving system settings
- Micro SD card holder
- Infrared LED for IR printer operation
- Level shifters to interface with the HP41 bus
- Real Time Clock (on the module version only)

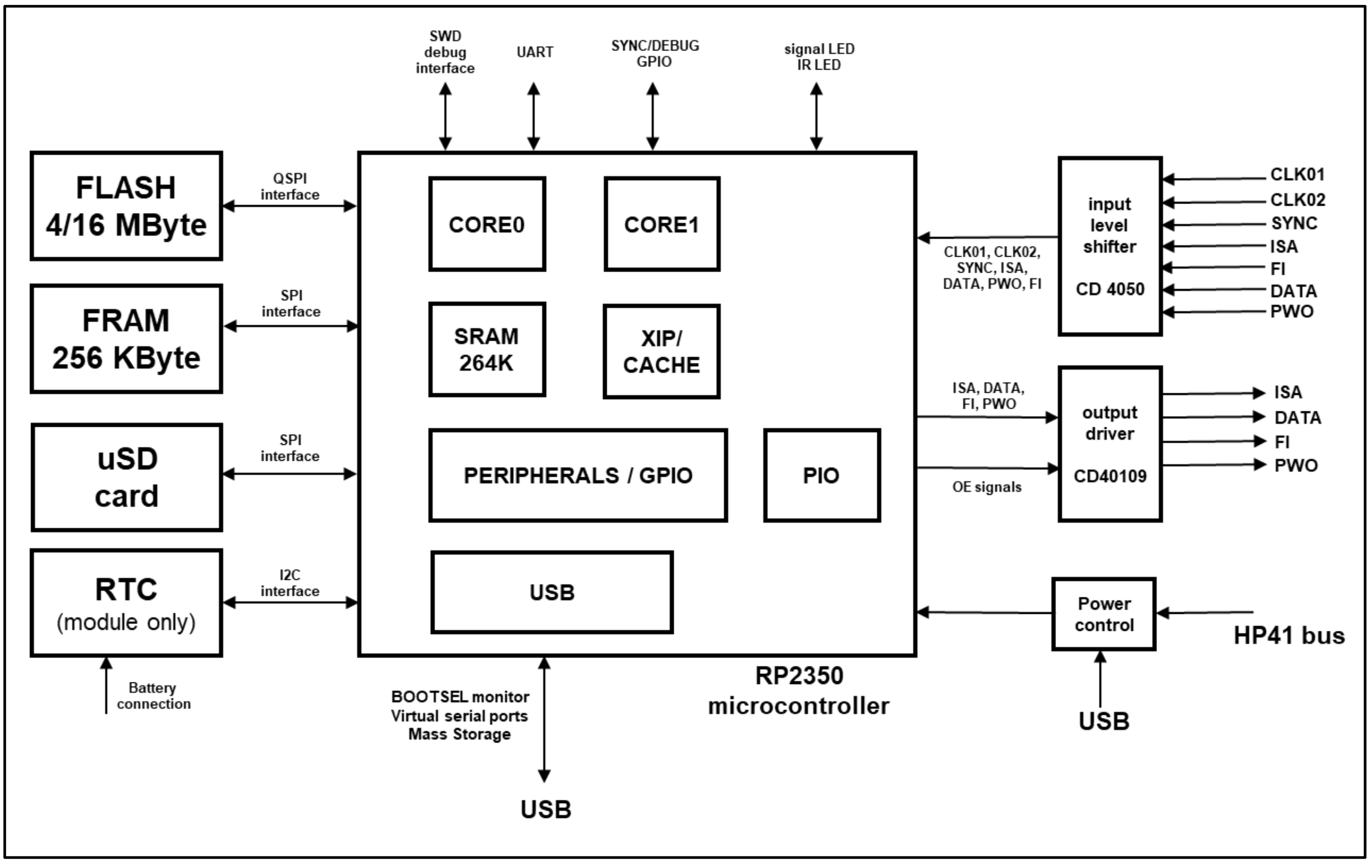
RP2350 development boards are available from several vendors, including the Raspberry Pico2 which is used in the prototyping phase of this project.

In this project both ARM cores are used, and two of the PIO blocks are used to almost their full capacity for the HP41 bus interfacing. The RP2350 has two RISC-V Hazard3 processor cores that can be enabled instead of the ARM cores, these are not used on the TULIP4041.

The TULIP4041 comes in two different versions:

- TULIP DevBoard: a hand solderable board with all interfaces available on headers for development and debugging. This version is mainly intended for my own development to replace the fragile breadboard. It uses a standard Pico2 board (or 100% pin compatible) and comes with a connector board for plugging in the HP41
- TULIP Module a long module sized PCB with all smd components assembled except the HP41 module connector. This unit has the size of an HP-IL module and is housed in a custom 3D printed module housing

IMPORTANT: the HP41 module connector is NOT included with either product!



TULIP4041 BLOCK DIAGRAM

6. TULIP4041 Firmware Architecture

The TULIP4041 firmware has three main parts:

- PIO state machines running the low-level HP41 bus interfacing, timing and synchronization
- Core1, running the HP41 emulation layer in sync with the HP41 bus interface
- Core0, running the initialization, user interface, communication, and other non-critical tasks

6.1. PIO State Machines

The RP2350 microcontroller has 3 PIO (Programmable I/O) blocks, of which 2 are used for low level HP41 bus interfacing, synchronization and timing. Each PIO block has up to four state machines, and a code space of 32 instructions. Data between the cores is exchanged using software fifo's (with the queue functions). The TULIP4041 implementation uses 2 PIO blocks to almost their full capacity.

In the description of the state machines the HP41 bit timing starts at T0, the time that data bit 0 appears on the HP41 bus. The counting is a bit different from most MLDL-type hardware that starts counting at the end of the SYNC pulse.

For synchronization between the state machines two GPIO signals are used: SYNC_TIME and T0_TIME. The nature of the HP41 timing and the requirements of the other state machines made the use of GPIO signals necessary, as interrupts in the PIO blocks have some limitations.

6.2. SYNC State Machine

After initialization the SYNC state machine waits for the first rising edge of SYNC and then enters its main loop. This is the main state machine that ensures correct synchronization, CLK counting and control of the external signals to synchronize the other state machines. This state machine is also used for ISA input.

- The external synchronization GPIO signal SYNC_TIME becomes active during each SYNC cycle, even when no SYNC appears on the HP41 bus (in case of a non-instruction fetch or the second word of an instruction (XQ/GO, LDI or a peripheral instruction)
- 10 HP41 bit times are counted, and at each rising edge of CLK01 a bit from ISA is pushed in the state machines ISR (Input Shift Register)
- After the 10th bit ISA is sampled again together with SYNC, in order to give the emulation layer access to the SYNC status during the instruction fetch.
- The ISR is then pushed to the state machine RX FIFO, to be read by the emulation layer in core1. This happens during T54, briefly before the end of the SYNC cycle.
- SYNC_TIME is driven low
- The state machine waits two CLK02 cycles before the start of T0
- At the start of T0 (rising edge of CLK02) the external GPIO T0_TIME is driven low (it is an active low signal)
- At the end of T0 (next rising edge of CLK02) the T0_TIME signal is de-asserted (driven high)
- From T1 the state machines starts counting (with a counter initialized during PIO initialization) and samples data from ISA. At the end of the ISA address the address is autopushed to the RX FIFO
- The state machine continues counting until the counter reaches 0 at exactly the start of SYNC

Upon a PWO event (falling or rising edge) the SYNC state machine is reset by a forced jump to the first instruction to wait for the first rising edge of SYNC. This is done by an interrupt handler.

The main emulation loop in core1 can do a blocking wait for data from the RX FIFO. The first data it receives is the ISA instruction, these are 12 bits in the following format:

SYNC status		**ISA instruction (10 bits)**									
SYNC	bit 9	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

The operation of the SYNC state machine captures the instruction bit 9 a second time, plus the status of SYNC.

After processing the software does a second blocking wait for receiving the ISA address. The 16 least significant bits of the captured data is the 16 bit address.

6.2.1. DATAIN State Machine

For sampling the DATA line a separate state machines uses the T0_TIME signal to synchronize. The state machine does a constant sampling of DATA on the rising edge of CLK01 and uses the AUTOPUSH feature so it does not have to count the bits. This state machine runs at 12.5 MHz to prevent waiting for both CLK01 edges. By using a slower clock it can skip the wait for the falling edge of CLK01 by using a delay of 15 clocks after the IN instruction.

- The state machine starts with a blocking wait for the rising edge of SYNC_TIME, this will usually be the first SYNC after a PWO event
- The state machine starts sampling DATA on the rising edge of CLK01 until it sees T0_TIME low
- When T0_TIME is low the data in the ISR is pushed and the main loop of the state machine is entered again

The state machine relies on AUTOPUSH with a count of 32. This means that at T32 the first 32 bits of data appear in the RX FIFO, and at T0 the remaining 24 bits.

The main software loop must empty the RX FIFO at each cycle to prevent the state machine to block.

Upon a PWO event (rising or falling edge) the state machine gets a forced jump to the start, waiting again for a new SYNC cycle.

6.2.2. DEBUGOUT State Machine

This is a very simple state machine, simply sending the TX FIFO contents to a debug output. This can be used by the main loop to indicate its position by putting a code or number of bits in the TX FIFO. This state machine uses AUTOPULL at 16 bits and consists of only one OUT instruction.

The debug output is used together with the T0_TIME and SYNC_TIME outputs for tracing the inner workings of the TULIP4041, and these signals are available on separate GPIO pins.

6.2.3. ISAOUT State Machine

The ISAOUT state machine has three functions:

- Driving ISA output (and ISA output enable) when it has to provide data during the instruction time (SYNC_TIME high)
- Driving ISA for one bit-time during T0 to transmit a carry status after an instruction has requested a status (typically a peripheral status, in practice used only by the HP82143A printer)
- Drive ISA when the calculator is idle (light or deep sleep) to wake up the calculator for an I/O event

The state machine is normally stalled at a blocking pull. When data arrives in the TX FIFO is will then wait for the start of T0_TIME to output one bit for the carry status transmission. At the end of T0 it will return to the blocking pull from the TX FIFO.

To transmit an ISA instruction, the main software loop must do a forced jump to the label isa_inst_out. Here is a blocking pull from the TX FIFO, and upon receiving data it will wait for the start of SYNC_TIME and then transmit the 10 instruction bits at each rising edge of CLK01. When done it will return to the blocking wait for the carry status. This setup has been chosen to allow the shortest possible time for transmitting the carry bit, since we have only two clocks after receiving the instruction and sending the carry at T0_TIME.

A wake-up of the calculator is done outside the state machine. Since the calculator is not running (PWO is low) it does not make sense to run the state machine as it depends on CLK02. Instead a C routine will do the following:

- Use a pio set instruction to drive ISA_OUT high and a sideset to drive ISA_OE
- Wait 20usecs
- Use a pio set instruction to de-assert ISA_OUT and ISA_OE

Like the DATAIN state machine, this state machine also runs at 12.5 MHz.

6.2.4. FIIN State Machine

The FIIN state machine is used to capture the state of the FI signal. It is identical to the DATAIN state machine and actually uses the same code, but with its own context. The FI input is used for tracing the HP41 bus only and is not available on the TULIP Module, only on the DevBoard. On the module version the FI output is send to the Trace buffer. The Module version has a single bit level shifter that may be used for FI input, this requires a wire to be hand soldered and a firmware implementation (which is not available). The firmware for the module version does not initialize this state machine.

6.2.5. DATAOUT State Machine

The DATAOUT state machine drives the DATA output signals and the DATA output enable signal. Input is the 56 bit pattern to be output on the DATA line. The state machine only drives the DATA output enable when data is present in the TX FIFO. It is possible to only put one byte or word in the TX FIFO, this word will be shifted out to DATA with the LSB first, and the rest of bits on DATA will be zero. The first bits will then end up in the S&X of the C register. This state machine runs at 12.5 MHz.

6.2.6. FIOUT State Machine

The FIOUT state machine is almost identical to the DATAOUT state machine to drive the FI output enable signal. Since the FI signal on the HP41 bus is an active low signal, we only drive the output enable of the output driver IC, the input of the driver is tied to GND. The OE signal is active high. The input to the FIOUT state machine is a 56 bit word, with the first 3 bits of a set flag high, the 4th bit is low to allow the bus to settle. The original specs of the HP41 demand pre-charging this signal to high after being driven, but tests have shown that this is not necessary.

6.2.7. IROUT State Machine

The IROUT state machine drives the IR LED with the correct Redeye protocol and timing. The state machine is designed to send half bits to the IR led. This is a trade-off between available code space in the PIO block and required functionality. To meet the IR timing requirements the state machine needs to runs at the double IR carrier frequency of 32768 kHZ which is achieved by properly setting the state machines clock divider.

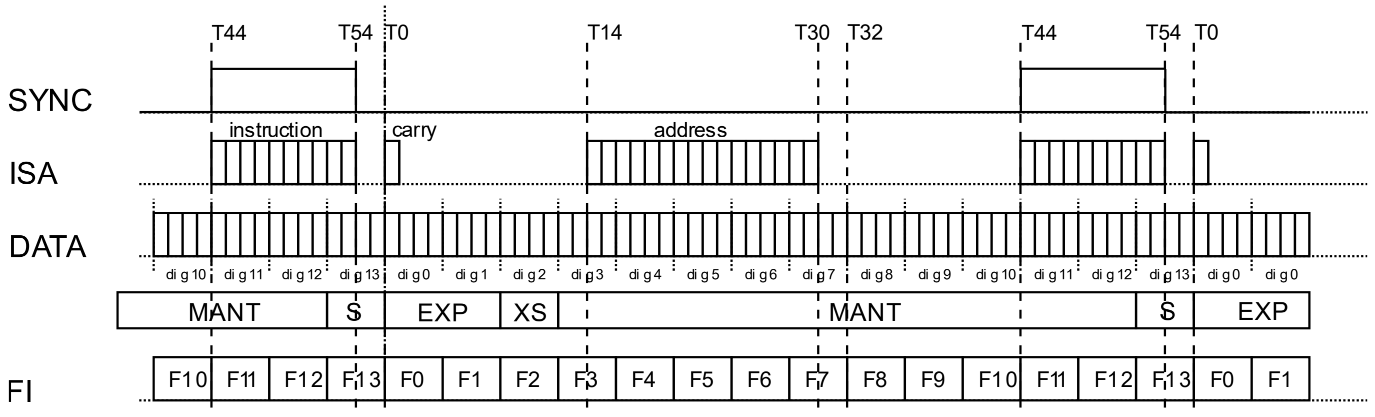
To output a frame the C level software needs to prepare a specific pattern (with checksum) of half bits as follows:

- start frame : 3 hi-lo transitions
- 0-bit : 1 hi-lo, 1 lo-lo
- 1-bit : 1 lo-lo, 1 hi-lo
- the irout state machine sends the following:
  input 0-bit : send lo-lo frame. put 01 in the output frame (lsb sent first)
  input 1-bit : send hi-lo frame, put 10 in the output frame (lsb sent first)

Summarizing, for sending an 'A' character, the frame to send to the state machine is the following:

```
-   0000.0111.1010.0110.0110.0101.0101.0110
-          ^^^ start bits
-               ^^^........................^^  payload 24 bits
```

6.3. Principle of Operation, core1



The RP2350 core1 runs the time critical software that does the actual emulation of HP41 peripherals and interaction with the HP41 bus. It runs in sync with the HP41 bus and is driven by the data it receives from the PIO State Machines. These provide data at specific HP41 bus events. To keep up with the timing of the bus the response must be in time for the next critical event. The core1 main loop runs from SRAM to prevent cache misses when running from FLASH.

The core1 software is started by core0, and core0 runs the main TULIP application program, controlling the Command Line Interface and all communication (USB, SD Card) and non-time critical peripheral operation. The DEBUGOUT is typically driven just before all the Tx events.

On a regular HP41 the time from T0 to T0 is between 152 and 159 microseconds. The time between two rising edges of the CLK is about 2.8 microseconds. It is possible to modify an HP41 to increase the clock speed with a T0-T0 time of about 106 microseconds. Preliminary tests have shown that the TULIP firmware can handle that speed.

6.3.1. Initialization

All state machines are initialized to wait for one of the synchronization signals, or to wait until valid data appears in the TX FIFO. The SYNC state machine waits for the first SYNC appearance. When PWO goes high (calculator switched on and going to RUN mode) most state machines are forced to their waiting state.

The main core1 loop starts with a blocking wait for data to appear from the first SYNC after power on and returns to this waiting state when the calculators is switched off or goes into STANDBY mode (PWO low).

6.3.2. T54, Start of core1 loop

After startup and initialization of the system and the PIO's the core1 software is idle and waits for the first data to arrive from the SYNC state machine, this will always be the ISA INSTRUCTION. The SYNC state machine is triggered by the PWO interrupt handler to wait for the first SYNC after power on, and when that appears it will capture the ISA bits during SYNC time, and present the data during T54. At this event the main loop in core1 gets the data and can start the HP41 emulation.

At T54 the software receives 12 bits of the instruction (or data) as presented on ISA. The lower 10 bits (0..9) are the actual instruction bits, the highest bit (bit 11) represents the status of SYNC during the instruction.

The main challenge immediately after T54 is to ensure that the deadline at T0 is met. At T0 the software must have presented the first 32 bits for the DATA line in case any instruction came by that requires driving the DATA line. These are all variations of the READ instruction and peripheral instructions to read registers. In addition, some peripheral instructions require to drive the carry (during T0_TIME) on the bus. These instructions must be handled first and within 2 HP41 CLK cycles.

The FIOUT state machine also requires that any flags to be driven on the FI line are presented to the state machine before T0_TIME.

Before T0 the data must be sent to the DATAOUT state machine, and if any flags are to be driven these must be sent to the FIOUT state machine. Driving the carry during T0 is done by the ISAOUT state machine by writing a single '1' bit to its TX FIFO.

6.3.3. T0, DATA complete

After the critical work is done to be ready for T0, the main loop does a busy wait for data arriving from the DATAIN state machine and as long as PWO is high. These are the remaining 24 high bits of DATA. When the calculator is going into light or deep sleep, PWO will be low and there may not be any more data coming since all the clocks will stop and the DATAIN state machine may stall.

Important now is to do a check if PWO is indeed high. If it is low the RX FIFO of the SYNC state machine is emptied, and also the PWO interrupt handler will ensure that all state machines are in a known and stable state. This can also be used to put the RP2350 in a low power mode.

When PWO is still high after T0 the core1 loop can continue to process any instructions that did not have high priority. Before that the information for the HP41 bus tracing is completed by reading the FIIN RX FIFO. That runs in sync with the DATAIN state machine and its data is now also available. The complete trace information is pushed (non-blocking) into the trace queue. If the trace queue was full the trace sample is discarded, and an overflow will be noticed by the tracer software in core0 reading the queue.

Emulation actions that should be done now are the following:

- Pending write that was waiting for the DATA to be completed
- Pending read (completion of read that was handled between T54 and T0) of the higher data bits, these must be provided to the DATAOUT state machine before T32
- RAMSLCT instruction, this is used to write any cached data back to FRAM due to the FRAM speed. We must wait until T32 when data becomes available to know the new selected register

- HP-IL (receive frame handling), Wand (handling of incoming data)
- Bank switching
- HEPAX instructions (not implemented yet)
- All other supported instructions that do not need any DATA

The software has time for all this until T30, which is the next deadline, although this is not critical. After all activities are done the next step is to get into a blocking read of the SYNC state machine.

6.3.4. T30, ADDRESS complete

The ISA address is complete at T30. The start of the address is fully handled by the SYNC state machine, all we have to do is wait for data to be ready. It is also not so relevant if the activities before have taken a bit longer, as long as the next activities are completed before the next deadline, which is at T54.

When the ISA address is complete this can be used to get data from any of the active emulated ROM images (and taking bank switching into account). In the case of FRAM this may take a bit of time due to the speed of the SPI interface that controls the FRAM. Since the ISAOUT state machine is normally waiting for a carry to be sent (which must be done with priority between T54 and T0) this state machine must receive a forced jump to the correct offset in the PIO code with a pio_sm_exec() function. After that instruction has been issued the data from the ROM image can be pushed in the ISAOUT TX FIFO. The deadline for providing the instruction word is at T44.

6.3.5. T32, DATA first 32 bits complete

After T30 the main core1 loop waits for the data bits 00..31 from the DATAIN state machine at T32. This is close to T30, but not critical. At this time any operations needing any of the lower data bits can be executed. Examples of these instructions are:

- WROM, write word to QROM
- Peripheral instructions using data from the C register
- RAMSLCT , mark address and prefetch if the selected register is in our FRAM
- PRPHSLCT, mark active peripheral
- All WRIT instructions, cache data bits and mark a pending write for use after T0 (when remaining data bits are available)
- Write to HP-IL registers. When a write is done to the HP-IL output register the frame should be sent to the HP-IL out queue to trigger the actual sending of the data (by core0)

In essence, all needed work is now done, as long as any of the above operations are completed before the next T54.

At this point also the HP-IL status register is used to check if any flags need to be driven to set the flag output register that is used to drive the FIOUT state machine. This must be done before T0 to ensure driving the FI line in time.

6.3.6. PWO interrupt handler

Critical in the operation of TULIP4041 is the handling of the PWO signal. An interrupt handler is used on both the rising and falling edge of PWO. This allows the state machines to be brought into a known state and to enable proper synchronization to the HP41 signals.

It should be noted that after the initial SYNC the state machines fully rely on counting the CLK01 edges in the SYNC state machine. There is no possibility to resync if a clock is missed.

6.4. Core0 firmware

The core0 firmware runs the tasks that are not time critical, and does the initialization of the IO, TinyUSB stack, SD card, buffers, PIO, starting the core1 firmware and other housekeeping. It then enters an endless loop with the following calls:

- PowerMode_task: checks the HP14 powermode and logs this. Puts the TULIP4041 in low power mode if needed (not in the bETA software)
- tud_task: call to the TinyUSB stack to process any data requests pending for USB communication
- runCLI: call to the Command Line Interface to process incoming commands
- trace_task: to process HP41 bus trace information data coming from the core1
- print_task: process any data to be printed (both HP82143 and IR printing) coming from core1
- HPIL_task: process HPIL data request (incoming and outgoing) and control the HP-IL scope output
- TULIP_IF_task: process any requests from the emulation layer for the TULIP ROM (commands, files, etc)
- Any other tasks used for handling traffic to and from peripherals

The tasks above all deal with the communication between TULIP4041 and the USB connection and/or the file system on the SD card. The tasks in core0 should prevent blocking as much as possible. The USB interface and the CLI rely on polling in the main loop to handle all communication, and when another task is blocking the USB communication is also halted. During some of the tasks it may be necessary to call the Tiny USB stack to process data, especially for tasks that take a bit more time or send/receive data to one of the USB serial ports.

The RP2350 has a default clock speed of 150 MHz. To reduce power consumption the TULIP fimware runs at 125 MHz.

7. TULIP4041 interfaces

The TULIP4041 has a number of interfaces to the outside world:

- HP41 module interface, using level shifters to convert the 6V HP41 signals to the 3.3 V RP2350 compatible levels, and vice versa.
- RP2350 debug interface
- RP2350 USB interface
    - In BOOTSEL mode it offers the possibility to upload a new firmware image and to program FLASH memory
    - In normal running mode (TULIP4041 firmware) offers the following virtual serial ports
        - Command Line Interface (CLI)
        - HP41 mcode/bus trace output
        - HP-IL communication
        - HP-IL scope output
        - Printer data (output from TULIP4041 only)
        - Mass Storage (MSC) device for exposing the micro SD card to the host computer
- Micro SD card
- Real Time Clock (only on the Module version) for possible TIME module emulation
- Serial interface (digital 3.3V level only)
- RP2350 GPIO pins
- IR led
- On-board signalling LED
- Auxiliary signals for multi-core communication, debugging en future additional functions
- I2C signals for connecting I2C peripherals (used for the RTC) like an external display

The TULIP4041 interfaces with the HP41 using level shifters. These are needed to convert the 6V HP41 signals to the 3.3 V RP2350 compatible levels, and vice versa. All other signals are 3.3V (please check the RP2350 datasheet for details). The RP2350 is now certified for 5V signaling, but that is still not suitable for the 6V signals of the HP41.

7.1. TULIP4041 PICO/ RP2350 pinout

This paragraph handles the I/O as seen from the RP2350 microcontroller, the Pico2 board (for use with the TULIP DevBoard) and the final products (DevBoard and Module).

RP2350 GPIO	Development board function (Pico board pinout)		Module version (RP2350 pinout)	
GP0	UART TX/I2C0 SDA	probe/debug	UART TX/I2C0 SDA	debug/aux serial port
GP1	UART RX/I2C0 SCL	probe/debug	UART RX/I2C0 SCL	debug/aux serial port
GP2	FI input	FI input	I2C1 SDA	RTC/display/aux I2C
GP3	IR output/PWO out	IR output/PWO out	I2C1 SCL	RTC/display/aux I2C
GP4	SPI0 RX	FRAM SO	FRAM SO	FRAM SO
GP5	SPI0 CSn	FRAM CS	FRAM CS	FRAM CS
GP6	SPI0 SCK	FRAM SCK	FRAM SCK	FRAM SCK
GP7	SPI0 TX	FRAM SI	FRAM SI	FRAM SI
GP8	SPI1 RX	uSD card DO	uSD card DO	uSD card DO
GP9	SPI1 CSn	uSD card CS	uSD card CS	uSD card CS
GP10	SPI1 SCK	uSD card SCK	uSD card SCK	uSD card SCK
GP11	SPI1 TX	uSD card SI	uSD card SI	uSD card SI
GP12	CLK01	HP41 CLK01 bus input	CLK01	HP41 CLK01 bus input
GP13	CLK02	HP41 CLK02 bus input	CLK02	HP41 CLK02 bus input
GP14	ISA_in	HP41 ISA bus input	ISA_in	HP41 ISA bus input
GP15	SYNC_in	HP41 SYNC bus input	SYNC_in	HP41 SYNC bus input
GP16	DATA_in	HP41 DATA bus input	DATA_in	HP41 DATA bus input
GP17	PWO_in	HP41 PWO bus input	PWO_in	HP41 PWO bus input
GP18	ISA_out	HP41 ISA bus output	ISA_out	HP41 ISA bus output
GP19	ISA_OE	HP41 ISA output enable	ISA_OE	HP41 ISA output enable
GP20	DATA_out	HP41 DATA bus output	DATA_out	HP41 DATA bus output
GP21	DATA_OE	HP41 DATA output enable	DATA_OE	HP41 DATA output enable
GP22	FI_OE	HP41 FI output enable	FI_OE	HP41 FI output enable
GP23	(NA on Pico2 pins)	not used	SPARE1	HP41 PWO output (optional) HP41 FI Input (optional)
GP24	(NA on Pico2 pins)	VBUS present	VBUS present	USB power connected
GP25	(NA on Pico2 pins)	on-board LED	on-board LED	activity/diagnostics LED
GP26	T0_TIME	PIO synchronization	T0_TIME	PIO synchronization
GP27	SYNC_TIME	PIO synchronization	SYNC_TIME	PIO synchronization
GP28	P_DEBUG	debug output	P_DEBUG	debug output
GP29	(NA on Pico2 pins)	not used	IR output	IR output

Signals in green have the same function on both the DevBoard and Module. The differences between the DevBoard and Module version are the following:

- The module version has a separate level shifter for the FI input, which is not connected to any input. Optionally this may be wired to a GPIO input to enable FI tracing. By default the tracer on the module version will only show FI signals as driven by the TULIP
- The DevBoard has a shared output for the IR led and PWO out, and only one of these can be used. This is selectable with a jumper
- On the DevBoard GPIO2 and GPIO3 can be used for I2C when FI input and PWO/IR output are disabled with an open jumper

- On the Module version the PWO output can be connected using a 0-Ohm resistor to the SPARE1 signal (GPIO23), this is not fitted by default

On the Module version the signals for the Serial Wire Debug port (SWDIO and SWCLK) are available, plus the RUN signals (can be grounded to reset the processor) and BOOTSEL (ground during USB plugging to get the processor in BOOTSEL mode).

7.2. TULIP4041 memory layout

The TULIP4041 has a number of different memory types

- 520 KByte on-chip SRAM, used by the RP2350 for its own data and execution of critical parts of the program. This memory is dual ported for fast access by both cores. When used, the TraceBuffer typically takes up most of the available SRAM. Biggest user of RAM is the queue used for the tracebuffer. In a future firmware version it is anticipated that a copy of emulated User Memory will reside in RAM.
- 4 MByte FLASH memory (on the Pico2 board), used for program storage (and program execution) and storage of ROM images. The first 1 MByte is reserved for code storage, the rest is available for ROM images
- 16 MByte FLASH memory on the module version, of which the first 1 Mbyte is used for firmware code storage, leaving 15 MByte for the file system
- 256 KByte FRAM, used for QROM images, Extended/Expanded/User memory emulation and persistent storage of settings
- Micro SD card, storage used for ROM images, user programs (RAW files) and LIF container for HP-IL drive simulation
- Internal OTP (One Time Programmable) memory, used for storing the device serial number

All storage is managed by the firmware under control of the User Interface (CLI). The SD card must be formatted by the user in a host system with the FAT or exFAT file system, and the subdirectories with ROM images and RAW files must be created in that system as well.

7.3. FLASH memory layout

The first 1 MByte (minus 4K) of FLASH memory is reserved for storing the firmware image. This is handled automatically by the RP2350 development tools and bootloader. This means that a software image (including any data it contains) cannot be larger that appr. 1 MByte. Current image size is about 258 KByte (August 2025). Software is executed directly from the FLASH memory, unless a part is marked as critical, in that case it will run from SRAM. The TULIP4041 core1 critical loop runs from SRAM for example. FLASH code is cached, and a cache miss may lead to unacceptable delays in software execution. All static arrays, such as ROM images embedded in the firmware (24Kbyte) and the lookup tables for the disassembler are in FLASH and are part of the firmware image.

The start address of the software in FLASH memory is 0x10000000 (XIP_BASE), the start address of the ROM images (offset in FLASH) is at 0x100000 (allowing 1 MByte for the firmware). ROM images may be stored in in FLASH in ROM, MOD1 or MOD2 format. The contents are accessed directly by the emulation layer for ROM access, no images need to be copied to SRAM when plugging into the HP41 ROM map. To keep track of the ROM images in FLASH and FRAM a rudimentary filesystem is maintained.

The 4K area just below the FLASH File System is used to program the owner string, this starts at (offset in FLASH) address 0x00FF000.

The RP2350 supports FLASH sizes from 2 to 16 MByte, and this limits the number of ROM images that can be in FLASH at any time. The standard Pico2 board has a total of 4 MByte FLASH, which applies to the DevBoard. The amount of FLASH on the final module version is 16 Mbyte.

It must be noted that FLASH memory requires a special sequence to write (and erase before writing), while reading can be at very high speed. Any operations to write to FLASH are time consuming and can only be done when the HP41 emulation core is idle (calculator OFF) and no USB communication is happening. Typically all interrupts are switched off during a FLASH erase cycle. FLASH memory can be erased only in blocks of 4 Kbyte and written to in blocks of 256 bytes.

7.4. FRAM memory layout

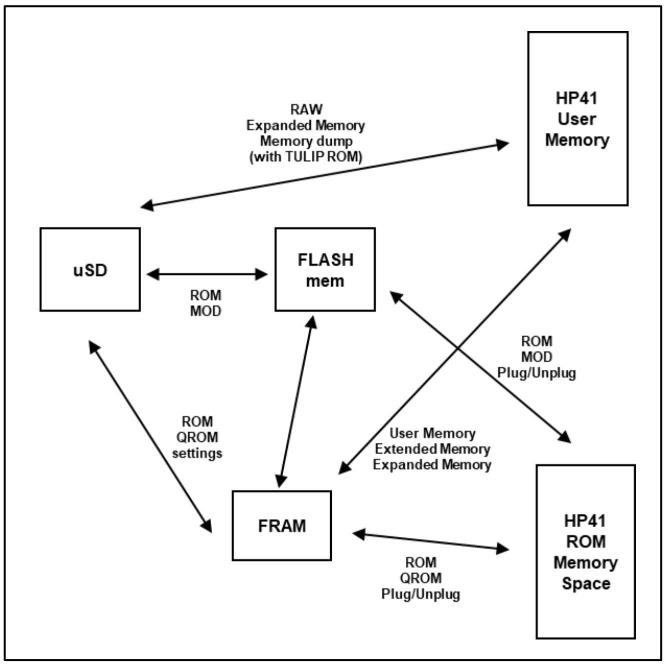
FRAM is used for persistent storage that requires byte based read and write access. This is the case for QROM images (MLDL pages or HEPRAM pages). FRAM also contains the TULIP4041 configuration settings and (emulated) User, Extended and Expanded memory. ROM images are stored in the same format as in FLASH using a simple file system. It is reasonably fast but in some cases data must be copied to SRAM for fast enough access. FRAM does not require a battery to keep its contents. A file system similar to that in FLASH is used to organize the data.

7.5. Micro SD card storage

The micro SD card is provided by the user and can have any reasonable size larger than 2 GByte (the used FAT library does not support SDSC cards, only SDHC and better). It should be formatted with the FAT or exFAT (recommended) file system and contains the following information:

- Root directory with the LIF container files (for use with a virtual HP-IL drive) and configuration backup files
- (optional) MOD directory containing your own repository of .MOD files (MOD1 and MOD2)
- (optional) ROM directory containing your own repository of .ROM files
- RAW directory containing your collection of .RAW files (HP41 user programs). These files can be accessed only by the special TULIP ROM (not available yet).
- In practice use your own directory with a repository of ROM and MOD files that you want to have available in FLASH for plugging. Firmware functions allow easy mass import for the uSD card into FLASH and upgrade in case of changes
- ROM images must be imported in FLASH (using the CLI) before these images can be virtually plugged

7.6. Storage of ROM images



ROM images can be stored in 3 formats: MOD1, MOD2 and ROM. Most common and easy to use is the MOD1 format. MOD2 is not very common, but can be used without any special action just like MOD1 files. Both have the .MOD file extension. The only difference between the two is that a MOD1 image is compressed and contains only the 10-bit words, while the MOD2 image contains the 16-bit words of the image, for example for use with the HP41CL. ROM files require a bit more care when using. The full 16-bit word is NOT used in the TULIP4041, only the 10-bit word is used.

The advantage of a MOD file is that it can contain multiple ROM images and has meta information such as the preferred page to be plugged in and many other attributes. The TULIP4041 uses this information, and especially the information about the hardware, to enable the hardware features for an HP-IL module for example. When plugging a .ROM file the user must manually choose the page and enable any hardware specific features.

The place for offline storage for ROM images is on the SD card. In order to be used by the TULIP4041 with your HP41 a ROM image must be in either FLASH memory or in FRAM. With the user interface you can copy your desired ROM images to FLASH or FRAM. Only images in FLASH or FRAM can be ‘plugged’ in a (virtual) slot of your HP41. ROM images in FLASH are organized in a basic file system.

7.7. TULIP4041 power consumption

The HP41 system is generally a very low power system. The RP2350 however is a very performant and somewhat power hungry processor. Care should be taken to properly manage power consumption in combination with the HP41.

When running the TULIP4041 consumes up to 20 mA, and the processor is running constantly when the HP41 is running. A microSD card will consume around 1-2 mA when idle, and up to 20-30 mA extra when in use. Some high-speed cards consume even more, and it is advised to check power consumption of the cards. Programming (writing to) FLASH memory increases power consumption with about 10mA. During power up of the TULIP the uSD card can sometimes use op to 20-30 mA while being mounted.

[NOT YET SUPPORTED]: When not powered by USB the TULIP4041 enters low power mode after about 10 seconds when the HP41 goes into STANDBY (also referred to as LIGHT SLEEP) or OFF (DEEP SLEEP) mode The power consumption is then still around 2.7 – 3 mA, and this will drain the HP41 battery quickly. The TULIP4041 is powered by the HP41 at all times when not powered by USB. Power is sourced by the HP41 battery directly and not by the HP41 regulator. The processor will NOT power down when one of the virtual serial ports is active or during file operations or FLASH/FRAM programming. When a USB power bank is connected, no serial ports are active and the power bank will power the TULIP4041 in low power mode. Please be aware that many USB power banks require a minimum current to prevent them from being shut down and that the low current of the TULIP may cause the power bank to shut off power.

When powered by USB the calculator is always powered by its own battery.

Advice 1: Connect the TULIP4041 to a USB power source whenever possible and practical

Advice 2: Remove the TULIP4041 from your calculator when not in use and when it is not connected to a USB power source.

When powering down the TULIP4041 will save all relevant settings and memory contents to non-volatile storage. These contents are then safe when the TULIP4041 is removed from the calculator.

7.8. PCF8523 RTC (TULIP module only)

The TULIP module has a PCF8523 Real Time Clock IC on board. This is intended to be used for emulation of the TIME module (not yet implemented). The RTC is connected to the processor using the I2C bus, and the I2C signals are also available on the I/O header of the module version to connect other devices.

For keeping the time while the system is not powered a backup battery can be connected. The Module version comes with a separate battery carrier suited for a 3.3V CR1620 battery and is connected with wires to the main board. Switchover to the battery backup is automatic and the RTC signals a low battery when the battery voltage is under 2.50V. The actual voltage may be as low as 1.8V to keep the time. While TIME module emulation is not implemented, or if you are simply not using the TIME module emulation the TULIP does not use the RTC timekeeping feature and connecting a battery is not necessary.

8. HP41 device emulation on the TULIP4041

The main function of the TULIP4041 is to emulate devices plugged on the HP41 bus. There may be a debate about the term emulation versus simulation. My take on the matter is that the HP41 sees a real device on its bus and has no idea if it is a genuine device or something that behaves like it. I think the term emulation is correct here.

The TULIP device may be plugged in any convenient port, the port address signals B3 and B4 are not connected.

TULIP4041 interacts with the HP41 system in various ways, and this chapter gives an overview of the ways of interaction. Very basically, the device captures all events on the HP41 system bus and responds to address and instructions. The main functions are:

1. Bus tracing: passive catching of all bus traffic and presenting in an understandable way
2. ROM emulation: watching the address on the bus, and responding with a data word whenever the address is in the range of the emulated ROM
3. QROM/MLDL emulation: catching the WROM (0x040) instruction and the information on the DATA line to write a word to QROM if the address matches the correct address range
4. Peripheral emulation: catching the SELP n instruction for the emulated device and executing the special device instructions (mainly for the HP82143 printer and HP-IL)
5. Peripheral emulation: catching the PRPHSLCT instruction for the emulated device and use the READ and WRIT instructions to transfer data
6. User memory emulation: catching the RAMSCLT and READ/WRIT instructions and take appropriate action to emulate memory modules and Extended or Expanded Memory
7. Special instruction emulation: monitor special instructions like bank switching, HEPAX special instructions, HP41CL instructions
8. HP41 carry control: drive the carry flag during T0 when requested
9. HP41 FI control: drive the FI line at the relevant bit time to indicate an emulated device requests servicing
10. HP41 power control: drive the ISA line to wake up the calculator
11. HP41 reset control: drive the PWO line to interrupt and reset the calculator

8.1. ROM / QROM emulation

The ROM emulation is relatively simple. The TULIP maintains a table with the mapping of virtual ROMs that are plugged and compares the received address on ISA with this table and presents the result (if any) on ISA during the instruction time. Banks switching is not supported in the first BETA version but will be in a future version. The ROM mapping is kept in FRAM and will remain valid after the next power cycle or reset.

Please be aware of possible port or XROM conflicts since the TULIP firmware does not know of any physical modules plugged in the calculator, and that also applies to the calculator type (HP41C, CV or CX). Also be careful when using an HP41CL with possible virtual ROMs plugged.

ROM images are placed in FLASH or FRAM and some are embedded in the firmware (HP-IL, IL Printer and the HP82143A printer). ROM images can be programmed (imported) in FLASH or FRAM from the micro SD card using the Command Line Interface. In the first versions only the ROM format is supported, support for MOD images will be added in a later version.

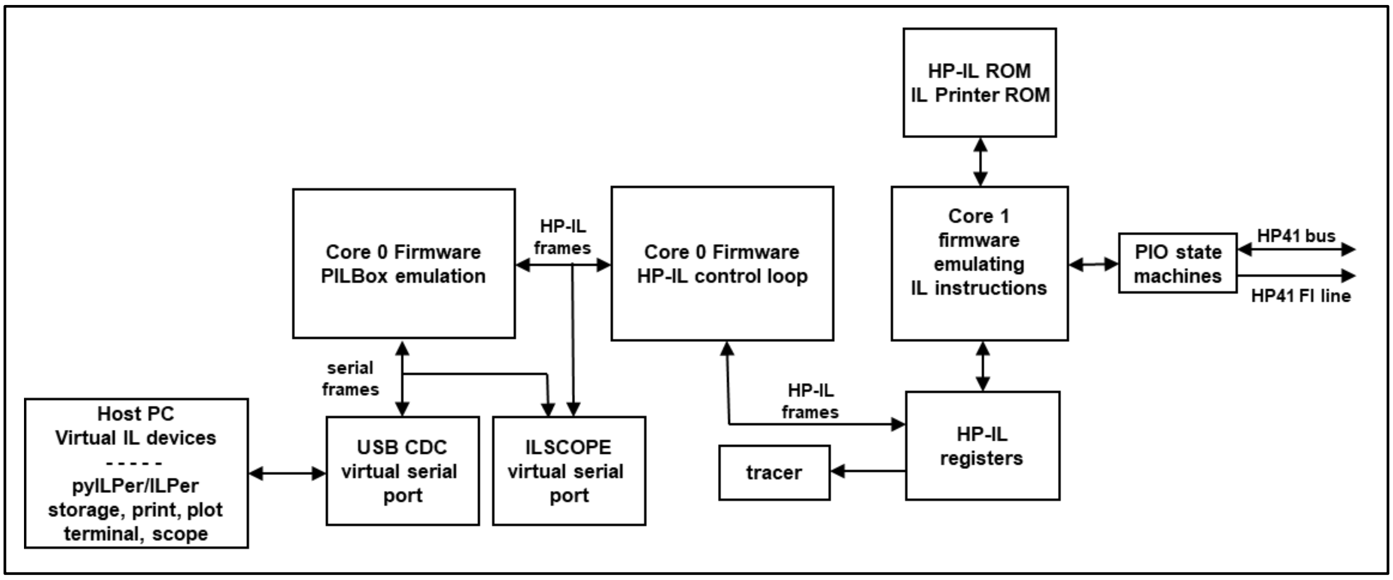
The TULIP can emulate QROM (or MLDL RAM) as part of the ROM mapping. QROM must always be mapped in FRAM, because only FRAM is writeable with the WROM instruction. FRAM storage is persistent and will withstand a power cycle or reset. QROM is currently not yet supported.

Any ROMs needing support of specific instructions (HEPAX, HP-IL, printer etc) must have the specific instruction emulation support enabled.

8.2. HP-IL emulation

The TULIP HP-IL emulates a virtual IL loop. This means that it connects with a host computer via a USB virtual serial port to virtual HP-IL devices running on the host computer. There are provisions to expand the possibilities in future firmware versions. TULIP emulates the registers of the 82160A HP-IL module and the instructions to communicate with these registers and in addition uses the FI signals towards the HP41 system bus. The emulation is based on the V41 sources by Christoph Giesselink and EMU41 by Jean-Francois Garnier.

A (virtual) serial link is used for the connection with the host computer, and the translation of HP-IL frames to serial is that of the PIL Box (by Jean Francois Garnier), which is emulated by the TULIP4041. The host software for virtual HP-IL will see a PILBox connected.



When no Host PC or virtual serial port is connected the HP-IL virtual loop is always closed internally. For more advanced functionality other ROMs can be plugged. The Plotter ROM is tested for example.

In the current firmware version there are a few limitations:

- It is not yet possible to put the TULIP system in device mode (with the IL Development ROM for example)
- RFC/CMD frame handling as in the original PILBox is not implemented. As a result multiple RFC frames may be sent to the Virtual IL devices.
- AUTOIDY mode is not implemented

For debug and study purposes the HP-IL registers are sent every cycle to the tracer queue and can be shown in the tracer (using the tracer virtual serial port and a terminal emulator). With the CLI this can be enabled or disabled. The tracer supports the HP-IL instructions in the disassembler.

HP-IL frames and PILBox serial traffic are sent to the ILSCOPE virtual serial port for analyzing or debugging your HP-IL applications. This can be controlled by the CLI.

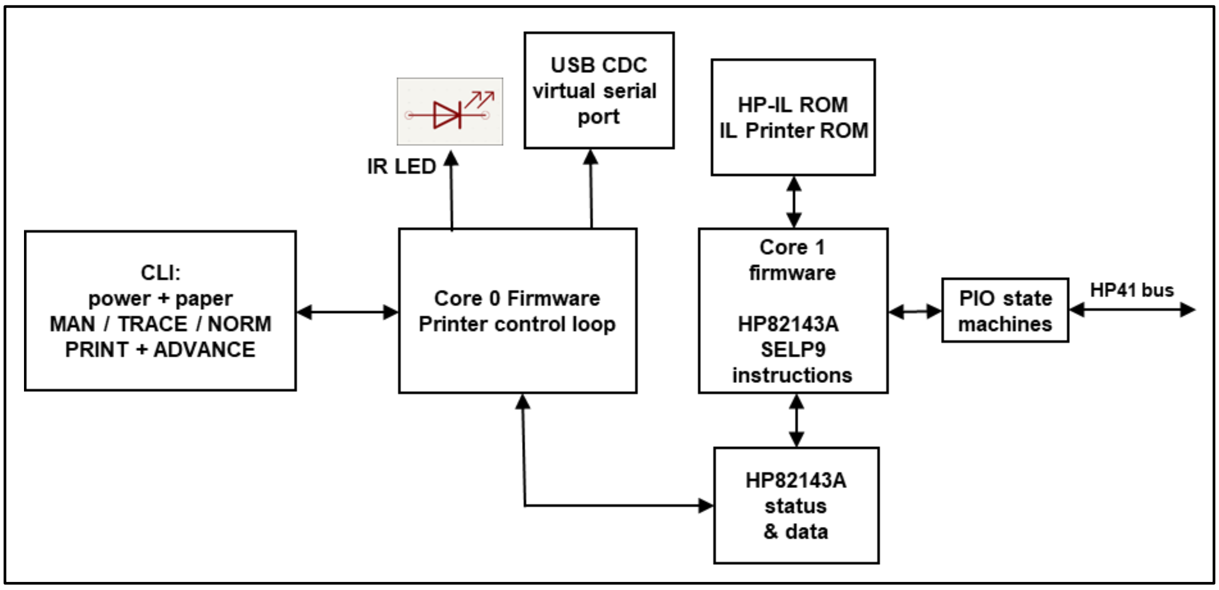
When there is no connection with a serial device the HP-IL loop is closed. When there is a serial connection and this is not a virtual HP-IL device (like ILPer or pyILPer) then the loop is open, and any HP-IL operations will result in

TRANSMIT ERR. Only when a virtual HP-IL device is connected the loop will be closed again. Changes in the status of the PILBox connection are shown in the CLI. In some cases a **TRANSMIT ERR** or a slow down of HP-IL traffic can occur when the TULIP firmware is heavily loaded with traffic on several virtual serial ports at the same time such as showing the tracer and the IL Scope at the same time.

The HP-IL printer can be disabled (like the little switch on the HP-IL module) by simply not plugging it or unplugging it. The IL Printer ROM will completely disappear and is not parked in Page 4, so Page 4 is available for other fun stuff. The HP82143 Printer ROM can be plugged instead and can co-exist with the HP-IL module.

8.3. HP82143A printer emulation

The HP82143A printer emulation is based on earlier experiences with the printer emulation for the HP41CL and the USB Printer module by Diego Diaz. Main source of knowledge is the document "HP82143A Printer Study" by Doug Wilder.


USB CDC virtual serial port
HP-IL ROM IL Printer ROM
IR LED
CLI: power + paper MAN / TRACE / NORM PRINT + ADVANCE
Core 0 Firmware Printer control loop
Core 1 firmware HP82143A SELP9 instructions
PIO state machines
HP41 bus
HP82143A status & data

The printer emulation is done by implementing the printer status register and decoding the SELP 9 instructions. The HP82143A printer is (as far as I know) the only device that uses the HP41 capability to transfer the carry status by driving ISA at T0 when requested (with a SELP 9 instruction), and that feature is implemented in the ISA output state machine.

The CLI is used to control the keys and switches that are normally on the printer itself. Be aware that (like in real life) the printer is default without paper, so before use paper must be loaded using the CLI.

Printer output is sent to one of the virtual serial ports for use by the HP82240 simulator (from Christoph Giesselink). This simulator must be put in the proper mode for the HP82143A printer. Graphics printing is supported. The printer output is also sent (if enabled with a jumper on the DevBoard) to the infrared LED. The bytes printed to the IR port are *not* compatible with the real HP82240 IR printer and are intended to be used with an IR receiver (serial or USB) connected to a host computer and the HP82240 simulator in HP82143A mode.

There a few implementation limitations:

- The internal printbuffer (the queue between core1 and core0 firmware) is 100 bytes. In addition there is a large buffer (about 1 KByte) for the USB virtual serial port. Normally the printing to the USB serial port is much faster than the HP41 can keep up with, but in theory this could lead to a **PRINTER BUSY** message if the core0 software cannot empty the queue fast enough. This has been tested by throttling the output bytes, currently there is no throttling and bytes are sent as fast as they come in

- When no virtual serial port is connected the printer emulation will work, but the printed data is simply discarded internally and sent only to the IR LED
- When printing to the infrared LED there is no throttling implemented
- When printing the IR LED is always used, it cannot be disabled in the current firmware. **On the DevBoard when using the printer emulator jumper 3 must be open!** JP3 enables PWO output and will reset the calculator!
- When using IR printing jumper 1 must be closed to enable the IR led (DevBoard only)
- There may be a few minor differences in output to the simulator compared to a real HP82143 printer. This is under investigation.

8.4. HP82153A emulation: the Wand Barcode Reader

The HP82153A Wand is an interesting device. It allows entry of programs and even keys by scanning a barcode. The TULIP emulates the Wand by implementing a single byte register that is accessed when peripheral 0xFE is selected. The emulation itself is relatively simple, when the FI flags 0 and 2 are set something from the Wand is available for reading from that register. When the register is empty the flags are reset. Behind that register is a queue structure that is filled by the Core0 code for simulating the paper keyboard or simulated scanning of a program. In addition the Wand hardware has the ability to switch the HP41 on by briefly driving the ISA line, this behavior is implemented as well. The CLI allows sending many functions, keyboard codes and programs to the Wand emulation.

The simulated Wand scanning in Core0 occasionally does a busy wait, and this may interfere with virtual HP-IL traffic.

8.5. User and Extended Memory emulation

The TULIP4041 can emulate User Memory. This is the memory used for the HP41 status registers, program memory and Extended Memory. In the BETA version only emulation of Extended Memory is possible. This is valid only for the Extended Memory modules that can be plugged. The TULIP currently does not support the Extended Functions module or its built-in memory. Using the CLI the user may plug and unplug 0, 1 or 2 Extended Memory modules. When 0 is used all plugged modules are unplugged, with a value 1 only the first Extended Memory module is plugged (and module 2 unplugged if it was plugged).

• 0 modules:	no Extended Memory	
• 1 module:	0x201..0x2EF	Extended Memory Module 1
• 2 modules:	0x201..0x2EF and 0x301..0x3EF	Extended Memory Module 1 + 2

The contents of the Extended Memory modules are saved in FRAM and will survive a power cycle. The memory contents are not erased when a module is unplugged.

Reading a full HP41 register (8 bytes) from FRAM is relatively slow and not fast enough to support a READ instruction. Therefore a single HP41 register (if it exists) is cached from FRAM in the RP2350 RAM when a RAMSLCT (DADD=C) occurs. This register is then ready for actual reading when a READDATA (C=DATA) instruction happens. This works fine for the existing Extended Functions ROM, but none of the READ n (C=REGN) are supported. For practical purposes also the WRIT n (REGN=C) instructions are currently not supported, only

WRITDATA (DATA=C, 0x2F0) is implemented. This will most likely be resolved in a future firmware version by caching the entire User Memory contents in RAM.

NOET YET IMPLEMENTED: When using an HP41C it is possible to emulate User Memory modules, both Single and Quad. It is not possible to combine physical modules with User Memory modules plugged with the TULIP. In a HP41CV, -CX or -CL this option does not make sense.

8.6. TULIP emulation of HP41 instructions

The TULIP4041 emulates HP41 instructions by capturing these from the bus and comparing it with one of the following supported instructions.

READDATA [0x038, C=DATA]

Copy the active selected register (selected with RAMSLCT) to C. The individual Class 0 READ instructions are not (yet) supported

WRITDATA [0x2F0, DATA=C]

Copy C to the active selected register. This individual Class 0 WRIT instructions are not (yet) supported

SELP 9 [0x264, PERTCT]

Select peripheral to take control, peripheral 9 is the HP82143 printer. After this instruction the following instructions are ignored by the CPU and control the actions of the printer. If the msb of that instruction is 1 control will be given back to the CPU. During the instructions SYNC is low.

BUSY?	[0x003]	Set carry of the printer is busy
POWON?	[0x043]	Set carry if the printer is on
ERROR?	[0x083]	Set carry if printer errors
RDPTRN	[0x03A]	or C=STATUS, return printer status word to C[10..13]
PRINTC	[0x007]	send byte C[0..1] to the printer buffer
RTNCPU	[0x005]	return control to the CPU

SELP 0 .. SELP 7

Select the HP-IL peripheral and immediately selects a register 0..7. Refer to the HP41 Programming Handbook for details

RAMSLCT

Select a block of 16 registers in User Memory indicated by C[0..2]. The TULIP firmware caches this register

PRPHSLCT

Selects the peripheral indicated by C[0..2], only used for Wand emulation (peripheral = 0x0FE)

ENBANKx

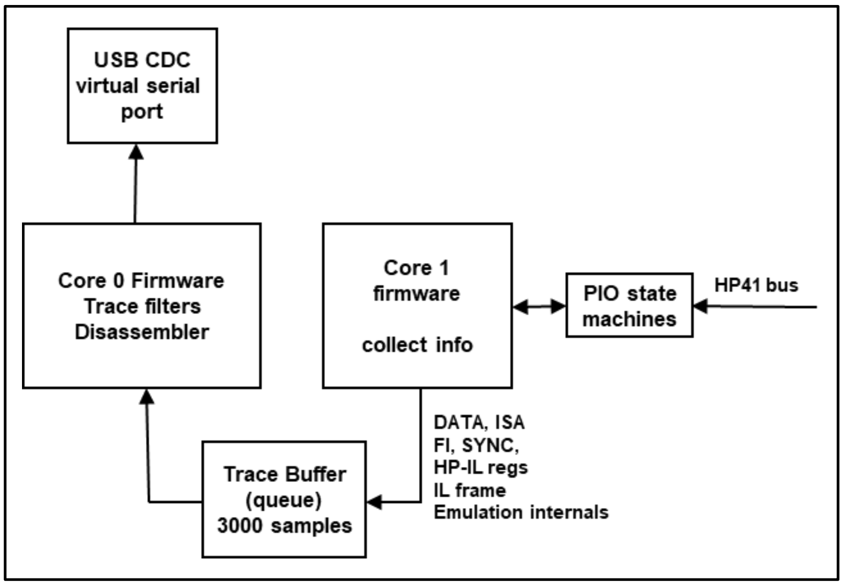
Enables the indicated Bank. See the paragraph on Bank Switching

WROM

Writes the word (actually only the 10 lsb's) in C[0..2] to the QROM address in given in C[3..6]

9. HP41 Bus tracing

HP41 bus tracing is extremely useful when testing and debugging mcode programs, and to discover how the HP41 software or existing peripherals work. It can also be used to mimic some HP41 peripherals, such as using the data to drive an additional display to mirror the existing HP41 display (not yet supported in the TULIP).


USB CDC virtual serial port
Core 0 Firmware Trace filters Disassembler
Core 1 firmware collect info
PIO state machines
HP41 bus
Trace Buffer (queue) 3000 samples
DATA, ISA
FI, SYNC,
HP-IL regs
IL frame
Emulation internals

The Bus Tracing unit in the TULIP catches the following information during each HP41 cycle in the critical core1 loop:

- Instruction counter (reset after each PWO event)
- ISA address (16 bits)
- ISA instruction (10 bits)
- SYNC state during the ISA instruction (1 bit)
- DATA (56 bits)
- FI line (56 bits, at each bit time) (only on the TULIP DevBoard). On the TULIP Module FI tracing is limited to the flags that are driven by the TULIP. Flags driven by physical devices on the HP41 bus (like the TIME module in an HP41CX) are not visible to the TULIP module (there may be a possibility to implement this)
- Carry status (ISA state at T0) when output by the TULIP
- RAMSLCT selected register (only captured when a RAMSCLT instruction was executed)
- (optional) HP-IL registers (9* 8 bit register) and HP-IL frame in and frame out
- Active bank (derived from earlier ENBANKx instructions)

Briefly after T0 the information above is sent to the TraceBuffer, but only under the following conditions:

- Bus tracing is enabled
- The TraceBuffer is not full. If the buffer was full, the sample is discarded. This is an overflow condition which will be recognized by the core0 firmware because the sample counters are not consecutive

The TraceBuffer is a queue structure, the core1 part will never do a blocking wait and checks if the queue is not full before writing a sample, otherwise the sample will be discarded. The default size of the TraceBuffer is 6000 samples and can be changed in the CLI (reboot required).

The TraceBuffer is read by the core0 non-critical loop and before reading will check if there is any data in the buffer to prevent that it blocks the application.

A single trace line consists of the data captured during a single T0-T0 cycle. Remember that this cycle presents the DATA (usually the C-register), the address (in ISA) and the instruction (also on ISA) fetched at this address. The instruction is then executed in the next cycle, but the C-register is not updated yet. The effect of the instruction on the C-register is shown in the cycle after that on the DATA line.

```
20  0208-1  1  046  0.0000000000.0.00  ...  R2FD  C0  FI--------------  C=0 S&X
21  0209-1  1  3F0  0.0000000000.0.00  ...  R2FD  C0  FI--------------  PRPHSLCT
22  020A-1  1  270  0.0000000000.0.00  3F0  R2FD  C0  FI--------------  RAMSLCT
23  020B-1  1  130  0.0000000000.0.00  270  R000  C0  FI--------------  LDI
24  020C-1  0  169  0.0000000000.0.00  ...  R000  C0  FI--------------  169
25  020D-1  1  106  0.0000000000.0.00  ...  R000  C0  FI--------------  A=C S&X
26  020E-1  1  378  0.0000000000.1.69  ...  R000  C0  FI--------------  READ 13(c)
27  020F-1  1  17C  1.A70016919C.1.9A  ...  R000  C0  FI--------------  RCR 6
```

In the example above the first LDI 169 instruction is fetched in sample 23 and 24 (note that SYNC is low in cycle 24, 0 in the 3rd column, indicating the fetch of a literal). While the CPU executes this during sample 25, this is not shown on DATA yet. But the next instruction is already fetched. Sample 26 shows the result on DATA from the LDI 169. This can be confusing, but please understand that the cycles are shown as they appear on the bus for T0 to the next T0.

When showing all sampled data by the tracer the stream to the (virtual) serial port will generally limit the performance, and after 8000 to 10.000 samples (depending on the host showing the trace) the TraceBuffer will overflow. This will never block the operation of the TULIP4041, you will simply miss samples (indicated by an O for overflow in the trace display). With clever filtering (for example to filter out some of the standard loops in the system ROM for keyboard checking and debounce) the performance will be much better, or even when you filter out the complete mainframe ROM (0x0000-0x5FFF) for example, depending on your ROM configuration and what information you are looking for. This allows you to do near real-time tracing of your HP41 system on mcode level.

Using the Tracer is a significant load on the TULIP4041 firmware in core0, due to the high data density especially to the USB serial port. In some cases characters or part of a traceline can be skipped. This can best be resolved by being smart about the filtering, and this also prevents overflows. This has been seen specifically on Windows systems, Linux systems seem to behave better. Using the Tracer may impact performance of the emulated HP-IL loop and in rare cases lead to a transmit error.

Due to the behavior of the Tracer during PWO events the first and last traces may show incorrect information.

To save a trace log use the features of your terminal emulator. First thing to do is set the terminal buffer depth. In Teraterm this is set with Setup->Window->Scroll Buffer. When a trace is finished it is then very easy to select the samples of interest and copy/paste into a text editor. Alternatively you can activate a logging function if available.

Using the CLI the Tracer functionality can be managed and trace filters and triggers can be set.

The output of the Tracer is streamed to one of the USB Serial Ports, connect to it with your favorite terminal emulator (for example TeraTerm on Windows or minicom on Linux). When the virtual serial port is not connected all trace samples will be discarded. The output has the following columns (subject to change):

```
[1] [2] [3]    [4] [5]  [       6        ]  [7]  [ 8] [9]  [       10      ]  [  11    ]
 O   0  0193-1  1  201  0.000002C048.0.FD  ...  R010  C0  FI01234---------  ...
     1  0001-1  0  006  0.000052C048.0.FD  ...  R010  C0  FI--------------  ?NC GO 0180
     2  0002-1  1  2B5  0.000052C048.0.FD  ...  R010  C0  FI--------------  ...
     3  0003-1  0  006  0.000052C048.0.FD  ...  R010  C0  FI--------------  ?NC GO 01AD
     4  01AD-1  1  001  0.000052C048.0.FD  ...  R010  C0  FI--------------  ...
     5  01AE-1  0  100  0.000052C048.0.FD  ...  R010  C0  FI--------------  ?NC XQ 4000
     6  4000-1  1  000  0.000052C048.0.FD  ...  R010  C0  FI--------------  NOP
     7  01AF-1  1  2E0  0.000052C048.0.FD  ...  R010  C0  FI--------------  DSPOFF

                 [  11    ]          [    12   ]  [                13                   ]
                 ?FI 8 ?FRAV         IL> 023 DAB  Reg E0  A0* 40  40  01  03  01  00  00
```

[1] Tracer status, one character with the following meaning

[space]	normal traceline
=	tracelines before this line are skipped due to a filter
O	an overflow occurred in the TraceBuffer
T	Trigger occurred on this address (not in current firmware)

[2] Traceline sample counter, counts all traces to identify how many lines are skipped. Counter is reset upon a PWO event. The counter is generated by the core1 emulation layer and will be non-consecutive in case of a filter or overflow, and can be used as an indication of the number of samples lost or skipped.

[3] Address + active bank. Active Bank can be 1-4, and uses the 'standard' bank switching scheme as follows:

ENBANKx instruction in Page 3 switches banks in Page 5 (HP41CX behaviour)
ENBANKx instruction in the Port Pages switches banks in all Pages in that Port
ENBANKx instruction in any other Page switches banks in that Page only

This may conflict with some specific hardware modules. The active Bank for all Pages is reset to Bank 1 when the calculator goes in STANDBY mode (PWO event), except when the ZEPROM Sticky Bankswitching mode

[4] SYNC status during ISA Instruction, 1 when this is an instruction fetch, 0 for data fetch or under peripheral control

[5] Data or instruction (instruction when SYNC is 1, otherwise it is a data fetch, peripheral instruction or second word of an instruction)

[6] Contents of DATA line, formatted like a register with sign, mantissa, exponent sign and exponent. Most of the time the C register is output to DATA, except during a peripheral or memory read. In that case it contains the read value

[7] 16 bit indicator of the instruction (including the SYNC status as read from the state machine), appears only when the firmware has a potential instruction match that is handled by the TULIP firmware, is empty otherwise. Used for firmware verification

[8] Current valid RAMSLCT memory address

[9] Carry output during T0 (C0 when not set, C1 when set), only used by the HP82143 printer emulation

[10] FI line status, 14 flag positions, will show hex flag number in the correct position. The TULIP module version will show only the flags that are driven by the TULIP, it has no access to flags from other devices. The Development Board actually traces the FI line and shows all flags from other peripherals in the calculator (TIME for example in an HP41CX). The very first trace line may show incorrect FI information

[11] Disassembled instruction. Currently only JDA type mnemonics supported, peripheral instructions are not decoded except HP-IL instructions

[12] HP-IL frame, > indicates output, < is input, only shown when enabled and when the HP-IL module is (virtually) plugged. The frame is decoded

[13] HP-IL registers (when enabled) show all registers of the HP-IL module, only when there is a change. A changed register is indicated with *

Tracing of HP-IL frames and registers can be enabled or disabled in the CLI. The tracer shows 9 registers, R1R is only the read part of Register 1, R1W is the Write part of R1 since HP-IL register 1 has different functions for read and write (as implemented in EMU41 and V41).

```
[  11    ]          [    12   ] [13  R1R R2  R3  R4  R5  R6  R7  R8  R1W]
?FI 8 ?FRAV         IL> 023 DAB  Reg E0  A0* 40  40  01  03  01  00  00
```

The disassembler is a very simple lookup table and therefore very fast, but with limitations. Currently only JDA (Jacobs-DeArras) mnemonics are supported. A data fetch (SYNC low during ISA instruction time) is shown as hexadecimal literals. Only 2-word XG/GO's are not handled by the lookup table but constructed in core0, and the first word of such an instruction is shown with 3 dots. 3-word relative GOTO/GOSUB are not decoded. Mainframe labels are not decoded (this may be implemented in a next version). Peripheral instructions for HP-IL are disassembled.

Pressing a key in the tracer window will enable or disable the tracer and pause the listing. Since the tracer will be disabled it will not continue at the halted address!

Possible future features of the HP41 bus tracer are:

- Advanced pass or block filter by user provided address range
- Trigger on the Nth occurrence of a trigger
- Trigger on a data or instruction fetch (instead of an address)
- Trigger inside a specified Bank
- Add labels to the disassembler
- Disassembly of peripheral instructions
- Support for other mnemonic types (HP and ZENROM)
- Dynamic sizing of the TraceBuffer (this is limited by available memory)
- Set a trigger to enable a trigger pulse to an external output to allow tracing with an external logic analyzer or oscilloscope

[NOT YET IMPLEMENTED] The Tracer has the feature to pass or block samples, and to set or clear a trigger condition. It is not possible yet to apply the filters or triggers to a specific bank.

- Pass: simply pass all samples. Default is to pass all samples.
- Block: samples that match a specified address or address range will be blocked, and not shown in the trace listing
- Trigger: samples that match a trigger address are marked in the listing, and traces are listed starting at this address. An option can filter out specific ranges (apply blocked addresses), and an option can block all samples until the Trigger condition is met. There is currently no pre-trigger buffer (yet)
- Trigger End: after a Trigger, the listing of samples stops when there is a Trigger Address match. As an option, sampling can stop after a given number of samples

The current software only supports a limited number of pre-defined filters and Page ranges.

A number of system loops to block is pre-programmed and can be enabled using the CLI:

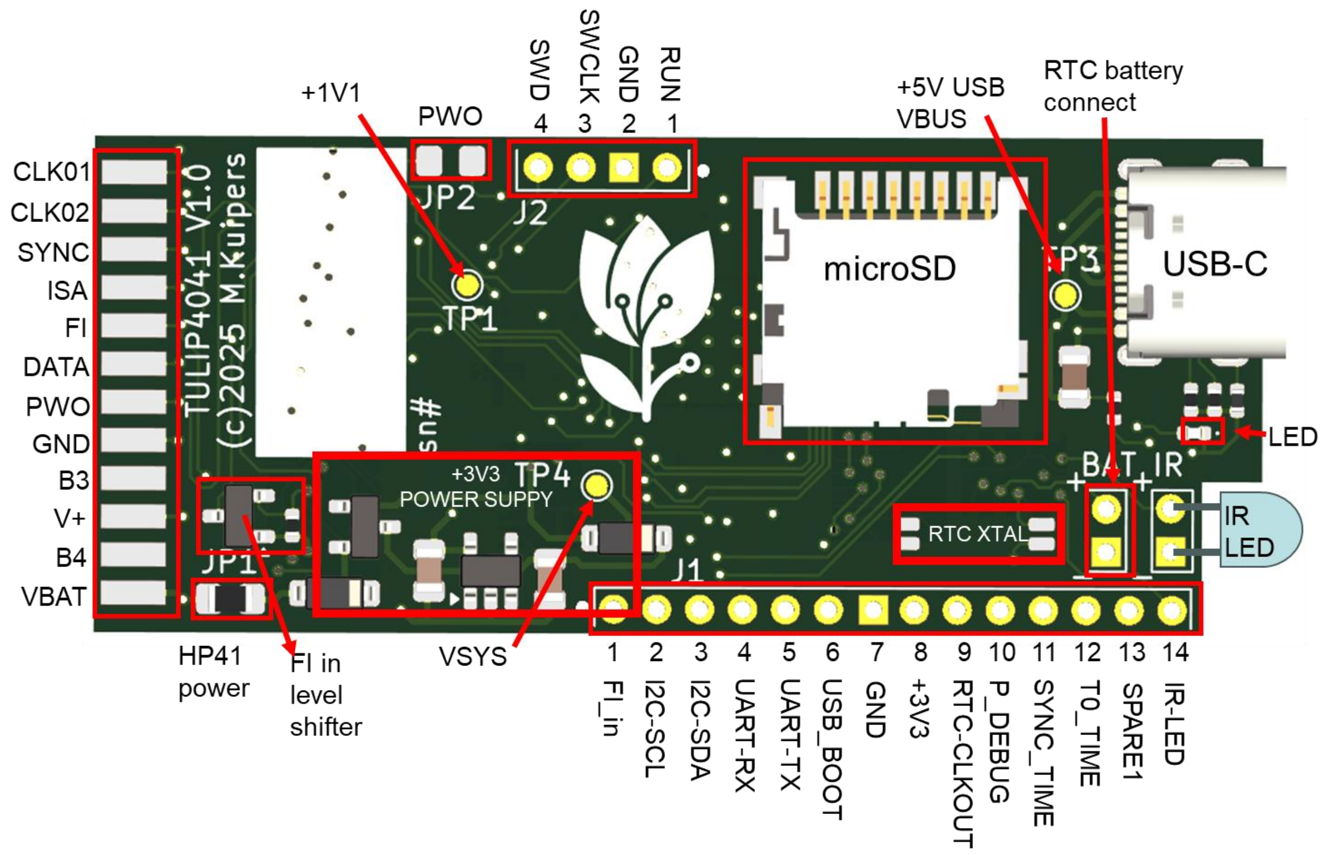
```
0x0098 - 0x00A1       RSTKB and RST05
0x0177 - 0x0178       delay for debounce
0x089C - 0x089D       BLINK01
0x0E9A - 0x0E9E       NLT10 wait for key to NULL
0x0EC9 - 0x0ECE       NULTST NULL timer
```

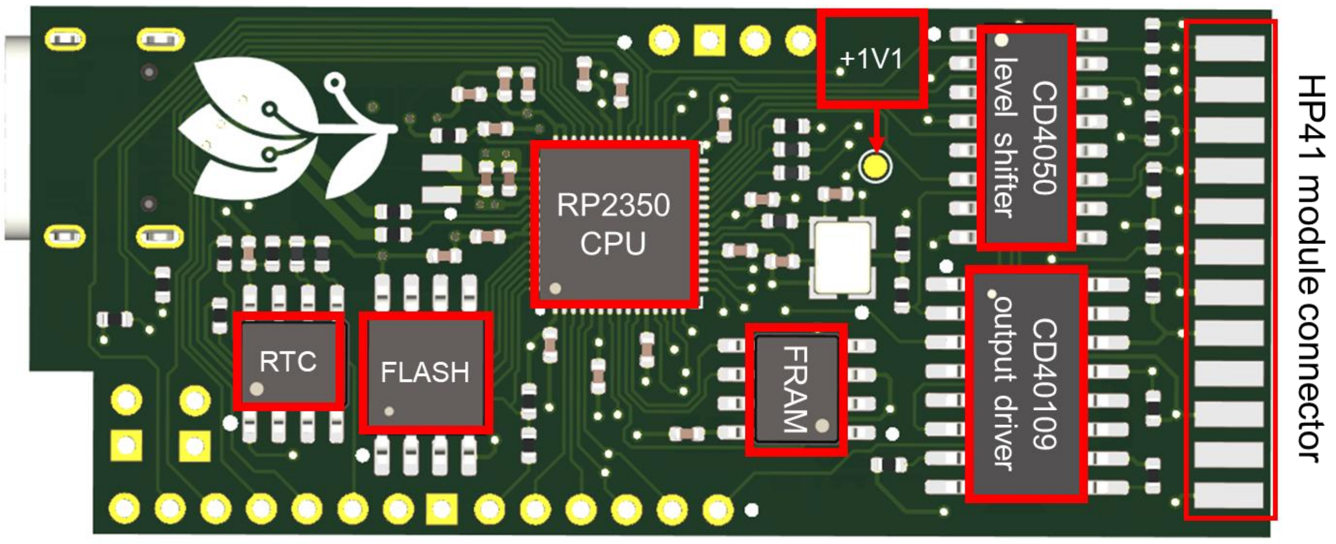
Finally a special note for HP41CL owners. When your CL runs in any of the Turbo modes, you will not see every single operation of processor on the bus. Instead the HP41CL appears to be fetching NOPs all the time while internally it is running at a higher bus speed. The HP41CL switches back to normal speed when its needs to access a peripheral on the outside (which includes the display and external ROMs), and only those cycles will be visible in the tracer. In case you want to see all cycles on the HP41CL the speed must be set to the normal HP41 speed.

15.1. TULIP Module IO connections

The TULIP module has the following connections:

- HP41 Module connector
    - The signals of the HP41 bus. The B3 and B4 signals for port identification are not connected
    - The TULIP uses power straight from the HP41 battery when it is not powered by USB. The power consumption is about 20mA (when not using the uSD card) which is a significant load and the batteries will drain more quickly
- USB-C connector
    - For the mass storage (uSD card) and s virtual serial ports
    - TULIP uses power from USB even when the HP41 is connected
- SWD Debug interface
    - This is a 4-pin pad intended for a 2mm header or direct soldering
    - Offers the SWDIO and SWCLK signals plus GND for connection a host computer with one of the available debug options for the CPU to allow low level debugging of firmware
    - The RUN signal is available. Connecting this to GND will rest the CPU
    - Pin 1 is indicated by a white dot on the PCB silkscreen
- IO Breakout
    - This is a 14-pin pad intended for a 2mm header or direct soldering
    - A collection of IO signals for possible extension and/or debugging
    - A pin next to the GND pin is the USB_BOOT and has the same function as the pushbutton on the PICO board to force the processor into BOOTSEL mode
    - Pin 1 is indicated by a white dot on the PCB silkscreen
- There is a blue signaling LED on the PCB that can be controlled by firmware, and it is connected to GPIO25, the same pin as on the PICO development board
- Two holes are used to connect the infrared LED
- Two holes on the PCB are used for the RTC backup battery


CLK01
CLK02
SYNC
ISA
FI
DATA
PWO
GND
B3
V+
B4
VBAT
HP41 power
FI in level shifter
VSYS
+1V1
PWO
SWD 4
SWCLK 3
GND 2
RUN 1
J2
JP2
TP1
TULIP4041 V1.0
(c)2025 M.Kuipers
sn#
+3V3 TP4 POWER SUPPY
JP1
+5V USB VBUS
RTC battery connect
microSD
TP3
USB-C
LED
BAT
IR
IR LED
RTC XTAL
J1
1 FI_in
2 I2C-SCL
3 I2C-SDA
4 UART-RX
5 UART-TX
6 USB_BOOT
7 GND
8 +3V3
9 RTC-CLKOUT
10 P_DEBUG
11 SYNC_TIME
12 T0_TIME
13 SPARE1
14 IR-LED


+1V1
CD4050 level shifter
CD40109 output driver
HP41 module connector
RP2350 CPU
RTC
FLASH
FRAM

HP41 Module Connector		**Module version (RP2350 pinout)**
CLK01	input	
CLK02	input	
SYNC	input	
ISA	input/output	
FI	input/output	
DATA	input/output	
PWO	input/output	Optionally driven by TULIP, JP2 must be closed
GND	HP41 GND	
B3	not connected	
V+	HP41 regulated power ~6V	
B4	not connected	
VBAT	HP41 battery	Power for TULIP when no USB connected
Testpoints		
TP1 (top), TP2 (bottom)	+1.1V	From internal CPU regulator
TP3	VBUS, USB +5V	
TP4	VSYS	USB or VBAT power after FET
J2 DEBUG		
1 - RUN	CPU RUN signal	Short to GND to reset CPU
2 - GND	System GND	
3 - SWCLK	Debugger CLK	Use with PicoProbe
4 - SWD	Debugger Data	Use with PicoProbe
J1 GPIO		
1 - FI in	FI input from FI level shifter	Not connected to CPU GPIO, possibly connect to pin13 for future implementation
2 - I2C SCL	GPIO3	I2C connected to RTC, available for other peripherals
3 - I2C SDA	GPIO2	I2C connected to RTC, available for other peripherals
4 - UART RX	GPIO1	UART used by firmware, future other use
5 - UART TX	GPIO0	UART used by firmware, future other use
6 - USB_BOOT	USB-BOOT	Short to GND for BOOTSEL mode while connecting USB
7 - GND	System GND	
8 - +3V3	System +3V3	
9 - RTC-CLKOUT	CLK/INT	Output from RTC
10 - P_DEBUG	GPIO28	Used by firmware, future other use
11 - SYNC_TIME	GPIO27	Used by firmware
12 - T0_TIME	GPIO26	Used by firmware
13 - SPARE1	GPIO23	Connected to PWO via J2 or future other use
14 - IR LED	GPIO29	IR_LED output before R11
Battery Backup		
BAT+	RTC battery backup	
BAT-	RTC battery backup GND	

16. References

All firmware files, documentation and schematics are available at my GitHub pages: https://github.com/mjakuipers/TULIP-DevBoard.

Several videos about my calculator activities, including the TULIP, are available on my YouTube channel: https://www.youtube.com/@TheMeinable

Reference	URL	Description
HP Museum Forum	https://www.hpmuseum.org/forum/index.php	The best place to get help and general information
V41 emulator	HP41.org and https://hp.giesselink.com/v41.htm	HP41 emulator for Windows
HP-IL emulation	https://hp.giesselink.com/hpil.htm	Emulated hardware on your PC, to use with the PIL Box or the V41 emulator
PIL Box	http://www.jeffcalc.hp41.eu/hpil/index.html	HP-IL to PC USB based link
EMU41	http://www.jeffcalc.hp41.eu/hpil/index.html	HP41 emulator, DOS based
HP41.org	www.hp41.org	The best source for manuals, books, ROM images
Clonix, NoVRAM and HP82143 emulator	https://www.clonix41.org/	Hardware plug-in configurable modules, and a printer emulator
41CL Homepage	http://systemyde.com/hp41/index.html	Home page for the 41CL Calculator and MAXX module
SDK41	www.hp41.org	Old-school mcode toolchain for DOS
41CL Other Docs	http://systemyde.com/hp41/documents.html	Includes the OSX3 manual
41CL File area	http://systemyde.com/hp41/files.html	Download links of the current ROM repository and various tools, which includes the CL Updater and clreader/clwriter plus some other goodies
HP-IL virtual devices	https://github.com/bug400/pyilper	pyILPer, a python based HP-IL emulator with printer and mass storage emulation
Calypsi	https://www.calypsi.cc/	Modern mcode and RPN toolchain with debugger for Linux, MacOS and Windows
Raspberry Pi RP2350 SDK	https://www.raspberrypi.com/documentation/pico-sdk/index_doxygen.html	All info on the SDK

Raspberry Pi RP2350 chip	https://www.raspberrypi.com/products/rp2350/	All info on the RP2350 chip
Raspberry Pi Pico2 module	https://www.raspberrypi.com/products/raspberry-pi-pico-2/	All info on the Pico2 module
PCF8523 RTC	https://www.nxp.com/part/PCF8523T#/	Datasheet of the RTC chip
KiCad	https://www.kicad.org/	Website of KiCad, used for the design of the TULIP PCB's
Book	HP41CX Programmers Handbook by Poul Kaarup	Available from hp41.org. A must have for mcode programmers
HP41UC by Leo Duran	https://sourceforge.net/projects/hp41uc/ File Converter (LIF. P41, BIN, RAW. etc), Compiler (text to binary), De-compiler (binary to text), Barcode Generator (PCL3, Postscript images)	Program to convert HP41 user code between various formats including generating barcode

17. Change log

Version	date	description
00.01.01	June 2024	Initial version for beta release
00.01.02	September 2024	Edited and expanded for first public release
00.01.03	September 2024	Preparing for change to RP2350/Pico2 and first public PCB’s
00.01.04	October 2024	Migrated to RP2350/Pico2, firmware updates documented
00.02.01	March 2025	Update to firmware specifications
00.03.01	June 2025	Update to firmware 0.9 BETA 1 and preliminary Module version
00.03.02	August 2025	Update to firmware 0.92 BETA 2
		Added references, general edits
00.04.02	December 2025	Update to firmware 0.96, split user manual from background
		Added references, general edits
		wand emulation added