

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.

Meindert Kuipers

Email: meindert@kuiprs.nl

June 2025

1. Credits

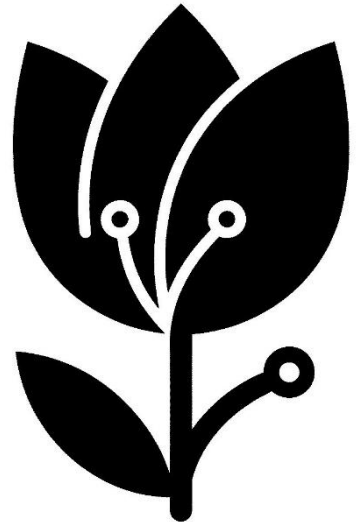
Many thanks to Andrew Menahue for inspiring this project, his cooperation and the design of the TULIP housing, Thomas Fänge for his contributions, cooperation and testing. HP-IL emulation is based on V41 and EMU41 by Christoph Giesselink and Jean-Francois Garnier, PILBox emulation is based on the PILBox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

Copyright © 2025 Meindert Kuipers. This document and all sources and schematics are subject to the MIT license conditions.

[Table of Contents](#)

Introduction.....	1
1. Credits	1
2. Version info	1
3. Conventions.....	2
4. Copyrights and Disclaimer	2
Table of Contents.....	3
5. TULIP4041 overview	6
6. TULIP4041 Firmware Architecture	9
6.1. PIO State Machines	9
6.2. SYNC State Machine	9
6.2.1. DATAIN State Machine	10
6.2.2. DEBUGOUT State Machine	10
6.2.3. ISAOOUT State Machine	10
6.2.4. FIIN State Machine	11
6.2.5. DATAOUT State Machine	11
6.2.6. FIOUT State Machine	11
6.2.7. IROUT State Machine	12
6.3. Principle of Operation, core1	12
6.3.1. Initialization	13
6.3.2. T54, Start of core1 loop.....	13
6.3.3. T0, DATA complete	13
6.3.4. T30, ADDRESS complete.....	14
6.3.5. T32, DATA first 32 bits complete	14
6.3.6. PWO interrupt handler	15
6.4. Core0 firmware	15
7. TULIP4041 interfaces	16
7.1. TULIP4041 PICO/ RP2350 pinout	17
7.2. TULIP4041 memory layout	18
7.3. FLASH memory layout	18
7.4. FRAM memory layout	19
7.5. Micro SD card storage	19

7.6.	Storage of ROM images.....	19
7.7.	TULIP4041 power consumption	20
7.8.	PCF8523 RTC (TULIP module only).....	20
8.	HP41 device emulation on the TULIP4041	21
8.1.	ROM / QROM emulation.....	21
8.2.	HP-IL emulation	22
8.3.	HP82143A printer emulation.....	23
8.4.	User and Extended Memory emulation.....	24
9.	HP41 Bus tracing.....	25
10.	Using the TULIP4041	30
10.1.	Virtual serial ports.....	30
10.2.	BOOTSEL mode and firmware upgrade	31
10.3.	Reset of the TULIP4041	32
10.4.	TULIP4041 power consumption	32
10.5.	Micro SD card.....	32
11.	ROM images, User memory and the file system	33
11.1.	Importing files and the Flash File Manager.....	34
11.2.	Files containing QROM.....	36
11.3.	Plugging ROM images	37
11.4.	Bank switching in the TULIP	39
11.5.	Embedded ROM images	41
11.6.	Hardware emulation	41
11.7.	Unplugging ROM images	41
11.8.	HP41 User Memory	42
11.9.	Using the mcode tracer	45
12.	TULIP4041 Firmware programming or update.....	47
13.	TULIP4041 Quick Start.....	48
14.	TULIP4041 Command Line Interface reference [TO BE UPDATED]	49
15.	Limitations	59
16.	TULIP4041 Hardware: Module version	60
16.1.	The TULIP housing.....	60
16.2.	Soldering the connector	60
16.3.	Preparing for first use	61

16.4.	The micro SD card	62
16.5.	Other tests	62
16.6.	TULIP Module IO connections	63
17.	TULIP4041 Hardware: Development Board	66
17.1.	Assembling the DevBoard	66
17.2.	Prepare the micro SD card.....	68
17.3.	Getting the DevBoard up and running.....	68
18.	References	73
19.	Change log	75

5. [TULIP4041 overview](#)

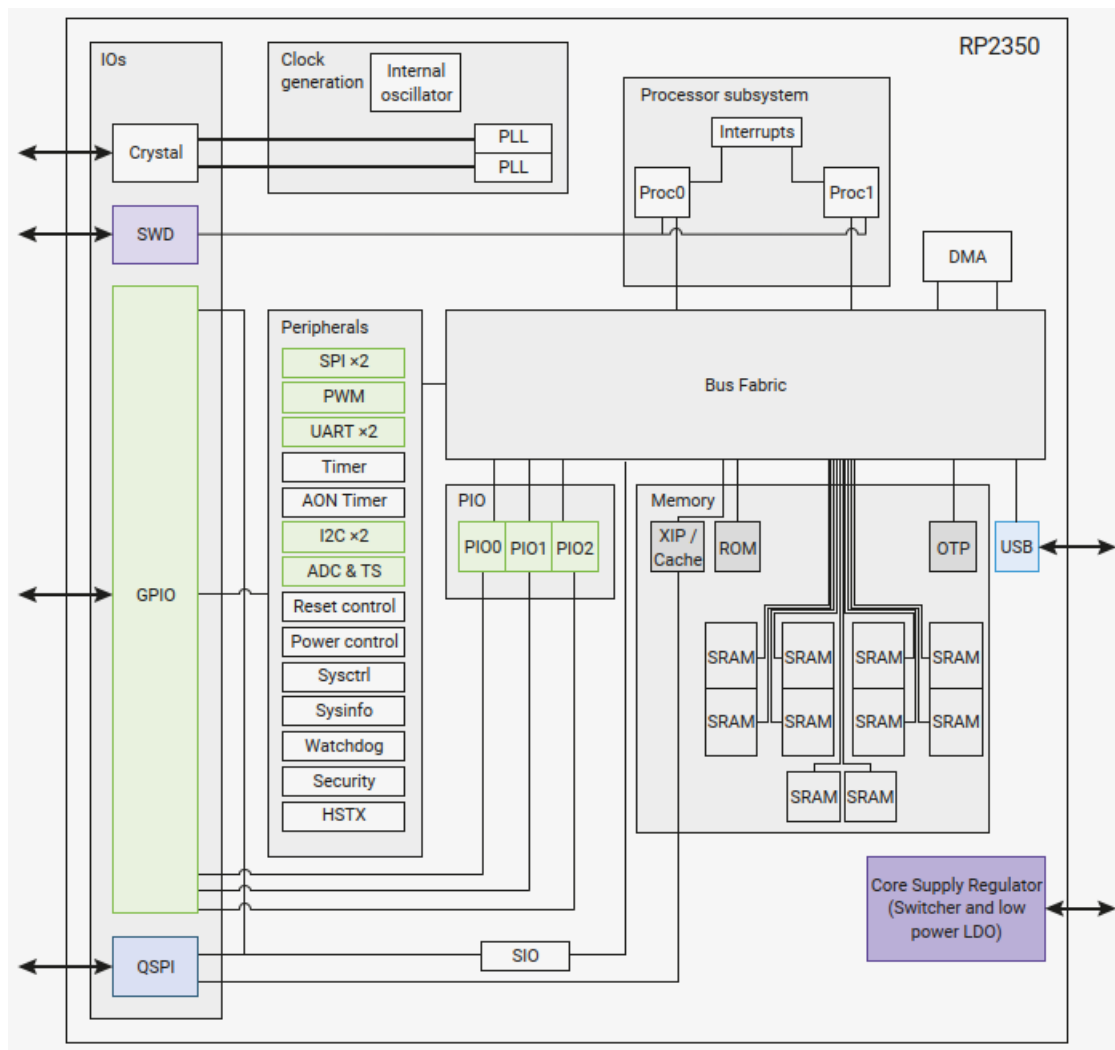
The TULIP4041 is a smart peripheral for the HP41 series of calculators. It is capable of emulating nearly all of the original memory and peripheral modules and offers a number of new features for the HP41. Some of the features it supports:

- 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.

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)

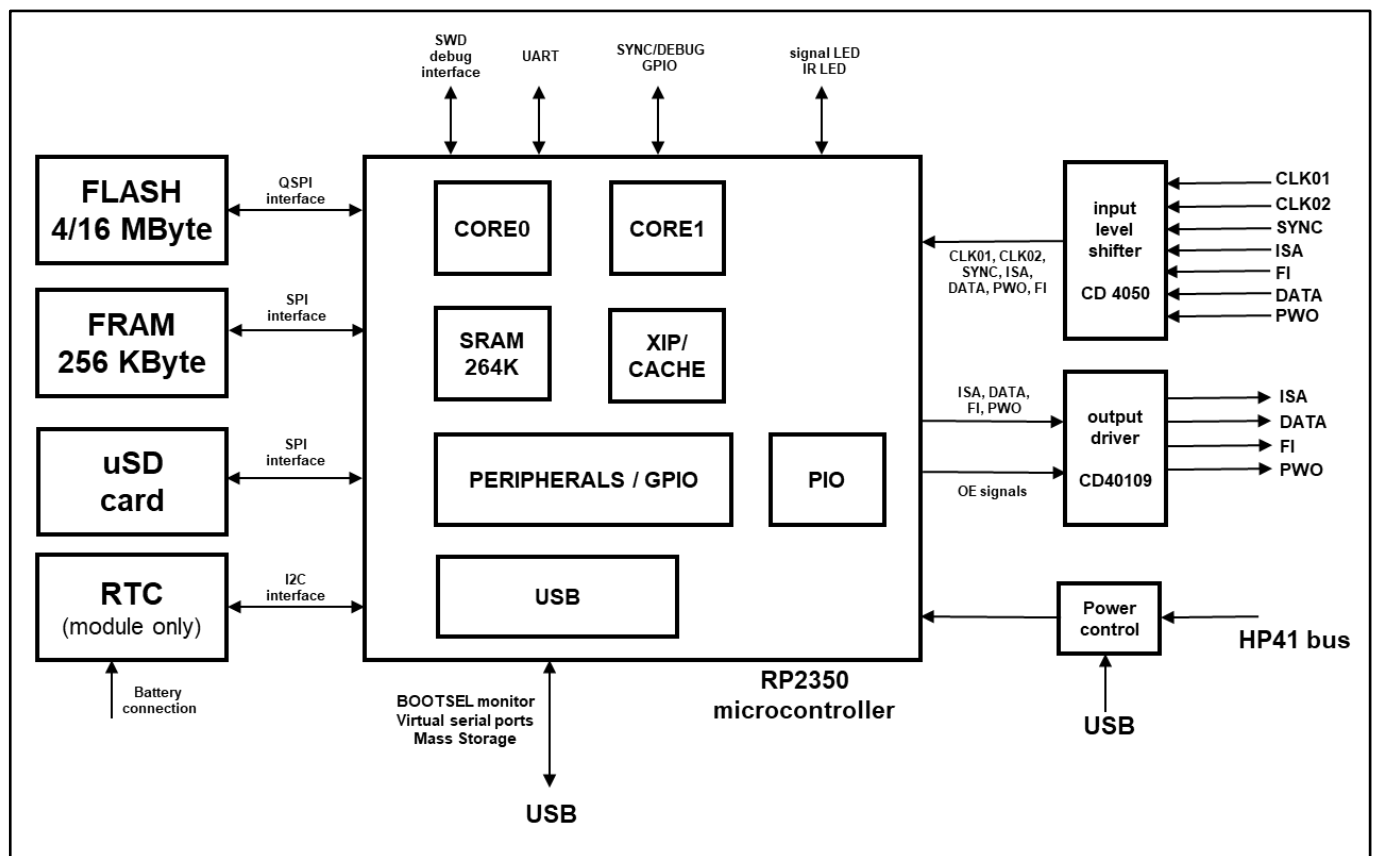
RP2350 development boards are available from several vendors, including the Raspberry Pico2 which is used in the prototyping phase of this project.

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 it 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:

- Do a forced jump to the label isa_inst_out, where a blocking pull is from the TX FIFO
- Put a single '1' bit in the TX FIFO, this will pull the bit from the TX FIFO and advance the state machine to a blocking wait for SYNC_TIME (which does not happen)
- Do a forced jump to the label isa_out with a sideset of 0, here is the out instruction to output one bit to ISA and waits for CLK02 rising edge (which does not happen). This drives ISA and the output enable
- Software then waits for about 20 usecs (about the time needed for the calculator to pick up the ISA and start to wake up)
- Do a forced jump back to the start of the ISAOUT state machine to the blocking pull to wait for the carry bit

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. The FI input requires an additional level shifter for which there is no space on the module. On the module version the FI output is sent 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).

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.

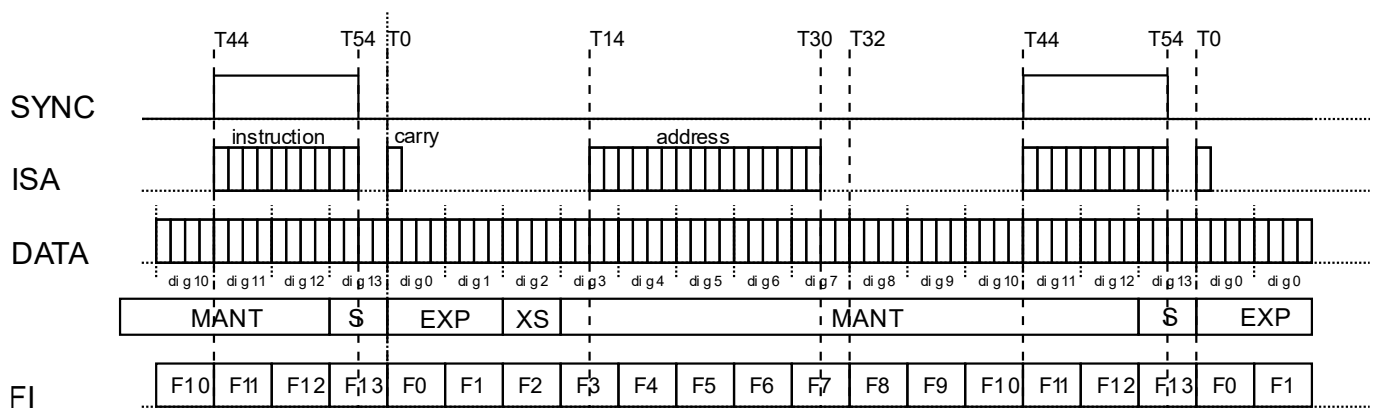
To output a frame the C level software needs to prepare a specific pattern (with checksum) of half bits as follows:

- 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 firmware 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).

7.1. TULIP4041 PICO/ RP2350 pinout

The TULIP4041 has a number of interfaces to the outside world. This paragraph handles the I/O as seen from the RP2350 microcontroller, the Pico 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.
- 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.

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 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 than appr. 1 MByte. Current image size is about 245 KByte (March 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 delay in software execution. All static arrays, such as ROM images embedded in the firmware and the lookup tables for the disassembler are in FLASH.

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 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 yet to be defined. Initial experience with the DevBoard indicates that 4 Mbyte is more than enough for everyday use. The Module has 16 MByte of FLASH memory.

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.

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.

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.

7.5. Micro SD card storage

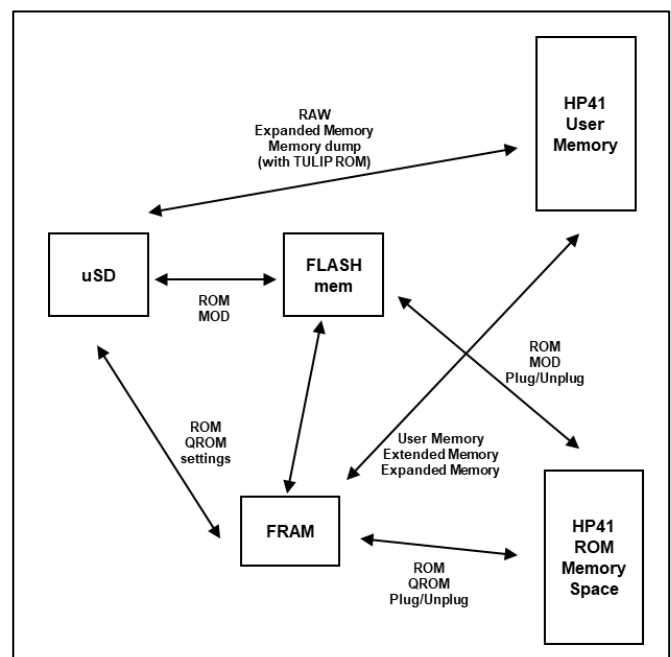
The micro SD card is provided by the user and can have any reasonable size larger than 2 GByte (the used FAT library does not support SDSC cards, only SDHC and better). It should be formatted with the FAT or exFAT (recommended) file system and contains the following information:

- 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
- In practice use your own directory with a repository of ROM and MOD files that you want have available in FLASH for plugging. Firmware functions allow easy mass import for the uSD card into FLASH and upgrade in case of changes

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. This file system and its support routines is currently in development.

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.

[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 (controlled by the firmware) 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. To 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.

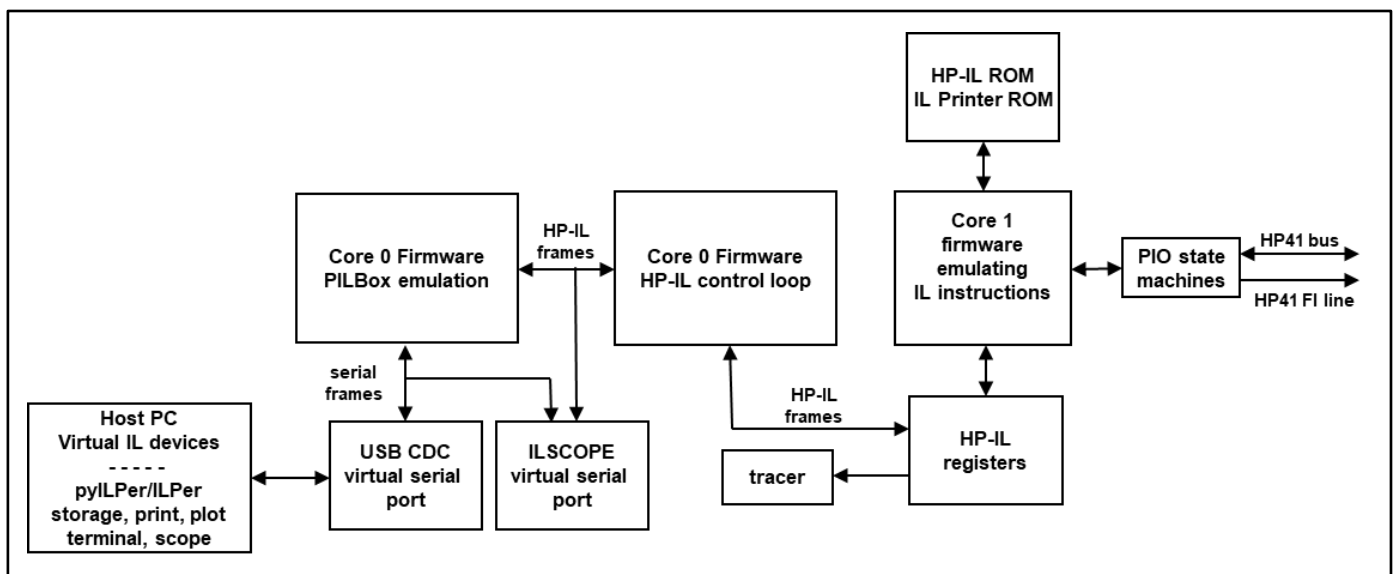
The TULIP can emulate QROM (or MLDL RAM) as part of the ROM mapping. QROM must always be mapped in FRAM, because only FRAM is writeable with the WROM instruction. FRAM storage is persistent and will withstand a power cycle or reset.

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.

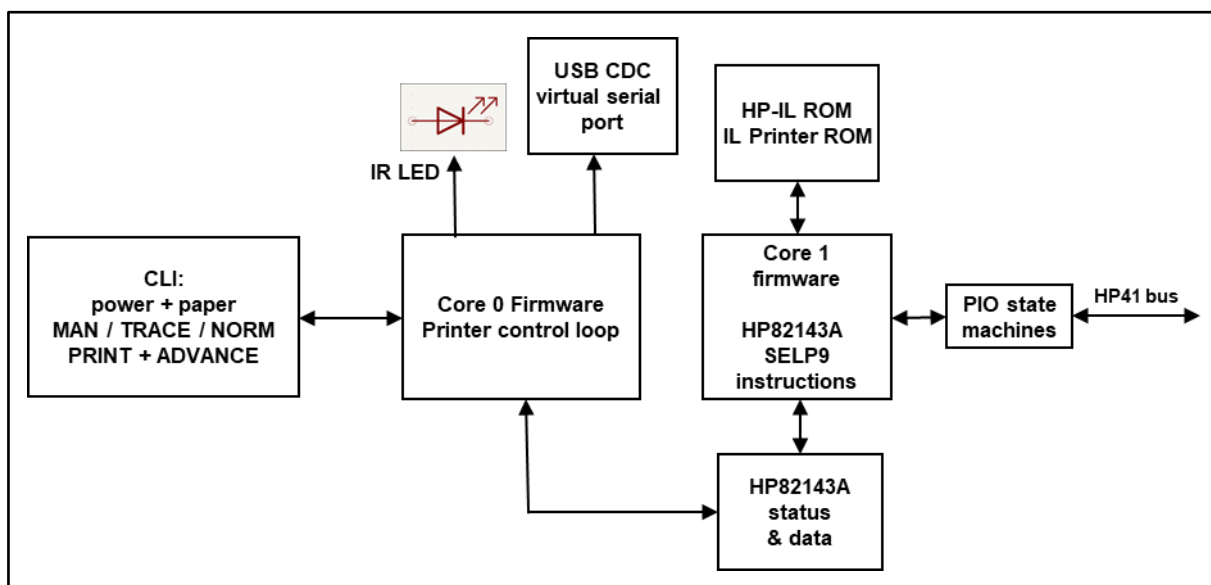
When there is no connection with a serial device the HP-IL loop is closed. When there is a serial connection and this is not a virtual HP-IL device (like ILPer or pyILPer) then the loop is open, and any HP-IL operations will result in

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 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.



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 are 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. 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

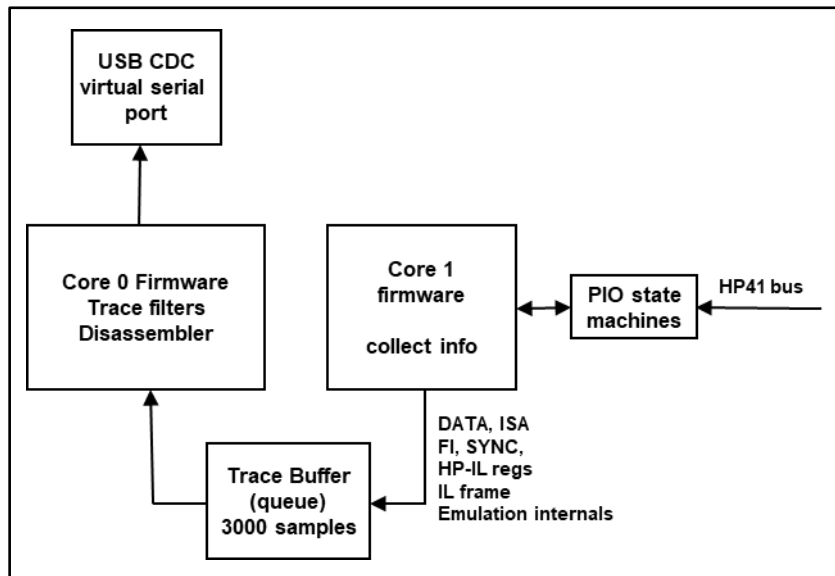
The contents of the Extended Memory modules are saved in FRAM and will survive a power cycle. The memory contents are not erased when a module is unplugged.

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.

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.

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).



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
- RAMSCLT 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 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, 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.

10. [Using the TULIP4041](#)

The primary interface for using the TULIP4041 is the USB interface. This offers a number of virtual serial ports (VCP) to the host system and a USB storage medium. On a Windows system these will be visible as COM ports, on Linux these will typically be `/dev/ttyACMx` ports). The next section assumes a Windows host computer. Due to the handling of these ports on a host system it can sometimes be tricky to identify a port, and this can change as well. The assignment of the COM port numbers appears to be somewhat arbitrary. On a Windows system, use the Device Manager -> Communication Ports to get a list of which COM ports are used. The storage medium is the micro SD card and a drive letter will be assigned by the host computer, even when no micro SD card is plugged on the TULIP.

When the TULIP4041 is connected (via USB) to a host computer the first action is typically to connect to the CLI with a terminal program.

Normally the TULIP4041 is plugged in an HP41 calculator for use. It is very well possible to connect a unit to USB without being plugged in an HP41.

The virtual devices will be visible only when valid firmware is loaded on the TULIP4041. Please refer to the section in one of the next paragraphs on instructions to load the TULIP firmware.

10.1. [Virtual serial ports](#)

The primary interface for the TULIP4041 is the USB interface. This offers a number of virtual serial ports (VCP) to the host system. On a Windows system these will be visible as COM ports. This section assumes a Windows host computer. The COM port numbers used here are indicative. Details of each port use is described in detail in later paragraphs.

All COM ports use 8 bits, 1 stop bit, no parity, and should be set to the highest possible baud rate, generally the highest baud rate will be used automatically. Regardless of the baud rate, the communication is always at maximum speed.

- **COM(1) or `/dev/ttyACM0`** : used for the TULIP Command Line Interface (CLI). Use a terminal program (for Windows I use Teraterm, but most of these tools will work just fine). This is the main tool to manage ROM images, plug/unplug modules and manage all aspects of the TULIP4041.
- **COM(2) or `/dev/ttyACM1`**: used for HP41 mcode/bus tracer output. Another instance of a terminal program should be connected to this COM(2) port to present the results. The TULIP4041 constantly monitors the HP41 system bus (where the modules are physically plugged) and provides an almost realtime stream of information with details about bus activity. CLI commands are used to control if and how the data is presented and filtered. Default is no tracing active at all. Tracing can have impact on the performance of the TULIP4041. Tracing uses memory based buffers with limited capacity and filtering should be used to prevent overflows of these buffers. Overflows are visualized in the trace results and do not impact the operation of the TULIP4041.
- **COM(3) or `/dev/ttyACM2`** : used for HP-IL communication. The TULIP4041 can emulate the HP-IL ROM (with printer) and virtual HP-IL devices on a host computer (ILPer or PyILPer) can be connected to COM(4) to offer a large number of virtual devices. Internally the TULIP4041 emulates a PILBox, the host computer will see a PILBox connected. When a host computer is not connected the firmware will automatically simulate a closed HP-IL loop. Using the CLI a virtual drive can be enabled in a container file on the SD card. When a host is connected but this is not a valid ILPer or pyILPer virtual HP-IL device the loop will appear open.

- **COM(4) or /dev/ttyACM3**: used for HP-IL frame tracing, much like an HP-IL scope. A terminal program should be connected to this port to present the results. In addition to HP-IL frames the serial communication to and from the emulated PILBox are shown (when enabled).
- **COM(5) or /dev/ttyACM4** : used for printer output from the HP82143A printer ROM (PRINTER 1E). When this ROM is plugged the output is sent to this port and the HP82240 simulator (for Windows) can be used for presenting the results. The HP82240 simulator should be set to HP82143A mode for correct results, including graphics. With the CLI the printer characteristics can be changed, such as printing to a terminal window. Other settings include TRACE/MAN/NORM mode. This can be done only with the CLI (or functions in the TULIP ROM). In addition it is possible to direct the HP82143A output to the Infrared LED.

Another serial port is available on GPIO 0 and 1, UART Tx and UART Rx. This serial port runs at 11520 baud, 8 databits, no parity and 1 stopbit. This port can be used to connect to a Pico Probe (or a USB to UART interface, connect only 3.3V equipment to the GPIO's!) and offer a very low level CLI that was used for development. In some cases low-level error messages are shown here. Use only when really needed in case of support. Do NOT use the offered FLASH programming functions! The function of this port may change in future firmware versions.

If a connection to one of the virtual serial ports is made or broken a message in the CLI will be shown.

10.2. [BOOTSEL mode and firmware upgrade](#)

The RP2350 processor uses BOOTSEL mode for a number of internal functions such as firmware update, FLASH programming and other settings. Also the update of the TULIP4041 firmware requires entering BOOTSEL mode. In this mode the processor enables a special USB interface (for FLASH programming for example). Please refer to the RP2350 documentation and tools for a complete description.

In BOOTSEL mode the RP2350 presents a USB disk drive to the connected host system. To upgrade or change the firmware, simply copy a valid firmware file with .UF2 extension to this drive. When the copy operation is complete the RP2350 processor will automatically reboot into the new firmware.

To enter BOOTSEL mode use one of the following methods:

- Use the CLI to issue the **system BOOTSEL** command, this is the preferred method. This will work only if there was already a TULIP firmware running. In rare cases the TULIP appears not to enter BOOTSEL mode, when that happens power cycle the unit, and try again.
- On the TULIP-DevBoard disconnect the USB cable, push the button on the Pico board, and while pushing the button connect the USB cable again and then release the button. The RP2350 should now be in BOOTSEL mode. This is also the preferred method for a new Pico board or a Pico board that had other software loaded
- On the TULIP-Module create a short between the USB-BOOT and GND while disconnecting and reconnecting USB (this is the same as pushing the button in the Pico board).
- With a development system and Pico Probe connected through the debug interface new firmware can be downloaded using the development tool (VS Code or OpenOCD)

IMPORTANT: due to the use of multiple virtual serial ports it is not possible to enter BOOTSEL mode under control of the PICOTOOL utility on the host computer when the TULIP4041 firmware is running. Once in BOOTSEL mode the various functions of PICOTOOL can be used.

10.3. [Reset of the TULIP4041](#)

A reset of the TULIP4041 can be achieved by removing all power and applying power again. Alternatively, connect the RUN pin on the Pico board to GND for a reset. The RUN pin is also exposed on the TULIP-Module, and shorting this pad to GND will also reset the unit.

The CLI can be used to reset the TULIP4041 using the **system REBOOT** command.

10.4. [TULIP4041 power consumption](#)

Please re-read section 7.7 : TULIP4041 power on the power characteristics of the TULIP4041.

10.5. [Micro SD card](#)

The TULIP4041 has slot for a micro SD card. This is used for central off-line storage of ROM images and it is necessary to use a card to program ROM images in FLASH and/or FRAM to be able to use the TULIP4041. Please understand the following:

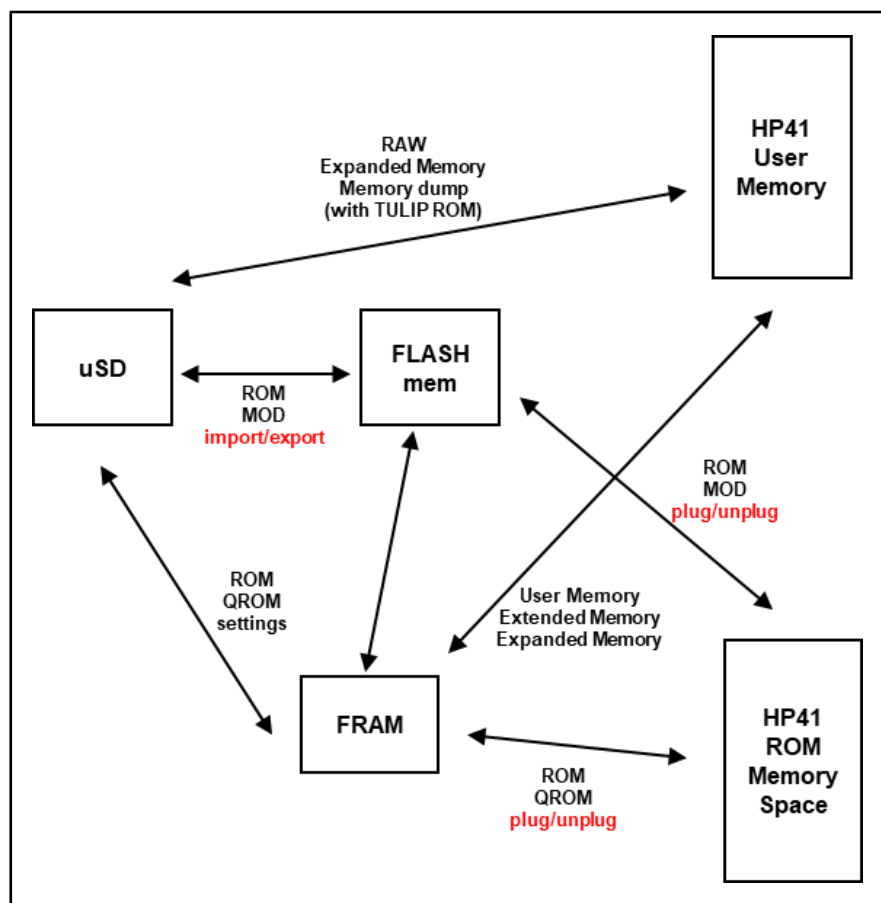
- **IMPORTANT:** The name of a directory or file in the micro SD card should not contain any spaces. This is not supported by the functions in the firmware unless the name is enclosed in quotes, which of course is inconvenient
- The microSD card must have a size of at least 2 GByte (SDHC and SDXC type cards), the FatFS library used does not support SDSC type cards
- Best plug or unplug a micro SD card only when the TULIP4041 is not powered
- Format the card on a host computer according to the FAT or exFAT file system. The TULIP4041 firmware uses the FatFS software to manage the files on the card, and only FAT or exFAT file systems are supported
- The exFAT file system is recommended, this is much faster than the FAT file system, especially when connecting the TULIP to a host computer
- Filenames of files that are to be imported in FLASH or FRAM are limited to 31 characters (and that includes the extension and dot) and should not have any spaces
- (optional) After formatting create a MOD and ROM subdirectory in the root directory. A RAW subdirectory or a LIF container is optional.
- (optional) Any MOD images that you plan to use should be put in the MOD subdirectory
- (optional) Any ROM images that you plan to use should be put in the ROM subdirectory
- Best practice is to copy all your favorite ROM and MOD files that you wish to be pluggable in a single directory
- Put the micro SD card in the slot on the TULIP device and power the device
- Use the **sdcard mount** in the CLI to activate the card, although this normally happens automatically
- Use the **dir** command, followed by the directory to list the contents of the card
- The micro SD card is connected to the processor with a single channel SPI connection which is not very fast
- The micro SD card is exposed as a removable USB drive on your host computer. Use the command **sdcard connect** to make the card visible, and **sdcard eject** to remove the card from the host PC. After ejecting the drive itself is still visible on the host computer but cannot be used

11. ROM images, User memory and the file system

The handling of ROM images is an essential part of the TULIP4041 system to allow these images to be visible to the HP41. The main source of any ROM image is the micro SD card, this is the place where you collect your ROM and MOD files. The micro SD card is easily accessible from a host computer as a disk drive and you can copy your ROM repository to a directory on the card.

The TULIP firmware cannot delete, copy or rename files in the micro SD card, this must be done by the host computer

In order to be (virtually) pluggable into the HP41 memory space a ROM image must be present in FLASH or FRAM memory. The **import** command (just like on the DM41X) copies a file from the uSD card to FLASH or FRAM. ROM images containing writeable QROM should be imported into FRAM memory.



The image type can be a MOD or ROM file. Both FRAM and FLASH can contain other file types:

.MOD	MOD1/MOD2 files
.ROM	ROM files (only uncompressed ROM files are supported)
.UMM	User Memory copy/backup
.EXT	Extended memory copy/backup
.EXP	Expanded memory copy/backup
.TRM	TULIP ROM mapping
.TGL	TULIP Global settings
.TTF	TULIP Tracer Filters, triggers and settings

11.1. Importing files and the Flash File Manager

The files in FLASH and FRAM are handled by a very basic Flash File Manager (FFM). It handles the file type, file name (up to 31 characters including the extension and the dot), file size and position of the next file. The files can be programmed from the micro SD card in FLASH or FRAM with the **import** command in the CLI. If there is no more room or a file with the same name already exists an error message will be given (unless the update option is used). Batch importing can be used to import all supported filetypes from a subdirectory. Before it can be used, the FFM must initialize storage with the **flash INIT** command. This creates the start of the file chain in FLASH by creating a file name "TULIP4041 FLASH HEADER".

A file can be removed from FLASH or FRAM with the **delete** command. Be aware that the image must be unplugged before deleting.

[NOT SUPPORTED YET] To save a ROM image back to the micro SD card, use the **export** command. This will typically be useful for QROM images that you have modified (with HEPAX for example). You can then easily copy the exported file to your host computer for further actions. The other files types can be imported and exported for archiving or sharing.

To find out which files are in FLASH and/or FRAM use the **list** command. To see which modules are already plugged use the **cat** command (just like your HP41). The **dir** command is used to list the files on the micro SD card.

Importing, exporting and deleting files can be done only when the calculator is off or in light sleep (PWO is low). Be aware that sometimes the HP41 wakes up by itself (due to an alarm or other peripheral) which may corrupt the file. Plugging and unplugging can be done in a running calculator, but must be done with extreme care.

Due to efficient storage in FLASH memory the granularity of files storage is 256 bytes. FLASH memory has a limited number of write cycles and wear may occur on memory locations that are frequently written to. According to the device datasheet the number of erase/write cycles should be at least 100.000. Theoretically this will give you about 273 years if you erase a FLASH sector every day. The file manager has some limited handling to prevent wear in FLASH memory. If the memory starts to develop problems the FLASH memory chip must be replaced. FLASH and FRAM storage may become fragmented and currently no tools for defragmentation are planned. Best approach when defragmentation becomes an issue is to save (export) all relevant files to the micro SD card and re-initialized storage.

IMPORTANT: During any FLASH erase or programming operation the TULIP must remain powered. Power loss may result in loss of data and/or a corrupt file system. During programming some FLASH contents are temporarily saved in RAM.

The size of a file in the flash system can be estimated as follows:

- Take the size of the file in bytes
- Add 40 bytes for the file system header
- Round up to the next 256-byte multiple

To prevent wear on the FLASH memory most new files are placed at the end of the existing files. Gaps (which can exist after removing files) can be filled by new imported files. Files can also be updated without prior deleting, in this case the size and type must be identical. Deleting a file means that the file is not completely erased, but the file type is set to erased and the file chain is kept intact. Only when a new file is imported in the space of a deleted file will the chain be updated. The **import** function will search in the FLASH File System for the smallest space where the new file will fit, otherwise the file will be programmed at the end of the chain.

Deleting files and subsequent programming of new files can lead to fragmentation of the Flash File System, and there are no tools to resolve this fragmentation. Given the size of FLASH (especially in the Module version) this will not become a common issue. Fragmentation can be resolved only by completely erasing all FLASH and importing the needed files again.

The FLASH memory can be completely erased (nuked) with the ***flash NUKEALL*** command. This erases the complete area reserved for the file system. It will not erase the part of FLASH where the firmware is. After erasing the filesystem must be initialized with the command ***flash INIT***, this will initialize the filesystem by creating the header file and the necessary system files. When the file chain is corrupted or other issues are present this is the only way to resolve this.

FRAM does not have a filesystem yet, and system data is saved at a number of fixed addresses

- ROMMAP, all the plugged ROMs, starting at 0x00000 (offset in FRAM)
- Persistent system settings, starting at 0x1D000
- Extended Memory, starting at 0x1E000
- The highest FRAM address is 0x3FFFF (256 Kbyte)

Before use the FRAM is automatically initialized. Both FLASH and FRAM can be inspected with the ***flash dump*** and ***fram dump*** commands.

FRAM supports an almost infinite number of write cycles. Since FRAM is used constantly by a running HP41 operations on FRAM can only be done when the calculator is not running.

[NOT YET SUPPORTED] Due to its limited capacity (256 Kbytes) only files that really need the random write access should be placed in FRAM. This includes a number of system files (requiring about **TBA** Kbyte) and files containing QROM images (see next paragraph). Any file can be forced to be imported in FRAM, but only relevant files (QROM and some system files) should be imported in FRAM. By default a file is always imported in FLASH, unless it is a MOD file where the first ROM image is tagged as RAM. A .ROM file can be manually forced in FLASH or FRAM if needed. Fragmentation can occur in FRAM as well.

[NOT YET SUPPORTED] Sometimes a ROM image is updated by the developer. To facilitate the update process it is possible to replace an existing file in FLASH or FRAM with an updated version of the image under the following conditions:

- The file name is identical
- The file size is identical
- The contents are different
- There is no need to first unplug the ROM image if it was plugged if:
 - The exact same bank assignment is used in case of a multi-bank module
 - The exact same page order is used in case of a multi-page image
 - There are no changes in the hardware support of a MOD file if it is plugged

When a ROM image is updated in the way described above multiple FLASH sectors may be erased containing parts of the ROM image to be updated. When multiple ROM images are updated a single FLASH sector may be erased multiple times thus increasing the risk or wear. This does not apply when updating ROM images in FRAM.

A special variation of the import function can be used for batch programming your complete ROM and MOD repository stored in a subdirectory on the micro SD card to FLASH. This saves you a lot of manual work. Prepare the directory with the files you intend to use and make an estimate of the amount of space used in the FLASH file system, to check if it will fit. Best is to start with fully erased (and initialized) FLASH, but that is not really necessary. In case of any updated files, you can simply run this command with the UPDATE option. The command

to be used is ***import [directory] ALL***. This will program all MOD and ROM files in the subdirectory “directory” to FLASH with the following rules:

- Existing files will be skipped
- **[NOT YET SUPPORTED]** When the UPDATE parameter is used the existing files will be replaced under the same rules as the single file update
- New files will be appended to the file system (or inserted in free slots)
- The import will stop after all files have been imported or when the end of the FLASH file system is reached
- Files will only be imported into FLASH, never in FRAM

When importing a single file the filename in the Flash File System will be in the case as you typed on the command line. When importing multiple files, the case will be as it appears in the directory listing on the micro SD card.

[NOT YET SUPPORTED] After importing one or more files the operation can be verified with the *compare* option. This can be used on a single file or on a complete directory. A use case is to check which files may need to be updated. As an extra the compare option will indicate for every file if it can be simply reprogrammed, or if a flash erase must be done first. When using the *compare* option the contents of FLASH or FRAM will never be modified.

Details of the commands to manipulate ROM images are described in the CLI reference. In short here is an overview of the relevant CLI commands:

delete [filename]

list <filename>

dir <directory>

import [directory] [ALL]

import [filename]

11.2. [Files containing QROM](#)

[NOT YET SUPPORTED] QROM stands for Quasi-ROM. This is a ROM image that is actually in RAM and can be written to with a special HP41 instruction (WROM, hex 040). This is the way the MLDL’s from the good old days work, and this also works in the HEPAX module, NoVRAM, MLDL2000 and the HP41CL. To be able to do the write instruction the ROM image must be placed in FRAM.

If a MOD file contains (one or more) QROM images it will automatically be imported in FRAM (if there is enough room), but only if the first image is a QROM image. It is possible to force ROM images in FRAM (for example if you want to edit these). The ROM image must be plugged before it can be accessed by the HP41. A QROM image is recognized by the RAM indicator in the page metadata.

Any ROM image in FRAM can be written to by the WROM (hex 040) instruction if it is plugged to the HP41. Using this instruction on an image in FLASH will have no effect. As with most MLDL type devices a QROM image can be write protected and also read protected. Protecting from reading by the HP41 can be useful to prevent the HP41 from responding in an undesired way to the QROM contents. This used to be the case when RAM chips were used in MLDL type devices. When not initialized the uninitialized RAM chips contains random data where are a guarantee for erratic calculator behavior. By disabling reads from the calculator, but by enabling writes the RAM contents can be initialized to zero (using a function in a specialized MLDL support ROM) to make reading safe. The ***qrom*** command is used for setting specific parameters of plugged modules.

<i>qrom <status></i>	list the status of the plugged modules, same as <i>cat</i>
<i>qrom p[x] <status></i>	detailed list for the indicated Page
<i>qrom p[x] read/noread</i>	read enable/disable, all banks in the Page
<i>qrom p[x] write/nowrite</i>	enable or disable writes (only for FRAM), all banks in the Page
<i>qrom p[x] clear <b[n]></i>	clear the complete Page to zero, indicate bank in needed

In all cases x is the Page number in hex, n is the bank number 1..4, default is bank 1

When a ROM image is plugged from FRAM, it is enabled by default for both reading and writing and for reading when plugged from FLASH. If this is a problem, simply plug the module while the calculator is off (good practice anyway) and adjust the settings before switching that calculator on.

A .ROM file does not have metadata and the user must force it to be imported in FRAM if that is the intention.

To optimize usage of FRAM it is NOT recommended to use MOD files with a mix of QROM and regular ROM images (for example some existing HEPAX MOD files).

11.3. [Plugging ROM images](#)

[plugging of .MOD files is not yet supported]

In order to make a ROM image visible to the HP41 it must first be (virtually) plugged in one of the ports. When a ROM image belongs to a peripheral (Printer or HP-IL module for example) or contains special hardware (like the HEPAX module) the hardware support for the special function must be enabled. For a ROM image in a .MOD file this is fully automatic in case the hardware is supported), but for a .ROM file this must be done by the user. The advantage of a .MOD file is also that multi-page images (like HEPAX or the Advantage ROM) are handled automatic. The automatic handling can only work if the TULIP knows if any physical modules are already plugged, and that cannot be done automatically. Therefore any modules that are physically plugged or already present in the HP41 (for example in the HP41CX) must be made known to the TULIP with the ***reserve*** command. This can only be done if you know the page it is plugged in. Pages 0, 1, 2 and 3 are always reserved for the HP41 system ROMS. Use the ***list*** command to get details of the ROM or MOD file you intend to plug.

<i>reserve [x] <name></i>	where x is the physical page number in hex. Optionally type the name (or any other descriptive text) of the module, this is used in the <i>list</i> command. This name or text can not contain any spaces
<i>reserve cx</i>	if you have an HP41CX to reserve Page 3 and 5.
<i>reserve printer</i>	to reserve Page 6 (for the HP82143A, IR or HP-IL Printer module)
<i>reserve hpil</i>	to reserve Page 7 if you have the real HP-IL module plugged
<i>reserve clear [X]</i>	clear the reservation in the named port, port number in hex
<i>reserve clear all</i>	clear all reservations. The reservations for the system Ports cannot be cleared

Port	Page	Address	All models	HP41CX	Typical HP41 use
4	F	0xF000 - 0xFFFF	Port 4, upper Page	Port 4, upper Page	
	E	0xE000 - 0xEFFF	Port 4, lower Page	Port 4, lower Page	cardreader
3	D	0xD000 - 0xDFFF	Port 3, upper Page	Port 3, upper Page	
	C	0xC000 - 0xCFFF	Port 3, lower Page	Port 3, lower Page	wand (if cardreader plugged)
2	B	0xB000 - 0xBFFF	Port 2, upper Page	Port 2, upper Page	
	A	0xA000 - 0xAFFF	Port 2, lower Page	Port 2, lower Page	
1	9	0x9000 - 0x9FFF	Port 1, upper Page	Port 1, upper Page	
	8	0x8000 - 0x8FFF	Port 1, lower Page	Port 1, lower Page	
	7	0x7000 - 0x7FFF	HP-IL Module	HP-IL Module	
	6	0x6000 - 0x6FFF	(IR)PRINTER/IL Printer	(IR)PRINTER/IL Printer	
	5	0x5000 - 0x5FFF	TIME Module	TIME Module/CX FNS	
	4	0x4000 - 0x4FFF	TAKE OVER ROM disabled IL Printer	TAKE OVER ROM disabled IL Printer	Service ROM, 4LIB
	3	0x3000 - 0x3FFF		CX FNS – bank 1	
	2	0x2000 - 0x2FFF	HP41C OS – ROM2	HP41CX OS – ROM2	
	1	0x1000 - 0x1FFF	HP41C OS – ROM1	HP41CX OS – ROM1	
	0	0x0000 - 0x0FFF	HP41C OS – ROM0	HP41CX OS – ROM0	

Regular User ROM space
Fixed module page if plugged
Be careful here
HP41CL/CX fixed pages
Not available in HP41CL



In case of an HP41CX, you must also inform the TULIP about any other plugged physical modules. When you use an HP41CL, remember that this is always an HP41CX configuration, and that any virtual ROM images in the HP41CL must also be plugged. This ‘plugging’ of a physical module only prevents the TULIP from plugging any ROM images in the reserved Page, it does not prevent you from plugging an identical ‘virtual’ module.

To plug a ROM image in the TULIP simply type the **plug** command with the name of the file. No need to enter the .ROM or .MOD extension. For a .MOD file nothing else is needed. If you have correctly informed the TULIP about any physical modules the ROM image will be automatically plugged in the best possible Page according to the MOD handling algorithm. You can still force a MOD file in a specific Page if you think this is better. For a .ROM file you must indicate a Page and optionally a Bank number. But if you do not enter a Page number the firmware will plug the ROM image anyway in the first available Page starting at Page 8. An error message will be shown if the ROM image cannot be plugged, for example when the Page is already taken.

[NOT YET SUPPORTED] If a .MOD file contains (one or more) QROM images it will automatically be imported in FRAM (if there is enough room), but only if the first image is a QROM image. It is possible to force .ROM images in FRAM (for example if you want to edit these). The ROM image must be plugged before it can be accessed by the HP41. A QROM image is recognized by the RAM indicator in the page metadata, but only for a .MOD file.

Plugging a .ROM file means that you must provide the target Page. The Bank must be entered when the image must be in any of the banks 2..4. When plugging multi-bank ROM images bank 1 should be plugged last unless you are certain the calculator will not run in between the **plug** commands.

Summary of commands to plug a module:

plug [filename] [p] plug the ROM image in Page p (p in hex)

[NOT YET SUPPORTED]

- plug [filename]*** no Page number required, will plug in the first available Page. Use in combination with the ***reserve*** command
- plug [filename] T*** use this for testing in which Page a file will be plugged, plugging will not take place
- plug [filename] p b*** force the ROM image in Page p and bank b (b=1..4), only for .ROM files. This is the only plug variation that allows overplugging, meaning that it is not necessary to unplug the ROM image in the Page/Bank first.

The plugged ROM images are maintained in a rom map by the TULIP in the CModule class and this is stored in FRAM.

[NOT YET SUPPORTED] This is a table stored in a file in FRAM named “default_map.trm” (extension .TRM is reserved for TULIP Rom Map). This file is created by the FRAM initialization process. A copy of this file is maintained in the TULIP RAM to speed up access. The active rom map can be copied to a file with another name, any .TRM file can be assigned as the active rom map with the ***rommap*** command. This active rom map will also be the default upon startup. Be aware that the rom map also contains the plugged physical modules, and it is therefore not always straightforward to move the active rom map to another calculator.

IMPORTANT: the rom map contains pointers to the ROM images in FLASH and/or FRAM. If the files are deleted from FRAM or FLASH the rom map will become invalid, but there are NO checks upon startup.

The CModule class maintains the following information:

- Filename and address in the FLASH/FRAM file system. For a multi-bank module this is the filename of Bank 1 (in case of a .ROM image)
- Module type: MOD1, MOD2, ROM, or physical (to manage real modules plugged)
- Pointer to each Bank
- Current active Bank and a Sticky bit for ZEPROM emulation
- Module status: read/write enabled, QROM, DIRTY (QROM changes to be done)

[NOT YET SUPPORTED] Overview of the ***rommap*** command:

- rommap <status> <[filename]>*** show the contents of the rom map [filename], or the active rom map
- rommap copy [filename]*** create a copy of the current active rom map to a new file [filename]
- rommap active [filename]*** make the file [filename] the active rom map
- rommap check [filename]*** check if the rom map [filename] is still valid and all files exist

To view the ROM images that are plugged use ***cat*** command:

- cat*** show the contents of the rom map [filename], or the active rom map
- cat [X]*** show details of the ROM plugged in Page X (in hex) with a hex dump of that ROM

11.4. [Bank switching in the TULIP](#)

Bank Switching is a delicate topic, as it is traditionally handled by the hardware in a module, and different implementations exist. It is therefore a bit tricky to generalize the behavior of bank switching, but necessary to emulate this as accurate as possible in the TULIP firmware. The TULIP monitors the HP41 bus for the bank switching instructions ENBANKx (x=1..4) for all bank for including this information in the tracer and for selection

the correct word from a ROM image. The active bank is maintained for every single page and is reset to Bank 1 when the calculator goes in STANDBY mode.

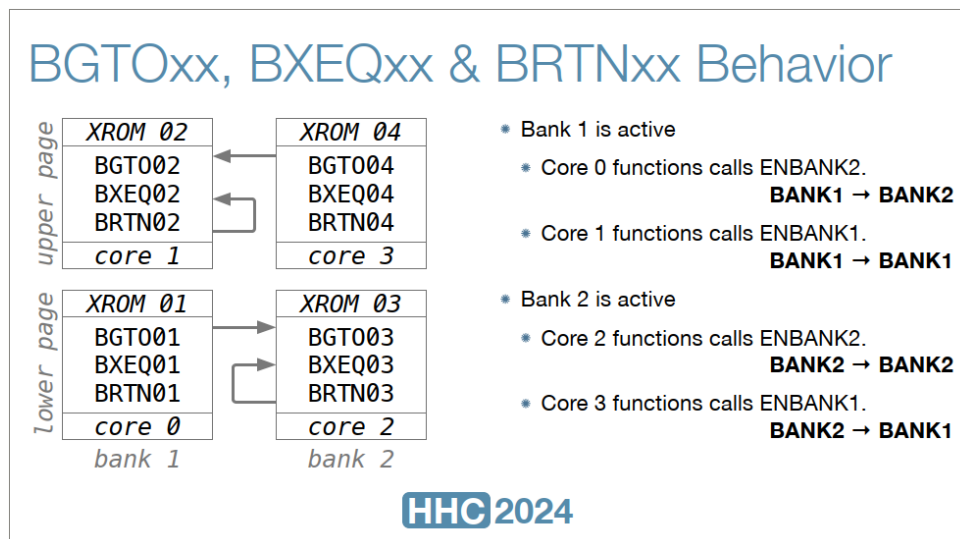
ENBANKx instruction in Page 3 switches banks in Page 5 (HP41CX behavior), used for the tracer

ENBANKx instruction in the Port Pages (Pages 8..F) switches banks in all Pages in that Port. For example switching a Bank in Page 9 also switches banks in Page 8.

ENBANKx instruction in any other Page (4..7) switches banks in that Page only. This enables bank switching in Page 4 and in Page 5 (if a CX is not used)

When a Bank is switched in a PORT and the other Page does not have an active ROM image plugged in that Bank then always the information from Bank 1 is fetched. This may conflict with some specific hardware modules.

A special case of bank switching is implemented for emulation of the ZEPROM. This must be specifically enabled for a Port with the **emulate** command, and the ZEPROM ROM images must be manually plugged in the correct Page and Bank, the ROM images are not automatically placed like in the real ZEPROM hardware. The main feature of the ZEPROM bank switching is that the banks are sticky and are not reset to Bank 1 when entering STANDBY mode or DEEP SLEEP. To achieve this the bank switching mechanism in the TULIP firmware checks in the active Page has the 'sticky' bit set, and upon a bank change marks the Page as dirty. On the next PWO event (calculator going in Light or Deep Sleep) the current state is then saved to FRAM if the dirty bit was set.



EXPLANATION OF ZEPROM BANK SWITCHING (BY SYLVAIN COTE)

The TULIP emulation of the ZEPROM follows the explanation in the figure above (thanks Sylvain). More details in his presentation at HHC2024.

11.5. [Embedded ROM images](#)

The TULIP has a number of built-in ROMs or Embedded ROMs. These are fully included in the firmware and do not have to be imported, and also are not visible with the *list* command. These Embedded ROMs are:

- HP-IL and HP-IL Printer ROM
- HP82143A Printer ROM

When plugging these ROM images the Page and hardware emulation is automatically selected (and disabled when unplugging). The ROM images are identical to the original devices.

<i>plug hpil</i>	plugs the Embedded HP-IL ROM image in Page 7 and enabled HP-IL emulation
<i>plug ilprinter</i>	plugs the Embedded HP-IL Printer image in Page 6 and enabled HP-IL emulation. The HP-IL ROM must be plugged first!
<i>plug printer</i>	plugs the HP82143A printer ROM in Page 6 and enables the emulation for this

Unplugging is done with the ***unplug*** command, and the correct Page number must be given. Emulation will then be disabled.

11.6. [Hardware emulation](#)

In order to support emulation of specific hardware this has to be enabled. In case you are using a real HP-IL module the HP-IL module in the TULIP must be switched off to prevent conflicts. When using the Embedded ROM images the emulation is automatic. It is very well possible to import the HP-IL ROM image and plug it in Page 7 (**this ROM must be plugged in Page 7!**). In that case the emulation inside the TULIP of the HP-IL hardware registers must be manually enabled with the ***emulate*** command.

<i>emulate</i>	show the status of emulation
<i>emulate hpil</i>	toggle HPIL hardware emulation
<i>emulate printer</i>	toggle HP82143A printer emulation
<i>emulate zeprom P</i>	toggle ZEPROM emulation in Page P (hex) for sticky bank switching
	See the separate topic on bank switching in paragraph 11.4.

Do NOT manually switch off emulation while a ROM is still plugged that needs the emulation. This will lead to unexpected events.

11.7. [Unplugging ROM images](#)

To remove a module and make it invisible to the HP41 you can use the ***unplug*** command. This simply removes a ROM image from the rom map. Unplugging is done by Page number. If you ***unplug*** from a Page which is reserved for a physical module this will be unplugged from the TULIP, remember to physically unplug as well.

<i>unplug p</i>	unplug all Banks in the indicated Page p (p in hex), also cancels a reserved Page
<i>unplug p b</i>	unplug the Bank b in Page p
<i>unplug all</i>	unplug all plugged ROMs except reserved Pages
<i>unplug ALL</i>	unplug all ROMs including all reserved Pages (except Page 0..3)

11.8. [HP41 User Memory](#)

The HP41 User Memory is the calculators memory that is used for the normal registers, stack, status registers and program memory. This is a single address space of 56-bit registers and is totally independent from the memory space where the ROM images and the HP41 operating system resides. The HP41 user can access the memory with the STO and RCL instructions, by accessing the stack, using Extended Memory functions and by entering programs.

HP41 Address	Name	Used for
0xC00 – 0xCFF	Expanded Memory 3	Expanded Memory in HP41CL/MAXX/TULIP
0x800 – 0x8FF	Expanded Memory 2	Expanded Memory in HP41CL/MAXX/TULIP
0x400 – 0x4FF	Expanded Memory 1	Expanded Memory in HP41CL/MAXX/TULIP
0x301 – 0x3EF	X-Memory	X-memory Module 2
0x201 – 0x2EF	X-Memory	X-memory Module 1
0x100 – 0x1FF	User Memory	User Registers HP41CV + CX, Quad Memory Module Programs, Key Assignments, Alarms, Buffers
0x0C0 – 0x0FF	User Memory	User Registers HP41C Programs, Key Assignments, Alarms, Buffers
0x040 – 0x0BF	X-Memory	X-memory in X-Functions module
[not used]		Non-existing memory
0x000 – 0x00F	HP41 Status registers	User stack, ALPHA, flags, return stack, SIZE info, OS use

A part of the User Memory is Extended Memory, managed by the Extended Functions ROM (built-in in the HP41CX). Direct access to this memory in mcode is done using the RAMSLCT (DADD=C) instruction to select a block of 16 registers, and with the READ and WRIT instructions. All access is always using all 56 bits, and the address is 10 bits wide, limiting the number of registers to 1024 (hex 400). The HP41 memory map contains a number of gaps (non-existent registers) which are vital for the correct operation of the HP41.

With the introduction of the HP41CL Monte Dalrymple defined Expanded Memory. This is User Memory beyond the original User Memory by using a 12-bit address and a new instruction EADD=C (0x0C0) to select this memory. With this method there are now 4096 registers available. Please refer to the HP41CL and MAXX documentation for further details. This type of memory is not yet supported in the TULIP firmware.

Depending on the type of HP41 you may have several types of User Memory already available in the calculator, the table below indicates what you would need extra to get a full configuration.

The current firmware only supports Extended Memory with the **xmem** command. You will need to use an HP41CX or a C/CV with the physical Extended Functions module.

xmem	shows the current status
xmem [n]	<i>n=0,1,2 Plugs 0, 1 or 2 Extended memory modules</i>

HP41 type	Built-in	Possible with TULIP
HP41C	0x000 - 0x0FF Only basic memory	1..4 single Memory modules Quad Memory Module Extended Functions + 2* Extended Memory Expanded Memory
HP41CV	0x000 - 0x0FF 0x100 - 0x1FF Basic memory+ Quad Memory	Extended Functions + 2* Extended Memory Expanded Memory
HP41CX	0x000 - 0xCFF 0x100 - 0x1FF 0x040 - 0x0BF Basic memory + Quad Memory Extended Functions + Basic Extended Memory	2* Extended Memory Expanded Memory
HP41CL	Everything including Expanded Memory	Lucky you, you have everything already
HP41 (any model) with MAXX	Everything including Expanded Memory	Don't bother, it's all in the MAXX

[NOT YET SUPPORTED]: paragraphs below about the *umem* command

If you already have an Extended Functions Module (which has the basic Extended Memory built in) you can happily keep it in your calculator. A mix of Memory Modules or Extended Memory modules in combination with the TULIP emulated User memory is not supported. This is good news as it will free up ports. In an HP41C or CV it is advised to remove the Extended Functions Module and use the TULIP emulation for this.

Expanded Memory can only be used with special functions. These are available for the MAXX module and the HP41CL, but not yet for the TULIP. Use of it is optional.

How to plug each type of memory depends on the configuration you have already. The command *umem* is used for most configurations. If you do not have the Extended Functions module (in case of an HP41C or CV) this must be plugged first with the *plug* command, just like any ROM image. When plugging the Extended Functions module as a .MOD file it will enable Extended Memory if the parameter XMemModules (in the MOD file metadata) is used. This parameter can be set (with a MOD file editor) to 0, 1, 2 or 3.

If you have a physical XFunctions module or an HP41CX, do not forget to use the *plug module* command to reserve the Page. Keep in mind that this command only reserves the Page and does NOT reserve the X-Memory space!

<i>umem <status></i>	shows the current status
<i>umem mem <0..4></i>	to plug a Memory modules (if you have an HP41C), using 4 is the equivalent of a Quad Memory module
<i>umem xfun <on/off></i>	Use the <i>on</i> option if you have an HP41CX or a physical plugged Extended Functions module and no Extended Memory modules plugged. This allows the correct installation of Extended Memory modules. Do not plug any physical Extended Memory modules
<i>umem xmem <0..3></i>	plug Extended Memory modules in an HP41CX or if you have a physical Extended Functions module. Will correctly plug the number of Extended Memory modules and the Extended Memory built in the Extended Functions module

umem xpmem <clear> enable all Expanded memory, use clear to disable it

umem <ext/mem/exp> dump shows all non-zero registers in the indicated memory type that are in the TULIP memory. Memory inside the HP41 cannot be shown

All types of the above mentioned User Memory is stored in files in FRAM, these files are created when the FRAM file system is initialized. This file always has the maximum size (always full Extended Memory for example). When starting the TULIP a copy of the file is made in main RAM to increase the speed of READ operations for the emulation layer. Write operations are immediately synchronized to FRAM and all these memory types will survive a power cycle.

[NOT YET SUPPORTED] File types in FRAM:

.UMM User Memory copy/backup
 .EXT Extended memory copy/backup
 .EXP Expanded memory copy/backup

Summarizing for your situation:

HP41CL or HP41 (any model) with MAXX: there is really nothing to do, the CL or calculator with MAXX is fully loaded with all available Extended and Expanded memory

HP41CX: Extended Functions and basic Extended Memory is built in. Remove any existing Extended Memory modules and use ***umem xmem 2*** to get the maximum possible amount of Extended Memory

HP41C/CV with a physical Extended Functions: treat like an HP41CX

HP41C/CV: use *plug EXT-FUNS.MOD*. Use ***umem status*** to find out how much Extended Memory you now have and use ***umem xmem 3*** to get the maximum amount of Extended Memory

HP41C: remove all memory modules (single or quad) and use ***umem mem 4*** to get the maximum amount of memory

HP41 type	CLI command	Result
HP41C	<i>umem mem 4</i>	Plug a Quad Memory module
HP41C + X-Functions module	<i>umem mem 4</i> <i>plug module p8</i> <i>umem xfun on</i> <i>umem xmem 2</i>	Plug a Quad Memory module Reserve the Page, assuming the Xfunctions module is plugged in Port 1 (Page 8) Announce a physical X-Functions module Add 2 Extended Memory Modules
HP41CV No modules	<i>plug ext-funs.mod</i> <i>umem xfun off</i> <i>umem xmem 3</i>	Plug Extended Functions module No physical X-Functions module (default anyway) Add X-Functions memory and 2 XMem modules
HP41CX	<i>umem xfun on</i> <i>umem status</i> <i>umem xmem 2</i>	Announce a physical X-Functions module Check how much memory is included Add 2 Extended Memory Modules if needed
HP41CL	nothing to do	All memory already built in the HP41CL
HP41 with MAXX	nothing to do	All memory already built in the HP41 and in the MAXX

For fun or testing the amount of memory modules can be reduced or increased, but you may lose data or lock up your calculator.

11.9. [Using the mcode tracer](#)

The mcode tracer is enabled whenever its serial port is connected. By default all samples are passed by the core1 critical loop to the queue, and the core0 software reads from the queue to display the samples. When the queue is full the core1 software will simply skip the sample, and the core0 software will miss a sample and indicate this with an O (for Overflow).

The tracer has a default size of 5000 samples. This can be changed using the ***tracer buffer*** command. A reboot of the TULIP is necessary for the new buffer size to become effective.

[filters not yet implemented]

The tracer has a feature to filter samples, and uses an array of all possible addresses in the HP41 ROM Memory space to quickly decide what to do with a sample. Although this is memory intensive it reduces the load on the code. Two bits per address indicate one of the following options:

- Pass: Pass the sample at the given address. Default is to pass all samples.
- Block: samples at this address are blocked and not shown in the trace listing
- Trigger Start: samples that match a trigger address are marked in the listing, and traces are listed starting at this address. Traces are shown until the Trigger End address appears or until a given number of samples are shown. The contents of the pre-trigger buffer are shown before the Trigger Start sample
- Trigger End: Sampling stops after the Trigger End address or earlier if a number of samples is given

When using Pass or Block the order of the commands is important. For example to view only samples with a certain address range (within your own ROM for example) the first command must Block everything, and a later command can then pass a range:

tracer block all block all samples

tracer pass p9 pass all samples in Page 9

or

tracer pass 9400 9600 to pass all samples between 0x9400 and 0x9600

A subsequent block command can be used to block a range within the range of passed addresses.

A number of standard addresses can be set automatically:

tracer block sysloop

tracer block illoop

tracer block sysrom blocks all samples in Page 0, 1, 2, 3, 5 (NOT Page 4!)

tracer block ilrom blocks all samples in Page 6 + 7

A complete Page can be blocked or passed by using p followed by the Page number.

Limitations:

- Pass, block and triggers apply to all banks of that Page
- Pass, block and triggers apply to both instruction fetch (SYNC high) and data fetch (SYNC LOW)

The tracer allows to set the size of the main trace buffer and the pre-trigger buffer. The default values are 5000 for the main tracer buffer, and 32 for the pre-trigger buffer. Keep in mind that the tracer is memory intensive, and one trace sample takes 52 bytes, this means that the default trace buffer of 5000 samples uses 253 Kbyte. And total available memory is 520 Kbyte. When the size of the main trace buffer is changed a restart of the TULIP firmware is needed to ensure that the available RAM memory is not fragmented. The maximum size of the main trace buffer is 8000 samples. The size of the pre-trigger buffer can be changed at will, maximum size of this buffer is 256 samples.

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

12. [TULIP4041 Firmware programming or update](#)

Updating the TULIP Firmware is relatively easy, just follow the steps below. When using the DevBoard and you are using a new Pico2 this board will always start up in the so-called BOOTSEL mode. When connecting to a host compute in BOOTSEL mode you will see a drive with the name RP2350. Simply copy the firmware file (the file with extension .uf2) to this drive and the ULIP processor will restart with the new firmware.

BE CAREFUL: The DevBoard and the Module use different firmware files!

ADVICE: Do the firmware upgrade with your HP41 disconnected!

OOPS: You may lose some of your TULIP settings (although unlikely) after a firmware upgrade

UNPLUG: al plugged Embedded ROM's (HP-IL, Printer) MUST be unplugged prior to a firmware upgrade

To enter BOOTSEL mode on the DevBoard:

- If no previous TULIP firmware was loaded you may use the PicoTool, or push the little button while plugging the USB cable. please refer to the Pico documentation for more information.
- If there is TULIP firmware loaded simply use the CLI command ***system BOOTSEL***

To enter BOOTSEL mode on the TULIP Module:

- Simply use the CLI command ***system BOOTSEL***, firmware is already loaded on the TULIP module
- In rare cases when the firmware is locked a bit of fiddling is required. Power cycle the TULIP and try again. If that does not work remove the TULIP from the calculator and disconnect the USB cable. Open the TULIP housing and locate the USB_BOOT pad (pin 6 on the large set of connector pads). Use a short piece or wire to connect this pad with GROUND (pad 7, next to pad 6). Now connect the USB cable again while keeping USB_BOOT and GROUND shorted. Release the connection between USB_BOOT and GROUND when the USB connection is made and the TULIP will be in BOOTSEL mode

Now you can copy the firmware file to the RP2350 drive from the host computer. Remember to use the correct firmware file, there are different version for the DevBoard and the TULIP Module

Once the TULIP4041 firmware is loaded it is not possible to use Picotool to get the processor in BOOTSEL mode.

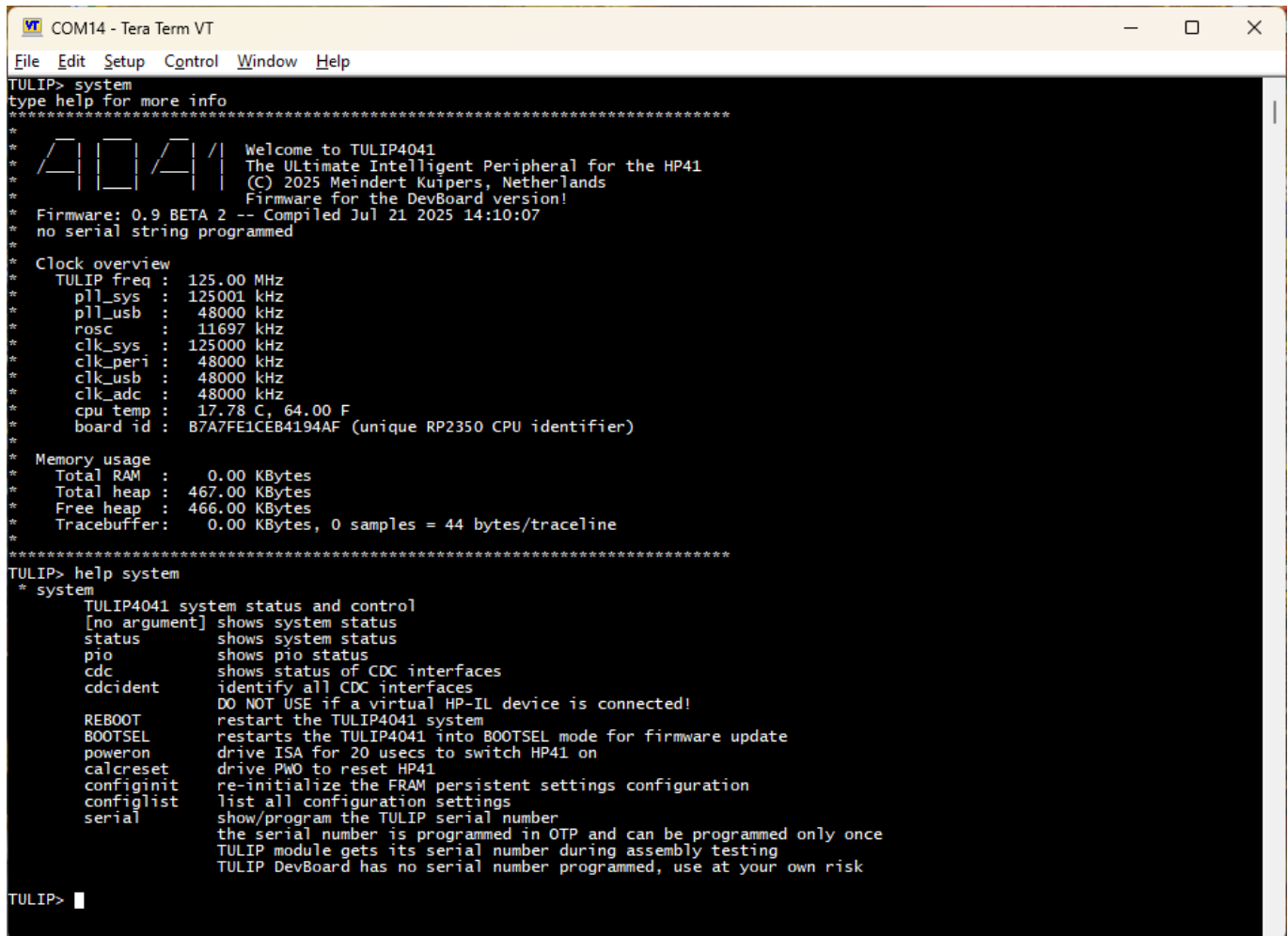
13. [TULIP4041 Quick Start](#)

The instructions below will guide you through the steps to get up and running with the TULIP4041 quickly.

1. Format a micro SD card (minimum 2 GByte) with the exFAT (recommended) or FAT file system
2. Create a repository in a subdirectory on the card of ROM and MOD files that you wish to use in your TULIP (can also be done after step 3). Let's call this myROMrepo
3. Plug the microSD card in your TULIP and connect it with your host PC, verify if the card is visible as a removable drive in your host computer
4. Connect a Terminal Emulator with the TULIP Command Line Interface. Try the newly discovered serial ports until you see the TULIP welcome message. Use the ***sdcard connect*** command in the CLI to connect with the host if the USB drive was not visible. When using the FAT file system the detection by the host can take some time, even up to several minutes when a large capacity SC card is used.
5. Update to the latest firmware (see the instructions in the previous chapter) if needed
6. Type the command ***dir myromrepo*** (do not care about the case of the subdirectory name) and verify if all your files are visible
7. Type the ***list*** command to check which ROMs are already in FLASH. If this is the first time the TULIP is used it will contain the ROM files used for production testing . Go to step 12 in case the list is garbled or if no files are visible at all. If the system is newly initialized it will show only the "TULIP4041 FLASH HEADER" file of type 41.
8. Type the command ***import myromrepo ALL*** (ALL must be uppercase!). This will copy your ROM repository to FLASH memory which is needed for the ROMs to be pluggable.
9. Use ***list*** again to verify if all your roms are imported
10. Connect the TULIP with your calculator. While doing that check which ports are physically occupied
You can now use the plug command. Say that you have imported the set of PPC roms, PPCL.rom and PPCU.rom. Assuming that you have Port 4 free (this is Page E and F) you can type (the case of the ROM name and the Page do not matter)
TULIP> ***plug PPCL.rom E***
TULIP> ***plug PPCU.rom F***
11. Use ***cat*** in the TULIP CLI to confirm that the ROMs are plugged and enter CAT 2 on the HP41 to confirm that the PPC ROM is now plugged in the calculator. The configuration is saved in FRAM and will be available again after a power cycle
12. In case the file system is corrupted you need to fully initialize the FLASH File System with the following steps:
TULIP> ***flash NUKEALL***
TULIP> ***flash INIT***
And go back to step 7

14. [TULIP4041 Command Line Interface reference \[TO BE UPDATED\]](#)

The TULIP4041 Command Line Interface (CLI) is accessible using the first (usually lowest numbered) virtual serial port. Upon a reboot or reconnect of the unit the CLI will show a welcome message with system information and a prompt. Commands can be issued at the prompt. The CLI buffers the commands given which can be recalled and edited. This paragraph gives an overview of the available commands. Details on plugging and unplugging ROM images and other features relevant to HP41 device emulation and tracing are given in the next paragraphs. Many commands have parameters, these are explained with the command. Commands are case sensitive, most commands can be entered in lower case, some critical commands or parameters have to be given in uppercase.



```

COM14 - Tera Term VT
File Edit Setup Control Window Help
TULIP> system
type help for more info
*****
*  A041 Welcome to TULIP4041
*      The ULtimate Intelligent Peripheral for the HP41
*      (C) 2025 Meindert Kuipers, Netherlands
*      Firmware for the DevBoard version!
*      Firmware: 0.9 BETA 2 -- Compiled Jul 21 2025 14:10:07
*      no serial string programmed
*
* Clock overview
*   TULIP freq : 125.00 MHz
*   pll_sys   : 125001 kHz
*   pll_usb   : 48000 kHz
*   rosc      : 11697 kHz
*   clk_sys   : 125000 kHz
*   clk_peri  : 48000 kHz
*   clk_usb   : 48000 kHz
*   clk_adc   : 48000 kHz
*   cpu temp  : 17.78 C, 64.00 F
*   board id  : B7A7FE1CEB4194AF (unique RP2350 CPU identifier)
*
* Memory usage
*   Total RAM : 0.00 KBytes
*   Total heap: 467.00 KBytes
*   Free heap : 466.00 KBytes
*   Tracebuffer: 0.00 KBytes, 0 samples = 44 bytes/traceline
*****
TULIP> help system
* system
  TULIP4041 system status and control
  [no argument] shows system status
  status        shows system status
  pio           shows pio status
  cdc           shows status of CDC interfaces
  cdcident      identify all CDC interfaces
                DO NOT USE if a virtual HP-IL device is connected!
  REBOOT       restart the TULIP4041 system
  BOOTSEL      restarts the TULIP4041 into BOOTSEL mode for firmware update
  poweron      drive ISA for 20 usecs to switch HP41 on
  calcreset    drive Pw0 to reset HP41
  configinit   re-initialize the FRAM persistent settings configuration
  configlist   list all configuration settings
  serial       show/program the TULIP serial number
                the serial number is programmed in OTP and can be programmed only once
                TULIP module gets its serial number during assembly testing
                TULIP DevBoard has no serial number programmed, use at your own risk
TULIP>

```

The description below applies to TULIP4041 firmware version 0.9 BETA 2. Functions indicated with a * are planned to be implemented in a later version. It may be possible to type the command, but no action will be taken.

Parameters in <> are optional, parameters in (square brackets) must be given.

help <command>

Shows all available commands with a short explanation. Shows the help for a specific command when a parameter is typed, for example *help system*

system <argument>**<no argument>**

Shows the welcome message with the system status, including memory usage

status

Shows the welcome message (same as the command system without a parameter).

pio

Shows the results of the PIO state machine initialization

cdc

Shows the status of the virtual serial ports connected to the USB port

cdcident

Sends a message to each connected virtual serial port to facilitate identification of the connected serial ports. Use only when a terminal emulator is connected to the ports. When a virtual IL device is connected this command may confuse the virtual device.

REBOOT

Resets the TULIP4041 unit. Note that a reboot can take a few seconds before the TULIP4041 is able to respond to the HP41 and before the CLI can be accessed. All virtual serial ports are disconnected, automatic reconnect depends on the host computer and the terminal emulator used. Only works when the HP41 is not running (HP41 PWO line is low) or no calculator is connected

BOOTSEL

Stops the TULIP4041 firmware and enters RP2350 BOOTSEL mode for example for a firmware upgrade. Only works when the HP41 is not running (HP41 PWO line is low) or no calculator is connected

poweron

Drives ISA for 20 microseconds to switch on the calculator. Works only when PWO is low. Driving ISA causes the HP41 to wakeup and execute the IO Service requests. If there is no active request the calculator will go to STANDBY mode again, this is typically not visible in the calculator display. The tracer will show the instructions executed. This command is equivalent to pushing the button on a Wand for example, except there is no interrupt routine executed

calcretset

Drives PWO for 10 microseconds to halt the calculator. Works only when PWO is high. This will immediately halt all running code in the calculator, and can be used to recover from an mcode loop or other kind of lock-up. Test this by starting a CAT 3 for example and then type *system calcretset* in the CLI while the catalog is running. The CAT listing will immediately halt. Back in the good old days this function was implemented in a small module device or as an addition to an MLDL. Very useful to recover from experimental mcode programming, it was therefore also known as a crash killer. Executing this command may upset the tracer as the synchronization may be lost.

configinit

Initialized the persistent system configuration in FRAM to its default values. Use only in case the configuration seems to be corrupted.

configlist

Shows the persistent system configuration. A bit cryptical, use only for support purposes.

serial (TULIP module only)

show/program the TULIP serial number. The serial number is programmed in processors' OTP and can be programmed only once, and is done during production testing of the module version.

clear

Clears the CLI console window, typically does not clear the complete scroll buffer. Use a command in your terminal emulator to do that.

**blink **

Blinks the LED on the Pico board. Blinks b times, when no value is given will blink 5 times. B must be between 1 and 9. Use a value of 0 to toggle the LED status

dir <subdir>

Shows a directory listing of the microSD card root directory or subdirectory (when given).

sdcard <parameter>

<without parameter>

Shows the uSD card status report and will mount the card

status

Shows the uSD card status report and will mount the card

mount

Mounts the uSD card and prepares for use. The uSD card is automatically mounted upon start of the TULIP. Use only when plugging a uSD card in a powered system

unmount

unmounts the uSD card and disables the use of it

mounted

Check if the uSD card is mounted

connect

enables use of the uSD card as a USB drive on the host computer when it has a valid USB connection to a host PC. Upon start of the TULIP the uSD card is mounted and connected

eject

disables the use of the uSD card as a USB drive from the host PC. The card is still available to the TULIP until unmounted. Use only after ejecting the TULIP drive on the host computer to prevent corruption of the cards filesystem.

plug <argument>

Plug a ROM image (virtually) in the HP41. Only .ROM files are supported

hpil

plugs the HP-IL module in Page 7

ilprinter

plugs the HP-IL printer module in Page 6. Works only when the HP-IL module is plugged. The HP82143A printer module will be unplugged if it was plugged before.

printer

plugs the HP82143A printer module in Page 6, unplugs the HP-IL printer if it was plugged.

filename.rom P

plugs the file "filename.rom" in Page P (P in hex) if that Page is available. The file is always plugged in Bank 1

filename.rom P B

plugs the file "filename.rom" in Page P (P in hex) and Bank B (B=1..4) if that Page and Bank is available

filename.rom <T>

By omitting the Page will plug the file "filename.rom" in the first available Page (always Bank 1) starting at Page 8. When a T (for Test) is added the selected Page will be reported but the file will not be plugged

unplug <argument>

Removes a ROM image from the HP41 (only images plugged in the TULIP4041)

P

Unplug the ROM in Page P (P in hex), does not apply to reserved Pages

P B

Unplug the ROM in Page P (P in hex) and Bank B

all

Unplug all plugged ROMs, except reserved Pages

ALL

Unplug all plugged ROMs including all reserved Pages

reserve <argument>

Reserve a Page for a physical module to assist the Plug function

P <comment>

Reserve Page P (P in hex), use the comment for your own use. Comments must be enclosed in parenthesis if it contains spaces

CX

Reserve Page 3 and 5 for the CX ROMS (Timer)

timer

Reserve Page 5 for the TIME module

printer

Reserve Page 6 for a Printer (HP82143, IL Printer or IR Printer)

hpil

Reserve Page 7 for the HP-IL module

clear [P]

Cancel the reservation for Page P (in hex)

clear all

Cancel all reservations

cat <argument>

Show the plugged and reserved ROM images

<no argument>

Show a listing of all plugged and reserved ROM images

**P **

Show the details of the ROM image (with a hex dump) plugged in the given Page, optionally the Bank can be given. Page number must be in hex

emulate <argument>

Enable or disables emulated hardware, toggles the emulation status

<no argument>

Show the status of the emulated hardware

status

Show the status of the emulated hardware

hpil

toggle HPIL hardware emulation, use when plugging the HP-IL module as a .ROM file instead of the embedded ROM

printer

toggle HP82143A printer emulation

zeprom P

toggle ZEPROM emulation in Page P (hex) for sticky bankswitching, P is the Page number in hex (0..F). This will apply for both the odd and even Page in a Port. The ZEPROM itself does not need to be plugged

printer <argument>

Controls the HP82143A printer emulation. When this ROM is not (virtually) plugged, most actions will be stored and become effective when the ROM is plugged. Most status settings (power, printmode and paper) are saved on powerdown.

<no argument>

Shows the printer status including the print mode and details of the status/control register

status

shows the printer status including the print mode and details of the status/control register

power

toggles the printer power mode. The printer must be ON for any printing operations. By default the printer is OFF (like the real one). Once switched on the printer will stay on

trace

puts the printer into TRACE mode, like moving the slider on the real printer

norm

puts the printer into NORMAL mode, like moving the slider on the real printer

man

puts the printer into MANUAL mode, like moving the slider on the real printer

paper

toggles the printer Out Of Paper status. Default is no paper, like in real life you must put paper in for the printer to work.

print

does a virtual push of the PRINT button. Prints X or ALPHA depending on the calculator mode, or enters PRINT (or PRA) in a program when in program mode.

adv

does a virtual push of the ADV button. Virtually advances paper in the connected simulator or enters ADV in a program when in program mode

irtest

Test the infrared LED with an IR receiver. Sends a simple string (without toggling). This is not a printer test string

xmem <argument>

Controls Extended Memory emulation. Without argument will show the current status(number of modules plugged). The Extended Functions (and its built in Memory) is not emulated. This must be physically plugged or available in the HP41CX. Do not use with an HP41CL, as it has its own Extended Memory emulation. Ensure to set the number of modules to 0 when using the HP41CL in combination with the TULIP4041.

<no argument>

shows the number of modules plugged (0, 1 or 2)

status

shows the number of modules plugged (0, 1 or 2)

dump

creates a listing of the Extended Memory contents. Only non-zero registers are shown. Use for your own interest and to verify the pattern for testing FRAM. The expected pattern for the FRAM test is below:

```
REG 200 = 0x4142414241424142
REG 201 = 0x4243424342434243
REG 202 = 0x4344434443444344
REG 203 = 0x4445444544454445
REG 204 = 0x4546454645464546
REG 205 = 0x4647464746474647
REG 206 = 0x4748474847484748
REG 207 = 0x4849484948494849
REG 208 = 0x494A494A494A494A
REG 209 = 0x4A4B4A4B4A4B4A4B
REG 20A = 0x4B4C4B4C4B4C4B4C
REG 20B = 0x4C4D4C4D4C4D4C4D
REG 20C = 0x4D4E4D4E4D4E4D4E
REG 20D = 0x4E4F4E4F4E4F4E4F
REG 20E = 0x4F504F504F504F50
```

0, 1 or 2

plugs the given number of Extended Memory modules. 0 will unplug any modules

PATTERN

fills the Extended Memory with a test pattern for testing the FRAM. Will overwrite any existing information, verify with *xmem dump*

ERASE

clears all emulated Extended Memory to 0's

tracer <argument>

Controls the HP41 bus tracer. Without argument will toggle tracer on/off. Commands may be given while the tracer is running. For more detailed information see the chapter about the HP41 Tracer. Pressing a key in the tracer window will toggle the tracer activity. Note that that will most likely lead to an overflow when resuming tracing again. Note that the tracer settings are not automatically saved, this must be done with the *tracer save* command. This is done to allow control of the tracer while the calculator is running, and a running calculator prevents access to FRAM, where the settings are saved.

<no argument>

shows tracer status

status

shows the tracer status

buffer <size>

shows the size of the main tracer buffer in number of samples. The size can be adjusted by giving the required number of samples, and a subsequent reboot is needed for the change to become effective. The size is maximum 10.000 samples

pretrig <size>

shows the size of the pre-trigger buffer (not in use yet) and allows the size of the pre-trigger buffer to be defined. Default is 32 samples, maximum is 256 samples

trace

toggle trace enable/disable

sysloop

toggle tracing of system loops:

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

These loops are regularly called for a delay or when waiting for a key time-out. When system loop tracing is disabled, the above address will be skipped in the trace output and will usually allow much longer traces without overflow

sysrom

toggle tracing of system ROMs in Pages 0 to 5. Useful when you are interested only in tracing your own ROM images

ilrom

toggle tracing of the HP-IL and Printer ROM in pages 6 and 7

hpil

toggle tracing of the HP-IL frames to the ILSCOPE virtual serial port

pilbox

toggle tracing of the PILBox serial frames to the ILSCOPE virtual serial port

ilregs

toggle tracing of the HP-IL frames and registers to the trace window. Disabling (if you are not interested in this) reduced the load on the tracer software

save

save the tracer settings in the FRAM persistent storage. This is done to allow control of the tracer while the calculator is running, and a running calculator prevents access to FRAM, where the settings are saved.

flash <argument>

Functions for managing FLASH memory.

<no argument>

shows the FLASH chip ID and size

status

shows the FLASH chip ID and size

dump <ADDR>

shows a hex dump of the FLASH memory contents for the FLASH File System. Always shows 4K bytes. Use a hex address (offset in the Flash File System) for the start address. Subsequent *flash dump* commands without an address will continue the dump

INIT

initializes the Flash File System by creating the header file and start of the chain, FLASH must be fully erased first

NUKEALL

fully erases the complete Flash File System, only the space reserved for this, never the Flash memory used for the firmware

fram <argument>

Functions for managing FRAM memory.

<no argument>

shows the FRAM chip ID and size

status

shows the FRAM chip ID and size

dump <ADDR>

shows a hex dump of the FRAM memory contents for the FRAM File System. Always shows 4K bytes. Use a hex address for the start address. Subsequent *fram dump* commands without an address will continue the dump

INIT

initializes the FRAM File System, for now limited to the portion of the ROM map

NUKEALL

fully erases the FRAM to all zero's

import <argument>

Import files from the uSD card into FLASH. Only supported file types can be imported, this is checked by the file extension. File name can be maximum 31 characters (including dot and extension). The filename must include any subdirectories (subdirectory is not counted in the filename length)

<no argument>

not valid

<filename>

import a single file in FLASH

<directory> ALL

import all files in a directory on the uSD card into FLASH. Only supported filetypes will be imported, existing files will be skipped. Files will be skipped if there is not enough room in the files system

[import functions below not yet supported]**<filename> FRAM****[NOT YET SUPPORTED]**

import a single file in FRAM

<filename> compare <FRAM>**[NOT YET SUPPORTED]**

compare a single file with the one in FLASH (default) or FRAM. Will tell if the files are identical or not and if the file (if in FLASH) can be updated without prior erasing

<filename> UPDATE <FRAM>**[NOT YET SUPPORTED]**

update a single file with the one in FLASH (default) or FRAM. For a successful update the following must apply:

- The file name is identical
- The file size is identical
- The contents are different
- There is no need to first unplug the ROM image if it was plugged if:
 - The exact same bank assignment is used in case of a multi-bank module
 - The exact same page order is used in case of a multi-page image
 - There are no changes in the hardware support of a MOD file if it is plugged

<directory> ALL compare**[NOT YET SUPPORTED]**

compare all files in a directory on the uSD card with the ones in FLASH. Can be used to find out which files need updating

<directory> ALL UPDATE**[NOT YET SUPPORTED]**

update all files in a directory on the uSD card with the ones in FLASH. The same conditions apply as for updating a single file

delete <filename>

Deletes a file from FLASH memory, and frees the space for new files. The file is marked for deletion only but will not show up with the *list* command

list <argument>

Shows files and details of the files in the Flash File System.

<no argument>

Shows a listing of all files with filename, type, size, offset of the file in the Flash File System and the ioffset of the next file.

ext

show an extended listing of all files including some details of the file content

all

shows all files including deleted files

<filename>

Shows an extended listing of a single file with details of the contents. A partial name can be entered, and then the first matching file will be shown.

rtc <argument>

(Module version only) Used for testing the PCF8523 RTC and backup battery

<no argument>

Shows the RTC status and scans the I2C bus for any other connected devices

status

Shows the RTC status and scans the I2C bus for any other connected devices

set YYMMDDHHMMSS

Set the RTC to the given date and time

get YYMMDDHHMMSS

Get the current date and time from the RTC

reset

Reset the RTC

dump

Dump all RTC registers

15. [Limitations](#)

- HP-IL and HP-IL printer emulation with PILBox emulation
 - RFC and CMD frames are not processed in the same way as in the hardware PILBox. This has no functional impact, the ILSCOPE window in the virtual HP-IL device on the host will be slightly different with too many RFC frames

16. [TULIP4041 Hardware: Module version](#)

The TULIP4041 module is a custom board that will fit in an HP41 long sized module (like the HP41 IL module). Its features are nearly identical to the DevBoard version, with a few different pin assignments. Added to the Module version is an RTC with battery backup for emulation of the TIME module). The only limitation of the Module version is that it cannot trace the FI signal from the HP41. The FI signal shown is limited to the FI signal being driven by the TULIP4041.

The TULIP module has 4 Mbyte of FLASH memory in an SOIC-8 package that is hand solderable. It can be replaced by a larger sized device if needed (requires recompilation of the firmware). A number of signals are available on holes for a 2mm header.

The TULIP4041 Module is fully assembled and tested and contains the TULIP firmware. It comes with a 3D printed housing. Depending on the chosen option it comes with or without a module connector.

TULIP module features:

- Comes with a 3D printed long module housing that is easy to open and close
- 4-layer PCB
- USB-C connector
- IO signals and CPU debug signals available on 2mm holes (no header provided)
- PCF8523 RTC (no battery provided) with backup battery carrier
- 4 Mbyte of FLASH and 256 Kbyte FRAM
- Micro SD card holder (no memory card provided)
- Signaling LED
- Infrared LED

16.1. [The TULIP housing](#)

The TULIP housing is very easy to open and close. Simply take the housing with your thumbs on the connector side and gently push the two halves apart. To close fit the two halves together and gently push the two halves until they snap in place. The TULIP board sits loose in the housing, take care that it does not fall off.

To insert or remove the micro SD card it may be easier to open the housing.

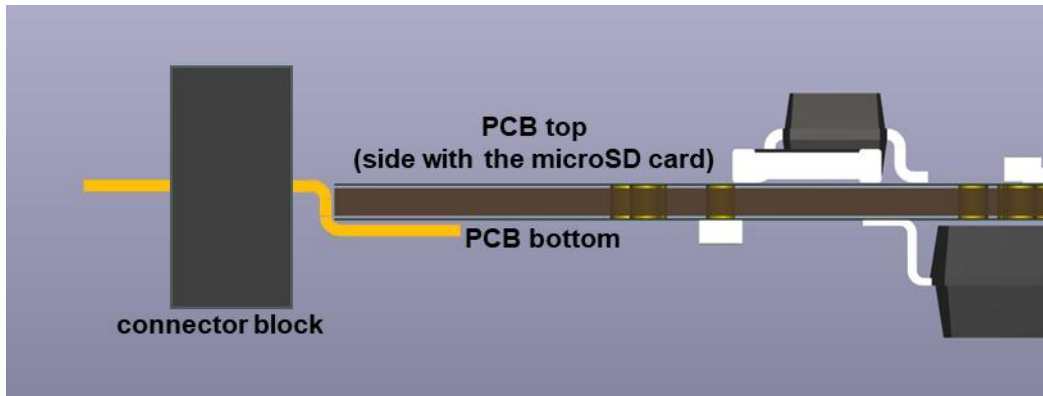
16.2. [Soldering the connector](#)

If you have ordered the TULIP module without connector, you must solder one yourself. Please follow the next steps. The connector solder pads are on both sides of the TULIP PCB. This is to allow easier testing in production and to prepare for a possible new connector solution. When using an connector harvested from a real module you must be aware that this must be soldered **on the bottom side of the PCB!**

Before attempting to solder the connector ensure that the TULIP Module is functional and that you can access the CLI, microSD Card, FRAM and FLASH memory, just to be certain. When you have the TULIP module without connector you need to harvest one from an older module. Be aware that the standard module connector is soldered on the **bottom** side of the TULIP PCB (the side with most components). It is important that the connector is well aligned and straight with the PCB. To achieve than best is to solder only one pin, and check alignment, reheat and position that pin as needed until the alignment is good. Use one or two pieces of cardboard, cut in narrow strips (thickness of about 1.6 mm) to support the PCB under the connector

pads. This is the ideal height. I find it easier to fix the PCB and connector with a bit of tape to prevent movement while soldering.

- Unsolder a connector from an original module, remove any remaining solder
- Remove the uSD card from the TULIP, just in case
- The HP41 connector is mounted on the BOTTOM side of the module, and that is the side where the level shifters are. The TOP is the side where the micro SD card holder is. See the picture below:



- Push the connector towards the PCB such that the edges of the contact touch the PCB. See the pictures below for the procedure.
- Carefully solder one pin of the connector and verify alignment, you could put the combination in the bottom housing to check. If you are satisfied then solder the other pins. Visually verify the soldering joints and check if the TULIP and connector fits in the housing and all pins are aligned.

First steps to test the connector:

- Ensure that there are no virtual ROM images plugged in the TULIP and remove the uSD card
- Remove the battery from your calculator and remove the USB cable from the TULIP
- Carefully insert the TULIP in your calculator, do not force it in. It should slide in the port and connect like a normal module. If you sense that more force is needed for insertion then try another port and visually check the pins in the module
- Now insert a battery in the calculator and check if the calculator is running normally. If you can measure the power consumption check if the current does not exceed 20-25 mA
- You may now insert a uSD card in the TULIP and connect with a host computer using USB

16.3. [Preparing for first use](#)

Before the first use of the TULIP module you must consider the following:

- To load virtual modules you must use a micro SD card unless you only want to use HP-IL or the Printer. Please follow the instructions to prepare a micro SD card below
- You will need a USB data cable with a USB-C connection on the TULIP end. The other end obviously should match your host computer
- Did you order the TULIP with an HP41 connector? If yes you can skip the paragraph about that
- Do you intend to use the RTC with a backup battery? If yes follow the instructions in one of the next paragraphs. Please note that the first software release does not support emulation of the TIME module and the RTC cannot be used from the HP41

- In case you plan to use the IO breakout or debug breakout please observe the pinout. The holes for both debug and IO have a 2mm pitch

The TULIP Module is fully tested before shipping. There is no need to test it like the Devboard.

16.4. [The micro SD card](#)

Prepare a micro SD card formatted with the exFAT (preferred) or FAT file system on your host computer. Any size larger than 2 GByte will normally work fine. Sizes under 2 GByte will most likely not work as the firmware does not support standard density cards. Keep in mind that high-speed cards may consume more power, and the interface to the micro SD card is not built for speed.

Create a subdirectory and copy your favorite ROM images as ROM and/or MOD files here. In the examples we will use a subdirectory *myROMrepo*. Also create a RAW directory for later use. The files may also be copied when the card is in the TULIP module.

- Remove (in a safe way) the micro SD card from the host computer and plug it in the slot of the TULIP module. The Card Detect signal is not used
- When powered and connected to a host PC with USB the SD card will be automatically recognized like a normal USB thumb drive. In some cases it may take several seconds until the card is recognized
- Using the user interface, verify with the command ***sdcard*** if the card is properly mounted by the firmware
- Unplugging the SD card in the TULIP is best be done when the module is powered down, or after unmounting and ejecting the card. The same applies for plugging a card, after plugging use the *mount* and *connect* commands

16.5. [Other tests](#)

Further testing is not really needed, but some tests could be useful.

- Test the IR LED: command ***printer irtest*** to send a test string to the IR LED
- Test the signalling LED: command ***blink 8*** to blink the LED 8 times
- Confirm presence of all serial ports on the host computer
- Verify FRAM operation (see the test of the DevBoard)
- Verify the RTC and backup battery operation with the ***rtc*** commands

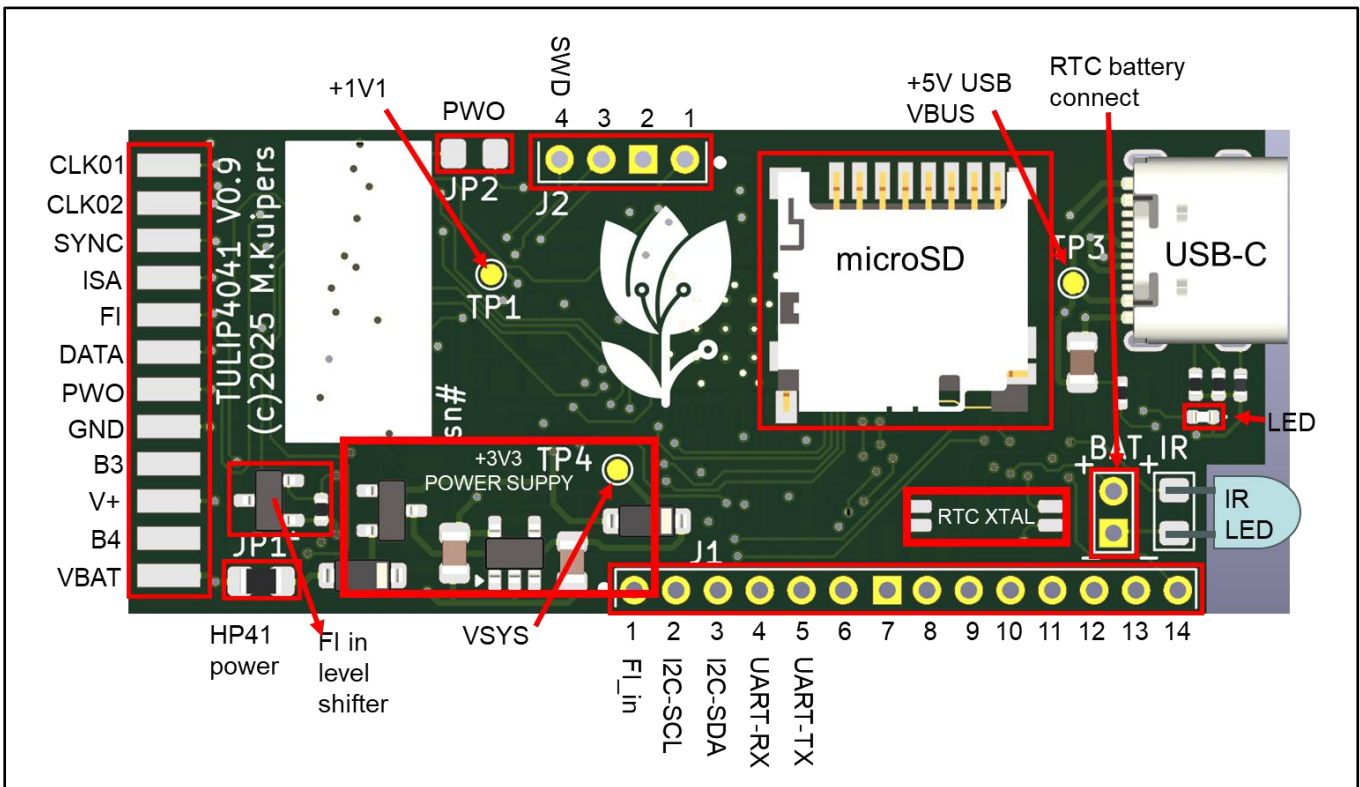
Prepare a micro SD card formatted with the exFAT (preferred) or FAT file system on your host computer. Any size larger than 2 GByte will normally work fine. Sizes under 2 GByte will most likely not work as the firmware does not support standard density cards. Keep in mind that high-speed cards may consume more power, and the interface to the micro SD card is not built for speed.

16.6. [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



PCB TOP AND I/O

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 - TO_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	

17. TULIP4041 Hardware: Development Board

The initial hardware for TULIP4041 is the TULIP-DevBoard, a development board for a commercial Pico module. On this board all signals are available on pins for testing and measurement. A separate module connector board is used for the physical interface to the HP41. There are some minor functional differences between the DevBoard and the module version. The DevBoard is intended to be assembled by the user. The DevBoard is designed for the original Raspberry Pico2 board. Please do not use the Pico-W board. Any other 100% pin-compatible alternative to the Pico (Pimorini for example) should work just fine but may require recompilation of the firmware to adjust for additional features (such as extra memory).

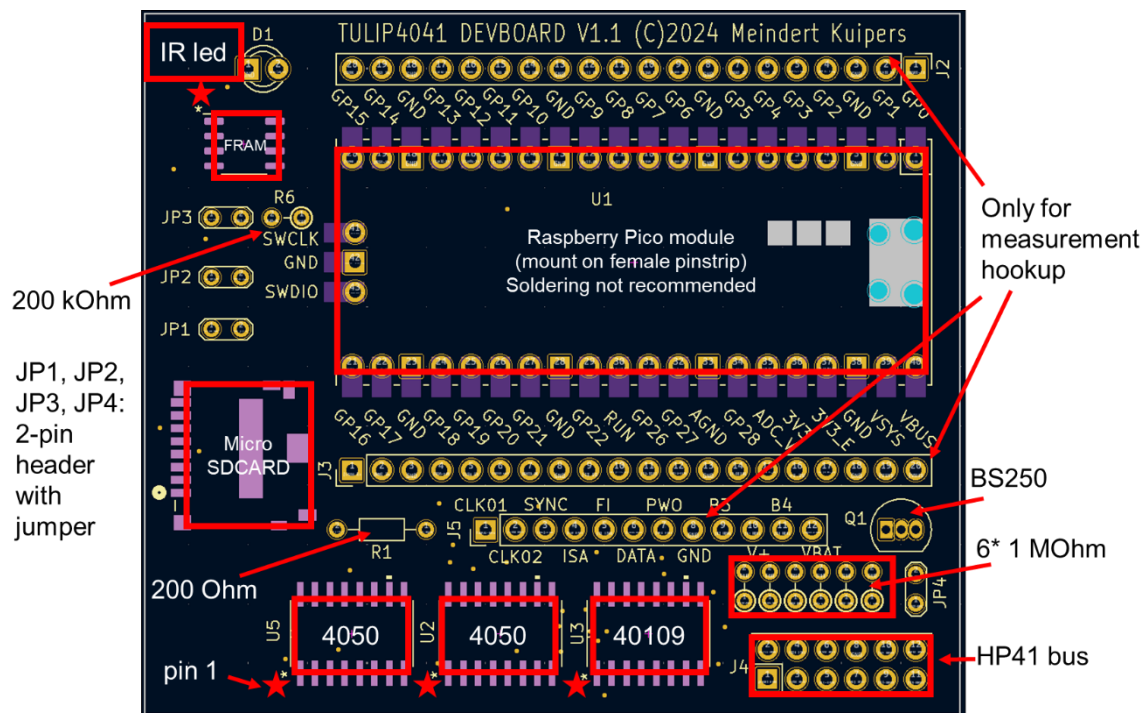
It is very well possible to solder the Pico2 board on the DevBoard, but this is not recommended. It is very hard to unsolder in case of a hardware failure or possible upgrade to another model. When ordering the female headers for the Pico board, ensure that the receptacles are large enough for the Pico pins. Typically turned receptacles may be too small.

The complete Bill Of Material (BOM) is available on the TULIP github pages.

Please follow the steps below in the given order. Do not connect with your precious calculator until instructed.

17.1. Assembling the DevBoard

The DevBoard requires some precise soldering. The figure below shows the position and orientation of the components, pin one is indicated with a red *. Best is to start with the smd components. The headers for probe or analyzer attachment are optional depending on your own preferences. Some of the connections here may be used for future expansions.



U5 is optional. This is a 4050 input level shifter and used only for the FI input tracing. If you do not want to trace FI this component can be left out, but it is highly recommended to keep this in.

To solder an SMD component, the way I do it (using a magnifier lamp):

- Apply a bit of solder on one of the edge pads
- Apply some flux on all pads
- Take the component and verify the value and orientation of pin 1
- Use tweezers to position the component, and solder the edge pad where you applied a bit of solder earlier. Correct the position where needed, such that all pins align with the respective pads
- Solder the pin on the opposite edge, verify the positioning and then solder all other pins
- Check the soldering of all pins with the magnifier

The IR led should be soldered with plenty of pin length above the PCB (about 10mm) and bent in a 90 degree angle away from the board.

Q1 is optional. Mount this if you want the DevBoard to be powered from the HP41. When USB is connected the DevBoard is always powered from USB.

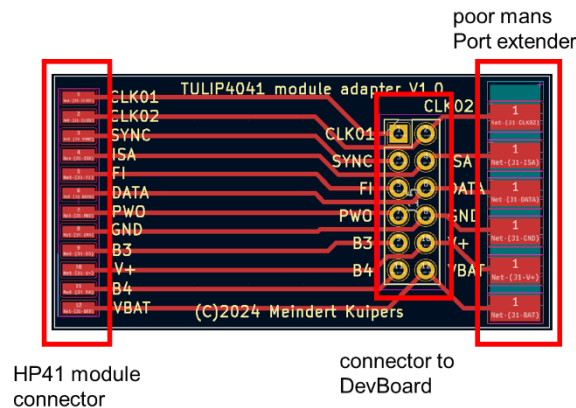
After soldering all components take some time to visually inspect all solder joints and verify the correct orientation before applying power.

Devboard connections:

J1	micro SD card holder
J2/J3	2* 20 pin header (optional) connections to all Pico pins for analyzer hookup or additional (future) expansions.
J4	2* 6 pin header, connection to the HP41 connector board (use a 12-pin flatcable or solder wires). Do not exaggerate the length of the cable, typically 20 cm will work just fine
J5	12 pin header (optional) connections to all HP41 signals for analyzer hookup
JP1	jumper to enable IR led output. When closed JP3 must be open and correct firmware settings must be observed
JP2	jumper to enable FI tracing (when U5 is mounted)
JP3	jumper to enable PWO output. When closed JP1 must be open and correct firmware settings must be observed. MUST BE OPEN WHEN USING THE HP82143 PRINTER EMULATION
JP4	jumper to enable powering the DevBoard from the HP41 BAT line. Only works if Q1 is fitted. Please re-read the section on power consumption
R2..R8	6* 1 MOhm weak pulldowns for stabilizing the HP41 input signals when no HP41 is connected
R1	is the current limiting resistor for the IR led. A value of 200-400 Ohm is just fine and will give you sufficient range to reach your printer or IR receiver at a distance up to 50 cm. Use a low value for more range, but this consumes more power and may draw too much current from the RP2350 I/O pin. The limit is around 12 mA for a single pin

HP41 ConnectorBoard connections:

- J1 2* 6 pin header, connection to the HP41 connector board (use a 12-pin flatcable or solder wires). The PCB has pads on both sides of the board and can be soldered on top of the connector as well, although this is not recommended.
- J2 HP41 module connector. Please be careful when positioning the connector to ensure proper connection with the HP41. Best is to place the PCB in bottom module case and then position and solder the connector. Keep the PCB with soldered connector in a module case to ensure proper alignment when plugging in the HP41 port.



The connector board has pads on the other end of the PCB to plug a physical module as a kind of poor mans port extender. These are not plated and not suited for a large number of cycles.

Connection between the ConnectorBoard and DevBoard can be done with individual soldered or plugged cables, or a flatcable with IDC headers. Cable length should be around 20 cm, longer may work but do not exaggerate.

17.2. [Prepare the micro SD card](#)

The BETA firmware version of the TULIP4041 does not use the SD card, but it will be in a next version of the firmware. There are functions to test the functionality and soldering of the SD card which are necessary.

Prepare a micro SD card formatted with the FAT file system on your host computer. Any size larger than 2 GByte will normally work fine. Create the following subdirectories in the root directory of the micro SD card (although it is not relevant yet)

- ROM, and put some .ROM files in here
- MOD, and add some .MOD files
- RAW, with some .RAW files
- And put some files in the root directory

Remove (in a safe way) the micro SD card from the host computer and plug it in the slot of the DevBoard.

17.3. [Getting the DevBoard up and running](#)

After visual inspection remove any jumpers, these will be used later. Do NOT connect with your calculator yet. First carefully seat the Pico2 board and verify if it is properly connected.

You may now connect the Pico2 board with your host computer using a USB cable. If the Pico2 board is new it does not have any firmware and will go straight into BOOTSEL mode. Otherwise disconnect from USB and put it into BOOTSEL mode by pushing the button on the Pico board while plugging the USB cable. It is also possible to use Picotool to enter BOOTSEL mode, please refer to the Pico documentation for more information.

With the USB cable plugged it makes sense to watch out for smoke or overheated components, although that is unlikely, but may depend on your soldering skills. If you have one it might be useful to use a USB power meter to check for excessive power consumption. Expected current should be under 20-30 mA.

Prepare your host computer with a terminal emulator and have the firmware file (with .uf2 extension) at hand. Keep the Pico connected with USB to your host PC. Do not connect your calculator yet.

Once the TULIP4041 firmware is loaded it is not possible to use Picotool for this. When the Pico is in BOOTSEL mode a USB disk drive is exposed to your host PC. Simply copy the firmware file (with extension .uf2) to this drive. The Pico will reset and start to run the TULIP4041 firmware. Your host PC may respond (if enabled) with sounds that new USB devices are connected.

Now verify if your system registers 5 new COM ports and a USB disk drive. The system may complain about a USB drive not being ready, that is fine.

Start the terminal emulator and connect with the TULIP4041 Command Line Interface (CLI). This is available on one of the new COM or ttyACM ports. Typically the CLI is on the lowest numbered (new) port, but this may differ between systems. The baud rate does not matter. When connected with the correct port the CLI will respond with a welcome message. Try multiple ports to find the CLI, and verify in the system settings of the host PC if all 5 COM ports are visible. When the CLI first connects it will immediately show the welcome message and prompt. You are now ready for the next steps to test the DevBoard and make it operational. The tests are intended to verify the soldering joints.

- **Initial situation**

The TULIP is connected with USB and the latest TULIP4041 firmware is loaded. The TULIP welcome message is shown in the terminal emulator connected to the CLI virtual serial port. The HP41 is NOT connected and NO jumpers are placed. Have a multimeter at hand. Your HP41 is near the test setup and has no modules plugged. The calculator may be an HP41C, -CV or -CX. It is possible to use an HP41CL if it is set to the lowest (original) speed and no (virtual) ROMs are plugged.

- **Find the USB Serial Ports**

To verify the COM port numbers of all serial ports open an instance of your favorite terminal emulator and connect to each USB virtual serial port you find. Do not worry about the baud rate. In the terminal with the CLI (the one with the TULIP welcome message) type the command: **system cdcident**. This will send an identification string to each port. Best make a note of the COM port number and function for later use. For the later tests it is best to close the instance of the terminal emulator connected to the HP-IL virtual serial port.

- **SD card test**

Put the SD card in the slot on the DevBoard and type **sdcard status** in the CLI. This will give the SD characteristics and mount the card. Then type **dir** to get a listing of the files in the root directory, and **dir rom** to list the files in the ROM subdirectory.

Type ***sdcard connect*** to expose the uSD card filesystem to the host computer, and check if that is indeed the case (the sdcard should be connected to the host PC automatically if a card was plugged during startup).

- **FRAM test**

FRAM is tested with the functions for Extended Memory. No plugging of a virtual module is needed. The emulated Extended Memory will be erased with this operation! First type the command ***xmem dump*** in the CLI. This will produce a listing of the emulated Extended Memory contents. Only non-zero registers will be listed. Then type the command ***xmem PATTERN***. This will program a test pattern in FRAM. Then type ***xmem dump*** again. If the registers show the test pattern then the FRAM is functioning correctly. The test pattern is a counting byte value in the bytes of FRAM, and will show as Extended Memory registers as follows:

```
REG 200 = 0x4142414241424142
REG 201 = 0x4243424342434243
REG 202 = 0x4344434443444344
REG 203 = 0x4445444544454445
REG 204 = 0x4546454645464546
REG 205 = 0x4647464746474647
REG 206 = 0x4748474847484748
REG 207 = 0x4849484948494849
REG 208 = 0x494A494A494A494A
REG 209 = 0x4A4B4A4B4A4B4A4B
REG 20A = 0x4B4C4B4C4B4C4B4C
REG 20B = 0x4C4D4C4D4C4D4C4D
REG 20C = 0x4D4E4D4E4D4E4D4E
REG 20D = 0x4E4F4E4F4E4F4E4F
REG 20E = 0x4F504F504F504F50
```

Now type ***xmem ERASE*** and then ***xmem dump***. You should see no registers (the dump function only shows non-zero registers).

- **Configuration test**

FRAM contains the persistent configuration settings and these should now be initialized. Type the command ***system configlist***. Item #92 must contain the value 4041 (Global settings initialized). The other values are the default settings

This concludes the first part of the test. In case of problems do the following:

- Visually inspect the DevBoard if all components are placed correctly, on the right place and with the correct orientation. Pin 1 is indicated with a * on the PCB silkscreen, and with a dot on the component itself
- Visually inspect all soldering joints (with magnifiers) of the FRAM and micro SD card holder for good soldering contacts and make certain that there are no shorts between component pins
- Check if the Pico board is properly seated and if all solder joints of the Pico connector are good. The TULIP firmware will run even when it is not plugged on the DevBoard.

You may now connect your calculator. Unpower the DevBoard (disconnect USB) and remove all physical modules from the calculator. Connect the DevBoard and ConnectorBoard with a 1-1 cable or individual wires. The B3 and B4 signals are not used and it does not matter which port is used. The calculator should have its own power source. Put the ConnectorBoard in an empty module shell for proper alignment with the HP41 module port.

Best practice is to first connect the DevBoard with USB, and then insert the module in the calculator.

- **Power test and initial connection**

Apart from the terminal emulator with the CLI also start an instance of the terminal emulator and connect it with the tracer virtual serial port (verify that with the **system cdcident** command). The CLI will give a message when the tracer port is connected.

Switch your calculator on and verify if it is still working correctly with the TULIP4041 connected. When the HP41 is running the LED on the Pico board will be briefly on. The window with the tracer connected should now show the activity from the calculator. Tracing should be enabled by default, otherwise enable the tracer with the **tracer trace** command. Pressing a key in the tracer windows also toggles the operation of the tracer.

Switch the calculator off.

Use a multimeter to verify the voltages on the HP41 BAT and V+ on the pins or pads at J5. These should both be around 6V, when the calculator is off V+ will be a bit lower.

- **HP41 interface test**

Switch the calculator on again and verify that there is activity in the tracer. This means that the input to the TULIP4041 from the HP41 is working. With the calculator off you can now close the jumper JP2, but only if you have U5 (the additional 4050 level shifter) mounted. This will allow you to trace the FI signal. If your U5 is not mounted then leave JP2 open. You will only see activity on the FI line when a relevant peripheral is connected, the FI signal should normally show only dashes.

In the CLI type the command **plug hpil**. This will plug the HP-IL module (but not yet the HP-IL printer ROM). Now switch the calculator on. In the trace windows you should see activity, and occasionally the FI lines in use. Do a CAT 2 on the calculator to verify that the HP-IL module is now plugged.

For the next tests the instance of the terminal emulator connected to the HP-IL virtual serial port must be closed. The firmware closes the virtual HP-IL loop when nothing is connected to that port. When something is connected the loop is not closed and a TRANSMIT ERR will result. Unless of course a virtual HP-IL device is connected on your host, like ILPer or pyILPer.

In the HP41 you can now key in the DIR command. This should return with the message NO DRIVE. If this does not happen first do a reboot of the TULIP system by typing the command **system REBOOT**, and try again. In this case you can skip plugging the HP-IL module, as this will remain plugged.

When successful you have now verified the connections of the level shifters. There are now 2 more tests for the connections.

First we will test the ISA power on feature. Make certain that your calculator is not running. It may be powered off or in standby mode. Also have the tracer window standby and keep an eye on it. In the CLI give the command **system poweron**, and you should see activity in the tracer (but no activity on the calculator). Be aware that this will not switch on the calculator when it was off!

Now close jumper JP3 and open jumper JP1. **Ensure that the printer ROM is NOT plugged!** We will now drive PWO low while the calculator is running, and this act like a hard emergency stop for the HP41. Enter a program with an endless loop and start it, or start a CAT 3 on the HP41. While the calculator is running give the CLI command **system calcreset**, and verify that the calculators halts and the tracer also stops (after the trace buffer is emptied). After this test open JP3 and close JP1 to prepare for the next and last test.

- **IR Led test**

Final test is for the infrared LED. For this you will need an IR to USB serial receiver and align with the IR LED on the DevBoard. Connect this to your host computer and start the HP82240 simulator in HP82143 mode (Windows only). On a no-Windows system best is to connect a terminal emulator. **Ensure that jumper 1 is closed and jumper 3 is open. Never plug the printer with jumper 3 closed!**

The TULIP IR printing currently does not work with a real HP82240 printer.

In the CLI type **plug printer** to plug the HP82143 ROM image and enable the emulation. Then do the following:

- Type **printer**, verify the status
- Type **printer power** if the printer was powered off
- Type **printer paper** if it was out of paper
- On the calculator key CAT 2 to verify of the printer ROM is plugged
- In the TULIP CLI key **printer trace** and run a CAT 2 again. You should see the catalog listing in the printer simulator, or a catalog listing with some control characters in a terminal emulator
- If you do not have the IR receiver, start a long printing operation, like PRP PRPLOT. Point your mobile phone camera straight into the IR LED and you should see a faint blue glow that stops when the printing is done
- From version 0.91 the CLI has a command that tests the IR LED by sending a string to the LED, to use this just type **printer irtest**, no need to plug the printer ROM.

You have now tested all functions and soldering of the DevBoard. Congratulations!

18. References

All firmware files, documentation and schematics are available at my GitHub pages:

<https://github.com/mjakuipers/TULIP-DevBoard>

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
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

19. [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

