41C Flexible Hardware Module

Every effort has been made to ensure the accuracy of the information contained herein. If you find errors or inconsistencies please bring them to our attention.

Notice:

"HP-41C", "HP-41CV", "HP-41CX" and "HP" are registered trademarks of Hewlett-Packard, Inc. All uses of these terms in this document are to be construed as adjectives, whether or not the noun "calculator", "CPU" or "device" are actually present.

Acknowledgements:

Nate Martin provided the designs of the various module housings.

Table of Contents

1. Introduction 4
2. FHM-based Time Module 6
3. FHM-based X-Functions + Double X-Memory 7
4. (to be announced) Module 8
5. (to be announced) Module 9
6. Implementation Details 10
7. Custom FPGA Programming 12
8. Instruction Memory Initialization 14
9. FHM-based Module Resources 18
10. Revision History 19
A1. FHM Schematic 20

Introduction

The 41C Flexible Hardware Module (FHM) is a small printed circuit board (PCB) designed to fit inside a standard HP-41 module housing. The FHM contains a low-power Field-Programmable Gate Array (FPGA) that can be programmed to implement a number of useful replacement 41C modules.

The FHM can easily be used to directly implement these 41C modules:

- Time Module
- Time Module plus two Extended Memory Modules
- Extended Functions/Memory Module plus two Extended Memory modules
- Two Extended Memory modules
- Service Module
- Printer Service Module
- Machine Language Development Lab (MLDL)

The FHM contains sufficient logic and memory resources to implement any number of new 41C modules. The FPGA used in the FHM, a Lattice Semiconductor ICE5LP2K, contains the following resources:

- 2048 logic cells (look-up table + flip-flop)
- 20 4096-bit memory blocks (for RAM or ROM)
- 48MHz (+/- 10%) High-speed oscillator
- 10KHz (+/- 10%) Low-speed oscillator
- PLL (phase-locked loop)
- 4 DSP blocks (MULT16 with 32-bit accumulator)
- 2 hardware I2C blocks
- 2 hardware SPI blocks

These FPGA resources are significant. The Time Module, which was the original target for the FHM, requires less than half of the FPGA logic resources and only half of the memory resources.

Ten memory blocks can be used to implement a normal 4K ROM image. Each memory block can implement sixty-four 41C registers.

The FHM circuit board contains a precision 32.768KHz oscillator and the possibility of limited I/O (maximum four signals, plus 3.3V and ground) through a Flat Flexible Cable (FFC) connector that is also used for programming the FPGA.

With additional external circuitry, it is likely that the FHM could be used to implement a number of other module functions:

- Printer controller
- IR printer controller
- IR remote control
- Barcode wand interface
- HP-IL interface
- Display controller for remote display
- Serial interface (UART, SPI, I2C)

The remainder of this document provides basic information for the various FHM-based modules that have already been implemented.

It should be noted that there is a pin-compatible FPGA with even more resources that could be used in a future version of the FHM. This FPGA, a Lattice Semiconductor ICE40UP5K, contains the following resources:

- 5280 logic cells (look-up table + flip-flop)
- 30 4096-bit memory blocks (for RAM or ROM)
- 4 16kx16 SPRAM blocks
- 48MHz (+/- 10%) High-speed oscillator
- 10KHz (+/- 10%) Low-speed oscillator
- PLL (phase-locked loop)
- 8 DSP blocks (MULT16 with 32-bit accumulator)
- 2 hardware I2C blocks
- 2 hardware SPI blocks

FHM-based Time Module

The FHM-based Time Module includes all of the user-visible features of the original Time Module, including the module software (version -1C). The current implementation does not include the special input and output functions that are available when the Time Module is disassembled, but these could be added to the logic design fairly easily.

The FHM-based Time Module draws more current than the original Time Module in some cases. This is an unavoidable byproduct of the advanced technology used to implement the FHM. The table below shows the typical current requirements for the FHM-based Time Module in the different calculator states.

Calculator state	FHM Time Module Current drain	HP Time Module Current Drain
Deep Sleep	220uA	10uA - 50uA
Light Sleep	230uA	?
Running	260uA	500uA
Running (Stopwatch)	320uA	?

The FHM-based Time Module only uses half of the memory blocks available in the FPGA, so it is also possible to include two X-Memory modules in the same design.

FHM-based X-Functions + Double X-Memory

This FHM-based design can be used with a 41C or 41CV to free up Ports while bringing the machine up to the full memory capabilities of the 41C system. This design includes version -1C of the Extended Functions software. An alternate version, that only includes the double X-Memory, can be used with 41CX machines.

The FHM version of the X-Functions Module and double X-Memory Modules draws more current than the combination of the original modules. This is an unavoidable byproduct of the advanced technology used to implement the FHM. The table below shows the typical current requirements for the FHM-based module in the different calculator states.

Calculator state	FHM Module Current drain	HP Module Current Drain
Deep Sleep	220uA	30uA
Light Sleep	230uA	30uA
Running	260uA	30uA

(to be announced) Module

The (to be announced) Module is an original FHM-based design that adds new capabilites to the 41C universe. The full capabilities of the (to be announced) Module are described in a stand-alone manual, but are summarized below.

	placeholder	

The (to be announced) Module uses a double-length module housing. Three pins on the FFC programming connector, plus the power and ground, communicate between the FHM board and the (to be announced) that takes up the remainder of the space in the enlarged module housing.

(to be announced) Module

The (to be announced) Module is another original FHM-based design that also adds new capabilites to the 41C universe. The full capabilities of the (to be announced) Module are described in a stand-alone manual, but are summarized below.

	placeholder	

The (to be announced) Module uses a double-height module housing. Four pins on the FFC programming connector, plus the power and ground, communicate between the FHM board and the (to be announced) that takes up the remainder of the space in the enlarged module housing.

Implementation Details

The schematic for the FHM board is shown in Appendix 1.

As mentioned previously, the FHM uses a Lattice Semiconductor ICE5LP2K FPGA. This device requires three regulated power supplies of 1.2, 2.5 and 3.3 volts. All three of these voltages are generated on the FHM board and are sequenced at power-up as required by the FPGA.

The configuration for the FPGA is held in the Non-Volatile Configuration Memory (NVCM) inside the FPGA. Using the NVCM to configure the FPGA reduces the number of external components required but also means that the FPGA **can only be programmed one time**. So the FHM, once programmed, cannot be modified.

In addition to the power-supply sequencing, the FPGA requires that a Configuration Reset signal be asserted until after the power supplies are stable. The FHM contains circuitry to do this automatically.

The FPGA requires six signals (plus power and ground) to do the programming of the NVCM, and these signals are brought to the FFC connector on the board for programming. Two of these signals are dedicated to programming, but the other four signals can be used for I/O to the FPGA after the initial configuration (loading the contents of the NVCM to the programmable array.) These signals are normal 3.3 volt inputs or outputs. Because an FFC can really only be used with a board-mounted connector, there is another board available (the same size as the FHM) that contains a matching FFC and a more user-friendly 1mm-pitch connector like that used on the 41CL board for programming and the serial port connection. This "translator" board is used in the (to be announced) Module to connect between the FHM and the (to be announced).

Although the FPGA contains a pair of configurable oscillators (48MHz and 10KHz), neither oscillator is accurate enough for most uses cases. So the FHM contains a 32.768 KHz oscillator for use as a precision frequency reference. This oscillator is always active, because there wasn't room to route a control signal from the FPGA to the oscillator to turn it off. This precision frequency reference can be used to "accurize" the FPGA oscillators. This technique is used in the (to be announced) Module and the (to be announced) Module to generate an accurate clock for the high-speed serial port.

The FHM also contains a dedicated power-on-reset circuit with the output routed to an FPGA input. This circuit is not strictly necessary, because the initial configuration of all of the storage elements in the FPGA is determined at configuration, which occurs at power-up.

The FHM contains voltage translators to drive the 41C bus, which runs at 6 volts. These drivers, for ISA, DATA and FI, allow the FHM to respond as instruction memory, data or peripheral register memory and with peripheral flags.

Because of space constraints, the FHM uses resistive dividers to translate from the 6 volt 41C bus to FPGA inputs. These resistive dividers are designed to minimize the current required and are only guaranteed to function at the normal 41C bus speed.

Again because of space constraints, the FHM does not decode the Port Identifier inputs. This means that **like the Card Reader and Time Module, the FHM must use a fixed page address for code, independent of its physical location in the calculator.** This should not be a problem in most cases, as logic can be included in the FPGA to relocate any code to a different page address under program control. This technique is used in the (to be announced) Module and the (to be announced) Module to relocate code away from the default page address.

The FHM board is designed to fit in a normal 41C module housing. In the original HP module housing the bus connector was soldered to the top of the printed circuit board (PCB) in the module, and the bottom of the circuit board rested on the lower half of the module housing. This was possible because these PCBs only had components on one side. The FHM has components on both sides of the board, so the bus connector must be soldered to the land pads on the bottom of the FHM. The size of the FHM board is slightly smaller than the HP board to account for this mounting requirement.

When used with a 41CL, it is possible to wire an FHM directly to solder lands on the top of the 41CL board (V4 and V5 only) and affix the FHM to the top of the RAM package on the CL board. This saves a Port.

Custom FPGA Programming

The FHM provides a way to create your own custom 41C module. This will require learning how to use the FPGA programming tools and acquiring FPGA programming hardware. The FPGA programming tools are free, and the FPGA programming hardware is fairly low cost (about $175.)

In this section, we will cover a number of traps that you should avoid if you choose to try to design something to load into an FHM. We discovered most of these during the design of the FHM-base Time Module and the (to be announced) Module.

1. Use the 41C bus interface, main state machine, instruction memory and register memory Verilog code from the FHM-based Time Module. This design works, and there is no point in reinventing the wheel.

2. The flip-flops in this FPGA have an active-high reset input. If you use an active-low reset signal it cannot be routed using a global route, which impacts performance.

3. There are only eight global buffers available, so choose how you use them wisely. Three of these global buffers will be needed for the PH1, PH2 and PWO signals. If you use the PWR_UP signal it will also need a global buffer.

4. The FPGA, as well as the synthesis tool, only works well with synchronous designs.

5. Pay very careful attention to clock-domain-crossing issues. The FHM-base Time Module source code should be used as an example of how to do clock-domain-crossing.

6. Using any of the four pins on the FFC for I/O can be tricky. Two of these pins have external pull-ups and one has an external pull-down, as recommended for programming. If you plan to use these pins for I/O the external resistors should probably be removed from the FHM board after programming to reduce supply current.

7. If you are not using any of the pins on the FFC, you still need to be careful to specify them properly in your design. Refer to the FHM-based Time Module for an example. The FPGA will enable an internal pull-up on uncommitted pins, which will conflict (to the tune of 65uA of wasted current) in the case of the pin with the external pull-down.

8. The FPGA flip-flops are always outputting a logic zero after configuration, even if they have an asynchronous set input. This can lead to unexpected deadlock states if you are not careful. The best way to avoid this is to design so that all flip-flops start out with a logic zero. The normally one-hot main state machine in the FHM-based Time Module provides an example of how to solve this issue.

9. The oscillators on the FPGA are not accurate, but the 32.768 KHz oscillator on the FHM is (to the tune of -90/+30 ppm over temperature.) Refer to the Verilog code for the

(to be announced) Module for an example of how to "accurize" the on-board oscillators.

10. The FPGA vendor does not provide a default FIFO macro for the FPGA memories, so you'll need to design your own. Refer to the Verilog code for the (to be announced) Module for an example of how to do this.

11. The FPGA vendor does not publish how the bits are arranged in the initialization strings except when using the by-8 memory configuration. Since an instruction memory will want to use the by-2 memory configuration, this is a problem. A BASIC program has been developed to translate from the .rom file format to the proper initialization strings. A listing of this program is provided in another section of this document.

12. The instruction memory, being RAM, can be made writeable. Refer to the Verilog code for the (to be announced) Module for an example of how to do this. This feature will allow the FHM to be used as a machine-language development lab (MLDL.)

13. None of our designs have used either the PLL, SPI, I2C or DSP blocks in the FPGA, so you're on your own there. If you use them and find any traps, which seems likely given our experience with other features in the FPGA, please send them to us for inclusion here.

14. Use the LSE synthesis tool, not the default Synopsys synthesis tool. The default synthesis tool does not properly handle memory block initialization.

15. Use the FHM-based Time Module schematic to create the pin assignments for the FPGA. Remember to always assign placeholders to the programming port pins so that you can disable the interal pullups on these pins.

16. Remember that the FHM uses the NVCM for FPGA programming, so it is one-time-programmable. Breadboard your design. An alternative would be to connect an external serial Flash to the programming pins via the FFC. We do not recommend this approach for two reasons: First, there is no visibility for internal signals with this approach. Second, the pins on the FFC cannot be used for I/O in this case.

Instruction Memory Initialization

The FHM can mimic a 4kx10 41C ROM (instruction memory) by using ten FPGA RAM blocks to create a 4kx10 memory. This requires that the memory blocks be configured for 2kx2 operation, which complicates the initialization of the RAM. Unfortunately, the FPGA supplier does not document how the initialization strings correspond to the memory contents when used as a ROM.

We ended up reverse-engineering how the bits in the initialization strings corresponded to the bits in a 2kx2 memory. The BASIC code shown below is what we use to translate between a normal 41C .rom file and the initialization strings for the block RAMs in the FPGA. This program outputs a separate initialization file for each instruction memory. Refer to the source code for the FHM-based Time Module for an example of how to use these outputs in the Verilog source code.

```
10 CLS
20 REM This program takes a module image file in .rom format and
30 REM translates it into Verilog 256-bit hex numbers ready to
40 REM load into 2-bit on-fpga memories.
50 PRINT "What is the module image filename?"
60 INPUT "(Without any .rom extension)   ", A$
70 F$ = A$

100 B0$ = F$ + "L0.txt"
101 B1$ = F$ + "L1.txt"
102 B2$ = F$ + "L2.txt"
103 B3$ = F$ + "L3.txt"
104 B4$ = F$ + "L4.txt"

110 A$ = A$ + ".rom"

120 OPEN A$ FOR BINARY AS #1
121 IF EOF(1) THEN PRINT "no such file ": CLOSE : END
122 ON ERROR GOTO 990

130 OPEN B0$ FOR OUTPUT AS #2
131 OPEN B1$ FOR OUTPUT AS #3
132 OPEN B2$ FOR OUTPUT AS #4
133 OPEN B3$ FOR OUTPUT AS #5
134 OPEN B4$ FOR OUTPUT AS #6

200 FOR BITNO = 0 TO 14 STEP 2

210 BBL$ = HEX$(BITNO)
211 BBH$ = HEX$(BITNO + 1)

220 R0$ = "defparam RAML_10.INIT_" + BBL$ + " = 256'h"
221 R1$ = "defparam RAML_10.INIT_" + BBH$ + " = 256'h"
222 R2$ = "defparam RAML_32.INIT_" + BBL$ + " = 256'h"
223 R3$ = "defparam RAML_32.INIT_" + BBH$ + " = 256'h"
224 R4$ = "defparam RAML_54.INIT_" + BBL$ + " = 256'h"
225 R5$ = "defparam RAML_54.INIT_" + BBH$ + " = 256'h"
226 R6$ = "defparam RAML_76.INIT_" + BBL$ + " = 256'h"
227 R7$ = "defparam RAML_76.INIT_" + BBH$ + " = 256'h"
228 R8$ = "defparam RAML_98.INIT_" + BBL$ + " = 256'h"
229 R9$ = "defparam RAML_98.INIT_" + BBH$ + " = 256'h"

230 FOR LINENO = 4032 TO 0 STEP -64

250 GET #1, (LINENO + BITNO + 49), BH3%
251 GET #1, (LINENO + BITNO + 50), BL3%
252 BL3% = BL3% AND 255
253 BH3% = BH3% AND 255

260 GET #1, (LINENO + BITNO + 33), BH2%
```



```
261 GET #1, (LINENO + BITNO + 34), BL2%
262 BL2% = BL2% AND 255
263 BH2% = BH2% AND 255

270 GET #1, (LINENO + BITNO + 17), BH1%
271 GET #1, (LINENO + BITNO + 18), BL1%
272 BL1% = BL1% AND 255
273 BH1% = BH1% AND 255

280 GET #1, (LINENO + BITNO + 1), BH0%
281 GET #1, (LINENO + BITNO + 2), BL0%
282 BL0% = BL0% AND 255
283 BH0% = BH0% AND 255

290 CV0 = ( (BL0% AND 1)          + ((BL1% AND 1)   * 2)  + ((BL2% AND 1)   * 4)  + ((BL3%
AND 1)   * 8))  AND 15
291 CV1 = (((BL0% AND 2)   / 2)   +  (BL1% AND 2)         + ((BL2% AND 2)   * 2)  + ((BL3%
AND 2)   * 4))  AND 15
292 CV2 = (((BL0% AND 4)   / 4)   + ((BL1% AND 4)   / 2)  +  (BL2% AND 4)         + ((BL3%
AND 4)   * 2))  AND 15
293 CV3 = (((BL0% AND 8)   / 8)   + ((BL1% AND 8)   / 4)  + ((BL2% AND 8)   / 2)  +  (BL3%
AND 8)       )  AND 15
294 CV4 = (((BL0% AND 16)  / 16)  + ((BL1% AND 16)  / 8)  + ((BL2% AND 16)  / 4)  + ((BL3%
AND 16)  / 2))  AND 15
295 CV5 = (((BL0% AND 32)  / 32)  + ((BL1% AND 32)  / 16) + ((BL2% AND 32)  / 8)  + ((BL3%
AND 32)  / 4))  AND 15
296 CV6 = (((BL0% AND 64)  / 64)  + ((BL1% AND 64)  / 32) + ((BL2% AND 64)  / 16) + ((BL3%
AND 64)  / 8))  AND 15
297 CV7 = (((BL0% AND 128) / 128) + ((BL1% AND 128) / 64) + ((BL2% AND 128) / 32) + ((BL3%
AND 128) / 16)) AND 15
298 CV8 = ( (BH0% AND 1)          + ((BH1% AND 1)   * 2)  + ((BH2% AND 1)   * 4)  + ((BH3%
AND 1)   * 8))  AND 15
299 CV9 = (((BH0% AND 2)   / 2)   +  (BH1% AND 2)         + ((BH2% AND 2)   * 2)  + ((BH3%
AND 2)   * 4))  AND 15

300 R0$ = R0$ + HEX$(CV0)
301 R1$ = R1$ + HEX$(CV1)
302 R2$ = R2$ + HEX$(CV2)
303 R3$ = R3$ + HEX$(CV3)
304 R4$ = R4$ + HEX$(CV4)
305 R5$ = R5$ + HEX$(CV5)
306 R6$ = R6$ + HEX$(CV6)
307 R7$ = R7$ + HEX$(CV7)
308 R8$ = R8$ + HEX$(CV8)
309 R9$ = R9$ + HEX$(CV9)

310 NEXT LINENO

320 PRINT #2, R0$ + ";"
321 PRINT #2, R1$ + ";"
322 PRINT #3, R2$ + ";"
323 PRINT #3, R3$ + ";"
324 PRINT #4, R4$ + ";"
325 PRINT #4, R5$ + ";"
326 PRINT #5, R6$ + ";"
327 PRINT #5, R7$ + ";"
328 PRINT #6, R8$ + ";"
329 PRINT #6, R9$ + ";"

330 NEXT BITNO

340 CLOSE #2
341 CLOSE #3
342 CLOSE #4
343 CLOSE #5
344 CLOSE #6


500 B0$ = F$ + "H0.txt"
501 B1$ = F$ + "H1.txt"
502 B2$ = F$ + "H2.txt"
503 B3$ = F$ + "H3.txt"
504 B4$ = F$ + "H4.txt"

530 OPEN B0$ FOR OUTPUT AS #2
531 OPEN B1$ FOR OUTPUT AS #3
```



```
532 OPEN B2$ FOR OUTPUT AS #4
533 OPEN B3$ FOR OUTPUT AS #5
534 OPEN B4$ FOR OUTPUT AS #6

600 FOR BITNO = 0 TO 14 STEP 2

610 BBL$ = HEX$(BITNO)
611 BBH$ = HEX$(BITNO + 1)

620 R0$ = "defparam RAMH_10.INIT_" + BBL$ + " = 256'h"
621 R1$ = "defparam RAMH_10.INIT_" + BBH$ + " = 256'h"
622 R2$ = "defparam RAMH_32.INIT_" + BBL$ + " = 256'h"
623 R3$ = "defparam RAMH_32.INIT_" + BBH$ + " = 256'h"
624 R4$ = "defparam RAMH_54.INIT_" + BBL$ + " = 256'h"
625 R5$ = "defparam RAMH_54.INIT_" + BBH$ + " = 256'h"
626 R6$ = "defparam RAMH_76.INIT_" + BBL$ + " = 256'h"
627 R7$ = "defparam RAMH_76.INIT_" + BBH$ + " = 256'h"
628 R8$ = "defparam RAMH_98.INIT_" + BBL$ + " = 256'h"
629 R9$ = "defparam RAMH_98.INIT_" + BBH$ + " = 256'h"

630 FOR LINENO = 8128 TO 4096 STEP -64

650 GET #1, (LINENO + BITNO + 49), BH3%
651 GET #1, (LINENO + BITNO + 50), BL3%
652 BL3% = BL3% AND 255
653 BH3% = BH3% AND 255

660 GET #1, (LINENO + BITNO + 33), BH2%
661 GET #1, (LINENO + BITNO + 34), BL2%
662 BL2% = BL2% AND 255
663 BH2% = BH2% AND 255

670 GET #1, (LINENO + BITNO + 17), BH1%
671 GET #1, (LINENO + BITNO + 18), BL1%
672 BL1% = BL1% AND 255
673 BH1% = BH1% AND 255

680 GET #1, (LINENO + BITNO + 1), BH0%
681 GET #1, (LINENO + BITNO + 2), BL0%
682 BL0% = BL0% AND 255
683 BH0% = BH0% AND 255

690 CV0 = ( (BL0% AND 1)          + ((BL1% AND 1)   * 2)  + ((BL2% AND 1)   * 4)  + ((BL3%
AND 1)   * 8))  AND 15
691 CV1 = (((BL0% AND 2)   / 2)   +  (BL1% AND 2)         + ((BL2% AND 2)   * 2)  + ((BL3%
AND 2)   * 4))  AND 15
692 CV2 = (((BL0% AND 4)   / 4)   + ((BL1% AND 4)   / 2)  +  (BL2% AND 4)         + ((BL3%
AND 4)   * 2))  AND 15
693 CV3 = ((((BL0% AND 8)  / 8)   + ((BL1% AND 8)   / 4)  + ((BL2% AND 8)   / 2)  +  (BL3%
AND 8)        )  AND 15
694 CV4 = (((BL0% AND 16)  / 16)  + ((BL1% AND 16)  / 8)  + ((BL2% AND 16)  / 4)  + ((BL3%
AND 16)  / 2))  AND 15
695 CV5 = (((BL0% AND 32)  / 32)  + ((BL1% AND 32)  / 16) + ((BL2% AND 32)  / 8)  + ((BL3%
AND 32)  / 4))  AND 15
696 CV6 = (((BL0% AND 64)  / 64)  + ((BL1% AND 64)  / 32) + ((BL2% AND 64)  / 16) + ((BL3%
AND 64)  / 8))  AND 15
697 CV7 = ((((BL0% AND 128) / 128) + ((BL1% AND 128) / 64) + ((BL2% AND 128) / 32) + ((BL3%
AND 128) / 16)) AND 15
698 CV8 = ( (BH0% AND 1)          + ((BH1% AND 1)   * 2)  + ((BH2% AND 1)   * 4)  + ((BH3%
AND 1)   * 8))  AND 15
699 CV9 = (((BH0% AND 2)   / 2)   +  (BH1% AND 2)         + ((BH2% AND 2)   * 2)  + ((BH3%
AND 2)   * 4))  AND 15

700 R0$ = R0$ + HEX$(CV0)
701 R1$ = R1$ + HEX$(CV1)
702 R2$ = R2$ + HEX$(CV2)
703 R3$ = R3$ + HEX$(CV3)
704 R4$ = R4$ + HEX$(CV4)
705 R5$ = R5$ + HEX$(CV5)
706 R6$ = R6$ + HEX$(CV6)
707 R7$ = R7$ + HEX$(CV7)
708 R8$ = R8$ + HEX$(CV8)
709 R9$ = R9$ + HEX$(CV9)

710 NEXT LINENO
```



```
720 PRINT #2, R0$ + ";"
721 PRINT #2, R1$ + ";"
722 PRINT #3, R2$ + ";"
723 PRINT #3, R3$ + ";"
724 PRINT #4, R4$ + ";"
725 PRINT #4, R5$ + ";"
726 PRINT #5, R6$ + ";"
727 PRINT #5, R7$ + ";"
728 PRINT #6, R8$ + ";"
729 PRINT #6, R9$ + ";"

730 NEXT BITNO

840 CLOSE
850 END

990 PRINT "Error in input file": CLOSE : END
```


FHM-based Module Resources

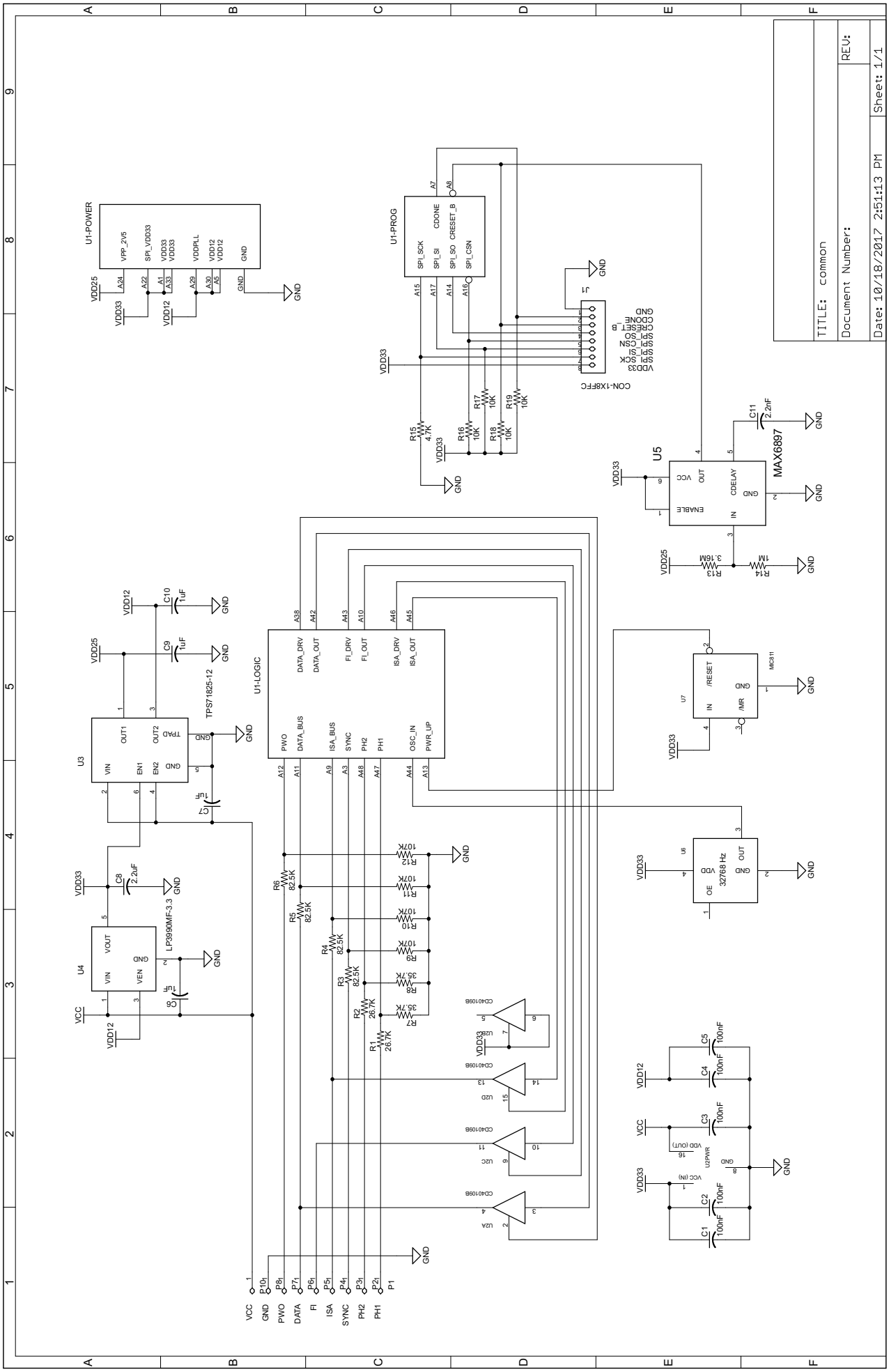
To give you an idea of the capabilities of the FHM, the table below shows the resource requirements for the various FHM-based modules that have already been designed.

Module	LUTs (2048)	DFFs (2048)	RAMs (20)	ROMs (20)	GBs (8)
Time	834	627	-	10	5
Time + 2 x X-MEM	1058	720	8	10	6
X-FNS + 2 x X-MEM	375	189	10	10	4
2 x X-MEM	228	96	8	-	4
(to be announced)	1259	763	19	-	5

Revision History

09/19/2018	Preliminary release

FHM Schematic


TITLE: common
Document Number:
REV:
Date: 10/18/2017 2:51:13 PM
Sheet: 1/1
U1-POWER
U1-PROG
U1-LOGIC
U5
MAX6897
U3
TPS71825-12
U4
LP3990MF-3.3
CON-1X8FFC
32768 Hz
CD4010B
VDD25
VDD33
VDD12
VCC
GND
VDD33
SPI_SCK
SPI_SI
SPI_SO
SPI_CSN
CRESET_B
CDONE
VDD33
GND