10555 Scripps Trl, San Diego, CA 92131
858-877-3485 858-408-9550
http://www.pulserain.com

PulseRain FP51-1T Microcontroller

Technical Reference Manual

Sep, 2017

This page is intentionally left blank.

Table of Contents

References

1. MCS-51 Programmer's Guide and Instruction Set, Intel Corporation.
2. Building Embedded Systems – Programmable Hardware, Changyi Gu, APress Media, July 2016 http://www.apress.com/us/book/9781484219188
3. SDCC Compiler User Guide, SDCC 3.6.0, 2016-06-04, Rev 9615 (http://sdcc.sourceforge.net)
4. Computer Organization and Design - The Hardware / Software Interface (3rd edition), David A. Patterson and John L. Hennessy, Morgan Kaufmann Publication, 2005
5. Wishbone B4, WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores, OpenCores Organization, 2010
6. PulseRain M10 – I2C, Technical Reference Manual, Doc# TRM-0922-01007, Rev 1.0.0, 09/2017 https://github.com/PulseRain/M10I2C/raw/master/extras/M10_I2C_TRM.pdf
7. PulseRain M10 – PWM, Technical Reference Manual, Doc# TRM-0922-01009, Rev 1.0.1, 10/2017, https://github.com/PulseRain/M10PWM/raw/master/extras/M10_PWM_TRM.pdf
8. PulseRain M10 – microSD, Technical Reference Manual, Doc# TRM-0922-01006, Rev 1.0.0, 09/2017 https://github.com/PulseRain/M10SD/raw/master/extra/M10_SD_TRM.pdf
9. PulseRain M10 – Voice CODEC, Technical Reference Manual, Doc# TRM-0922-01001, Rev 1.0.3, 09/2017, https://github.com/PulseRain/M10CODEC/raw/master/extras/M10_CODEC_TRM.pdf
10. PulseRain M10 – ADC, Technical Reference Manual, Doc# TRM-0922-01003, Rev 1.0.0, 09/2017 https://github.com/PulseRain/M10ADC/raw/master/extra/M10_ADC_TRM.pdf
11. PulseRain M10 – JTAG, Technical Reference Manual, Doc# TRM-0922-01008, Rev 1.0.0, 09/2017 https://github.com/PulseRain/M10JTAG/raw/master/extras/M10_JTAG_TRM.pdf
12. PulseRain M10 – SRAM, Technical Reference Manual, Doc# TRM-0922-01004, Rev 1.0.0, 09/2017 https://github.com/PulseRain/M10SRAM/raw/master/extras/M10_SRAM_TRM.pdf
13. PulseRain M10 – DTMF, Technical Reference Manual, Doc# TRM-0922-01002, Rev 1.0.0, 09/2017 https://github.com/PulseRain/M10DTMF/raw/master/extras/M10_DTMF_TRM.pdf
14. PulseRain M10 – Serial Port, Technical Reference Manual, Doc# TRM-0922-01005, Rev 1.0.0, 09/2017, https://github.com/PulseRain/M10SerialAUX/raw/master/extras/M10_Serial_TRM.pdf
15. PulseRain M10 – Quick Start Guide, Doc#QSG-0922-0039, Rev 1.2.0, 10/2017 https://github.com/PulseRain/Arduino_M10_IDE/blob/master/docs/M10_quick_start.pdf
16. C++ preprocessor __VAR_ARGS__ number of arguments, https://stackoverflow.com/questions/2124339/c-preprocessor-va-args-number-of-arguments
17. C Pre-Processor Magic, by Jonathan Heathcote, http://jhnet.co.uk/articles/cpp_magic
18. MicroChip 23A1024/23LC1024 1Mbit SPI Serial SRAM with SDI and SQI Interface Datasheet, DS20005142C
19. Si3000 Voice Band CODEC with Microphone / Speaker Drive, Rev 1.4, Silicon Laboratories, 12/2010

Acronyms and Abbreviations

Acronyms / Abbreviations	Definition
ACK	Acknowledge
ADC	Analog to Digital Converter
API	Application Program Interface
BCD	Binary-Coded Decimal
CISC	Complex Instruction Set Computer
CODEC	Coder-Decoder
DPTR	Data Pointer
DTMF	Dual Tone Multi Frequency
FPGA	Field Programmable Gate Array
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IO	Input and Output
IRQ	Interrupt Request Line
ISA	Instruction Set Architecture
ISR	Interrupt Service Routine
JTAG	Joint Test Action Group
LSB	Least Significant Bit
MCU	Microcontroller Unit
MSB	Most Significant Bit
NOP	No Operation
OCD	On-chip Debugger
OS	Operating System
PC	Personal Computer or Program Counter
PSW	Program Status Word
PWM	Pulse Width Modulation
RISC	Reduced Instruction Set Computer
SDCC	Small Device C Compiler
SFR	Special Function Register
SPI	Serial Peripheral Interface
SRAM	Static Random-Access Memory
UART	Universal Asynchronous Receiver-Transmitter
Wi-Fi	Wireless Fidelity

1 Introduction

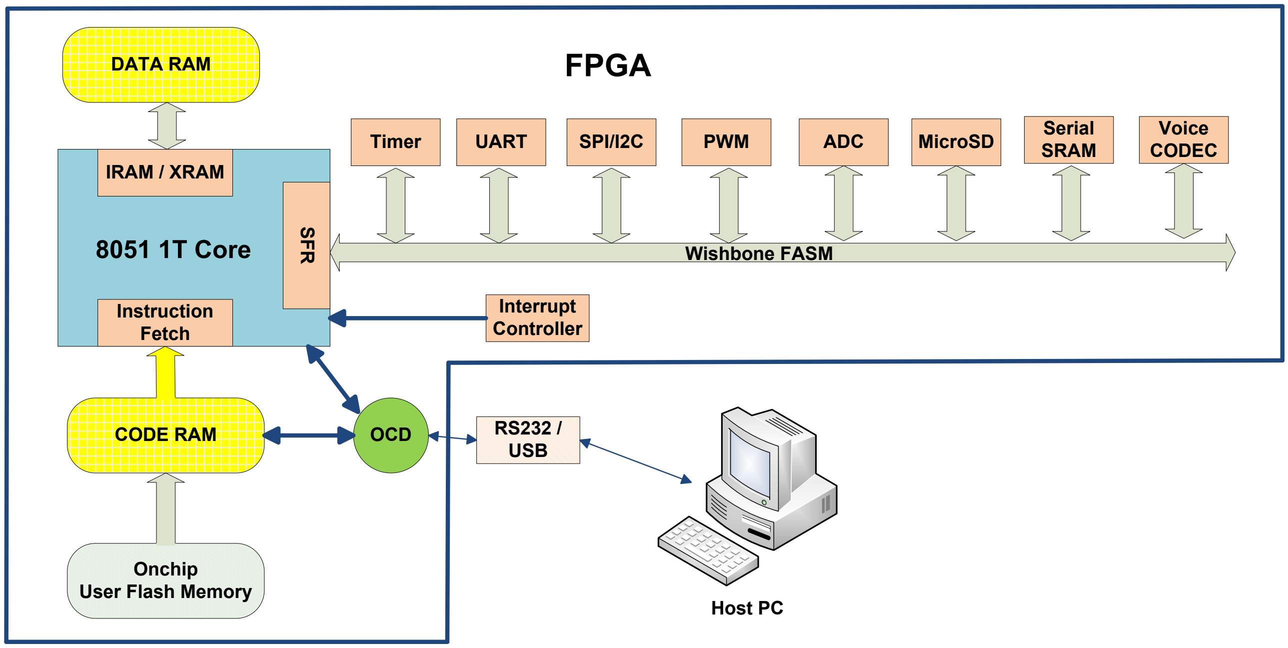

Figure 1-1 The Whole Picture

Putting MCU core into FPGA is becoming a universal practice these days. However, most MCU soft cores available today are tied to specific FPGA vendors, with close source implementation. And there will be oodles of hoops for engineers to jump through if they decide to migrate from one proprietary core to another. To break the status quo, PulseRain Technology has come up with the **open source** FP51-1T core (Named after the legendary P-51 Mustang Fighter-bomber.) for a more portable FPGA solution.

As shown in Figure 1-1, the FP51-1T is a high performance 8-bit MCU core, compatible with Intel 8051 ISA. With a crafty RISC implementation, this core can achieve single clock cycle execution for most instructions, while pushing the clock rate above 96MHz. (Silicon Proven on the low speed -8 grade device in Intel/Altera MAX 10 family).

To facilitate debugging, an OCD (On Chip Debugger) is also provided along with the MCU core. The OCD can communicate with host PC through RS232 protocol. And it supports features like code downloading, single step execution, hardware breakpoint etc. A software package for Arduino IDE is also provided to support the Arduino Language.

And with great flexibility, new peripherals can be added and customized through Wishbone bus interface. The revision A0 of FP51-1T MCU supports the following peripherals:

- Timer

- UART
- I2C
- PWM
- ADC
- microSD
- Serial SRAM (Microchip 23LC1024)
- Voice CODEC (Silicon Lab Si3000)
- JTAG UART

To facilitate the software development, PulseRain Technology has also provided Arduino libraries for those peripherals. Some application specific libraries are also offered, like the following:

- Library for decoding DTMF tones
- Library to control the ESP8266 Wi-Fi chip

And all the code can be found on PulseRain Technology's public repositories: http://git.pulserain.com

2 Hardware

2.1 Processor Core

2.1.1 Hardware Architecture

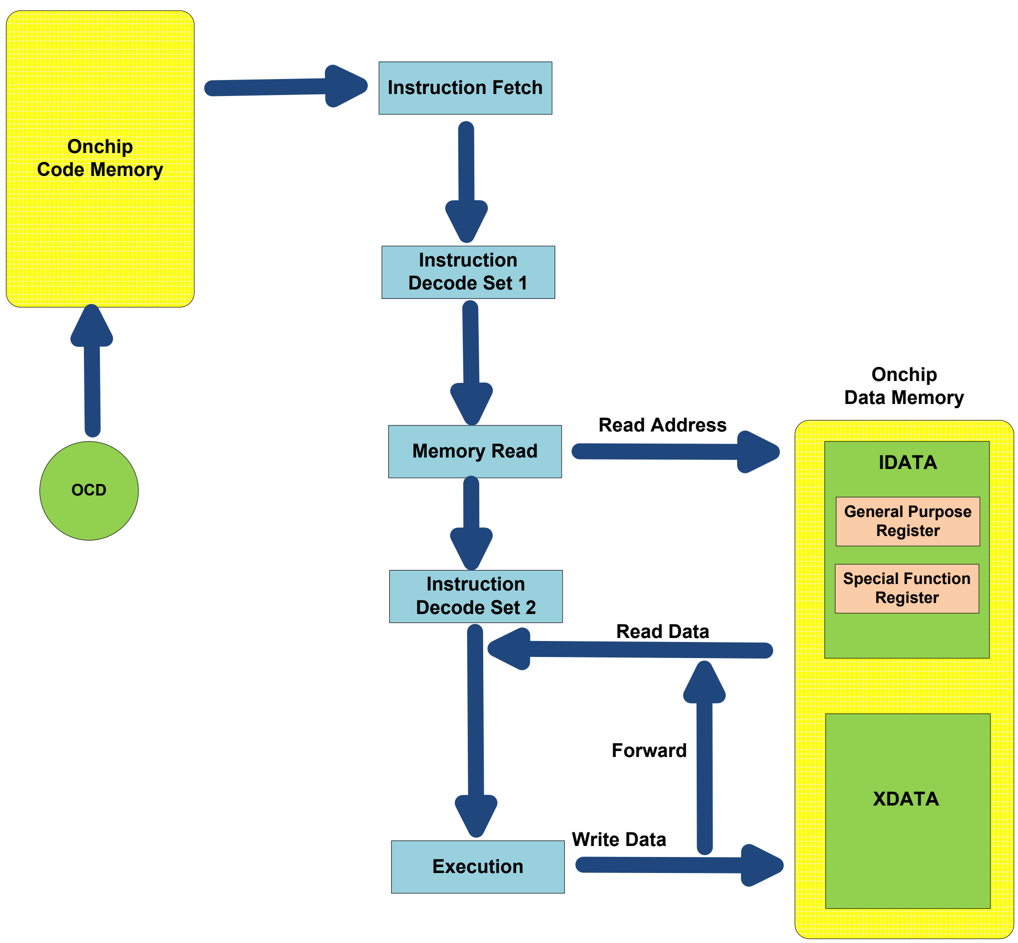

Figure 2-1 FP51-1T Architecture

As shown in Figure 2-1, the FP51-1T processor core has a Harvard memory architecture with 5 pipeline stages. In other words, the memory is divided into two separate address spaces: The memory space for code and the one for data. The code memory is addressed by a 16-bit PC (Program Counter), so the maximum code memory space is 64KB. The data memory space is divided into two sub spaces: IDATA and XDATA. The IDATA space is addressed by an 8-bit address, which has the maximum space size of 256 bytes. The XDATA is addressed by a 16-bit address, which theoretically has a maximum space size of 64KB. On FP51-1T, the IDATA and XDATA are physically merged into one block memory, so the actual available XDATA space on FP51-1T is 64KB - 256 Byte.

2.1.1.1 Registers

Within the IDATA address space, some of the addresses are used by memory mapped registers. These registers are as following:

- General Purpose Register
  The general-purpose registers are divided into 4 banks, with 8 registers in each bank. The address for those registers are shown in Table 2-1. At any moment, only one bank is active, and the active bank is specified by the PSW (Program Status Word).

Bank	Register	Address
0	R0 – R7	0 – 7
1	R0 – R7	8 – 15
2	R0 – R7	16 – 23
3	R0 – R7	24 – 31

Table 2-1 Address Mapping for General-Purpose Registers

- SFR (Special Function Register)
  The SFRs are mapped into the addresses between 0x80 and 0xFF. They are mainly used for control of processor core and peripherals. Those registers for peripheral control will be discussed in later sections. And those SFRs specific to the processor core are listed below in Table 2-2.

Address	Register	Description	Bit Address (MSB ~ LSB)
0x81	SPL	The lower 8 bits of stack pointer	N/A
0x82	DPL	The lower 8 bits of data pointer (DPTR)	N/A
0x83	DPH	The higher 8 bits of data pointer (DPTR)	N/A
0x87	PCON	Power Control (N/A on FP51-1T)	N/A
0xD0	PSW	Program Status Word	0xD7 ~ 0xD0
0xE0	ACC	Accumulator	0xE7 ~ 0xE0
0xE6	MCU_REVISION	Revision of the MCU (Read Only).	N/A
0xE7	XPAGE	Page Index for XDATA memory access	N/A
0xEF	SPH	The higher 8 bits of stack pointer	N/A
0xF0	B	B Register. Used by MUL / DIV instructions	0xF7 ~ 0xF0

Table 2-2 Address Mapping for SFR (Processor Core Related)

- ➢ Stack Pointer
  The FP51-1T supports 16-bit stack pointer, for which the lower 8 bits are stored in SPL while the higher 8-bits are in SPH.

- DPTR (Data Pointer)
  The XDATA can be accessed with a 16-bit data pointer (DPTR). The lower 8 bits of DPTR are stored in DPL while the higher 8 bits are in DPH.

- PSW (Program Status Word)
  The bits of PSW are defined in Table 2-3.

Bits	Name	Default	Description
0	CY	0	Carry flag. This flag will be set after an arithmetic instruction
1	AC	0	Auxiliary carry flag. This flag will be set if there is a carry from the lower nibble to the higher nibble during ADD / SUB operation.
2	F0	0	Reserved for future expansion
4 : 3	RS	0	Bank selection for general purpose registers (Table 2-1)
5	OV	0	Overflow flag
6	UD	0	Reserved for future expansion
7	P	0	Parity bit, P = ^ACC

Table 2-3 Bit Definition for PSW

- MCU_REVISION
  Currently only Revision A0 is supported.

- XPAGE
  As mentioned early, the XDATA can be accessed with a 16-bit address. One way to form this 16-bit address is to use the DPTR. The other way is to use a page window, for which the higher 8 bits of the address (page index) is stored in the XPAGE register, and the lower 8 bits are stored in a general-purpose register (R0 – R7). In this way, the XDATA can be addressed by MOVX instruction with a page size of 256 bytes.

Note: As mentioned early, the SFR occupies the address from 0x80 to 0xFF. In the classic 8051, the internal RAM that overlaps with SFR can still be accessed through indirect addressing mode, such as "ADD A, @R0". In this way, there could be a saving of 128 bytes of memory. But for modern day FPGAs, block RAM is no longer a scarce resource. The FP51-1T thus chooses NOT to support such address overlay in favor of concise design and higher clock rate. In other words, when it comes to the address from 0x80 to 0xFF, all the addressing mode will get funneled to the SFR. In the eyes of classic 8051, the FP51-1T can equivalently be viewed as if it only carries 128 bytes of internal RAM and a large sum of external RAM.

2.1.1.2 Pipeline

Notwithstanding the fact that 8051 has a CISC ISA, it is still possible to translate those instructions into microoperations and apply a RISC implementation on top of it (Ref [2]). And this is exactly the approach that FP51-1T is taken. The first 8051 came out with 12 clock cycles per instruction (called 12T). With a RISC implementation like FP51-1T, the instruction can be executed at a maximum throughput of 1 clock cycle per instruction. (That's why it has a "1T" in its name.)

As illustrated in Figure 2-1, the FP51-1T is implemented with a 5-stage pipeline. The 5 stages are:

- Instruction Fetch
- Instruction Decode for Set 1
- Memory Read
- Instruction Decode for Set 2
- Execution

The instructions are divided into two sets, set 1 and set 2. Those instructions in set 2 don't have to access the memory and are less complicated than those in set 2. And they are decoded separately to balance the work load so that higher clock rate can be achieved. The data memory, including those registers, are implemented as a simple dual port memory (one read-port, one write-port). In this way, the Memory Read Stage and the Execute State can work simultaneously to avoid structural hazards in pipeline (Ref [4]).

And the data hazards are handled like the following:

- Automatically insert NOPs if data dependency is detected between instructions
  For example, in the following code
  MOV R0, R1
  ADD A, @R0
  The read address of the second instruction cannot be immediately determined until the first instruction is fully executed. In this case, the FP51-1T will automatically insert NOPs between those two instructions

- Data Forwarding
  If the write address of the first instruction is the read address of the second instruction, the result of the first instruction will be forwarded to the second instruction through a bypass path, as illustrated in Figure 2-1. In this way, the pipeline can be kept running without stalling. An example of such case is the following:
  INC R1
  ADD A, R1
  In the above example, the *R1* will be updated with data forwarding without any pipeline stall.

And pipeline does have to be flushed for interrupt or branch instructions.

2.1.1.3 OCD (On-chip Debugger)

The FP51-1T comes with an OCD, which can be used to load code into the code memory. And it can also be used to setup breakpoints to pause the pipeline. The OCD supports a serial interface for frame format. Practically the OCD is often connected through a UART/USB bridge to a host PC for interaction.

2.1.2 ISA (Instruction Set Architecture)

8051 ISA is widely adopted and has a large ecosystem. Unlike AVR or ARM, 8051 is NOT beholden to a single company. In fact, you can find a plethora of vendors that supply various flavors of enhanced 8051 cores world-wide. (As of today, there are more than 70 of them.). Part of its popularity comes from the fact that it is not burdened by license/royalty/patent, as it has fallen into public domain thanks to its longevity and vitality. That's why it is picked for FP51-1T as a good option of open source implementation.

2.1.2.1 Addressing Mode

For an ISA, it always has various ways to specify the operand, which is called addressing mode. For 8051, there are 5 different address modes:

1. The Immediate Data

   In this mode, the operand is embedded in the instruction itself, like the following:

   ADD A, #99 ; A <= A + 99

   In the above example, the immediate value 99 is added to the accumulator.

2. The Direct Addressing Mode

   In this mode, the operand is specified by a memory address embedded in the instruction, like the following:

   ADD A, 99 ; A <= A + (99)

   In the above example, the content of address 99 is added to the accumulator.

3. The Register Direct Addressing Mode

   In this mode, the operand's value is stored in a general-purpose register, like the following:

   ADD A, R1 ; A <= A + R1

   In the above example, the data stored in R1 is added to the accumulator.

4. The Register Indirect Addressing Mode

   In this mode, the operand's memory address is stored in a general-purpose register, like the following:

   ADD A, @R1 ; A <= A + (R1)

   In the above example, the data stored in R1 will be used as memory address, and data will be read out from this address and added to the accumulator.

5. Offset Addressing Mode

   In this mode, the operand's address is obtained by adding two registers together, like the following:

   MOVC A, @A + DPTR ; A <= (A + DPTR)

   In the above example, the memory address is obtained by adding A and DPTR. Data will be read out from this address and saved in the accumulator.

2.1.2.2 Bit Addressable Location

Some of the memory locations in IDATA can be manipulated bit-wise by those instructions listed in Table 2-7. These memory locations are:

- memory address 0x20 ~ 0x2F
  There are total 128 bits in this region by little endian addressing. Namely the LSB of address 0x20 is mapped to bit address 0, while the MSB of address 0x2F is mapped to bit address 0x7F.

- Some of the SFR address
  For those processor-core-related SFRs, their bit address mapping is shown in Table 2-2. For those peripheral-related-SFRs, their bit address mapping will be discussed later in Section 2.2.2.

2.1.2.3 Instruction Definition

With the above 5 addressing modes, there are total of 255 different instruction operations. And these instructions can be classified into the following 5 categories:

- Arithmetic Instructions
- Logical Instructions
- Data Transfer Instructions
- Instructions for Bit Operations
- Branching Instructions

And those instructions are detailed below in Table 2-4, Table 2-5, Table 2-6, Table 2-7 and Table 2-8. Fortunately, users don't have to program directly in the assembly language. With the Arduino compatible library provided by PulseRain Technology, users can program with more user-friendly Arduino language, as later sections will demonstrate.

Mnemonics	Opcode	Bytes	Cycles	Description
INC A	0x04	1	1	A <= A + 1
INC direct	0x05	2	1	(direct) <= (direct) + 1
INC @Ri (i = 0 ~ 1)	0x06 ~ 0x07	1	1	(Ri) <= (Ri) + 1
INC Ri (i = 0 ~ 7)	0x08 ~ 0x0F	1	1	Ri <= Ri + 1
INC DPTR	0xA3	1	1	DPTR <= DPTR + 1
DEC A	0x14	1	1	A <= A – 1
DEC direct	0x15	2	1	(direct) <= (direct) – 1
DEC @Ri (i = 0 ~ 1)	0x16 ~ 0x17	1	1	(Ri) <= (Ri) – 1
DEC Ri (i = 0 ~ 7)	0x18 ~ 0x1F	1	1	Ri <= Ri – 1
ADD A, const	0x24	2	1	A <= A + const
ADD A, direct	0x25	2	1	A <= A + (direct)
ADD A, @Ri (i = 0 ~ 1)	0x26 ~ 0x27	1	1	A <= A + (Ri)
ADD A, Ri (i = 0 ~ 7)	0x28 ~ 0x2F	1	1	A <= A + Ri
ADDC A, const	0x34	2	1	A <= A + const + CY
ADDC A, direct	0x35	2	1	A <= A + (direct) + CY

Mnemonics	Opcode	Bytes	Cycles	Description
ADDC A, @Ri (i = 0 ~ 1)	0x36 ~ 0x37	1	1	A <= A + (Ri) + CY
ADDC A, Ri (i = 0 ~ 7)	0x38 ~ 0x3F	1	1	A <= A + Ri + CY
SUBB A, const	0x94	2	1	A <= A - const – CY
SUBB A, direct	0x95	2	1	A <= A - (direct) – CY
SUBB A, @Ri (i = 0 ~ 1)	0x96 ~ 0x97	1	1	A <= A - (Ri) – CY
SUBB A, Ri (i = 0 ~ 7)	0x98 ~ 0x9F	1	1	A <= A - Ri – CY
DIV AB	0x84	1	9	A <= higher_byte(A / B), B <= lower_byte (A / B)
MUL AB	0xA4	1	3	A <= lower_byte(A * B), B <= higher_byte (A * B)
DA A	0xD4	1	5	Decimal Adjust for BCD (See Ref [1])

Table 2-4 Arithmetic Instructions

Mnemonics	Opcode	Bytes	Cycles	Description
ORL direct, A	0x42	2	1	(direct) <= (direct) or A
ORL direct, const	0x43	3	1	(direct) <= (direct) or const
ORL A, const	0x44	2	1	A <= A or const
ORL A, direct	0x45	2	1	A <= A or (direct)
ORL A, @Ri (i = 0 ~ 1)	0x46 ~ 0x47	1	1	A <= A or (Ri)
ORL A, Ri (i = 0 ~ 7)	0x48 ~ 0x4F	1	1	A <= A or Ri
ANL direct, A	0x52	2	1	(direct) <= (direct) and A
ANL direct, const	0x53	3	1	(direct) <= (direct) and const
ANL A, const	0x54	2	1	A <= A and const
ANL A, direct	0x55	2	1	A <= A and (direct)
ANL A, @Ri (i = 0 ~ 1)	0x56 ~ 0x57	1	1	A <= A and (Ri)
ANL A, Ri (i = 0 ~ 7)	0x58 ~ 0x5F	1	1	A <= A and Ri
XRL direct, A	0x62	2	1	(direct) <= (direct) xor A
XRL direct, const	0x63	3	1	(direct) <= (direct) xor const
XRL A, const	0x64	2	1	A <= A xor const
XRL A, direct	0x65	2	1	A <= A xor (direct)
XRL A, @Ri (i = 0 ~ 1)	0x66 ~ 0x67	1	1	A <= A xor (Ri)
XRL A, Ri (i = 0 ~ 7)	0x68 ~ 0x6F	1	1	A <= A xor Ri
CLR A	0xE4	1	1	Clear Accumulator
CPL A	0xF4	1	1	Invert Accumulator
RL A	0x23	1	1	A <= {A[6 : 0}, A[7]}
RLC A	0x33	1	1	{A, C} <= {A[6 : 0], C, A[7]}
RR A	0x03	1	1	A <= {A[0], A[7 : 1]}
RRC A	0x13	1	1	{C, A} <= {A[0], C, A[7 : 1]}
SWAP A	0xC4	1	3	A <= {A[3 : 0], A[7 : 4]}

Table 2-5 Logical Instruction

Mnemonics	Opcode	Bytes	Cycles	Description
MOV A, const	0x74	2	1	A <= const
MOV direct, const	0x75	3	1	(direct) <= const
MOV @Ri, const (i = 0 ~ 1)	0x76 ~ 0x77	2	1	(Ri) <= const
MOV Ri, const (i = 0 ~ 7)	0x78 ~ 0x7F	2	1	Ri <= const
MOV direct1, direct2	0x85	3	1	(direct1) <= (direct2)
MOV direct, @Ri (i = 0 ~ 1)	0x86 ~ 0x87	1	1	(direct) <= (Ri)
MOV direct, Ri	0x88 ~ 0x8F	2	1	(direct) <= Ri
MOV @Ri, direct (i = 0 ~ 1)	0xA6 ~ 0xA7	2	1	(Ri) <= (direct)
MOV Ri, direct	0xA8 ~ 0xAF	2	1	Ri <= (direct)
MOV A, direct	0xE5	1	1	A <= (direct)
MOV A, @Ri (i = 0 ~ 1)	0xE6 ~ 0xE7	1	1	A <= (Ri)
MOV A, Ri (i = 0 ~ 7)	0xE8 ~ 0xEF	1	1	A <= Ri
MOV direct, A	0xF5	2	1	(direct) <= A
MOV @Ri, A (i = 0 ~ 1)	0xF6 ~ 0xF7	1	1	(Ri) <= A
MOV Ri, A	0xF8 ~ 0xFF	1	1	Ri <= A
MOV DPTR, const	0x90	3	1	(DPTR) <= const
MOVC A, @A + PC	0x83	1	6	A <= (A + PC) // move code into Accumulator
MOVC A, @A + DPTR	0x93	1	6	A <= (A + DPTR) // move code into Accumulator
MOVX A, @DPTR	0xE0	1	1	A <= (DPTR)
MOVX A, @Ri (i = 0 ~ 1)	0xE2 ~ 0xE3	1	1	A <= (Ri)
MOVX @DPTR, A	0xF0	1	1	(DPTR) <= A
MOVX @Ri, A (i = 0 ~ 1)	0xF2 ~ 0xF3	1	1	(Ri) <= A
XCH A, direct	0xC5	2	1	A <=> (direct)
XCH A, @Ri (i = 0 ~ 1)	0xC6 ~ 0xC7	1	1	A <=> (Ri)
XCH A, Ri	0xC8 ~ 0xCF	1	1	A <=> Ri
XCHD A, @Ri (i = 0 ~ 1)	0xD6 ~ 0xD7	1	3	A[3 : 0] <=> (Ri)[3 : 0]
PUSH direct	0xC0	2	1	push direct data onto stack
POP direct	0xD0	2	1	pop direct data onto stack

Table 2-6 Data Transfer Instruction

Mnemonics	Opcode	Bytes	Cycles	Description
CLR bit	0xC2	2	1	(bit) <= 0
CLR C	0xC3	1	1	CY <=0
SET bit	0xD2	2	1	(bit) <= 1
SET C	0xD3	1	1	CY <=1
CPL bit	0xB2	2	1	(bit) <= ~(bit)
CPL C	0xB3	1	1	CY <= ~CY
ANL C, bit	0x82	2	1	CY <= CY and (bit)
ANL C, /bit	0xB0	2	1	CY <= CY and (not (bit))
ORL C, bit	0x72	2	1	CY <= CY or (bit)
ORL C, /bit	0xA0	2	1	CY <= CY or (not (bit))
MOV C, bit	0xA2	2	1	CY <= (bit)
MOV bit, C	0x92	2	1	(bit) <= CY

JC rel	0x40	2	2/9	If (CY == 1) then
				(PC) <= (PC) + rel + 2
				else
				(PC) <= (PC) + 2
				end if

JNC rel	0x50	2	2/9	If (CY == 0) then
				(PC) <= (PC) + rel + 2
				else
				(PC) <= (PC) + 2
				end if

JB bit, rel	0x20	3	2/9	If ((bit) == 1) then
				(PC) <= (PC) + rel + 2
				else
				(PC) <= (PC) + 2
				end if

JNB bit, rel	0x30	3	2/9	If ((bit) == 0) then
				(PC) <= (PC) + rel + 2
				else
				(PC) <= (PC) + 2
				end if

JBC bit, rel	0x10	3	2/9	If ((bit) == 1) then
				(PC) <= (PC) + rel + 3
				(bit) <= 0
				else
				(PC) <= (PC) + 3
				end if

Table 2-7 Instructions of Bit Operation

Mnemonics	Opcode	Bytes	Cycles	Description
ACALL addr11	8'b???1_0001	2	9	Absolute Call (See Ref [1] for more detail.)
LCALL addr16	0x12	3	9	Long Call (See Ref [1] for more detail.)
RET	0x22	1	10	Return from subroutine call
RETI	0x32	1	10	Return from Interrupt
AJMP addr11	8'b???0_0001	2	9	Absolute Jump (See Ref [1] for more detail.)
LJMP addr16	0x02	3	9	Long Jump (See Ref [1] for more detail.)
SJMP rel	0x80	2	1	Short Jump (See Ref [1] for more detail.)
JMP @A + DPTR	0x73	1	9	Indirect Jump (See Ref [1] for more detail.)
JZ rel	0x60	2	2/9	Jump if (A == 0)
JNZ rel	0x70	2	2/9	Jump if (A != 0)
CJNE A, const, rel	0xB4	3	2/9	Subtract immediate data from Accumulator, modify carry bit, and jump if they are not equal
CJNE A, direct, rel	0xB5	3	2/9	Subtract direct data from Accumulator, modify carry bit, and jump if they are not equal
CJNE @Ri, cont, rel (i = 0 ~ 1)	0xB6 ~ 0xB7	3	2/9	Subtract immediate data from indirect data, modify carry bit, and jump if they are not equal
CJNE Ri, const, rel (i = 0 ~ 7)	0xB8 ~ 0xBF	3	2/9	Subtract immediate data from register, modify carry bit, and jump if they are not equal
DJNZ direct, rel	0xD5	3	2/9	Decrement direct data, and jump if not zero
DJNZ Ri, Rel (i = 0 ~ 7)	0xD8 ~ 0xDF	2	2/9	Decrement register, and jump if not zero
NOP	0x0	1	1	No Operation

Table 2-8 Branching Instruction

2.2 Peripherals

2.2.1 Wishbone FASM Interface

As illustrated in Figure 1-1, all the peripheral registers for FT51-1T are mapped into the SFR address space, and are accessed through Wishbone FASM interface (Ref [5]). All peripherals share the same Wishbone bus, and the processor core is the only master on the bus.

For master read, the signals are as following on the master side:

Signal Name	Description
RD_STB_O	Read Strobe, always high for FP51-1T
RD_ADR_O	Read Address, 8-bit shared signal
RD_DAT_I	8-bit read data. The data from each peripheral are mux-ed into RD_DAT_I. And the RD_DAT_I should be valid on the next cycle after RD_ADDR_O is provided. If long read latency is expected, software should insert NOP to compensate for the read delay.
RD_ACK_I	Read Acknowledge from the slave. For FP51-1T, this signal it not used and can be tied to 1

Table 2-9 Wishbone FASM Read Signal

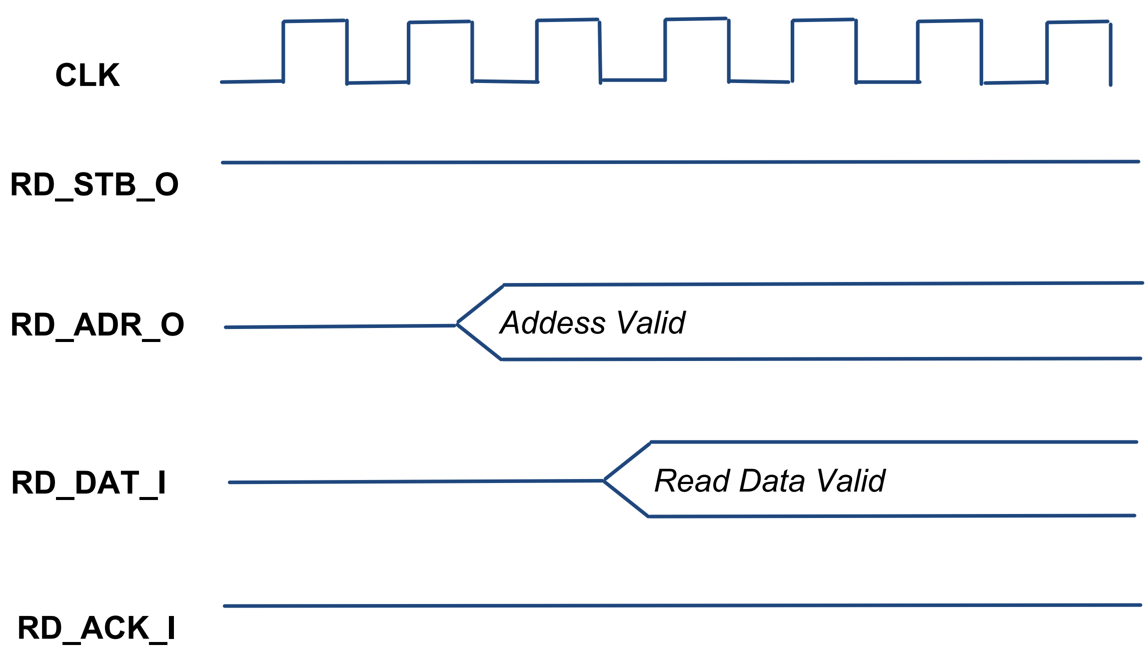

Figure 2-2 Timing Diagram for Master Read

For master write, the signals are as following on the master side:

Signal Name	Description
WR_STB_O	Write Strobe, always high for FP51-1T
WR_WE_O	Write Enable
WR_ADR_O	Write Address, 8-bit shared signal
WR_DAT_O	8-bit write data, available at the same time as WR_ADR_O

Table 2-10 Wishbone FASM Write Signal

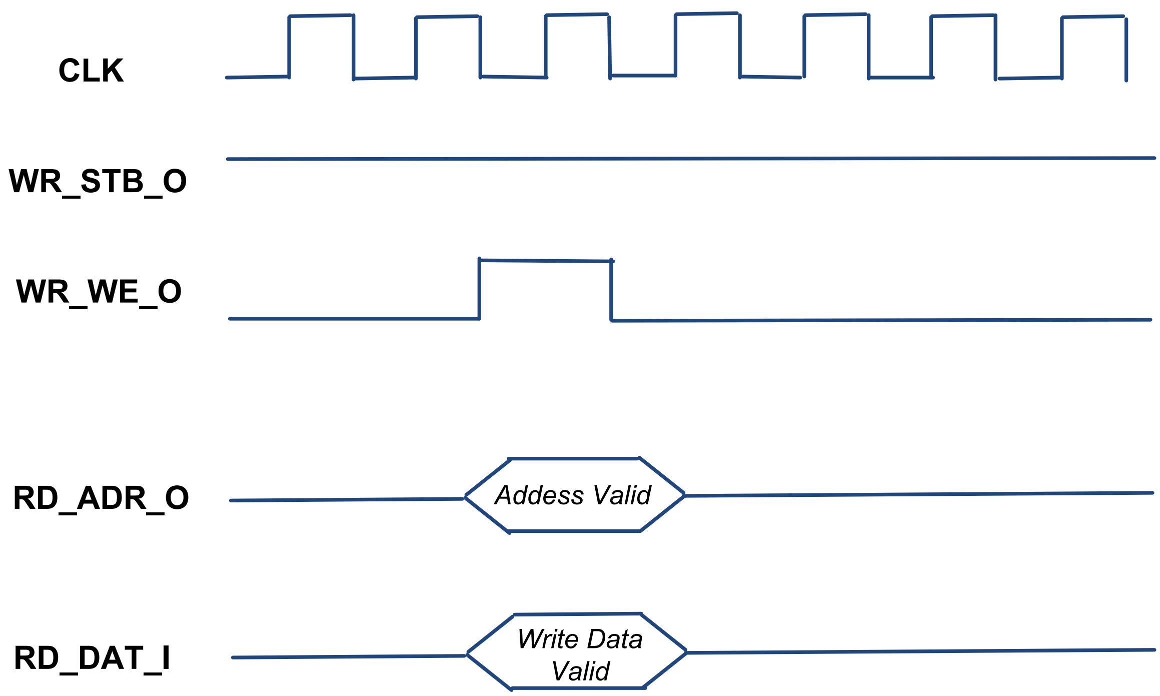

Figure 2-3 Timing Diagram for Master Write

2.2.2 SFR (Special Function Register)

2.2.2.1 Peripheral Address Mapping

As mentioned early, all the peripherals have their registers exposed to the processor core as SFR, and these SFRs are accessed through Wishbone FASM interface. For FP51-1T Rev A0, those peripheral registers are mapped into the SFR with the following addresses shown in Table 2-11:

Address	Register Name	Description	Bit Address (MSB ~ LSB)
0x80	P0	IO Port 0	0x87 ~ 0x80
0x88	TCON	Timer Control	0x8F ~ 0x88
0x89	TMOD	Timer Mode	N/A
0x8A	TL0	Timer 0 lower byte	N/A
0x8B	TL1	Timer 1 lower byte	N/A
0x8C	TH0	Timer 1 higher byte	N/A
0x8D	TH1	Timer 1 higher byte	N/A
0x90	P1	IO Port 1	0x97 ~ 0x90
0x98	SCON	UART Control (Serial Port Control)	0x9F ~ 0x98
0x99	SBUF	In / Out buffer for the UART (Serial Port)	N/A
0x9A	SCON_AUX	Control for the auxiliary UART	N/A
0x9B	SBUF_AUX	In / Out buffer for the auxiliary UART	N/A
0xA0	P2	IO Port 2	0xA7 ~ 0xA0
0xA8	IE	Interrupt Enable (See Table 2-13 for more detail.)	0xAF ~ 0xA8
0xB0	P3	IO Port 3	0xB7 ~ 0xB0
0xB8	IP	Interrupt Priority (See Table 2-14 for more detail.)	0xBF ~ 0xB8
0xC0	DEBUG-COUNTER-LED	See Section 2.2.14.	N/A
0xD1	I2C_CSR	Registers for I2C. See Ref [6] for more detail.	N/A
0xD2	I2C_ADDR-DATA		N/A
0xD3	PWM_CSR	Registers for PWM. See Ref [7] for more detail.	N/A
0xD4	PWM_DATA		N/A
0xD7	SD_CSR	Register for microSD, See Ref [8] for more detail.	N/A
0xD8	SD_CMD		N/A
0xD9	SD_ARG0		N/A
0xDA	SD_ARG1		N/A
0xDB	SD_ARG2		N/A
0xDC	SD_ARG3		N/A
0xDD	SD_BUF		N/A
0xDE	SD_DATA_IN		N/A
0xDF	SD_DATA_OUT		N/A
0xE8	CODEC_WRITE_DATA_LOW	Register for voice CODEC (Silicon Lab Si3000). See Ref [9] for more detail.	N/A
0xE9	CODEC_WRITE_DATA_HIGH		N/A
0xEA	CODEC_READ_DATA_LOW		N/A
0xEB	CODEC_READ_DATA_HIGH		N/A
0xEC	CODEC_CSR		N/A
0xED	CHIP_ID_DATA_CSR	Chip ID for Intel MAX 10 FPGA.	N/A
0xF1	P0_DIRECTION	IO direction for Port 0	N/A

Address	Register Name	Description	Bit Address (MSB ~ LSB)
0xF2	P1_DIRECTION	IO direction for Port 1	N/A
0xF3	P2_DIRECTION	IO direction for Port 2	N/A
0xF4	P3_DIRECTION	IO direction for Port 3	N/A
0xF5	ADC_DATA_HIGH	Register for the on-chip ADC in Intel MAX 10 FPGA. See Ref [10] for more detail.	N/A
0xF6	ADC_DATA_LOW		N/A
0xF7	ADC_CSR		N/A
0xF8	JTAG_UART	Register for JTAG UART. See Ref [11] for more detail.	N/A
0xF9	SRAM_INSTRUCTION		N/A
0xFA	SRAM_DATA	Register for SPI SRAM (Microchip 23LC1024). See Ref [12] for more detail.	N/A
0xFB	SRAM_ADDRESS2		N/A
0xFC	SRAM_ADDRESS1		N/A
0xFD	SRAM_ADDRESS0		N/A
0xFE	SRAM_CSR		N/A

Table 2-11 Peripheral Address Mapping

2.2.2.2 Bit Addressable Location in SFR

As shown in the last column of Table 2-11, some of the SFRs are bit addressable, which means they can be manipulated bit-wise by those instructions listed in Table 2-7.

2.2.3 Interrupt Controller

2.2.3.1 Interrupt Vector

As shown in Figure 1-1, there is an interrupt controller external to the process core. This interrupt controller will collect IRQ request from multiple sources, and generate interrupt vector addresses for the processor core. For FP51-1T Rev A0, 7 interrupt sources are supported, and the interrupt vector addresses are arranged as the following in Table 2-12:

ISR Address (in Code Memory)	Interrupt Source	Description
0x0000	Power-on Reset	Put a long jump here to jump to the start of user program
0x0003	External INT0	The INT0 pin of the MCU, level signal expected
0x000B	Timer0	Pulse Signal
0x0013	I2C Slave	Interrupt for I2C Slave controller, level signal expected
0x001B	Timer1	Pulse Signal
0x0023	UART	Level Signal
0x002B	ADC	Level Signal
0x0033	CODEC (Si3000)	Pulse Signal

Table 2-12 Interrupt Vector Table

2.2.3.2 Interrupt Enable / Disable

The 7 interrupt sources can be enabled/disabled both individually and globally by a SFR called IE (Interrupt Enable). The bit map of the IE register is detailed below in Table 2-13.

Byte Address	Bits Index	Bit Address	Default	Description
0xA8	0	0xA8	0	Enable bit for External INT0
	1	0xA9	0	Enable bit for Timer0 Interrupt
	2	0xAA	0	Enable bit for I2C slave interrupt
	3	0xAB	0	Enable bit for Timer1 Interrupt
	4	0xAC	0	Enable bit for UART Interrupt
	5	0xAD	0	Enable bit for interrupt from on-chip ADC
	6	0xAE	0	Enable bit for CODEC (Si3000) Interrupt
	7	0xAF	0	Global Interrupt Enable bit

Table 2-13 Bit Map for SFR: IE (Interrupt Enable)

2.2.3.3 Interrupt Priority

The interrupt control of FP51-1T supports two priority levels: high (1) and low (0). And each interrupt's priority can be configured individually by a SFR called IP (Interrupt Priority). The bit map of the IP register is detailed below in Table 2-14.

Byte Address	Bits Index	Bit Address	Default	Description
0xB8	0	0xB8	0	Priority bit for External INT0
	1	0xB9	0	Priority bit for Timer0 Interrupt
	2	0xBA	0	Priority bit for I2C slave interrupt
	3	0xBB	0	Priority bit for Timer1 Interrupt
	4	0xBC	0	Priority bit for UART Interrupt
	5	N/A	0	Priority bit for interrupt from on-chip ADC
	6	N/A	0	Priority bit for CODEC (Si3000) Interrupt
	7	N/A	0	Reserved

Table 2-14 Bit Map for SFR: IP (Interrupt Priority)

Interrupts with low priority can be overridden by those with high priority. When two interrupts with the same priority hit simultaneously, they will be served in a round robin fashion.

2.2.4 IO Port

The FP51-1T MCU has 4 IO ports, and each port is 8-bit wide. They are all bit-addressable, as shown in Table 2-11.

These ports are bi-directional, and the direction for each IO pin is controlled by the SFR **Px_DIRECTION**, where x = 0 ~ 3. A bit 1 in Px_DIRECTION will set the correspondent IO pin of Port x as output, while a bit 0 will set the IO pin as input.

2.2.5 Chip ID

For Intel/Altera MAX 10 FPGA, each device has a 64-bit chip ID that can uniquely identify the device, and this 64-bit number can be obtained by using the "Altera_unique_chip_ID" IP core. For FP51-1T Rev A0 MCU, a Wishbone wrapper has been provided to integrate this IP core into the peripheral bus, with the following register mapping:

Address	R/W	Register Name	Description
0xED	RW	CHIP_ID_DATA_CSR	Chip ID for Intel MAX 10 FPGA.

Table 2-15 Address Map for SFR: CHIP_ID_DATA_CSR

To obtain the 64-bit Chip ID, the following operations need to be carried out:

1. Write 0xFF to CHIP_ID_DATA_CSR
2. Read CHIP_ID_DATA_CSR for 8 consecutive times to read out the 64-bit Chip ID, with the MSB coming out first.

2.2.6 I2C

I2C protocol, both master and slave mode, is supported by FP51-1T Rev A0. The details of the PulseRain I2C controller and the correspondent software library can be found in

Ref [6]: PulseRain M10 – I2C, Technical Reference Manual, Doc# TRM-0922-01007, Rev 1.0.0, 09/2017 https://github.com/PulseRain/M10I2C/raw/master/extras/M10_I2C_TRM.pdf

2.2.7 PWM

By default, the FP51-1T Rev A0 has 6 PWM output that can be configured individually. The details of the PulseRain PWM controller and the correspondent software library can be found in

Ref [7]: PulseRain M10 – PWM, Technical Reference Manual, Doc# TRM-0922-01009, Rev 1.0.1, 10/2017, https://github.com/PulseRain/M10PWM/raw/master/extras/M10_PWM_TRM.pdf

2.2.8 microSD

The SPI mode of Secure Digital Card is supported by FP51-1T Rev A0, for which a hardware microSD controller is offered by PulseRain Technology. This microSD controller has 1KB (2 disk sectors) of data buffer for ping/pong operation. On top of that, the correspondent software library also supports read/write at file system level. For more information, please refer to

Ref [8]: PulseRain M10 – microSD, Technical Reference Manual, Doc# TRM-0922-01006, Rev 1.0.0, 09/2017 https://github.com/PulseRain/M10SD/raw/master/extra/M10_SD_TRM.pdf

2.2.9 Voice CODEC (Silicon Lab Si3000)

The FP51-1T Rev A0 also supports a voice CODEC from Silicon Lab, with the part number of Si3000. The details of the CODEC controller and the correspondent software library can be found in

Ref [9], PulseRain M10 – Voice CODEC, Technical Reference Manual, Doc# TRM-0922-01001, Rev 1.0.3, 09/2017, https://github.com/PulseRain/M10CODEC/raw/master/extras/M10_CODEC_TRM.pdf

In addition, a DTMF decoder library has been offered based on the CODEC hardware and software. And its correspondent details can be found in

Ref [13], PulseRain M10 – DTMF, Technical Reference Manual, Doc# TRM-0922-01002, Rev 1.0.0, 09/2017 https://github.com/PulseRain/M10DTMF/raw/master/extras/M10_DTMF_TRM.pdf

2.2.10 ADC (Analog to Digital Converter)

Some of the Intel/Altera MAX10 FPGAs have built-in ADCs, and the converted results can be obtained by using the IP core provided by Intel/Altera. Accordingly, PulseRain Technology has provided a Wishbone wrapper in FP51-1T Rev A0 to integrate the ADC IP core into peripheral bus. And the details of the ADC hardware/software can be found in

Ref [10]: PulseRain M10 – ADC, Technical Reference Manual, Doc# TRM-0922-01003, Rev 1.0.0, 09/2017 https://github.com/PulseRain/M10ADC/raw/master/extra/M10_ADC_TRM.pdf

2.2.11 JTAG UART

As an auxiliary way of debugging, the JTAG port of Intel/Altera MAX10 FPGAs can also be used as a UART. The FP51-1T Rev A0 contains a Wishbone wrapper that supports the JTAG UART function. The details can be found in

Ref [11], PulseRain M10 – JTAG, Technical Reference Manual, Doc# TRM-0922-01008, Rev 1.0.0, 09/2017 https://github.com/PulseRain/M10JTAG/raw/master/extras/M10_JTAG_TRM.pdf

2.2.12 Serial SRAM (Microchip 23LC1024)

The FPT51-1T Rev A0 supports a serial SRAM (128KB) from Microchip, with the part number of 23LC1024. The details of the SRAM controller and the correspondent software library can be found in

Ref [12]: PulseRain M10 – SRAM, Technical Reference Manual, Doc# TRM-0922-01004, Rev 1.0.0, 09/2017 https://github.com/PulseRain/M10SRAM/raw/master/extras/M10_SRAM_TRM.pdf

2.2.13 UART

The FP51-1T Rev A0 supports two UARTs, both can reach a baud rate up to 921600 bps. The details of the UART controller and the correspondent software library can be found in

Ref [14]: PulseRain M10 – Serial Port, Technical Reference Manual, Doc# TRM-0922-01005, Rev 1.0.0, 09/2017, https://github.com/PulseRain/M10SerialAUX/raw/master/extras/M10_Serial_TRM.pdf

2.2.14 Debug-Counter-LED

To help debug, the FP51-1T Rev A0 has a register called DEBUG-COUNTER-LED, with the address of 0xC0 (Table 2-11). It is actually a combination of watch dog timer and LED flashing. Internally there are two counters rolling:

1. The flashing counter, which controls the frequency of the LED flashing. And it is a 27-bit free rolling counter. So if the LED ever flashes, it will flash at a frequency of 96Mhz / (2**27) = 0.7 Hz.

2. The watch dog counter, which is a 16-bit value. When the DEBUG-COUNTER-LED register is being written with an 8-bit value Y, the watch dog counter will be loaded with a value of {Y, 8'hFE}. And the watch dog counter will also keep incrementing until it reaches 16'hFFFF, which will trigger the watch dog and stop the flashing of LED. To reset the watch dog that has been triggered, simply write zero the DEBUG-COUNTER-LED register.

Thus, to avoid "being bitten by the dog", it is suggested to write to the DEBUG-COUNTER-LED periodically with a small value (such as 1). And the LED is supposed to keep flashing under normal circumstance.

2.3 OCD (On-Chip Debugger, beta)

2.3.1 Overview

As shown in Figure 1-1 and Figure 2-4, an optional OCD can be attached to the MCU to access the code RAM and obtain MCU's internal status.

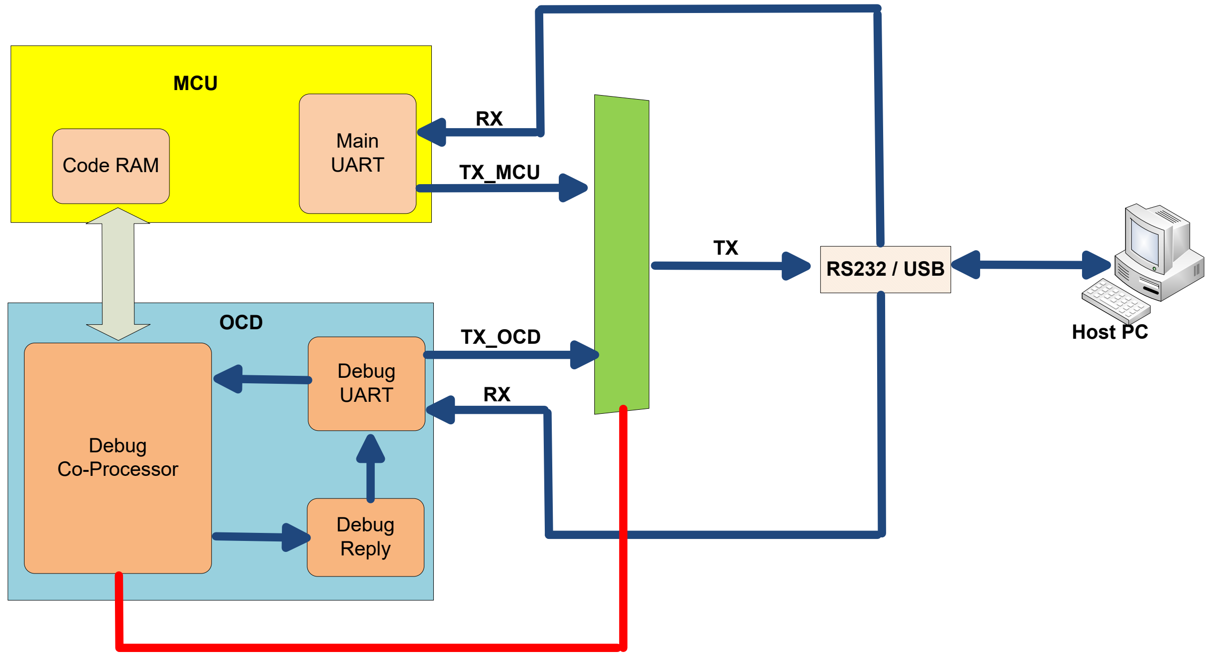

Figure 2-4 OCD (On-Chip Debugger)

The MCU and the OCD can share the same UART port, as illustrated in Figure 2-4. The RX signal goes to both the MCU and OCD, while the TX signal has to go through a mux. And that mux is controlled by the OCD. After power-on reset, the MCU has the TX by default. But a valid debug frame sending from the host PC can let OCD to reconfigure the mux and switch the TX to OCD, for which the code RAM and MCU's internal status can be accessed and controlled.

Unlike Arduino, the FP51-1T MCU does not use software bootloader to interact with the host PC, because its hardware OCD can take on chores like code downloading. Thus, the precious code RAM can all be dedicated to doing the real job.

2.3.2 Debug Frame Format

The OCD takes debug frames from the host PC, and reply with an ACK frame. For each transaction, the host PC is always the initiator of the transaction.

The debug frame (from host PC to OCD) has the format shown in Table 2-6:

Byte Index	Description
0	Sync Word, always 0x5A
1	Sync Word, always 0xA5
2	Sync Word, always 0x01
3	Frame Type
4 ~ 9	Payload
10 ~ 11	CRC16

Table 2-16 The Format of Debug Frame

The frame type in Table 2-6 has 8 bits. Among the 8 bits, bit 7 ~ bit 1 are the 7 bits value of frame type. Bit 0 is supposed to toggle between frames, so that a missing frame can be detected.

The valid frame type values (bit 7 ~ bit 0) are as following:

- 7'h6D: 4-byte Code Ram Read
  In this case, byte 4 in Table 2-6 has the higher 8-bits of the 16-bit memory address. And byte 5 has the lower 8-bits of the address.

- 7'h2D: Pause On with ACK
- 7'h3D: Pause Off with ACK
  Those two frames will pause or resume the processor core. The payload in this frame is not used.

- 7'h7D: Break On with ACK
- 7'h1D: Break Off with ACK
  Those two frames will enable/disable hardware breakpoint in the processor core. The hardware can support up to two hardware break points. And the addresses of those two break points are as following:
  code address for break point A: byte 4 – byte 5, where byte 4 has the MSB while byte 5 has the LSB
  code address for break point B: byte 8 – byte 9, where byte 4 has the MSB while byte 5 has the LSB
  byte 6 – byte 7 are not used.

- 7'h4D: Processor reset with ACK
  Reset the process core and set the PC (Program Counter) to zero. The payload in this frame is not used.

- 7'h45: RUN Pulse with ACK
  Use this frame for single step execution. The payload in this frame is not used.

- 7'h2F: Read CPU Status
  Use this frame to read CPU status, which includes the following:
    - PC (Program Counter)
    - Processor stall flag

  Please see the correspondent ACK frame in Table 2-21 for more detail.

- 7'h5D: 4-Byte Code RAM write with ACK
  Use this frame to write 4 bytes of data into code RAM, with the following format for payload (byte 4 – 9 in Table 2-16).

Payload Byte Index	Description
0	The higher 8-bits of the 16-bit code memory address
1	The lower 8-bits of the 16-bit code memory address
2	Bit 31 – bit 24 of the 32-bit data
3	Bit 23 – bit 16 of the 32-bit data
4	Bit 15 – bit 8 of the 32-bit data
5	Bit 7 – bit 0 of the 32-bit data

Table 2-17 The Payload Format of 4-byte Write Frame

- 7'h5B: 128 Byte Code RAM write with ACK
  Use this frame to write 128 bytes of data into code RAM, with the following format

Byte Index	Description
0	Sync Word, always 0x5A
1	Sync Word, always 0xA5
2	Sync Word, always 0x01
3	Frame Type {7'h5B, toggle_bit}
4 ~ 5	16-bit code RAM address. Byte 4 is the
6 ~ 9	byte 0 – 3 of the total 128 payload
10 ~ 11	CRC16 of byte 0 ~ 9
12 ~ 135	byte 4 – 127 of the total 128 payload
136 ~ 137	CRC16 of byte 12 ~ 135

Table 2-18 The Format of 128-byte Write Frame

- 7'h6F: Read Data Memory
- 7'h2B: Write Data Memory
  Reserved for Future Expansion

- 7'h2A: UART Select
  Use this frame to switch the UART TX pin between MCU and OCD, as illustrated in Figure 2-4. The payload format of this frame is as following:

Payload Byte Index	Description
0 ~ 4	Not used
5	The bit 1 of this byte determines the mux setting in Figure 2-4. Set bit 1 to low will let the MCU has the shared UART, while a high value in bit 1 will swing the UART to OCD

Table 2-19 The Payload Format of "UART Select" Frame

2.3.3 ACK Frame Format

The ACK frame from OCD to host PC has the similar format as that of debug frame, but the sync word is different. As mentioned early, each transaction is initiated by the host PC, the ACK is only a response to a debug frame received early.

The ACK frame (from host PC to OCD) has the format shown in Table 2-20 The Format of ACK Frame:

Byte Index	Description
0	Sync Word, always 0x80
1	Sync Word, always 0xA5
2	Sync Word, always 0x5A
3	Frame Type
4 ~ 9	Payload
10 ~ 11	CRC16

Table 2-20 The Format of ACK Frame

The frame type in ACK frame will have the same value as its correspondent debug frame. The respective payload formats are as following:

- 7'h2D: Pause On with ACK
- 7'h3D: Pause Off with ACK
- 7'h7D: Break On with ACK
- 7'h1D: Break Off with ACK
- 7'h4D: Processor reset with ACK
- 7'h2A: UART Select
  Payload is not used for those frame types

- 7'h2F: Read CPU Status
  The payload format for this frame is as following:

Payload Byte Index	Description
0	always zero
1	The LSB of this byte is the debug_stall_flag. This flag will be set to 1 when the processor core is on-hold
2 - 3	RESERVED
4 - 5	The 16-bit value of PC (Program Counter), with MSB in byte 4 and LSB in byte 5

Table 2-21 The Payload Format of ACK - Read CPU Status

- 7'h5D: 4-Byte Code RAM write with ACK
- 7'h5B: 128-Byte Code RAM write with ACK
  The payload format for those frames are as following:

Payload Byte Index	Description
0 - 1	The 16-bit starting address, received from the correspondent debug frame
2	always 8'hAA
3	always 8'hAB
4	always 8'hAC
5	always 8'hAD

Table 2-22 The Payload Format of write - ACK

- 7'h6D: 4-byte Code Ram Read
  The payload for this frame will contain both the read address and the 32-bit value read from code RAM:

Payload Byte Index	Description
0 - 1	The 16-bit read address (A), received from the correspondent debug frame
2	The value in address (A)
3	The value in address (A + 1)
4	The value in address (A + 2)
5	The value in address (A + 3)

Table 2-23 The Payload Format of read – ACK

2.4 Port List

The port list for FP51-1T MCU Rev A0 is shown below in Table 2-24:

Group Name	Signal Name	In/Out	Bit Width	Description
Clock / Reset	clk	Input	1	Clock input, 96MHz
	reset	Input	1	Asynchronous reset, active low
Instruction Memory r/w	inst_mem_we	Input	1	write enable for instruction memory (code memory)
	inst_mem_wr_addr	Input	16	write address for instruction memory
	inst_mem_data_in	Input	32	write data for instruction memory, little endian
	inst_mem_re	Input	1	read enable for instruction memory
	inst_mem_re_addr	Input	16	read address for instruction memory
	inst_mem_re_enable_out	Output	1	enable out for instruction memory read
	inst_mem_data_out	Output	32	instruction memory read data, little endian
External Interrupt	INTx	Input	2	INT0 and INT1 for external interrupt
UART	UART_RXD	Input	1	RX pin for the main serial port
	UART_TXD	Output	1	TX pin for the main serial port
	UART_AUX_RXD	Input	1	RX pin for the auxiliary serial port
	UART_AUX_TXD	Output	1	TX pin for the auxiliary serial port
IO Port	P0	In/Out	8	IO port 0

Group Name	Signal Name	In/Out	Bit Width	Description
	P1	In/Out	8	IO port 1
	P2	In/Out	8	IO port 2
	P3	In/Out	8	IO port 3
Debug	pause	Input	1	pause the processor core
	break_on	Input	1	enable the hardware breakpoint
	break_addr_A	Input	16	address for hardware breakpoint A
	break_addr_B	Input	16	address for hardware breakpoint B
	run_pulse	Input	1	pulse for processor core single step execution
	debug_stall	Output	1	flag to indicate if the processor core is suspended
	debug_PC	Output	16	The current PC value of processor core
	timer_pulse_out	Output	1	Pulse Out of Timer 0
	debug_led	Output	1	debug_led out, See Section 2.2.14.
	debug_counter_pulse	Output	1	A pulse will be generated when DEBUG_COUNTER_LED register is written with a non-zero value. See Section 2.2.14.
SRAM	mem_so	Input	1	SRAM Serial Out. See Ref [18] for more detail.
	mem_si	Output	1	SRAM Serial In. See Ref [18] for more detail.
	mem_hold_n	Output	1	SRAM Hold, active low. See Ref [18] for more detail.
	mem_cs_n	Output	1	SRAM chip select, active low. See Ref [18] for more detail.
	mem_sck	Output	1	SRAM Serial clock. See Ref [18] for more detail
Voice CODEC	Si3000_SDO	Input	1	CODEC Serial Data Out. See Ref [19] for more detail.
	Si3000_SDI	Output	1	CODEC Serial Data In. See Ref [19] for more detail.
	Si3000_SCLK	Input	1	CODEC Serial Clock. See Ref [19] for more detail.
	Si3000_MCLK	Output	1	CODEC Main Clock. See Ref [19] for more detail.
	Si3000_FSYNC_N	Input	1	CODEC Frame Sync, active low. See Ref [19] for more detail.
	Si3000_RESET_N	Output	1	CODEC reset, active low. See Ref [19] for more detail.
microSD Card	sd_cs_n	Output	1	SD Card chip select, active low
	sd_spi_clk	Output	1	SD Card SPI clock
	sd_data_out	Input	1	Serial data from SD card to FPGA
	sd_data_in	Output	1	Serial data from FPGA to SD card
ADC	adc_pll_clock_clk	Input	1	2MHz clock from external PLL for the on-chip ADC
	adc_pll_locked_export	Input	1	The PLL locked signal from external PLL
I2C	sda_in	Input	1	I2C SDA input
	scl_in	Input	1	I2C SCL input
	sda_out	Output	1	I2C SDA output (sda_in/sda_out should be merged with a tri-state at FPGA top level. So are scl_in/scl_out)
	scl_out	Output	1	I2C SCL output
PWM	pwm_out	Output	6	PWM output

Table 2-24 Port List for FP51-1T MCU

2.5 Repository

The System Verilog code for the FP51-1T MCU can be found on GitHub

https://github.com/PulseRain/PulseRain_FP51_MCU

The correspondent processor core and peripherals can be found in its "submodules" folder.

2.6 RTL Simulation

To run RTL simulation on the FP51-1T MCU, please see the detailed procedures mentioned in

Ref [15]: PulseRain M10 – Quick Start Guide, Doc#QSG-0922-0039, Rev 1.2.0, 10/2017
https://github.com/PulseRain/Arduino_M10_IDE/blob/master/docs/M10_quick_start.pdf

3 Software

With a dexterous manipulation of C compiler, the FP51-1T MCU can provide a software interface that is compatible with the Arduino. In other words, through Arduino IDE, users can write sketches in Arduino Language and program the FP51-1T MCU the same way it works for AVR chips.

3.1 Compiler

Presently there are several commercial vendors that provide C/C++ compilers for 8051 instruction-set, such as IAR Systems and ARM/Keil. However, to make the FP51-1T MCU 100% open source, decisions have been made to use the SDCC (Ref [3]) as the default C compiler for FP51-1T.

As mentioned early, the FP51-1T MCU has internal XDATA, and it uses XPAGE register (Section 2.1.1.1) to access the internal XDATA. The address of the XPAGE register for FP51-1T is 0xE7 (Table 2-2), and the SDCC compiler needs to be configured with this XPAGE address.

The biggest shortcoming for traditional 8051 architectures is that it has a relatively small stack space (256 bytes), as it is limited by the single byte SP (Stack Pointer). And the FP51-1T has taken several approaches in both hardware and software to alleviate this constraint:

- The FP51-1T has added another stack pointer register (SPH, in Table 2-2) to extend the stack pointer to be 16 bits

- The SDCC compiler has been configured to work in large memory model, so that the stack will reside in the XDATA space

- Most API functions are compiled as non-entrant function calls to save stack space

3.2 Arduino Language

With the SDCC compiler, and FP51-1T is able to present a software programming interface that is compatible with the Arduino Language.

3.2.1 setup() and loop()

For processors, there are generally two ways to do the control flow (Ref [2]):

1. Use an embedded OS, with tasks being handled by threads or processes.
2. Use an endless super-loop, with tasks being handled sequentially, and the asynchronous events are handled in ISR.

The second option is more straightforward and does not require complicated scheduler, at the expense of resource efficiency. For microprocessor, especially for those whose jobs can be handled with less than 64KB of code, options 2 is often the preferred the choice. And this is also the approach that both Arduino and FP51-1T take.

With the Arduino IDE, the user will provide a sketch that must contain the following two functions:

- setup()
- loop()

The setup() will be called only once, while the loop() will be invoked repetitively. In fact, behind the scene, the Arduino will also wrapper those two functions in the way shown in Figure 3-1.

```
#include "Arduino.h"

void main()
{
        setup();

        while(1) {
                loop();
        }
}
```

Figure 3-1 Wrap setup() and loop() in main() Function

3.2.2 Data Type

Compatible with the Arduino Language, the following data types are defined in Arduino.h:

```
typedef unsigned long    uint32_t;
typedef long             int32_t;

typedef unsigned short   uint16_t;
typedef short            int16_t;

typedef unsigned char    uint8_t;
typedef signed char      int8_t;

typedef uint8_t          byte;
typedef uint16_t         word;
```

List 3-1 Data Type for FP51-1T

Please note that String object is currently not supported by FP51-1T's software library.

3.2.3 APIs

Compatible with Arduino Language, the following APIs are supported by FP51-1T's software library:

3.2.3.1 Digital IO

- *void pinMode (uint8_t pin, uint8_t mode)*

Parameters:

pin: IO pin index (0 ~ 15). The mapping of pin index to IO port is as the following:

pin index (0 ~ 7) => Port A (0 ~ 7)

pin index (8 ~ 15) => Port B (0 ~ 7)

mode: INPUT or OUTPUT (Behind the scene, INPUT / OUTPUT are defined as the following:

#define INPUT 0

#define OUTPUT 1

Return Value: None

Remarks: Use this function to configure the direction of the IO pin

- *void digitalWrite (uint8_t pin, uint8_t value)*

Parameters:

pin: IO pin index (0 ~ 15).

value: HIGH or LOW (Behind the scene, HIGH / LOW are defined as the following:

#define HIGH 1

#define LOW 0

Return Value: None

Remarks: Use this function to set the logic value of the IO pin

- *uint8_t digitalRead (uint8_t pin)*

Parameters:

pin: IO pin index (0 ~ 15).

Return Value: The current logic value of the IO pin (LOW (0) or HIGH (1))

Remarks: Use this function to get the current logic value of the IO pin

3.2.3.2 Analog IO

- *uint16_t analogRead(uint8_t channel_index)*

  Parameters:

  channel_index: index for analog channel (0 ~ 5)

  Return Value: The converted value from the designated analog channel (12-bit unsigned)

  Remarks: Use this function to get the current value of the designated A/D channel. For FP51-1T MCU, this function is part of the M10ADC library, so the user needs to include "M10ADC.h" before calling this function.

- *analogWrite(...)*

  Arduino uses *analogWrite()* function for PWM. And in this regard, the FP51-1T has a full-blown PWM library called M10PWM. For more details of the M10PWM library, please see

  Ref [7]: PulseRain M10 – PWM, Technical Reference Manual, Doc# TRM-0922-01009, Rev 1.0.1, 10/2017, https://github.com/PulseRain/M10PWM/raw/master/extras/M10_PWM_TRM.pdf

- *analogReference(...)*
  Arduino uses *analogReference()* function to specify the reference source for A/D Converter (Internal / External). For FP51-1T MCU, only internal reference is used at this point. Thus, this function is not available on FP51-1T.

3.2.3.3 Time

- *void delay (uint32_t delay_in_ms)*

  Parameters:

  delay_in_ms: delay value in millisecond

  Return Value: None

  Remarks: Use this function to delay in the granularity of millisecond.

- *void delayMicroseconds (uint32_t delay_in_us)*

  Parameters:

  delay_in_us: delay value in microsecond

  Return Value: None

  Remarks: Use this function to delay in the granularity of microsecond.

- *uint32_t millis ()*

  Parameters: None

  Return Value: Number of milliseconds passed since reset.

  Remarks: Use this function to get the number of milliseconds passed since reset

- *uint32_t micros ()*

  Parameters: None

  Return Value: Number of microseconds passed since reset.

  Remarks: Use this function to get the number of microseconds passed since reset

3.2.3.4 Interrupt

- *void interrupts()*

  Parameters: None

  Return Value: None

  Remarks: Use this function to enable interrupt globally

- *void noInterrupts ()*

  Parameters: None

  Return Value: None

  Remarks: Use this function to disable interrupt globally

3.2.3.5 ISR Handler

- *void attachIsrHandler(uint8_t index, void (*isr_handler_pointer)())*

  Parameters:

  index: IRQ index, the valid values for FP51-1T Rev A0 are:

  0 => External Interrupt 0,
  1 => Timer 0,
  2 => I2C Controller,
  3 => Timer 1,
  5 => On-chip ADC,
  6 => CODEC

  isr_handler_pointer: function pointer to the ISR handler

  Return Value: None

  Remarks: Use this function to attach ISR to the correspondent IRQ index. To detach the ISR, just set the isr_handler_pointer to NULL. This function is on par with Arduino Language's *attachInterrupt()* and *detachInterrupt()* functions.

  And by default, Timer 1 is used by the main serial port.

3.2.3.6 Serial

Like the Arduino Language, FP51-1T also supports Serial port as console in/out, and it supports the following functions:

- *Serial.begin()*
- *Serial.available()*
- *Serial.print()*
- *Serial.println()*
- *Serial.read()*
- *Serial.readBytes()*
- *Serial.write()*

The details of those functions can be found in

Ref [14]: PulseRain M10 – Serial Port, Technical Reference Manual, Doc# TRM-0922-01005, Rev 1.0.0, 09/2017, https://github.com/PulseRain/M10SerialAUX/raw/master/extras/M10_Serial_TRM.pdf

3.2.4 Default Arguments for Function Call

The SDCC compiler does not support default arguments for function calls. But with the help of pre-processor macros (Ref [16][17]), FP51-1T has managed to implement default arguments in its software libraries. The details of the implementation can be found in the "Arduino.h" inside FP51-1T software package.

3.3 Arduino IDE

The Arduino IDE supports 3rd party package for additional boards, such as PulseRain M10. And the FP51-1T MCU has been carried on the PulseRain M10 board as its default soft-core MCU. The procedures to add PulseRain M10 board support to Arduino IDE can be found in

Ref [15]: PulseRain M10 – Quick Start Guide, Doc#QSG-0922-0039, Rev 1.2.0, 10/2017 https://github.com/PulseRain/Arduino_M10_IDE/blob/master/docs/M10_quick_start.pdf

And this section will discuss the technical details of the M10/FP51-1T software package for Arduino IDE.

3.3.1 Package Repository

The M10/FP51-1T software package for Arduino IDE can be found on GitHub

https://github.com/PulseRain/Arduino_M10_IDE

In particular, the

https://raw.githubusercontent.com/PulseRain/Arduino_M10_IDE/master/package_M10_index.json

is the master JSON file that will provide indexes for the rest of the package, and the link above should be put in Arduino IDE's menu: File/Preferences/Additional Boards Manager URLs, as demonstrated in Ref [15].

3.3.2 Package Folder Structure

The folder structure of the M10/FP51-1T software package is shown in Figure 3-2. It has the following major files:

- The "Arduino.h" and "M10.c", which are part of the FP51 core. They contain the source code for those APIs mentioned in Section 3.2.3.
- The "main.c", which is illustrated in Figure 3-1
- The "boards.txt" and "platform.txt". The Arduino IDE will use these two files to determine to tools and parameters that are specific to the M10 board.
- Compiler and upload tools.

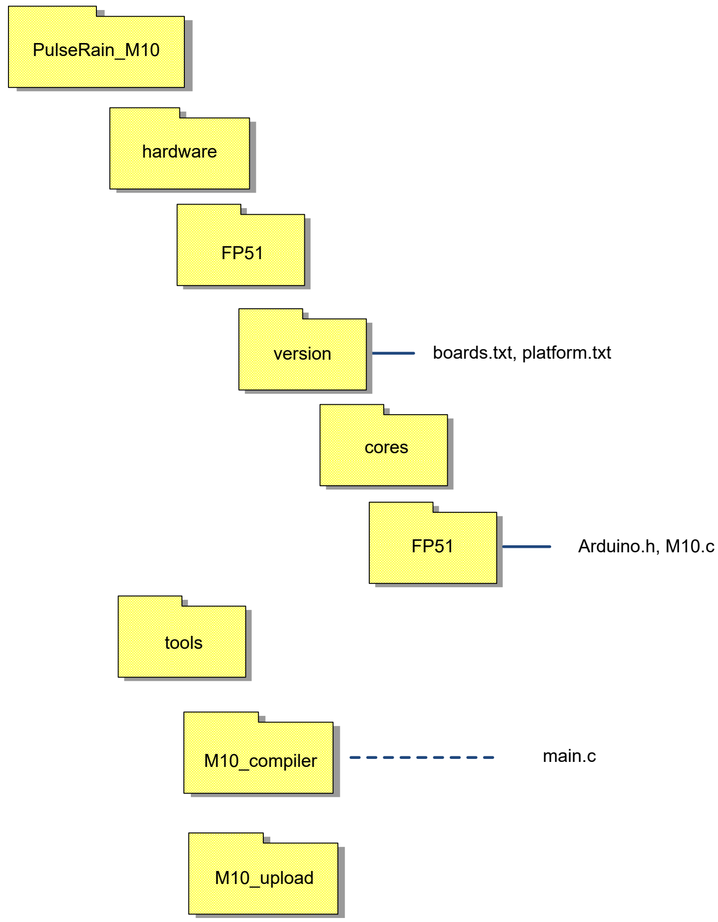

Figure 3-2 Package Folder Structure

3.4 Library

As mentioned early, the FP51-1T MCU supports a variety of peripherals with open source library, and their respective technical reference manual can be found in Ref [6][7][8][9][10][11][12][13][14]. The procedures to install and use those libraries can be found in Ref [15].

And the repository for those packages are:

- A/D Converter, M10ADC: https://github.com/PulseRain/M10ADC
- PWM, M10PWM: https://github.com/PulseRain/M10PWM
- JTAG, M10JTAG: https://github.com/PulseRain/M10JTAG
- I2C, M10I2C: https://github.com/PulseRain/M10I2C
- microSD, M10SD: https://github.com/PulseRain/M10SD
- voice CODEC, M10CODEC: https://github.com/PulseRain/M10CODEC
- Serial Port, M10SerialAUX: https://github.com/PulseRain/M10SerialAUX
- SRAM, M10SRAM: https://github.com/PulseRain/M10SRAM

- DTMF, M10DTMF: https://github.com/PulseRain/M10DTMF
- Sparkfun ESP8266 Shield, M10ESP8266: https://github.com/PulseRain/M10ESP8266

3.5 Example Sketches

To help users be familiar with the M10/FP51-1T, a variety of example sketches are provided in the following repository:

https://github.com/PulseRain/M10_Sketches