The BNF grammar and Coding for UCDL

1 The BNF grammar of UCDL

To access various types of heterogeneous chips uniformly, in this section, we build a comprehensive BNF grammar for our Unified Chip Description Language (UCDL). The basic rule of BNF is "<symbol>::=__expression__", where the "symbol" on the left of "::=" is interpreted by the expression on the right. For a better understanding, we first summarize the symbols used in UCDL as follows:

1)"<>" represents the non-terminal symbol. In this paper, its contents are syntactic variables of UCDL.
2)"|" means or. For example, X|Y|Z means the content is selected from one of X, Y, and Z
3)"[]" means that the content is optional. For example, <A>::=<B>[<C>] means that <B> is necessary for the expression of <A>, while <C> is optional and can be defaulted under certain conditions.

Based on this foundation, we introduce the BNF grammar of UCDL in three parts, *i.e.*, pin configuration, data format, and meta operation for chips, as follows.

1.1 Pin Connection Configuration

We define *PIN* as the keywords to dynamically configure the electrical connection on each pin of the deployed chip to support hot plug-and-play. Generally, the *PIN* statements contain four parameters *i.e.*, pin number, pin type, pin function, and connection properties. Its basic syntax is $PIN(j, type, function, connection)$, where the j-th pin of a specific chip should be configured with *type*, *function*, and *connection*. For instance, $PIN(3, I2C, DR, Open\text{-}Drain, (type1))$ means the third pin of the chip is set to be the data-read line of the I^2C bus, and its connection type is open-drain (type 1). In practical, to fit diverse chip interfaces, we also set additional parameters in the *PIN* keyword according to the configured interface (*e.g.*, I2C, SPI). The detailed BNF grammar can be summarized as:

▷**UCDL Grammar:** <pin configuration>::=Pin(<pin number>[1],<pin type>[2],[<pin function>[3]|<fitting voltage>[4]],[<connection properties>[5] | <I2S parameters>[6]],[<UART parameters>[7]])

1) Pin selection: <pin number>[1]::= 1 | 2 | 3 | 4 . . . . . 15 | 16
2) Parameter for pin types: <pin type>[2]::= (SPI | I2C | UART | 1-Wire | RS485 | I2S | PWM | PCM | SMBus) | (VDD | GND | VCC) | (SR | CW | ADC | DAC)
3) Parameter for pin functions: <function>[3]::=DW | DR | CLK | [RST] | [CS] | [Power In] | [Ground]
4) Parameters for voltage sitting: <fitting voltage>[4]::=0.1 | 0.2 | 0.3 $\cdots$ 17.9 | 18.0
5) Parameters for connection properties: <connection properties>[5]::= push-pull (in|out) | oper-drain (type 1 | type 2) | High-impedance
6) Parameters for I2S Bus: <I2S parameters>[6]::=<operating mode>, <payload size>,<frame size>,<sampling rate>
6.1) <operating mode>::=Standard | Left Justified | Right Justified
6.2) <payload size(frames)>::=1 | 2 | 3 | 4. . . . . 254 | 255
6.3) <frame-size(bit)>::= 16 | 32
6.4) <sampling rate>::=<coefficient><frequency unit>
6.5) <coefficient>::= 1 | 2 | 3 | 4 | 5 |$\cdots$ 998 | 999
6.6) <frequency unit>::= Hz | KHz
7) Parameters for UART Bus: <UART parameters>[7]::=<baud rate>, <payload size>,<parity bit>,<stop bit>,<hardware flow control bit>
7.1)<baud rate(bps)>::= 600 | 1200 | 2400 | 4800 | 9600 | 19200 | 38400 — 76800 | 115200
7.2) <payload size(byte)>::=1 | 2 | 3 | 4. . . . . 254 | 255
7.3) <parity bit>::=0 | 1
7.4) <stop bit>::=1 | 1.5 | 2
7.5) <hardware flow control bit>::= 0 | 1

The annotation of above signs [1]~[7] are as follows.

[1]<pin number> specifies the pin for connection configuration. In its initial state, our prototype employs a four-bit binary encoding for pin addresses, pointing to the connection configuration of 16 pins. The length of the pin address can also be adjusted according to IoT applications with more manageable pins. An n-bit address can point to the configuration of 2^n pins.

[2]<pin type> defines what kind of signal is transmitted on the target Pin. Its option can be divided into three categories:

- 1) Computer buses like SPI, I2C, USART, 1-Wire, RS485, I2S, PWM, PCM, and SMBus;
- 2) Direct voltage interaction. For digital signals (e.g., chip select), the user can choose the CW keywords to transmit control signals into a chip or select SR to read out state signals for feedback. For analog signals, the user can select DAC (digital-to-analog converter) to transmit an analog control signal for a chip or select ADC (analog-to-digital converter) to read out the analog signal from a chip;
- 3) Constant output, the user can choose VCC, or VDD for analog or digital power supply for a chip, respectively. The user can also select GND to connect the target pin to the ground.

[3]<pin function> defines the detailed functions of a pin if its type is a data bus. Specifically, computer buses usually consist of several different signal lines, *e.g.*, I2C contains a data line and a synchronized clock line. Therefore, when the pin type is a certain bus, it is necessary to further define its detailed function so that the LEGO device can select the required signal line in the bus kernel. In other cases (*i.e.*, direct voltage interaction and constant output), the signal on the target pin is directly controlled by the gateway instructions, so the detailed function is optional.

[4]<fitting voltage> The UCDL supports analog signal interaction and supply voltage fitting for chips. When the pin is set for analog signal interaction (ADC/DAC), the selected voltage is used as the reference voltage for ADC and DAC. The user may typically select 5V as the LEGO board provides this reference voltage initially. In addition, when the pin is set for I/O voltage fitting, its range is 3.0-18.0 V as it is achieved by the voltage shifter (MC14504B) on the LEGO board. Users could also choose other converters to change the voltage range according to different applications.

[5]<connection properties> The parameter defines the electrical characteristics of pin connections. The choices can be push-pull, open-drain, or high impedance, we discuss the details in Section. 3.

[6] <I2S parameters>. The I2S bus is dedicated to audio transmission and has some special parameters compared to other conventional digital buses (*e.g.*, I2C, SPI, RS485, 1-wire, etc.). When the selected bus is I2S, it needs to be supplemented with operating mode parameters, payload size, frame size, and sampling rate.

[7]<UART parameters>. The UART bus is also a dedicated bus that has additional configuration parameters. When a pin is set as a UART bus, it is necessary to configure its baud rate, payload size, parity bit, stop bit, and hardware flow control bit for data transmission.

1.2 Meta-Operations for Chip Control

We define four keywords: *DW* (Data Write), *DR* (Data Read), *CW* (Control Write), and *SR* (State Read) to carry the summarized four meta-operations. By orchestrating meta-operations in required timing, the required control logic for various chips can be dynamically generated in LEGO system. To achieve this, we define four types of variables and parameters in the statement of meta-operation statements, i.e., control keyword, pin number, operation type, processing delay. Its BNF grammar can be summarized as:

▷**UCDL Grammar:** <meta operation>::=<control keyword>[1](<pin number>,<operation type>[2],+<processing delay>[3]])

1) Operation definition: <control keyword>[1]::= DW | DR | CW | SR
2) Pin selection: <pin number>::= 1 | 2 | 3 | 4 . . . . . 15 | 16
3) type of meta-operations: <operation type>[2]::=<DW operation> | <DR operation> | <CW operation> | <SR operation>
3.1) <DW operation>::=0xYY ("YY" is a hexadecimal data, the length is up to 32 bytes.)
3.2) <DR operation>::= (n, X1, X2 $\cdots$ Xn) | (ADC, X1)
3.3) <CW operation>::= (H | L) | (ADC, *x.x*V (0.0≤ *x.x* ≤5.0V))
3.4) <SR operation>::= (H | L) | (DAC, *x.x*V (0.0≤ *x.x* ≤5.0V))
4) Timing control: <processing delay>[3]::=<delay multiples> × <time units>
4.1) <delay multiples>::=1 | 2 | 3 | 4. . . . . 998 | 999
4.2) <time units>::= nS | uS | mS | S

The annotation of above signs [1]∼[3] are as follows.

[1]<control keyword> defines the type of meta-operation, it has four categories: 1) select DW to write data to the target pin of a chip; 2) select DR to read data from a chip; 3) select CW to send a control signal to the target pin of a chip; 4) select SR to read out voltage states from a pin of the select chip.

[2]<operation type>. This parameter defines the detailed type of a meta-operation. It has 4 categories:

- 1) For data-write (DW) operation, the content is the write-in hexadecimal data with length up to 32 bytes;
- 2) For data-read (DR) operation, the content has two subcategories: a), for digital chips, the content is the length definition for data read out, where the unit is bytes, and the read-out data is marked as "X*i*" in each byte. By defining the returned data in each bits, it facilitates data converting for the application; b), for analog chips, the content is ADC and X*i*, which controls the LEGO device to sample the analog voltage on the target pin by the on-board 8-bit ADC, and return the sampled data.
- 3) For control-write (CW) operation, the content has two subcategories: a), for the digital chip, the input is logic level signal to pull the target pin to logic high (H) or logic low (L); b), for the analog chip, the input is an analog voltage, so the content is DAC (Digital to Analog Converter) and the target voltage. It controls LEGO device to schedule the onboard DAC and output target analog voltage to the target pin at a range from 0-5V.
- 4) For State-read (SR) operation, the content is logic high (H) or logic low (L), for example, SR (4, H) means to check the state of pin 4. Once its outputs turn to logic high (H), it will directly drive the LEGO device for local chip cooperation or upload the states to the gateway, if needed.

[3]<Timing control>. This parameter controls the operation interval between individual meta-operations, which is designed to give enough time for the chip to process. Specifically, as functional chips have a certain processing time for meta-operations, *i.e.*, it takes time for state transitions in its internal states machine (as discussed Section 4.2.2 of the paper manuscript), hence subsequent operations can only be initiated when the state transitions is complete, otherwise it will be ignored. For instance, if the user want to write the binary code $'00111001'$ into the chip, but the first three digits (001) are transmitted before the required state ready. In that case, the chip can only receive the last five digits (11001) and cannot operate correctly. To ensure the reliability of chip operation, an additional time delay is attached to each meta-operation to define the required processing time in the chip. For instance, $DW(j, 0x0b) + 6\mu S$ indicates that the chip requires 6 microseconds to process the data (0x0b) after it has been written into the j-th pin of the chip. Only after that, the chip can finish the code (0x0b) processing and transfers to the next state that is ready for the next operation. The required duration of each meta-operation can be found in the chip's data-sheet provided by the manufacturer.

1.3 Data Format Definition

We design the *DF* keyword to define the format of chip data out, which helps the gateway to convert the uploaded raw chip data into the target application data (*e.g.*, convert 11100010 to 36.8°C) for users. Considering, a chip may have multiple types of raw data output (*e.g.*, a BME280 sensor can output data on temperature, humidity, and air pressure). We design the basic *DF* syntax as $DF(j, imp, func, unit)$, where the j-th output is about the *imp* type of data and the data conversion method and unit are $func$ and *unit*, respectively. For instance, $DF(1, acc, x/64, g)$ means that the chip first returns the acceleration data; the data conversion function is $x/64$ (x is the raw data); and the unit of the result is g (*i.e.*, 9.8 m/S^2). The LEGO gateway utilizes the QScriptEngine [1] for data conversion, which supports the computing of strings with standard mathematical formulas (the details are presented in Section 4.2.3 of the paper manuscript). The BNF grammar of *DF* keywords can be summarized as:

▷**UCDL Grammar:** <data converting>::=DF(<type number>, <data type>, <equation string>, <data unit>)

1) <type number>[1]::= Y1 | Y2 ⋯ | Ym
2) <data type>[2]::= temperature | pressure | humidity. . . . . .
3) <equation string>[3]::=Y*i*=f(X*i*, X*j*, ...)
4) <data unit>::= kg | °C | N | psi. . . . . .

The annotation of this statement are as follows.

[1] <type number>: the function of this parameter is to distinguish the types of final data (not raw data) that can be output by the chip. For example, the BME280 chip can output temperature and humidity. To distinguish them, we can set the final temperature data as Y1 and the final humidity data as Y2.

[2] <data type>: the function of this parameter is to define the type of the data, it could be temperature, acceleration, pressure or other physical quantity if the chip is a sensor or an ADC(analog-digital converter). If the chip is not a sensor (*e.g.*, an EEPROM), the data type is null.

[3] <equation string>: the function of this parameter defines the equation to convert the raw data (X*i*) to the final application data (Y*j*). The function supports complex computing even the raw and final application data not correspond

[1]The site of QScriptEngine: https://doc.qt.io/qt-5/qscriptengine.html

Table 1: Specifications of COTS chips in UCDL

Specifications for Pins		Specifications for chip data converting		
Pin Type	**Description Name[1]**	**Data Type**	**Description Name**	**Data Unit**
Pin for analog power supply	VCC	Temperature	Temp, temperature	°C
Pin for digital power supply	VDD	Relative humidity	RH	%
Pin for push-pull output	Push-pull(output)	Pressure	P,Pre	Psi
Pin for push-pull input	Push-pull(input)	Blood oxygen concentration	SPO2	%
Pin for open-drain connection (type1)	Open-drain (type1)	Local temperature	LT, local temperature	°C
Pin for open-drain connection (type2)	Open-drain (type2)	Remote temperature	RT, remote temperature	°C
Pin for chip select functions	CS	Concentration of PM2.5	PM2.5 level	ppm
Pin for interrupt interactions	Int	Light intensity	light level, light intensity	Lux
Pin for clock signal transmission	Clock	Manufacturer ID	Manufacturer ID, MID	null
Pin for data signal transmission	Data	Concentration of CO2	eCO2	ppm
Pin for chip reset functions	RST	Concentration of TVOC[2]	eTVOC	ppm
Pin for I2C connection	I2C	Intensity of ultraviolet ray	UV Intensity	mW/cm^2
Pin for SPI connection	SPI	Rotation angle	Angle	°
Pin for I2S connection	I2S	Air alcohol concentration	Alcohol	ppm
Pin for SMBus connection	SMBus	Concentration of dust	Dust-density	ppm
Pin for 1-wire connection	1-Wire	Acceleration	Acceleration	g, m/s^2
Pin for UART connection	UART	Concentration of combustible gas	Gas	ppm
Pin for analog voltage output	ADC	Fluid flow velocity	Flow velocity	m/S
Pin for analog voltage input	DAC	Magnetic field intensity	Magnetic field	A/m

[1] Description name represents the specific name in UCDL syntax.
[2] TVOC means Total Volatile Organic Compounds

to one-to-one because the final data may be calculated from multiple raw data. For example, in ADXL362, the final acceleration data need to add temperature for compensation. Finally, the <data unit> defines the unit of the converted data. For example, if the data type is temperature, the unit is °C; if the data type is weight, the unit might be g (gram) or Kg (kilo gram).

1.4 Summary

Based on the above designs, users can make unified logical expressions of diverse functional chips (*e.g.*, sensory chips, flash memory chips, etc.) through simple syntax by writing UCDL, in aspects of electrical characteristic connections, data interaction and control interaction. For the ease of using, we also give the referenced specifications for COTS chips, as shown in Table 1 to make it readable for users. In summary, the UCDL design helps fast customization of IoT end devices by empowering chips plug-and-play. This is also helpful in accelerating the development of IoT applications.

2 UCDL Syntax Coding

In this section, we introduce operation process of UCDL statements,*i.e.*, how UCDL meta-operations are lowered to instructions, and how these instructions are lowered to underlying signals.

2.1 Overview

In UCDL programming, each instruction is directly linked to a meta-operations of a chip. For a UCDL statement, the gateway automatically maps its keywords and variables into a predefined code and connects them in series to form an independent instruction. Further, the instruction directly drives the hierarchical scheduler and Universal Signal Converting (USC) on LEGO devices (Section 6 of the paper manuscript) to generate the required signal chat controlling the deployed chips efficiently. In this section, we introduce the detailed coding rules with chip specifications for UCDL.

2.2 Coding Details

In a description file, the instructions for pin configurations (keywords: PIN) and logic control (keywords: DW, DR, CW, SR) are converted into gateway instructions, where the encoder maps the keywords and variables in UCDL statements to code and concatenates them to form the final gateway instructions. The coding map for UCDL parsing is summarized in Tables 2 and 3. In contrast, the instructions for data formats decoding (keywords: DF) is only running

Table 2: UCDL Syntax Coding (part1)

	Keywords	Code	Description
Coding For Keywords	PIN	1000	Set pin functions for a chip
	DF	1001	Define data format for a chip
	DW	1010	Write data to a chip
	DR	1011	Read data from a chip
	CW	1100	Send a control signal to a chip
	SR	1101	Read a status signal from a chip
	Pin Types	Code	Description
Coding For Pin Types	I2C	0001	Set Connections for I2C Bus
	SPI	0010	Set Connections for SPI Bus
	USART	0011	Set Connections for USART Bus
	1-Wire	0100	Set Connections for 1-Wire Bus
	RS485	0101	Set Connections for RS485 Bus
	I2S	0110	Set Connections for I2S Bus
	PWM	0111	Set Connections for PWM Bus
	PCM	1000	Set Connections for PCM Bus
	CW	1001	Set Control-Write Pin
	SR	1010	Set State-Read Pin
	Pin Functions	Code	Description
Coding For Pin Functions	CLK	0010	Clock signal transmission
	VCC	0011	Power supply (Analog)
	DW	0100	Data input(write)
	DR	0101	Data output(read)
	DW/DR	0110	Data Write/Read
	VDD	0111	Power supply (Digital)
	GND	1000	Ground
	Connection Types	Code	Description
Coding For Connection Types	Push-Pull (output)	11000	Set connection as input with push-pull
	Push-Pull (input)	00100	Set connection as output with push-pull
	Open-Drain (type 1)	01010	Set connection as type 1 of open-drain
	Open-Drain (type 2)	10001	Set connection as type 2 of open-drain
	High Impedance	00000	Set connection as high impedance

on the gateway, so it does not need a coding map, and the gateway directly extracts the content for data display. The encoding scheme of the 5 gateway instructions is as follows:

PIN instruction(keywords: *PIN*): The coding for Pin instruction has the main four cases:

- 1) Generally, the code format for pin configuration instruction consists of six fields, *i.e.*, 4-bit device code[2], 4-bit keyword code, 4-bit pin number code, 4-bit pin type code, 4-bit pin function code, and 5-bit connection type code. The general format fit for most type of digital buses, *i.e.*, I2C, SPI, 1-Wire, SMBus, etc.
- 2) When the pin is set for voltage fitting or analog signal interaction (with pin types of ADC, DAC and VDD, VCC), in its format, the 4-bits pin function code is changed to 8-bit voltage setting code. Hence, its code format is: 4-bit device code, 4-bit keyword code, 4-bit pin number code, 4-bit pin type code, 8-bit voltage setting code, and 5-bit connection type code.
- 3) When the pin is set for I2S connection, in its format, the 4-bits pin function code is changed to 24-bits I2S parameters. The I2S parameter contains 2-bit operating mode code, 8-bit payload size code, 2-bit frame size code, and 12-bit sample rate code (consists of 10-bit coefficient and 2-bit unit). Hence, its code format is: 4-bit device code, 4-bit keyword code, 4-bit pin number code, 4-bit pin type code, 24-bit I2S parameter code, and 5-bit connection type code.
- 3) When the pin is set for UART connection, in its format, the 4-bits pin function code is changed to 18-bits UART parameters. The UART parameter contains 4-bit baud rate code, 8-bit payload size code, 2-bit parity bit code, 2-bit stop bit code, and 2-bit hardware flow control bit code. Hence, its code format is: 4-bit device code, 4-bit keyword code, 4-bit pin number code, 4-bit pin type code, 18-bit UART parameter code, and 5-bit connection type code.

[2]In the initial stage, the system uses a 4-bit device code and a 4-bit pin address for the LEGO device. Hence a gateway can support up to 16 LEGO devices in its network.

Table 3: UCDL Syntax Coding (part2)

Coding for Voltage Setting	Voltage	Code	Voltage	Code
	0.1	00000001	...	...
	0.2	00000010	17.8	10110010
	0.3	00000011	17.9	10110011
	...	...	18.0	10110100
Coding for I2S Parameter	Operating mode	Code	payload size(frames)	Code
	Standard	01	1	00000001
	Left Justified	10	2	00000010
	Right Justified	11	...	...
	frame-size(bit)	Code	253	11111101
	16	01	254	11111110
	32	10	255	11111111
	Sampling rate (coefficient)	Code	Sampling rate (unit)	Code
	1	0000000001	Hz	01
	2	0000000010	KHz	10
	...	...		
	998	1111100110		
	999	1111100111		
Coding for UART Parameters	Baud rate(bps)	Code	Payload size(byte)	Code
	600	0001	1	00000001
	1200	0010	2	00000010
	2400	0011	...	...
	4800	0100	253	11111101
	9600	0101	254	11111110
	19200	0110	255	11111111
	38400	0111	Parity bit	Code
	76800	1000	0	01
	115200	1001	1	10
	Stop bit	Code	Hardware flow control bit	Code
	1	01	0	01
	1.5	10	1	10
	2	11		

DW instruction(keywords: *DW*): The code format for data write instruction consists of four fields, *i.e.*, 4-bit device code, 4-bit keyword code, 4-bit pin number code, and n-bit write-in data code (the data to write in the target chip).

DR instruction(keywords: *DR*) The code format for data read instruction consists of four fields, *i.e.*, 4-bit device code, 4-bit keyword code, 4-bit pin number code, and 4-bit read length definition code (number of Bytes read out from the target chip).

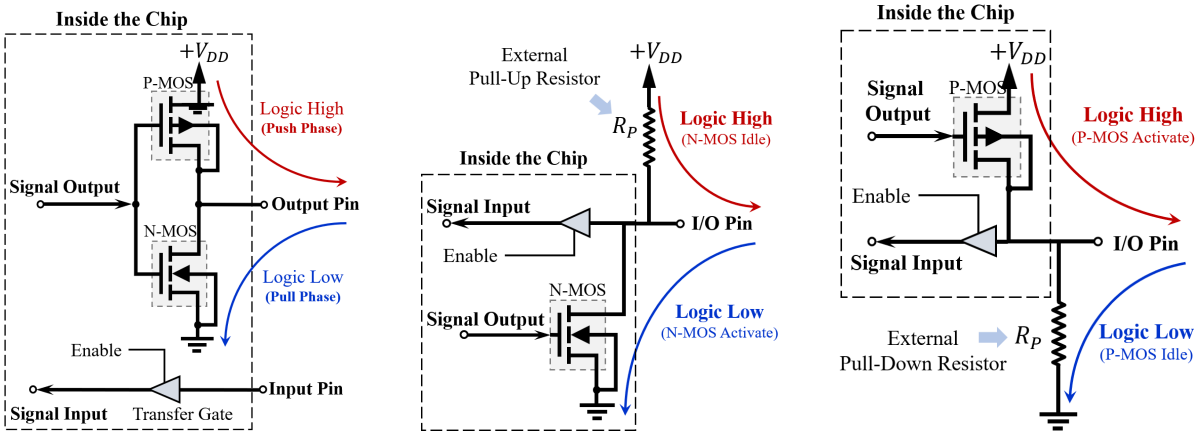
CW instruction(keywords: *CW*): The code format for control write instruction consists of four fields, *i.e.*, 4-bit device code, 4-bit keyword code, 4-bit pin number code, and 1-bit control flag code (to set the target pin of a chip to logic high (flag=1) or logic low (flag=0)).

SR instruction: The code format for state read instruction consists of three fields, *i.e.*, 4-bit device code, 4-bit keyword code and 4-bit pin number code.

Base on the coding rule, the gateway can directly parsing UCDL statements into instructions for chip control. For instance, instruction $'0011 - 1000 - 0101 - 0010 - 11000'$ means to set the 5-th pin of the LEGO device with ID 3 to SPI output for data writing, and the connection type is push-pull. When the device receives this instruction, the Universal Signal Conversion (USC) circuit acts accordingly to build up the required connections.

3 Connection type for COTS chips

In Section 1.1, we introduced the design of PIN keyword. One of its key functions is to dynamically configure electronic connections on each pin of the plugged chips. From the perspective of electrical characteristics, the connection type of functional chips can be categorized into two types: *push-pull* and *open-drain*, as illustrated in Figure 1. The push-pull architecture comprises two MOSFETs that provide the ability to output logic-high and logic-low signals (a).



(a) push-pull architecture. (b) open-drain architecture (type 1). (c) open-drain architecture (type 2).

Figure 1: Push-pull and open-drain architectures. (a) The push-pull architecture has capabilities to output both logic high and logic low signals unidirectional, which requires an independent pin for signal input. (b) and (c), the open-drain architecture removes one transistor to provide a data input channel. However, as a result, they can only output logic-low (type 1) or logic-high (type 2) signals, and requiring external pull-up or pull-down resistors in working.

However, this structure employs complementary MOSFETs (a P-MOS and an N-MOS), which fix the voltage at a specific logic level (logic-high or logic-low) on the signal line. By this, it can not relase the signal line for slave input, and can only support signal output for one-way data transmission. For example, the SPI BUS requires two independent ports for data input and output.

In contrast, the open-drain architecture replaces a MOSFET with a Tristate-Logic (TSL) gate to provide both signal input and output capabilities (b),(c). When signal input is required (to read data from a chip), such structure can turn off the MOSFET to release the signal line and activates the TSL gate to provide a channel for signal input. However, as removes a MOSFET, such architecture only supports unilateral level output (logic-low or logic-high), such as I^2C bus (type 1) and RS485 bus (type 2), and requires external pull-up (type 1) or pull-down (type 2) resistors for complete signal transmission. In that case, the pull-up/pull-down resistor provides the drive capability for another side to support bidirectional signal transmission for connected peripherals. In summary, different signal interfaces may have incompatible electrical characteristics.

Therefore, to provide the capability for chips plug-and-play, we designed a Pin Configuration unit in the Universal Signal Converting (USC) circuit as a solution to dynamically adapt both connections and electrical characteristics for chips deployed on LEGO devices (the details are presented in Section 6.2 of the paper manuscript).