The BNF grammar and Coding for UCDL

1 The BNF grammar of UCDL

To better present the design of our Unified Chip Description Language (UCDL), in this appendix section, we introduce its BNF grammar. 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, $PIN(3, I2C, DR, Open$ - $Drain, (type1))$ means the third pin of the chip is set to be the data-read line of the I^{2}C bus, and its connection type is open-drain (type 1).

To improve coding efficiency, 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 ··· 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 |··· 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. In subsequent applications, the

length of the pin address can be adjusted according to requirements to increase the number of 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> 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 realized. To achieve this, we define five 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 (note:)
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>::=(1, X1) | (2, X1, X2) | (3, X1, X2, X3) | $\cdots$ | (n, X1, X2 $\cdots$ Xn) | (ADC,X1)
3.3) <CW operation>::= (H | L) | (ADC, $x.x$V ($0.0 \leq x.x \leq 5.0$V))
3.4) <SR operation>::= (H | L) | (DAC, $x.x$V ($0.0 \leq x.x \leq 5.0$V))
4) Timing control: <processing delay>[3]::=<delay multiples> $\times$ <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. In this parameter, the user has four choices: 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 carries the detailed type of the 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 to upload the states to the gateway, which indicates the gateway to make subsequent operations.

[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 avoid this, 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 statement of meta-operation is guided by the keyword of DF, which is designed to convert the raw data of chip output to the application data for users. For example, convert "11100101" to 35.4°C. 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.

[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

[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 as the chip sensing. 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 (Xi) to the final application data (Yj). The function supports complex computing even the raw and final application data not correspond 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. It is linked to <data type>. 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.

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 logic by writing UCDL, in aspects of electrical characteristic connections, data interaction and control interaction. For the ease of using, we also give a standards of 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 USC circuit on LEGO devices to generate the required signal for chip control. The coding details are presented in Section.2.2.

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

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

statements to code and concatenates them to form the final gateway instructions. For the convenience of readers, this appendix chapter provides the coding map for UCDL parsing, summarized in Tables 2 and 3. In contrast, the instructions for data formats decoding (keywords: DF) is only running 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

[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		

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.

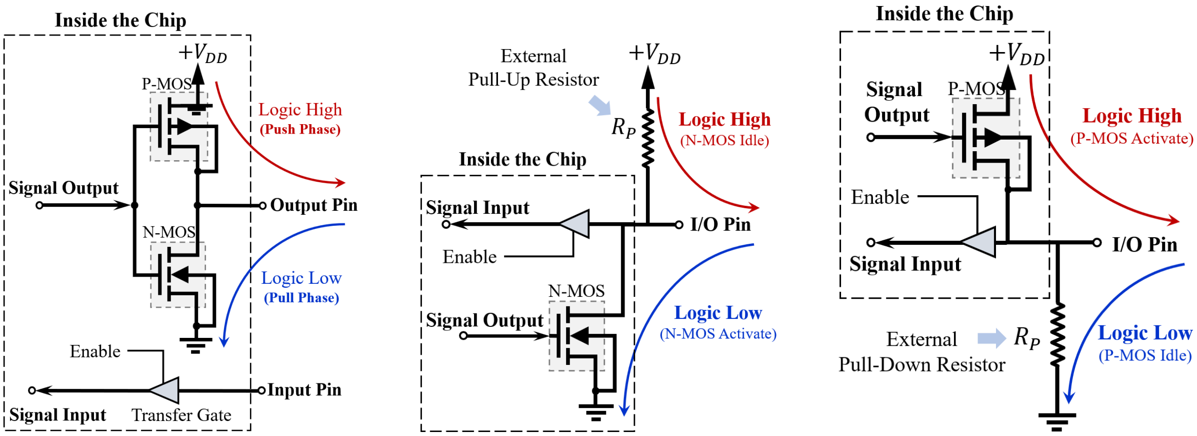
DW instruction(keywords: *DW***):** The code format for data write instruction consists of four fields, *i.e.*, 4-bit device code, 4-bit keyword code, 4-bit pin number code, and n-bit write-in data code (the data to write in the target chip).

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.



(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.

3 Connection type for COTS 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). 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).

Case Study for COTS Chips with UCDL

For a better induction, we provide a set of tables that describe the heterogeneous features of 20 different chips in detail through UCDL, demonstrating that UCDL has the capability of extensive support for COTS chips. In those tables, we discuss the detailed features for each chip, contains information of their type, name, communication protocol, ID, and how many detailed functions we have achieved (full or basic). Then, for each chip, we listed the detailed functions, and gave the UCDL control sequence for one of the functions. The listed functions represent the specific functions provided by the chip (not including pin configuration and data converting that provided by UCDL). Finally, we give a demonstration of data conversion for each chip, and add necessary annotations. For the data conversion, LEGO supports functions written in strings bu utilizing QScriptEngine[1], and we achieve complex data computing for chip outputs.

The heterogeneity characteristics of COTS chips are presented in three tables (in page 20-22). The first part (Table 1) is about different types of digital chips, which is controlled through different digital buses (*e.g.*, I2C, SPI, I2S); the second part (Table 2) is about different types of analog chips, which is controlled by ADC (analog-to-digital converter) and DAC (digital-to-analog converter) on LEGO device; the final part (Table 3) is the comparison of chips in the same function class. We provide the ID for each chip, based on which readers can easily find the original file in the anonymous open source link. It should be noted that there are a few differences in variable naming between the source file and the content discussed in the table, because we have made readability optimization in the table presentation. The details are as follows, where our UCDL gives a good support for all of them.

Table 1: **Features comparison of COTS Chips (Part 1: Digital chips in different functions)**

1	**Type: 3-Axis Digital Angular Rate Gyroscope Chip: FXAS21002 ID: 0xFAEE Protocol: SPI States: All 6 Functions Achieved**		
	Detailed Functions[1]	Control Sequence of function 1	Example of Data Conversion
	1. Read Angular Rate Register Data 2. From Standby Mode Change to Active Mode 3. From Ready Mode Change to Active Mode 4. Change to Ready Mode 5. Change to Active Mode 6. Reset Device and Change to Standby Mode	1), CW(2, L), +2us; 2), DW(3, 1, 0x81), +10us; 3), DR(5, 6, X1, X2, X3, X4, X5, X6), +60us; 4), CW(2, H), +2us;	1), DF(Y1, variable1, (Y1=X1<<8)+X2, null); 2), DF(Y2, variable2, (Y2=X3<<8)+X4, null); 3), DF(Y3, variable3, (Y3=X5<<8)+X6, null); 4), DF(Y4, X-angular, (Y4=Y1,Y1<32768; Y4=(Y1-65536),Y1>=32768), degree/s); 5), DF(Y5, Y-angular, (Y5=Y2,Y2<32768; Y5=(Y2-65536),Y2>=32768), degree/s); 6), DF(Y6, Z-angular, (Y6=Y3,Y3<32768; Y6=(Y3-65536),Y3>=32768), degree/s);
	Note[1]: The listed functions represents the specific functions provided by the chip.		**Annotation:**1)"Y4, Y5, Y6" are final data of 3-axis angular rates,respectively. 2),"Y1 to Y3" are intermediate variables. 3),"X1 to X6" are raw angular data read by Function 1.
2	**Type: Digital Microphone Chip: INMP441 ID: 0x7C60 Protocol: I2S States: 1 Basic Function Achieved (1/2 of all functions)**		
	Detailed Functions	Control Sequence of function 1	Example of Data Conversion (None[2])
	1. Get I2S Data	1), DR(2, I2S, DIRECT), +1s;	[2]**Annotation:** The I2S output data is directly send to the gateway and decode by third-party audio software.
3	**Type: Digital Accelerometer Chip: ADXL362 ID: 0xCCFC Protocol: SPI States: 3 Basic Functions Achieved (3/5 of all)**		
	Detailed Functions	Control Sequence of function 1	Example of Data Conversion
	1. Read Acceleration Data 2. Set Standby Mode 3. Set WAKE-UP Mode	1), CW(5, L),+2us; 2), DW(3,0x0B),+2us; 3), DW(3,0x08),+2us; 4), DR(4,1,X1),+6us; 5), CW(5, H),+2us;	1), DF(Y1, X-axis Acceleration, (Y1=X1/64, X1<128; Y1=(X1-256)/64, X1>=128), g); **Annotation:** 1), "Y1" is the final X-axis acceleration data. 2), "X1" is an 8bit signed raw data read by Function 1
4	**Type: 256Bytes EEPROM (Memory) Chip: AT24C02 ID: 0x94B5 Protocol: I2C States: 3 Basic Functions Achieved (3/5 of all)**		
	Detailed Functions	Control Sequence of function 1	Example of Data Conversion
	1. Read All Data 2. Set Address 3. Write Data	1), DR(5,0xA1,0x00,256), +100ms;	**Annotation:** The output data of Memory is displayed directly on the gateway. (not need to make the conversion)
5	**Type: 8-bit Temperature Sensor Chip: LM99 ID: 0x8169 Protocol: SmBus States: 2 Basic Functions Achieved (2/3 of all)**		
	Detailed Functions	Control Sequence of function 1	Example of Data Conversion
	1. Read Local Temperature Register Data 2. Read Remote Temperature Register Data	1), DR(5, 0x99, 0x00, 1, X1), +64ms;	1), DF(Y1, Temperature, (Y1=X1,X1<128; Y1=X1-256,X1>=256), degree C); **Annotation:** 1), "X1" is a 8bit signed raw data. 2), "Y1" is the final temperature data.

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

$\triangleright$ **Continue with table1 (Part1: Digital chips in different functions)**

6	**Type: Digital Baroceptor Chip: 2SMPB-02E ID: 0x010F Protocol: SPI States: 3 Basic Functions Achieved (3/4 of all)**		
	Detailed Functions	Control Sequence of function 1	Example of Data Conversion
	1. Read Compensation Variables 2. Read Temperature Register Data 3. Read Pressure Register Data	1), CW(2, L),+2us; 2), DW(3, 1, 0xA0),+2us; 3), DR(5, 25),+26us; 4), CW(2, H),+2us;	1), DF(Y1, variable1, Y1=X29*pow(2, 16)+X30*pow(2, 8)+X31, null); 2), DF(Y2, variable2, Y2=(X19*pow(2, 12)+X20*pow(2, 4)+X25&0x0F)/16, null); 3), DF(Y3, variable3, Y3=-6.3*pow(10, -3)+4.3*pow(10, -4)*(X21*pow(2, 8)+X22)/32767, null); 4), DF(Y4, variable4, Y4=-1.9*pow(10, -11)+1.2*pow(10, -10)*(X23*pow(2, 8)+X24)/32767, null); 5), DF(Y5, coe5, Y5=Y2+Y3*Y1+Y4*pow(Y1, 2), null); 6), DF(Y6, Temperature, Y6=Y5/256, degree C);
	Note: We present some special chips with complex data conversion that contain a large amount of compensation from different "read" functions. It indicates that our UCDL can support complex data computing for chip outputs.		**Annotation:** 1),"X19, X20, X21, X22, X23, X24, X25" are compensation variables read by Function 1. 2),"Y1-Y5" are intermediate variables. 3),"X29-X31" are raw temperature data read by Function 2. 4), Y6 is the final temperature data.
7	**Type: Digital LFV[3] Sensor Chip: SLF3S-0600F ID: 0x3938 Protocol: I2C States: 2 Basic Functions Achieved (2/3 of all)**		
	Detailed Functions	Control Sequence of function 1	Example of Data Conversion
	1. Read Flow Data 2. Soft Reset;	1), DW(1, 0x10, 0x36, 1, 0x08),+600us; 2), DR(1, NOADDR, 0x11, 3, X1, X2, X3),+100us;	1), DF(Y1, Flow Velocity, Y1=X1<<8+X2, ul/min); 2), DF(Y2, CRC, Y2=X3, null);
	Note[3]: LFV is the abbreviation of Liquid Flow Velocity.		**Annotation:** 1), "X1" and "X2" are the upper and lower 8-bits of the raw data for fluid velocity sensing, respectively. 2), "X3" is the CRC (Cyclic redundancy Check) code of "X1" and "X2", which helps the gateway to check the correctness of the uploaded data. 3), "Y1" is the final data for liquid flow velocity sensing.

Table 2: **Detailed features of COTS Chips (Part 2: Analog chips in different functions)**

8	**Type: Dual Axis Tilt Sensor Chip: AXISENSE-2 ID: 0xF038 Protocol: Analog Output States: All 1 Function Achieved**		
	Detailed Functions	Control Sequence of function 1	Example of Data Conversion
	1. Read Analog Angular (Achieved by ADC Sampling)	1), DR(3, ADC,X1), +10us; 2), DR(4, ADC,X2), +3ms;	1), DF(Y1, X_Axis, Y1=((5/pow(2,8))*(X1)-2.5)*45, Degree); 2), DF(Y2, Y_Axis, Y2=((5/pow(2,8))*(X2)-2.5)*45, Degree); **Annotation:** 1), "X1, X2" are the raw data output by ADC. 2), "X1, X2" are the final data for tilt angle in X-axis and Y-axis, respectively
9	**Type: UV (ultraviolet ray) Sensor Chip: ML8511A ID: 0x7DB9 Protocol: Analog Output States: All 1 Function Achieved**		
	Detailed Functions	Control Sequence of function 1	Example of Data Conversion
	1. Read Analog UV (Ultraviolet Ray) Value (Achieved by ADC Sampling)	1), DR(4, ADC,X1), +1ms;	1), DF(Y1, UV Intensity,Y1=((5/pow(2,8))*(X1)-1)*7.5, mW/cm2); **Annotation:** 1), "X1" is the raw data output by ADC. 2), "Y1" is the final data for UV intensity sensing
10	**Type: Analog Dust Sensor Chip: GP2Y1014AU ID: 0x126F Protocol: Analog Output States: All 1 Function Achieved**		
	Detailed Functions	Control Sequence of function 1	Example of Data Conversion
	1. Data Reading (Achieved by ADC Sampling)	1), CW(3, H), +280us; 2), DR(5, ADC, X1), +40us; 3), CW(3, L), +9680us;	1), DF(Y1, Dust-density, Y1=0.17*X1*5/255-0.1, mg/m^3); **Annotation:** 1), "X1" is the raw data output by ADC. 2), "Y1" is the final data for dust-density sensing.
11	**Type: Analog Alcohol Sensor Chip: MQ3 ID: 0x39ED Protocol: Analog Output States: All 3 Functions Achieved**		
	Detailed Functions	Control Sequence of function 1	Example of Data Conversion
	1. Get Raw Data (Achieved by ADC Sampling) 2. MQ3 warm up 3. Check if alcohol exists	1), DR(4, ADC, X1), +3ms;	1), DF(Y1, Y1=raw-data, X1/255*5, null); 2), DF(Y2, Alcohol, Y2=(Y1-1)*10*20, ppm); **Annotation:** 1), "X1" is the raw data output by ADC. 2), "Y1" is the analog voltage output by the chip. 3), "Y2" is the final data for alcohol concentration sensing

▷ **Continue with table2 (Part2: Analog chips in different functions)**

No.	Detailed Functions	Control Sequence of function 1	Example of Data Conversion
	Type: Analog Gas Sensor Chip: MQ5 ID: 0x454B Protocol: Analog Output States: All 3 Functions Achieved		
12	1. Get Raw Data (Achieved by ADC Sampling) 2. MQ5 warm up 3. Check if gas exists	1,) DR(4, ADC, X1), +3ms;	1,) DF(Y1, raw-data, Y1=X1/255*5, null); 2,) DF(Y2, gas concentration, Y2=(Y1-1)*10*20, ppm); **Annotation:** 1,) "X1" is the raw data output by ADC. 2,) "Y1" is the analog voltage output by the chip. 3), "Y2" is the final data for gas concentration sensing
	Type: Analog AMR[4] Sensor Chip: ADA4571 ID: 0xC790 Protocol: Analog Output States: All 1 Function Achieved		
13	1. Read Analog Output (achieved by ADC sampling)	1), DR(2, ADC,X1), +10us; 2), (4, ADC,X2), +3ms;	1), DF(Y1, variable1, Y1=X1, null); 2), DF(Y2, variable2, Y2=X2, null); DF(Y3, Angle, arc-tan(Y2/Y1)/2, Degree);
	Note[4]: The function of this chip is AMR (An-isotropic Magneto Resistivity) angular sensing.		**Annotation:** 1), "X1, X2" are raw data output by ADC, 2), "Y1,Y2" are the intermediate variables in the data conversion. 3), "Y3" is the final data for angle sensing.
	Type: Analog Rotary Position Sensor Chip: AMS22S ID: 0x1032 Protocol: Analog Output States: All 1 Function Achieved		
14	1. Read Analog Angular (Achieved by ADC Sampling)	1), DR(2, ADC,X1), +3ms;	1), DF(Y1, Angle, Y1=((5/pow(2,8))*(X1))/5*360, Degree); **Annotation:** 1), "X1" is the raw data output by ADC. 2), Y1 is the final data for angular position sensing

Table 3: **Detailed features of COTs Chips (Part 3: COTS chips in same function classes)**

No.	Detailed Functions	Control Sequence of Function 1	Example of Data Conversion
	Type: Ambient Light Sensor Chip: MAX9635 ID: 0x7B31 Protocol: I2C States: 2 Basic Functions Achieved (2/3 of all)		
15	1. Read Light Register Data in Continuous Mode 2. Set Continuous Mode	1), DR(5, 0x95, 0x03, 2, X1, X2),+7ms;	1), DF(Y1, variable1, Y1=(X1&0xF0)>>4, null); 2) DF(Y2, variable2, ((X1&0x0F)<<4)+(X2&0x0F), null); 3), DF(Y3, Lux, pow(2, EXP)*Y2*0.045, lux);
	Note: We have added some chips with similar functions. It indicates that COTS chips have heterogeneous characteristics even in the same function classes, and our UCDL can describe their features well.		**Annotation:**1), "X1" and "X2" are the raw data read out in Function 1 2), "Y1" and "Y2" are the intermediate variable in data conversion 3), "Y3" is the final data for ambient light sensing
	Type: Ambient Light Sensor Chip: TSL2562 ID: 0xC771 Protocol: SMBus States: 3 Basic Functions Achieved (3/4 of all)		
16	1. Read Register Data 2. Set Sensor Power down 3. Set Sensor Power up	1), DR(5, 0x53, 0xAC, 2, X1, X2), +100us; 2), DR(5, 0x53, 0xAE, 2, X3, X4), +400ms;	1), DF(Y1, variable1, Y1=X1+X2<<8, null); 2), DF(Y2, intermediate variable, Y2=X3+X4<<8, null); 3), DF(Y3, intermediate variable,Y3=Y2/Y1, null); 4), DF(Y4, Lux, (Y4=0.0315*Y1-0.0593*Y1*pow(CH10, 1.4),0<Y3<=0.52; Y4=0.0229*CH0-0.0291*Y2,0.52<Y3<=0.65; Y4=0.0157*Y1-0.0180*Y2,0.65<Y3<=0.80; Y4=0.00338*Y1-0.00260*Y2,0.80<Y3<=1.3; Y4=0,Y3>1.3), degree C);
	Note: The data conversion for this chip is relatively complex, but we can disassemble the process into multiple DF statements with intermediate variables. It proves that our UCDL can support complex computing for chip outputs effectively.		**Annotation:** 1), "X1-X4" are 8 bit raw data read by Function1, 2), "Y1,Y2,Y3" are intermediate variable of the data converting, 3), "Y4" is the final data for ambient light sensing.

▷ **Continue with 3 (Part3: COTS chips in same function classes)**

17	**Type: Relative Pressure Sensor Chip: MLX90809 ID:0xF921 Protocol: SENT States: All 1 Function Achieved**		
	Detailed Functions	Control Sequence of function 1	Example of Data Conversion
	1. Get Raw Data	1), DR(2, 3, X1, X2, X3), +1ms;	1), DF(Y1, variable1, X1<<8+X2<<4+X3, null); 2), DF(Y2, Pressure, (Y2=Y1/4096*3, Y1<2048; Y2=(Y1-4096)/4096*3-1.5, Y1>=2048), %FSO); **Annotation:** 1) "X1-X3" is the raw data read out by Function 1 2) "Y1" is the intermediate variable 3) "Y2" is the final data for relative pressure sensing
18	**Type:Digital Pressure Sensor Chip: MS5525DSO ID: 0xE52C Protocol: SPI States: 4 Basic Functions Achieved (4/6 of all)**		
	Detailed Functions	Control Sequence of function 1	Example of Data Conversion
	1. Read Pressure Register Data 2. Read Temperature Register Data 3. Read Calibration Data 4. Reset	1), CW(4, L),+2us; 2), DW(2, 1, 0x48),+9ms; 3), CW(4, H),+2us; 4), CW(4, L),+2us; 5), DW(2, 1, 0x00),+2us; 6), DR(1, 3, X13, X14, X15),+1us; 7), CW(4, H),+2us;	1), DF(Y11, variable1, Y11=C2*pow(2, 17)+(C3*dT)/128, null); 2), DF(Y12, variable2, Y12=C1*pow(2, 15)+(C3*dT)/128, null); 3), DF(Y13, Pressure, (D1*Y12/pow(2, 21)-Y11)/pow(2, 15), psi); **Annotation:** 1), "C1, C2, C3, dT" are computation results of other "DF" statements. 2), "Y11,Y12" are intermediate variables 3), "Y13" is the final data for air pressure sensing.
19	**Type: Digital Temperature Sensor Chip: DS18B20 ID: 0xAC1B Protocol: 1-Wire States: 3/5 Functions Achieved**		
	Detailed Functions	Control Sequence of function 1	Example of Data Conversion
	1. Read Temperature 2. Init SD18B20 3. Read 8 Bytes Serial Code	1), CW(2, L), +750us; 2), CW(2, H), +15us; 3), DW(2, 2, 0xCC, 0x44), +200us; 4), CW(2, L), +750us; 5), CW(2, H), +15us; 6), DW(2, 2, 0xCC, 0xBE), +200us; 7), DR(2, 2, X1, X2), +200us; composed of X1 and X2.	1), DF(Y1, Raw-data, X1+X2<<8, null); 2), DF(Y2, Temp, (y=Raw-data*0.0625, Raw-data<32768; y=(Raw-data-65536)*0.0625, Raw-data>=32768), degree); **Annotation:** 1), "X1,X2" are the upper and lower 8-bits raw temperature data. 2), "Y1" is the 16-bit of raw temperature data. 3), "Y2" is the final temperature data.
20	**Type: Temperature and RH Sensor Chip: ENS210 ID: 0xB1D2 Protocol: I2C States: All 10 Functions Achieved**		
	Detailed Functions	Control Sequence of function 1	Example of Data Conversion
	1. Read Temperature Registers 2. Disable Low-power Mode 3. Enable Low-power Mode 4. Temp & RH Continuous Mode, 5. Temperature Continuous Mode, RH Single Shot Mode 6. Temperature Single Shot Mode, RH Continuous Mode 7. Temperature Single Shot Mode, RH Single Shot Mode 8. Read PART_ID and UID 9. Read RH Register Data 10. Stop Continuous Measurement	1), DW(3, 0x86, 0x22, 1, 0x01), +130ms; 2), DR(3, 0x87, 0x30, 3, X13, X14, X15), +1ms;	1), DF(Y4, TinK, Y4=(X14*pow(2, 8) +X15)/64, K); 2), DF(Y5, TinC, Y5=Y4-273.15, C); 3), DF(Y6, TinF, Y6=Y5*1.8+32.0, F); **Annotation:** 1), "X13,X14,X15" is the raw data readed from function 1 2), "Y4 (TinK)" is the Kelvin temperature. 3), "Y5 (TinC)" is the Celsius temperature. 4), "Y5 (TinF)" is the Fahrenheit temperature.
		Note: RH is the abbreviation of relative humidity. We add this chip with many functions to illustrate that the UCDL can support COTS chips with complex functions. In this case, we have implemented all ten functions.	