UCDL Syntax Coding

1 Overview

The Unified Chip Description Language (UCDL) is designed to directly operate functional chips that are deployed on IoT devices. Because the LEGO system decouples the chip control logic from IoT devices to the gateway, the device only needs to perform the signal conversion between the gateway and onboard chips that no longer needs a microprocessor to run embedded programs for chip control. This makes the design of UCDL easy without considering the differences between different microprocessors.

In UCDL programming, each instruction is directly linked to a meta-operations of a chip. The gateway parses UCDL instructions by mapping the keywords and variables to the corresponding codes and connecting them in series to form a control flow. The coding details are presented in Section.2. When a chip is plugged in, the LEGO device uploads the plugged slot number together with the chip ID to the gateway, which helps the gateway to embed the chip location information in the downstream control flow. Therefore, the user does not need to care about the specific slot position when plugging and playing functional chips.

In practical applications, the gateway orchestration the generated downlink instructions by a novel designed three-layer instruction orchestration architecture (chip-functions-operation). By this, the generated instructions only need to issue once to write in the buffer framework of the target LEGO device when a new chip is plugged in, thus minimize communication cost.

2 Coding Details

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. In a description file, the instructions for pin configurations (keywords: PIN) and logic control (keywords: DW, DR, CW, SR) are converted into gateway instructions. In addition, 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, we discuss it in the end of this section.

The encoding scheme of the 5 gateway instructions is as follows:

- **PIN instruction(keywords: PIN):** The code format for pin configuration instruction consists of six fields, *i.e.*, 4-bit device code, 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.
- **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.

The code mapping of the keywords and variables of above five instructions is shown in Table 1. For instance, instruction $'0011-1000-0101-0010-11000'$ means to set the 5th 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 Adaptive Signal Conversion (ASC) circuit acts accordingly to build up the required connections.

In addition, as we discussed in early of this section, the keyword of DF is designed for the gateway to parse the data from the LEGO device and does not need to code into downlink instructions. The variables (j, imp, func, unit) are directly extracted on the gateway for data calculation and display. For instance, DF(1, X-axis acceleration, Y=(X[1]-127)/64, g) means the first output of the chip is X-axis acceleration, the data

[1] X is the original data output from the chip and Y is the final data for IoT applications.

Table 1: UCDL Syntax Coding

	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
	Pin Functions	Code	Description
Coding For Pin Functions	CLK	0010	Clock signal transmission
	VCC	0011	Power supply
	DW	0100	Data input(write)
	DR	0101	Data output(read)
	CW	0110	Control input(write)
	SR	0111	State output(read)
	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

converting function is Y=(X-127)/64 and the data unit is g. Hence, when a data (*e.g*, 10011010) is received, the gateway convert it to 0.203 by the data converting function, and display: X-axis acceleration=0.203g for users.

3 Current Limitations and future designs

In the current state, the difference in voltage level between different chips is not considered in the language, and the LEGO device supports chips to be interacted with I/O voltages in 1.2-3.3V by default. If a chip has I/O voltages out of this range, a level shifter should be added between the chip and LEGO devices, which will incur additional costs. In addition, the current UCDL design also does not support analogue chips. To access analog chips, an Analog-to-Digital Converter (ADC) should be added to the system, and it also incurs cost.

Those limitation shows up because the current version design is mainly focused on feasibility verification, so we only cover the most common cases. The above shortcomings will be settled in future works.