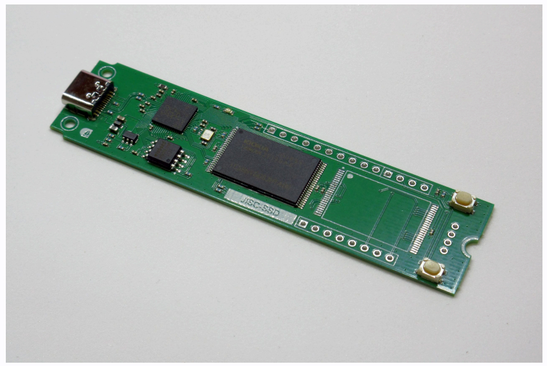
RPi Pico の NAND アクセスを高速化する

背景

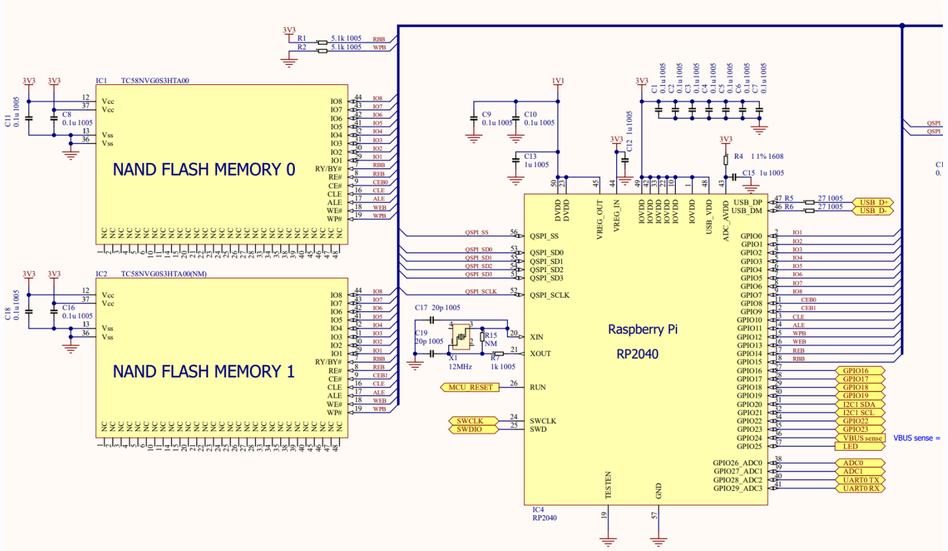
最近 SSD 自作キット JISC-SSD で ~~時折放置してますが~~ 遊んでいます。
途中まで C++で書いていましたが、気まぐれで Rust 再実装中...

RPi Pico: RP2040 Cortex-M0+ DualCore @133MHz
NAND Flash: KIOXIA TC58NVG0S3HTA00 1Gbit SLC ECC なし

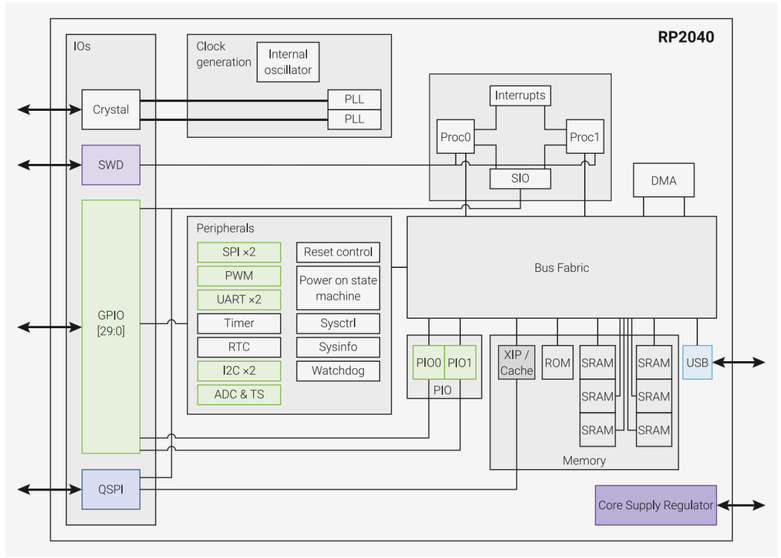
外観



回路図抜粋



RP2040 Block Diagram



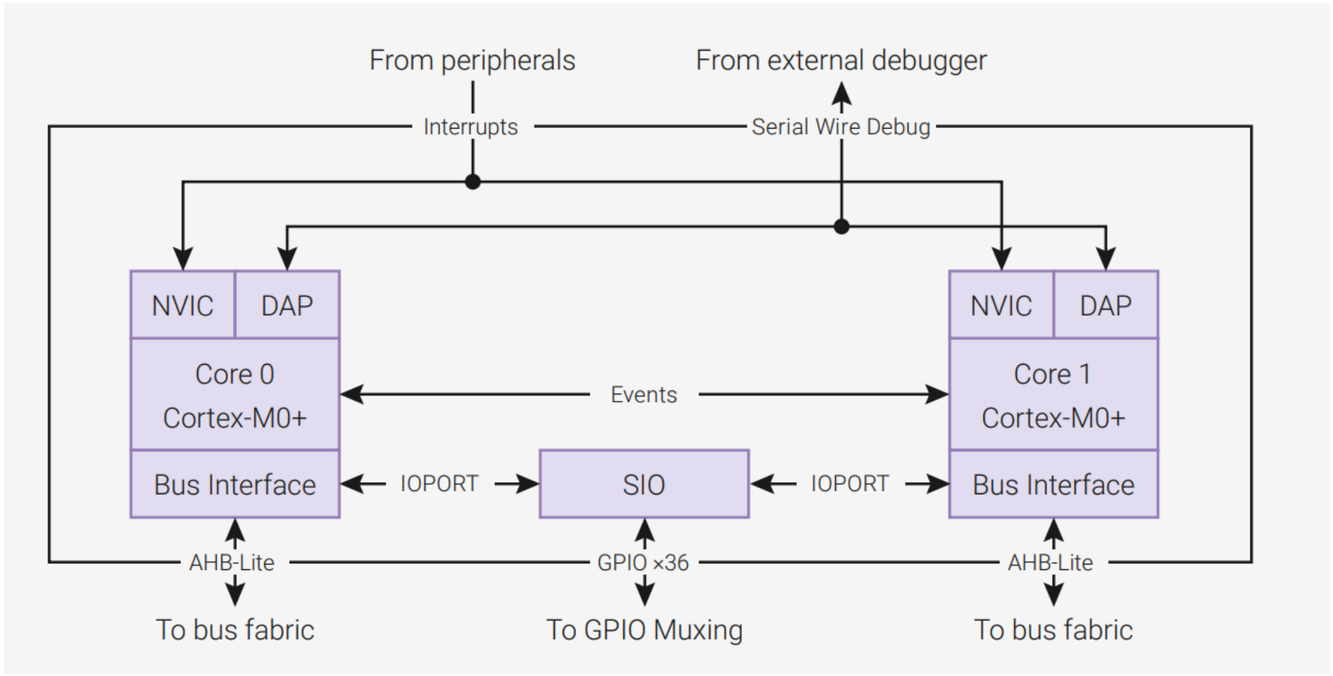
GPIO で NAND と直結

課題

NAND IC へのアクセスをもう少しいい感じにしたい

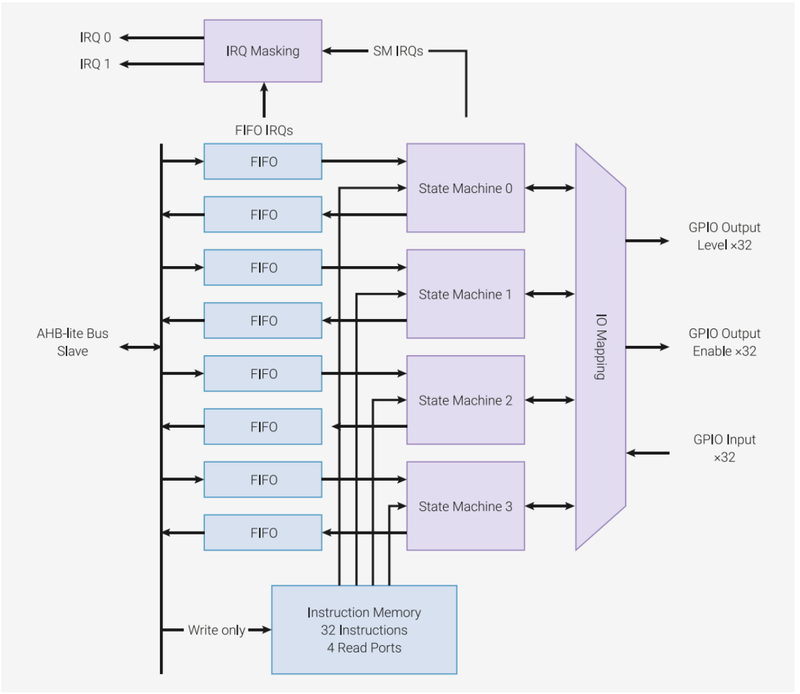
GPIO 直叩きでの通信は遅い（当然と言えばそう）
SIO (Single Cycle IO) もあるが、転送中 CPU Resouce 占有してしまう
PIO (Programmable IO) という小規模なシーケンサのような HW が乗っている

SIO



bus fabric 経由せず GPIO 叩ける

PIO Overview



State Machine 4 機

PIO 概要

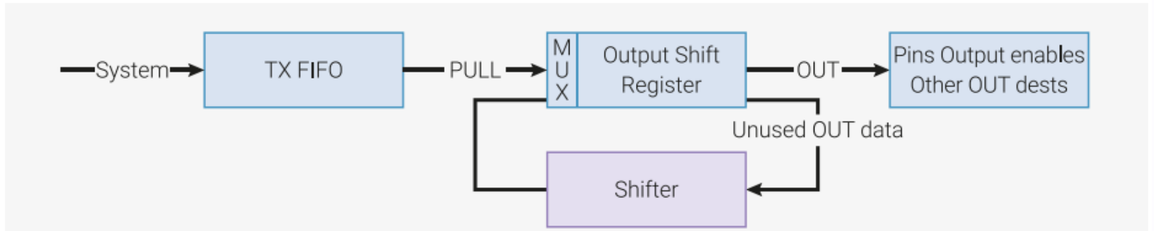
- 動作周波数はデフォルトで 125MHz, uCode は 32 命令まで
- 命令セットはデータ入出力+入出力時の bitshift、分岐命令 あたり
- GPIO の Read/Write/PinDir 変更、FIFO 経由で CPU/DMA とやりとりできる
- 命令実行と同時に固定サイクル遅延 (Delay)、もしくは特定ピンの H/L を変更できる (Sideset)
  - この 2 機能に割り当てられる bit は合わせて 5bit まで

命令セット

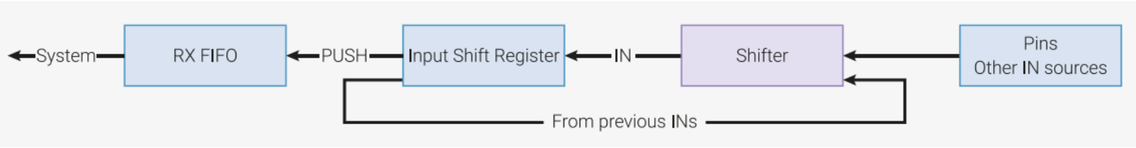
Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JMP	0	0	0	Delay/side-set					Condition			Address				
WAIT	0	0	1	Delay/side-set					Pol	Source		Index				
IN	0	1	0	Delay/side-set					Source			Bit count				
OUT	0	1	1	Delay/side-set					Destination			Bit count				
PUSH	1	0	0	Delay/side-set					0	IfF	Blk	0	0	0	0	0
PULL	1	0	0	Delay/side-set					1	IfE	Blk	0	0	0	0	0
MOV	1	0	1	Delay/side-set					Destination			Op		Source		
IRQ	1	1	0	Delay/side-set					0	Clr	Wait	Index				
SET	1	1	1	Delay/side-set					Destination			Data				

Shift Register

TX FIFO -> Output Shift Reg



Input Shift Reg -> RX FIFO



Delay/sideset の bit 割り振りは起動前に設定

NAND IC との通信

GPIO 複数ピンによるコマンド入力 + 双方向データ転送が必要

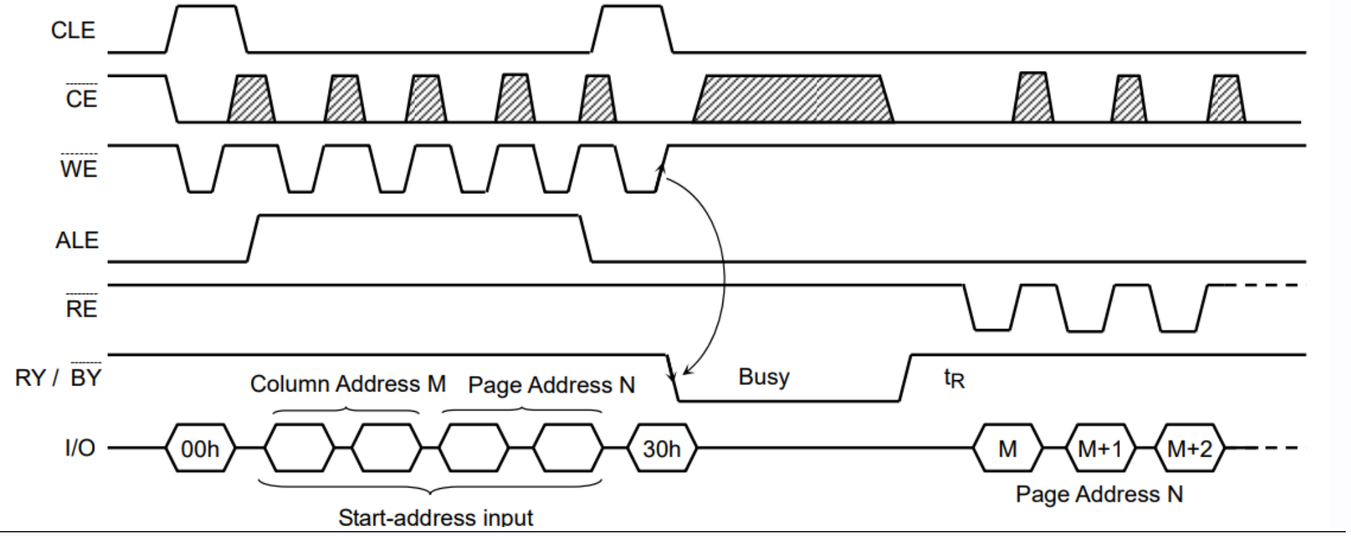
Pin	Function
/CS	Chip Enable
/WE, /RE	Write/Read Enable。転送クロック相当
ALE, CLE	Address/Command Latch Enable。入力データ区別
RY//BY	Ready/Busy: Busy なら Low
IO[7:0]	データ線。双方向

Logic & Command Table

Table 2. Logic Table

	CLE	ALE	$\overline{CE}$	$\overline{WE}$	$\overline{RE}$	$\overline{WP}$*1
Command Input	H	L	L	↑	H	*
Data Input	L	L	L	↑	H	H
Address Input	L	H	L	↑	H	*
Serial Data Output	L	L	L	H	↓	*
During Program (Busy)	*	*	*	*	*	H
During Erase (Busy)	*	*	*	*	*	H
During Read (Busy)	*	*	H	*	*	*
	*	*	L	H (*2)	H (*2)	*
Program, Erase Inhibit	*	*	*	*	*	L
Standby	*	*	H	*	*	0 V/V_{CC}

Read Operation



Cmd In (0x00)-> Addr In -> Cmd In (0x30) -> Wait RY/BY -> Data Out...

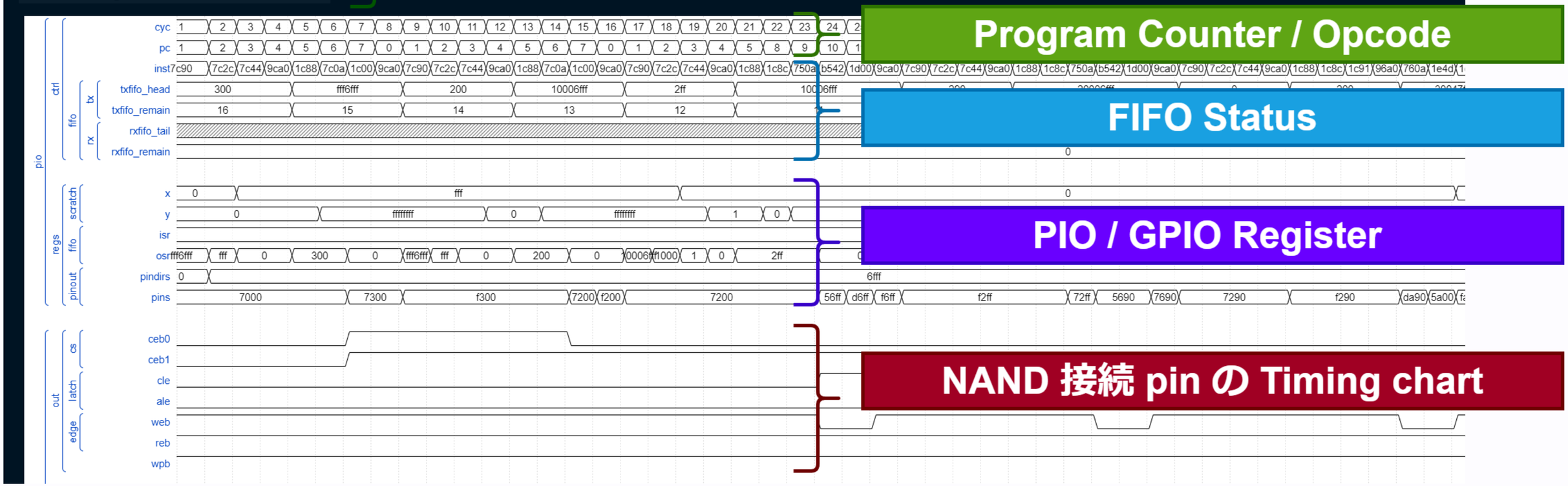
Simulation/Debug 環境

デバッガがないため、 Jupyter 上に論理検証環境構築

adafruit_pioasm + pioemu + pandas + wavedrom

	cycle	pc	event	ceb0	ceb1	io	io_dir
0	26	0	cmd_in	0	1	0x000000FF	0x000000FF
1	36	0	cmd_in	0	1	0x00000090	0x000000FF
2	47	16	addr_in	0	1	0x00000000	0x000000FF
3	60	21	data_out	0	1	0x0000001D	0x00000000
4	65	21	data_out	0	1	0x00000020	0x00000000
5	70	21	data_out	0	1	0x00000022	0x00000000
6	75	21	data_out	0	1	0x00000025	0x00000000
7	80	21	data_out	0	1	0x00000027	0x00000000

NAND IC / PIOとのやりとりをデータ出力 / Verify



Program Counter / Opcode

FIFO Status

PIO / GPIO Register

NAND 接続 pin の Timing chart

設計

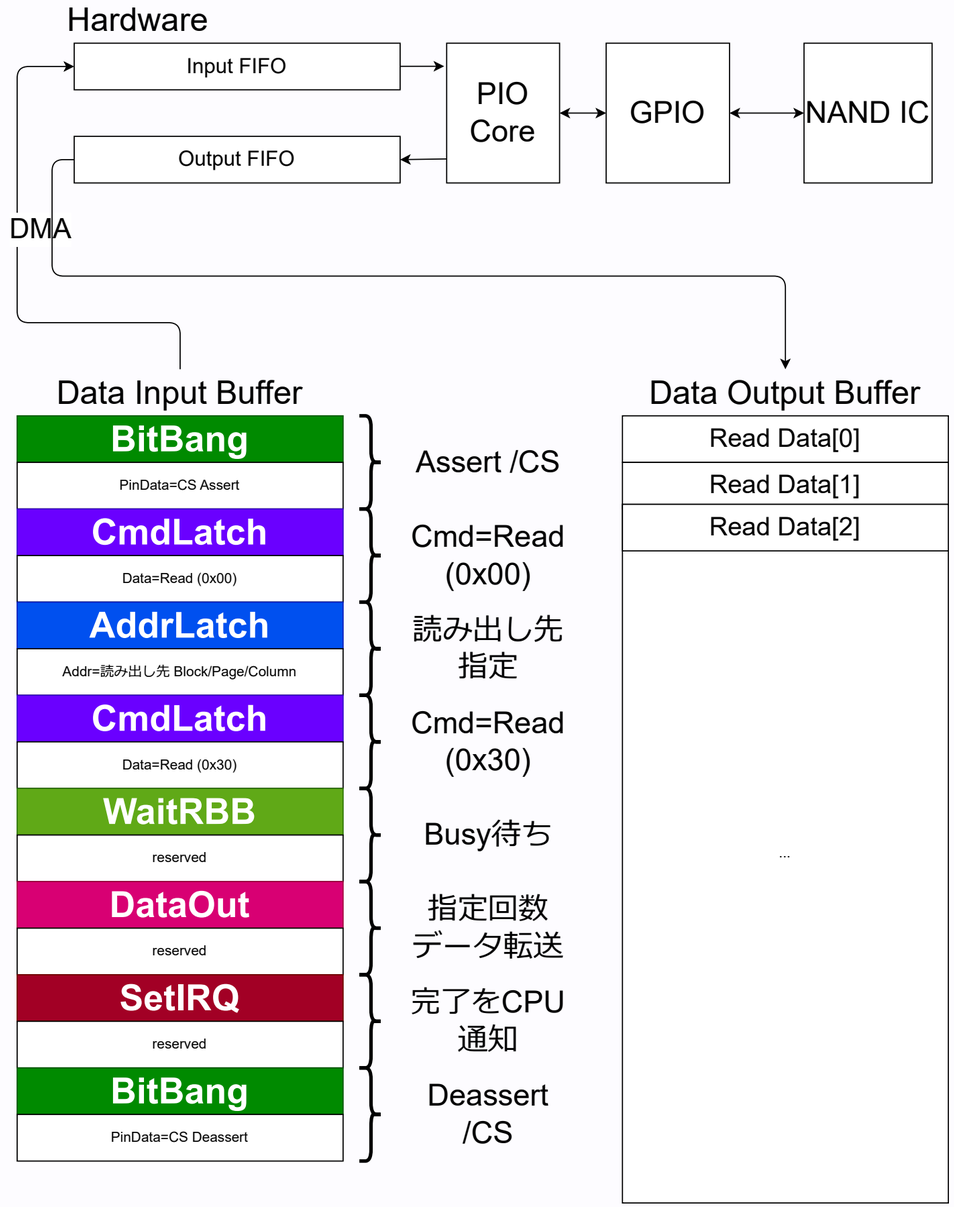
DMA で流すデータ上に独自コマンドセットを定義

CmdId を指定し、各シーケンスごとのルーチンを実装。CPU は Buffer 上にシーケンスを組む
(流すデータの順番があっていればよいので、連続領域に確保しなくても DMA を Chain で良い)

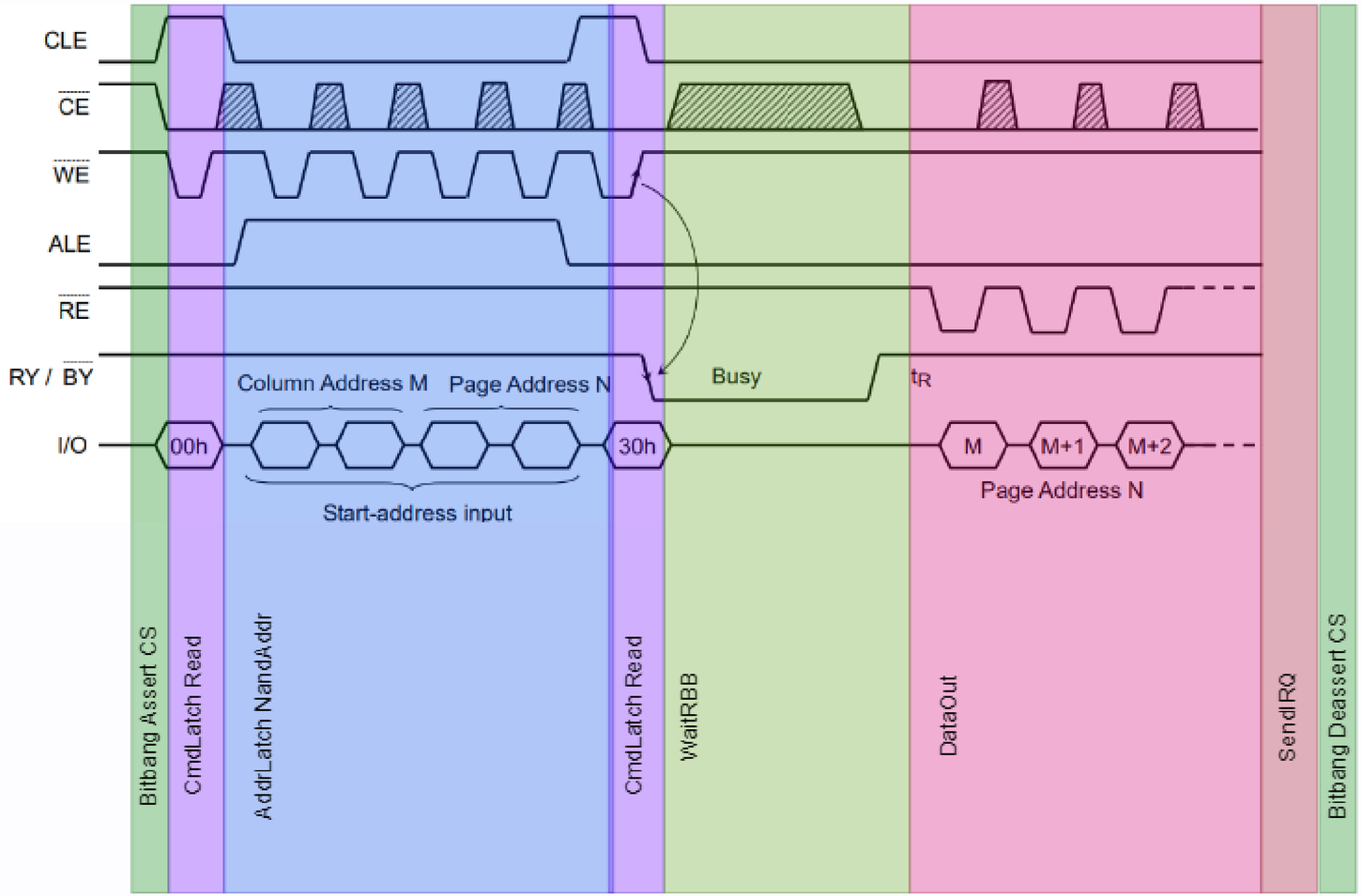
Data[0] assign	Field	Description
[31:28]	CmdId	コマンド指定
[27:16]	TransferCount	(AddrLatch/DataIn/DataOut のみ) 転送 byte 数指定
[15:0]	PinDirection	GPIO の入出力設定

CmdId	Data[1, ...]	Description
Bitbang	PinData	そのまま GPIO に流す
CmdLatch	NandCmdId	CLE 有効で Data 入力
AddrLatch	NandAddr	ALE 有効で Data 入力
DataOutput	-	指定回数 Data 出力 + GPIO Read
DataInput	TransferDatas	指定回数 Data 入力
WaitRbb	-	RB/BY pin が Ready になるまでループ
SetIrq	-	割り込み通知（転送完了を CPU に知らせる用）

PIO Process Diagram

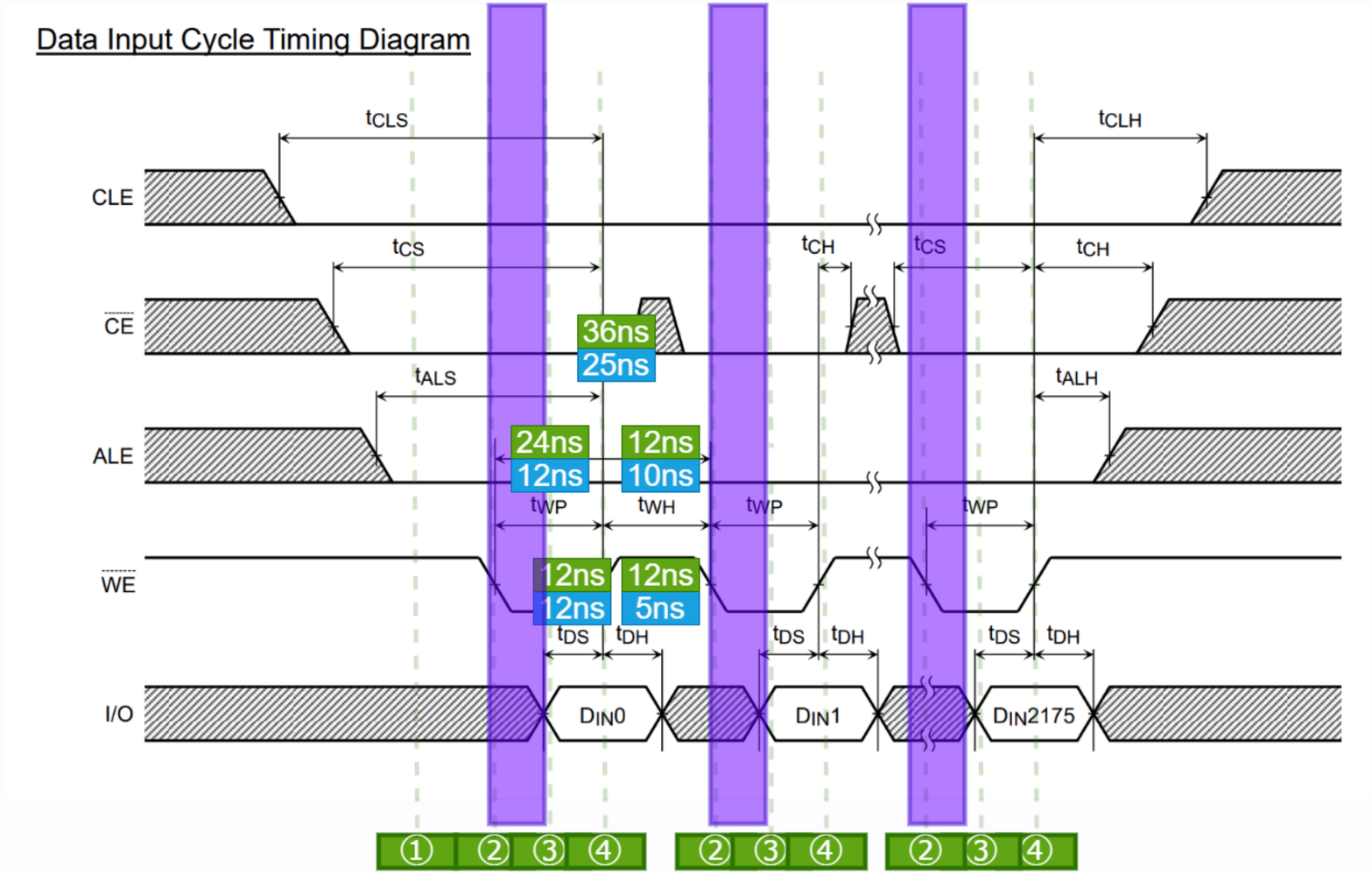


Sequence



性能 (Data Input の例)

f_{Max} で動かすと Data の Setup 満たさないので 83MHz (12ns) より下げて動かす必要あり。
GPIO Toggle よりはだいぶ速くなったが、AC 特性 Spec と比べるともうひと声。



AC Characteristic Condition

25 [ns/byte] => 36.15 [MB/s]

PIO Operation (f=83.3MHz)

36 [ns/byte] => 26.49 [MB/s]

GPIO Toggleよりは十分速いが...もう少し頑張れそう
→ Auto Pull使えばもう少し縮むかも

```
data_input_setup:
    jmp y--, wait_rbb_setup      side 0b11100 ; ①
data_input_main:
    pull block                   side 0b10100 ; ②
    out pins, 10                 side 0b10100 ; ③
    jmp x-- data_input_main      side 0b11100 ; ④
    jmp setup                    side 0b11100 ; ⑤
```

まとめ

所感

PIO は小規模ながら色々できて面白い
System Clock (Max 133MHz) で動くのでホビー用途なら十分速い
実機デバッグはしんどいので Sim 環境作っておいてよかった
FTL としての続きもぼちぼち実装します

出典

JISC-SSD(Jisaku In-Storage Computation SSD 学習ボード) - 株式会社クレイン電子
RP2040 Datasheet - Raspberry Pi Ltd
TC58NVG0S3HTA00 Datasheet - KIOXIA

Fin.