CPU T-4 Reference

CPU T-4 belongs to the fourth generation of Ternary CPU T series. It operates with trytes — 6-trit data arrays. It has unified RAM for instructions and data; address and data bus are both 6-trits in length. The CPU has three registers with one write channel and two read channels. In comparison to the previous generation, CPU T-4 has two less instruction conveyor stages (faster instruction loading), Instruction Decoder, reduced ALU (3 basic operations instead of 9 – smaller size) and few minor data path improvements. The important issue is that CPU T-4 can use binary clock.

Data path state variables

CPU T-4 has sixteen variables that control the state of the data path system.

Variable name	Variable meaning	Controlled states
ins-len	Instruction length in trytes	λ – 1 tryte 0 – 2 trytes 1 – 3 trytes
ic-src	Instruction counter source selector	λ – RAM output 0 – second instruction address 1 – second ALU operand
jmp-typ	Jump condition type	λ – complex condition 0 – no condition 1 – simple condition
jmp-con	Jump condition value	λ – ZF < 0 0 – ZF = 0 1 – ZF > 0
reg-act	Register action control	~~λ — fill~~ (forbidden) 0 — idle 1 — write
reg-src	Register write data source	λ — RAM output 0 — first register channel 1 — ALU output
reg-adr	Register address selector	λ — RZ 0 — R0 1 — R1
reg-ch1	First register channel selector	λ — RZ 0 — R0 1 — R1
reg-ch2	Second register channel selector	λ — RZ 0 — R0 1 — R1
ram-act	RAM action control	~~λ — fill~~ (forbidden) 0 — idle 1 — write
ram-src	RAM write data source	λ — first register channel 0 — second immediate 1 — ALU output

Variable name	Variable meaning	Controlled states
ram-adr	RAM address selector	λ — ALU output 0 — instruction counter 1 — first immediate
alu-act	ALU action selector	λ — Adder 0 — NANY 1 — Logical MUL (mask)
alu-op1	First ALU operand selector	λ — RAM output 0 — second register channel 1 — instruction counter
alu-op2	Second ALU operand selector	λ — first immediate 0 — first register channel 1 — second immediate
op2-mod	ALU's 2nd operand modification	λ — negate 0 — leave as is 1 — replace with const

Instructions and opcodes

CPU T-4 instructions have variable length: 1-3 trytes each. First tryte (obligate for every instruction) is instruction header «opcode». Table shows the meaning of opcode trits T0 … T5.

T5		T4	T3	T2	T1	T0
Instruction family:	1: mov	λ: mov reg → x 0: mov [i1] → reg 1: mov i2 → [i1]	λ: x = reg 1: x = [i1]	---	reg-ch1	reg-adr
	0: ALU	λ: alu reg, reg → [i1] 0: alu [i1], reg → reg 1: alu [i1], i2 → reg	alu-act	op2-mod	reg-ch1	reg-adr / / reg-ch2
	λ: Jxx	λ: reboot 0: jump 1: finish	ic-src	jmp-typ	jmp-con	alu-op2

The following table shows all possible instructions, their opcodes and how each opcode should be decoded by the Instruction Decoder to the set of state variables. Register with an over line must contain such value that allows the Instruction Decoder to guess the missing (unstated) register. Guessing method is pretty simple: Instruction Decoder looks at the two stated registers and selects the unstated one so that all three registers are present.

Instruction	Opcode	ins len	ic src	jmp typ	jmp con	reg act	reg src	reg adr	reg ch1	reg ch2	ram act	ram src	ram adr	alu act	alu op1	alu op2	op2 mod
I. MOV instructions																	
mov a → u	1λλ0au	λ	0	0	0	1	0	u	a	u	0	λ	1	0	0	0	0
mov [i1] → u	10λ00u	0	0	0	0	1	λ	u	0	u	0	1	1	0	0	0	0
mov a → [i1]	1λ10a0	0	0	0	0	0	0	0	a	0	1	λ	1	0	0	0	0
mov i2 → [i1]	111000	1	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0

Instruction	Opcode	ins len	ic src	jmp typ	jmp con	reg act	reg src	reg adr	reg ch1	reg ch2	ram act	ram src	ram adr	alu act	alu op1	alu op2	op2 mod
II. ALU instructions																	
add a, c → [i1]	0λλ0ca	0	0	0	0	0	1	a	c	a	1	1	1	λ	0	0	0
add [i1], c → u	00λ0cu	0	0	0	0	1	1	u	c	u	0	1	1	λ	λ	0	0
add [i1], i2 → u	01λ00u	1	0	0	0	1	1	u	0	u	0	1	1	λ	λ	1	0
sub a, c → [i1]	0λλλca	0	0	0	0	0	1	a	c	a	1	1	1	λ	0	0	λ
sub [i1], c → u	00λλcu	0	0	0	0	1	1	u	c	u	0	1	1	λ	λ	0	λ
sub [i1], i2 → u	01λλ0u	1	0	0	0	1	1	u	0	u	0	1	1	λ	λ	1	λ
nand a, c → [i1]	0λ00ca	0	0	0	0	0	1	a	c	a	1	1	1	0	0	0	0
nand [i1], c → u	0000cu	0	0	0	0	1	1	u	c	u	0	1	1	0	λ	0	0
nand [i1], i2 → u	01000u	1	0	0	0	1	1	u	0	u	0	1	1	0	λ	1	0
msk a, c → [i1]	0λ10ca	0	0	0	0	0	1	a	c	a	1	1	1	1	0	0	0
msk [i1], c → u	0010cu	0	0	0	0	1	1	u	c	u	0	1	1	1	λ	0	0
msk [i1], i2 → u	01100u	1	0	0	0	1	1	u	0	u	0	1	1	1	λ	1	0
III. Code flow instructions																	
jmp [i1]	λ0λ000	0	λ	0	0	0	1	0	0	0	0	1	1	0	0	0	0
jmp i1	λ0100λ	0	1	0	0	0	1	0	0	0	0	1	1	0	0	λ	0
jl [i1]	λ0λ1λ0	0	λ	1	λ	0	1	0	0	0	0	1	1	0	0	0	0
jl i1	λ011λλ	0	1	1	λ	0	1	0	0	0	0	1	1	0	0	λ	0
je [i1]	λ0λ100	0	λ	1	0	0	1	0	0	0	0	1	1	0	0	0	0
je i1	λ0110λ	0	1	1	0	0	1	0	0	0	0	1	1	0	0	λ	0
jg [i1]	λ0λ110	0	λ	1	1	0	1	0	0	0	0	1	1	0	0	0	0
jg i1	λ0111λ	0	1	1	1	0	1	0	0	0	0	1	1	0	0	λ	0
jle {jng} [i1], i2	λ00λ11	1	0	λ	1	0	1	0	0	0	0	1	1	0	0	1	0
jle {jng} i1, i2	λ01λ10	1	1	λ	1	0	1	0	0	0	0	1	1	0	0	0	0
jle {jng} i2, [i1]	λ0λλ11	1	λ	λ	1	0	1	0	0	0	0	1	1	0	0	1	0
jlg {jne} [i1], i2	λ00λ01	1	0	λ	0	0	1	0	0	0	0	1	1	0	0	1	0
jlg {jne} i1, i2	λ01λ00	1	1	λ	0	0	1	0	0	0	0	1	1	0	0	0	0
jlg {jne} i2, [i1]	λ0λλ01	1	λ	λ	0	0	1	0	0	0	0	1	1	0	0	1	0
jeg {jnl} [i1], i2	λ00λλ1	1	0	λ	λ	0	1	0	0	0	0	1	1	0	0	1	0
jeg {jnl} i1, i2	λ01λλ0	1	1	λ	λ	0	1	0	0	0	0	1	1	0	0	0	0
jeg {jnl} i2, [i1]	λ0λλ1	1	λ	λ	λ	0	1	0	0	0	0	1	1	0	0	1	0
finish	λ11000	λ	1	0	0	0	1	0	0	0	0	1	1	0	0	0	1
reboot	λλ1000	λ	1	0	0	0	1	0	0	0	0	1	1	0	0	0	1

Instruction Decoder

As CPU T-4 operates with encoded instructions, it needs the Instruction Decoder in order to decode them. The table above shows that ID can be built with the use of rational amount of

transistors, because many state variables are coloured with grey (what means that they do not involve many dependencies). The next table defines each state variable as the ternary function of the opcode trits T5 … T0. Note that these functions can be built with the use of threshold logic (what allows to use optimal number of transistors).

State var.	**Function**
ins-len	λ **if** (T5=1 **and** T4=T3=λ) **or** (T5=λ **and** T4≠0) **else** 1 **if** (T5>λ **and** T4=1) **or** (T5=T2=λ) **else** 0
ic-src	T3 **if** (T5=λ) **else** 0
jmp-typ	T2 **if** (T5=λ) **else** 0
jmp-con	T1 **always**
reg-act	1 **if** (T5=1 **and** T3=λ) **or** (T5=0 **and** T4>λ) **else** 0
reg-src	λ **if** (T5=1 **and** T4=0) **else** 0 **if** (T5=1 **and** T4=λ) **else** 1
reg-adr	T0 **always**
reg-ch1	T1 **always**
reg-ch2	T0 **always**
ram-act	1 **if** (T5>λ **and** reg-act=0) **else** 0
ram-src	λ **if** (T5=1 **and** T4=λ) **else** 0 **if** (T5=T4=1) **else** 1
ram-adr	1 **always**
alu-act	T3 **if** (T5=0) **else** 0
alu-op1	λ **if** (T5=0 **and** T4>λ) **else** 0
alu-op2	T0 **if** (T5=λ) **else** 1 **if** (T5=0 **and** T4=1) **else** 0
op2-mod	T2 **if** (T5=0) **else** 1 **if** (T5=λ **and** T4≠0) **else** 0