etherforth revival

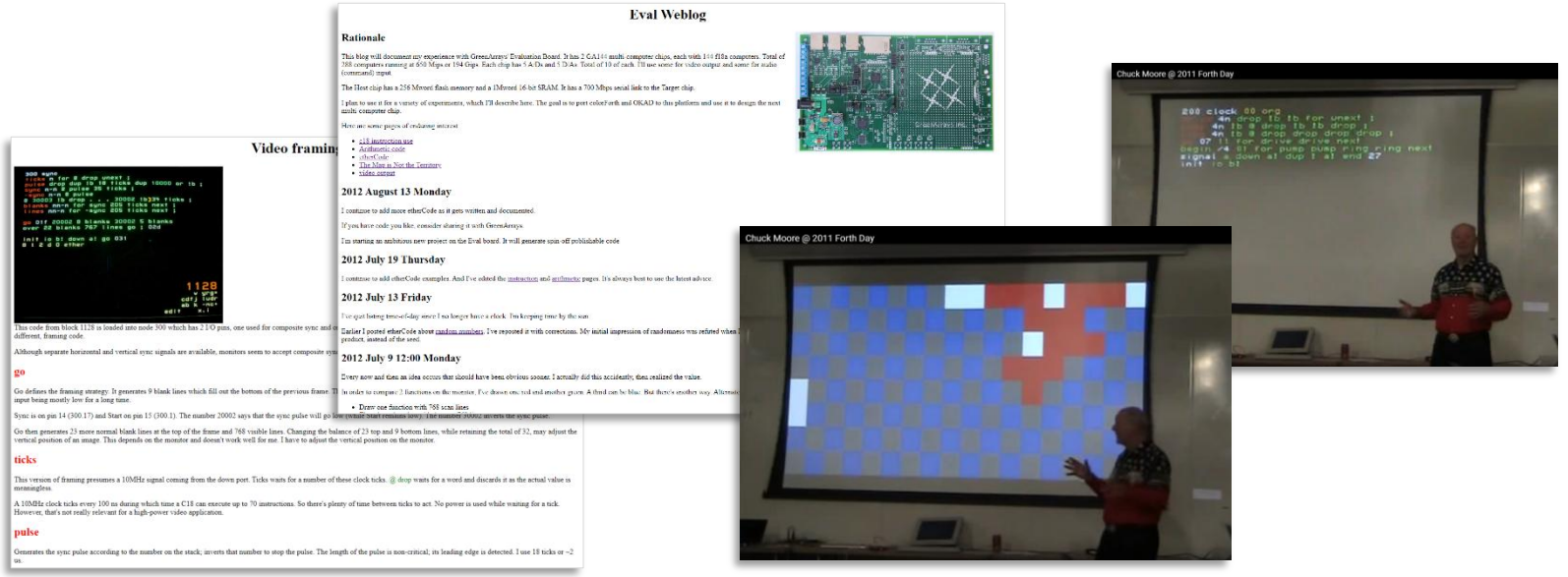
Programming GA144 using GA144 only

Daniel Kalny
on behalf of GreenArrays

Forth Day 2020

historical overview

2010 development started by Chuck Moore
2012 ether, sram, video, prototype compiler
2017 first experiments with Chuck's code
2019 usb keyboard
2020 etherforth - full development system



what is it?

etherforth is

- variant of colorForth, resident in GA144
- exploiting ether – code enabling nodes to communicate
- using pre-parsed words identified by colored tags
- standalone development system

etherforth is good for

- experimental work, exploring GA chips
- educational, hobby & demo apps

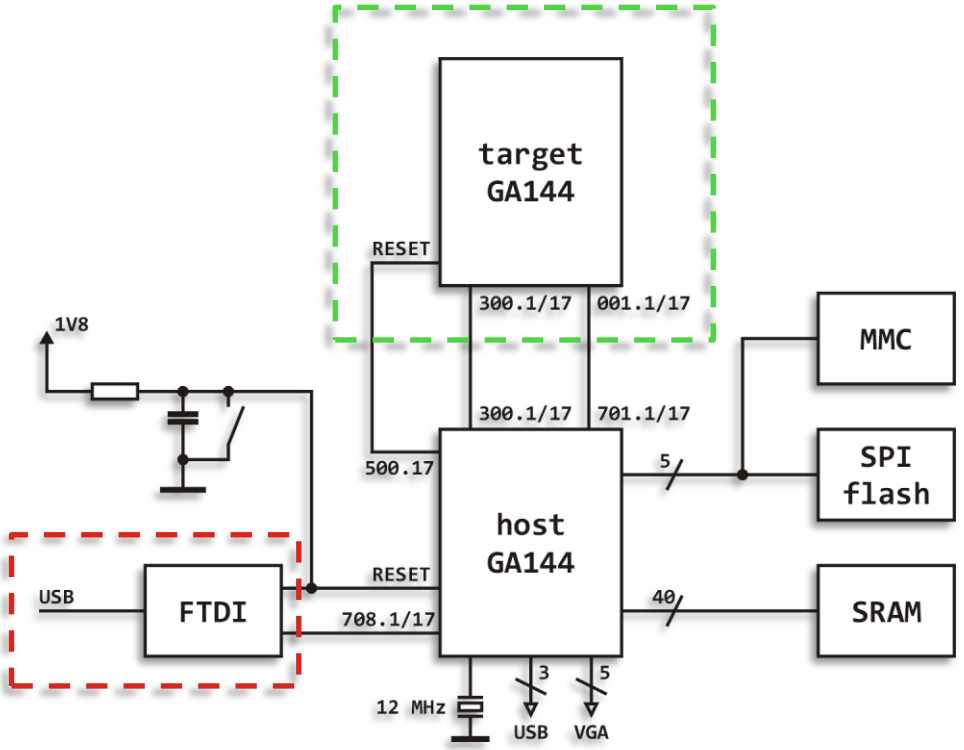
etherforth is not

- professional development tool

hardware

what etherforth needs to run?

- evaluation board EVB001 or similar
- 12 MHz clock source (ceramic resonator)
- vga display (640 x 480)
- keyboard (usb)
- dual voltage mmc
- spi flash (standalone only)



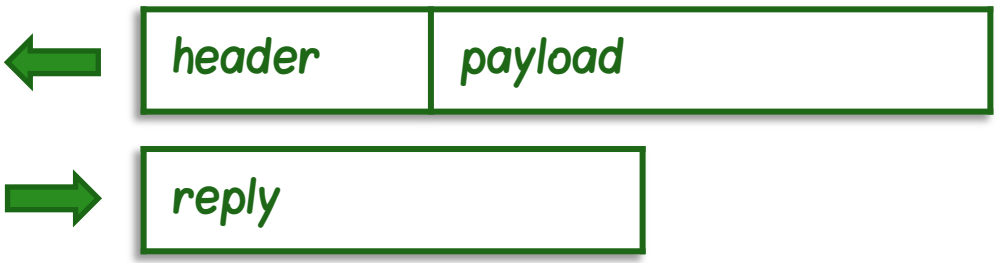
ether

software for routing messages between nodes

- code loaded into all nodes, multiport execute
- enables bootloading (ether overwritten with new code)
- facilitates internode communication

ether message

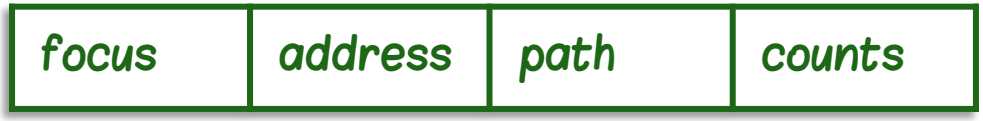
- header (4 words)
- payload (1 – 256 words)
- optional reply (0 – 511 words)



ether

message header

- focus (port call)
- address (ether entry point)
- path (describes route to follow)
- counts (payload + reply)

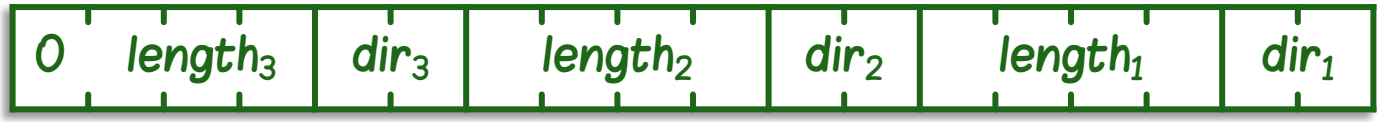


ether

path

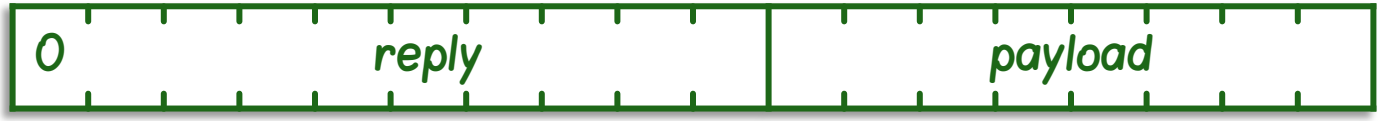
- three segments
- length (number of nodes to travel)
- direction (physical, not port names)

dir	direction
0	right
1	left
2	up
3	down



counts

- payload
- reply



pre-parsed words

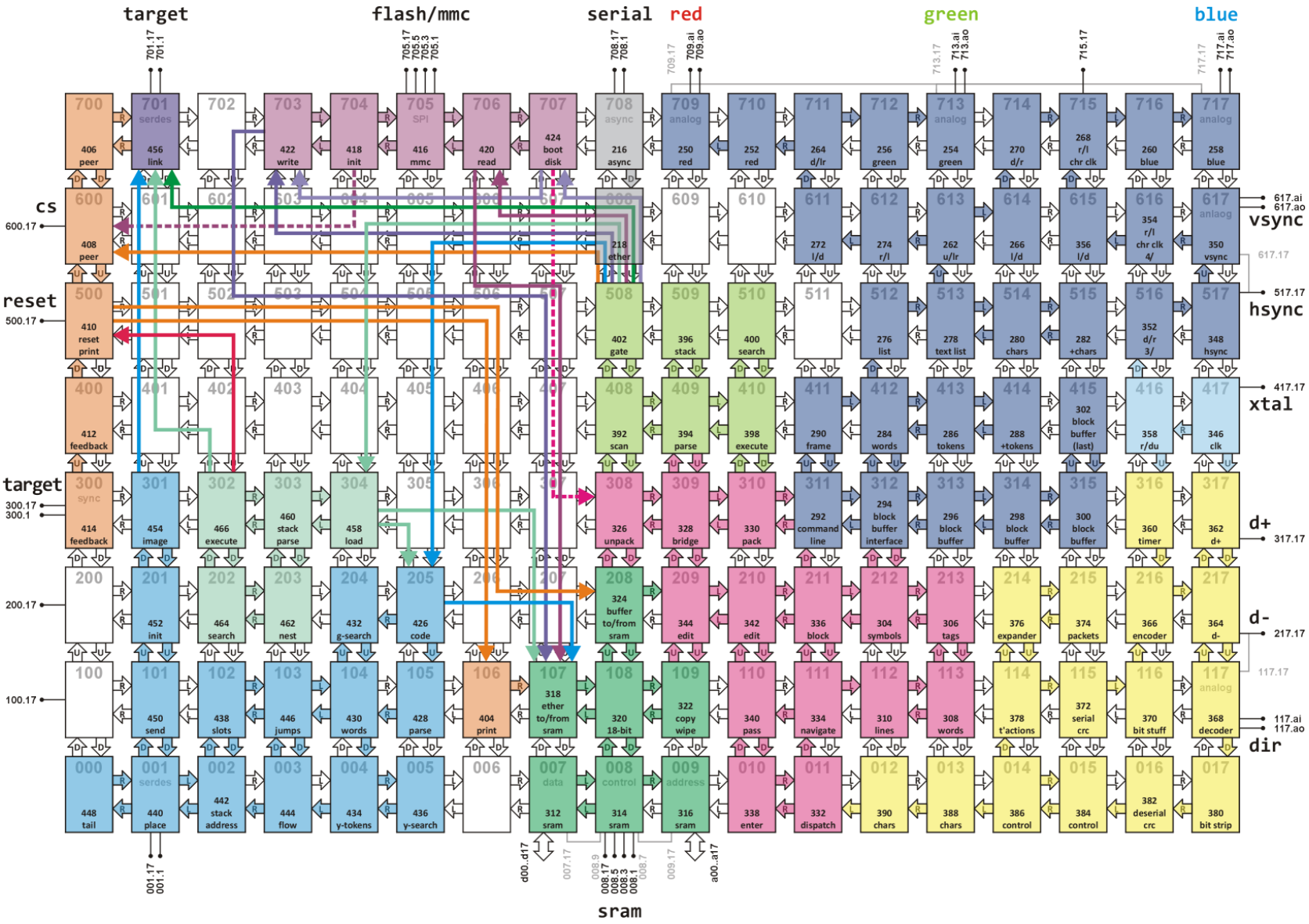
6-bit symbols = tags, characters, tokens

3 symbols packed in one 18-bit word

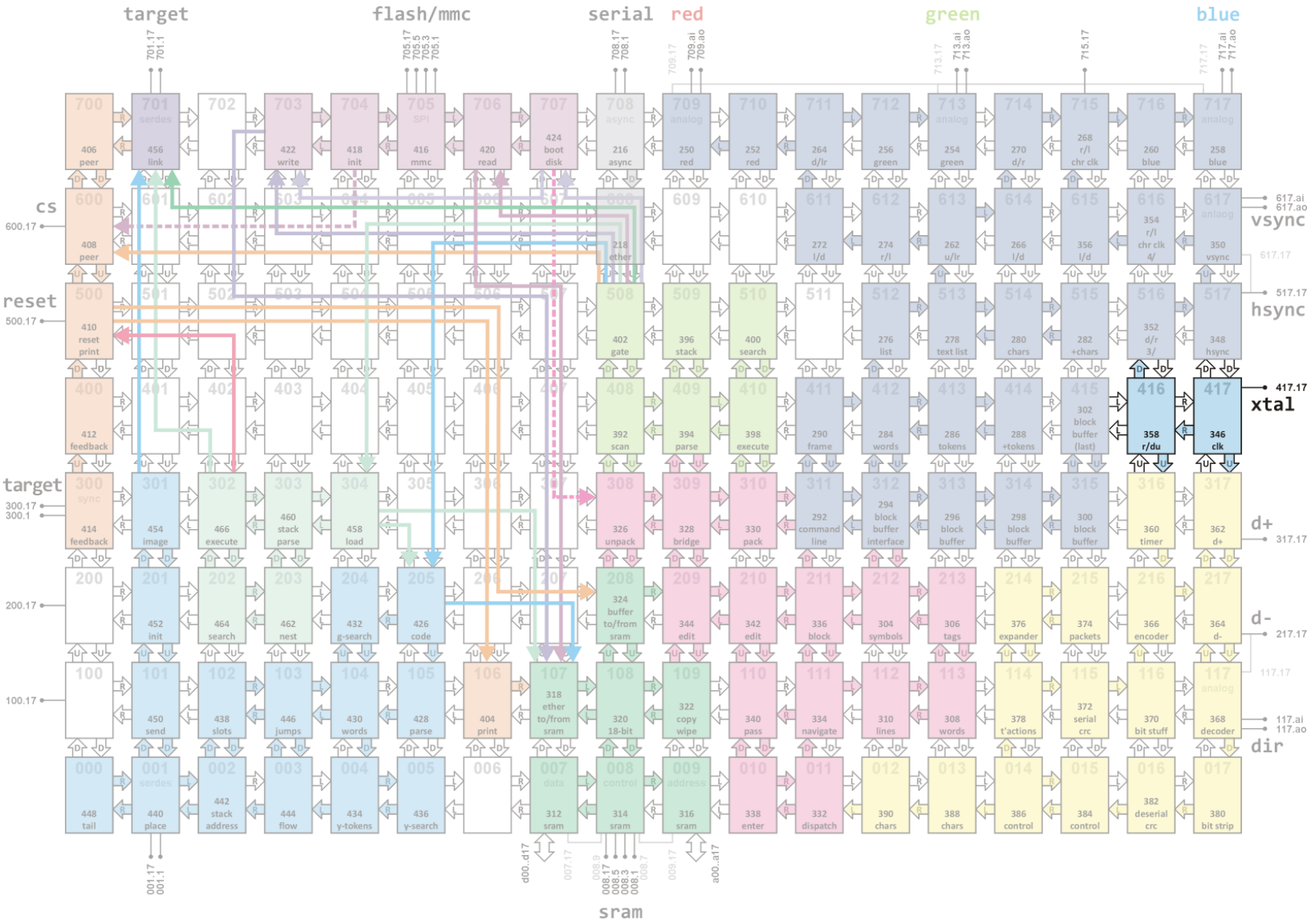
hex	tag	color
30	token	
31	token	
32	char	
33	hex	
34	hex	
35	char	
36	decimal	
37	char	
38	decimal	
39	eol	
3A	space	
3B	cursor	
3C	eob	
3D	char	
3E	char	
3F	char	

hex	char	token	hex	char	token	hex	char	token
00	0	;	10	g	+*	20	w	then
01	1	ex	11	h	2*	21	x	else
02	2	begin	12	i	2/	22	y	ahead
03	3	end	13	j	-	23	z	leap
04	4	unext	14	k	+	24	*	when
05	5	next	15	l	and	25	/	zif
06	6	if	16	m	or	26	@	till
07	7	-if	17	n	drop	27	!	-till
08	8	@p	18	o	dup	28	.	for
09	9	@+	19	p	pop	29	,	-when
0A	a	@b	1A	q	over	2A	;	left
0B	b	@	1B	r	a	2B	'	right
0C	c	!p	1C	s	.	2C	#	io
0D	d	!+	1D	t	push	2D	-	down
0E	e	!b	1E	u	b!	2E	?	up
0F	f	!	1F	v	a!	2F	+	data

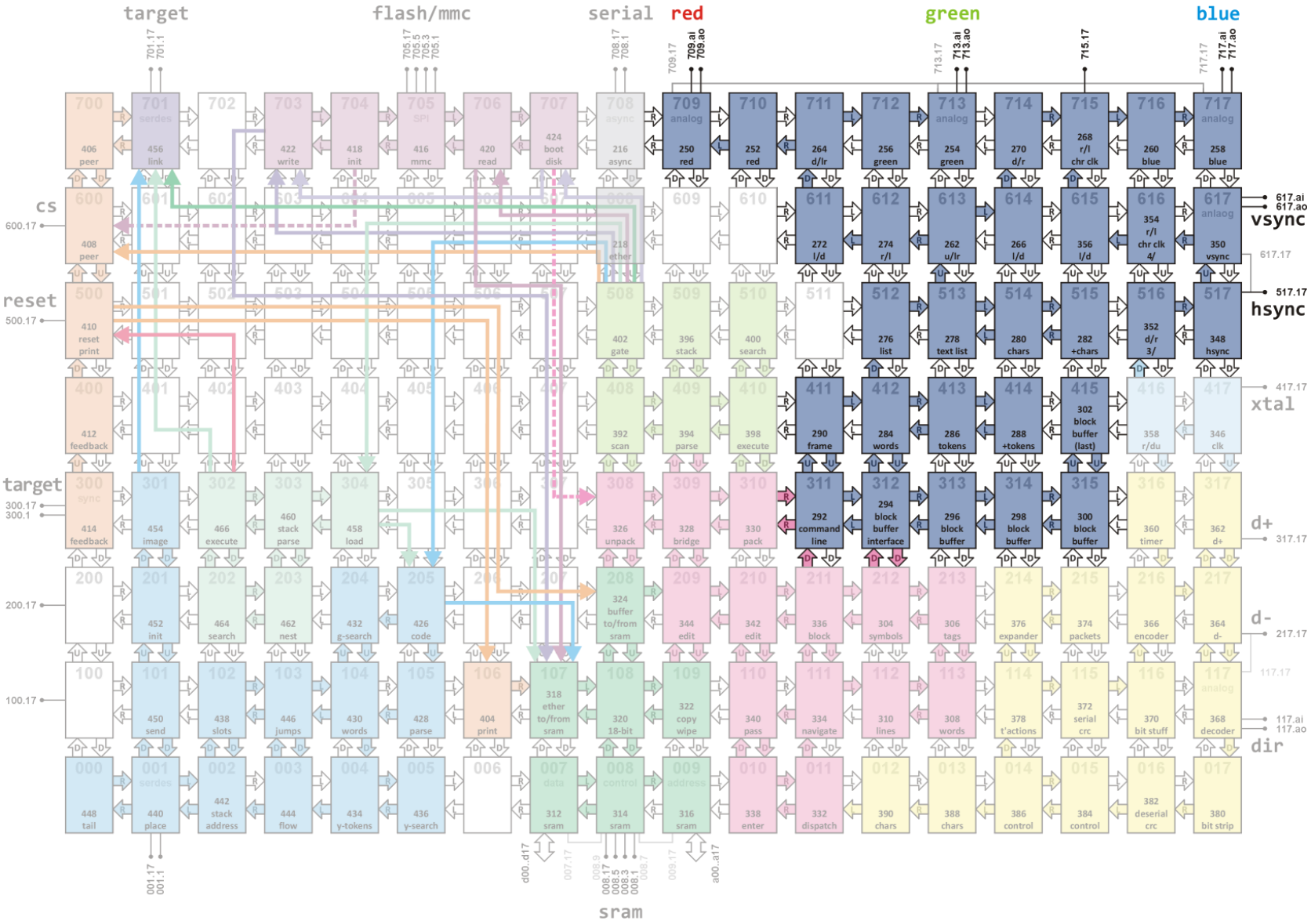
floorplan


target
flash/mmc
serial
red
green
blue
cs
reset
target
sram
vsync
hsync
xtal
d+
d-
dir
700 406 peer
701 serdes 456 link
702
703 422 write
704 418 init
705 SPI 416 mmc
706 420 read
707 424 boot disk
708 async 216 async
709 analog 250 red
710 252 red
711 264 d/lr
712 256 green
713 analog 254 green
714 270 d/r
715 268 r/l chr clk
716 260 blue
717 analog 258 blue
600 408 peer
601
602
603
604
605
606
607
608 218 ether
609
610
611 272 l/d
612 274 r/l
613 262 u/lr
614 266 l/d
615 356 l/d
616 354 r/l chr clk 4/
617 anlaog 350 vsync
500 410 reset print
501
502
503
504
505
506
507
508 402 gate
509 396 stack
510 400 search
511
512 276 list
513 278 text list
514 280 chars
515 282 +chars
516 352 d/r 3/
517 348 hsync
400 412 feedback
401
402
403
404
405
406
407
408 392 scan
409 394 parse
410 398 execute
411 290 frame
412 284 words
413 286 tokens
414 288 +tokens
415 302 block buffer (last)
416 358 r/du
417 346 clk
300 sync 414 feedback
301 454 image
302 466 execute
303 460 stack parse
304 458 load
305
306
307
308 326 unpack
309 328 bridge
310 330 pack
311 292 command line
312 294 block buffer interface
313 296 block buffer
314 298 block buffer
315 300 block buffer
316 360 timer
317 362 d+
200
201 452 init
202 464 search
203 462 nest
204 432 g-search
205 426 code
206
207
208 324 buffer to/from sram
209 344 edit
210 342 edit
211 336 block
212 304 symbols
213 306 tags
214 376 expander
215 374 packets
216 366 encoder
217 364 d-
100
101 450 send
102 438 slots
103 446 jumps
104 430 words
105 428 parse
106 404 print
107 318 ether to/from sram
108 320 18-bit
109 322 copy wipe
110 340 pass
111 334 navigate
112 310 lines
113 308 words
114 378 t'actions
115 372 serial crc
116 370 bit stuff
117 analog 368 decoder
000 448 tail
001 serdes 440 place
002 442 stack address
003 444 flow
004 434 y-tokens
005 436 y-search
006
007 data 312 sram
008 control 314 sram
009 address 316 sram
010 338 enter
011 332 dispatch
012 390 chars
013 388 chars
014 386 control
015 384 control
016 382 deserial crc
017 380 bit strip
600.17
500.17
300.17
300.1
200.17
100.17
617.ai
617.ao
617.17
517.17
417.17
317.17
217.17
117.17
117.ai
117.ao
701.17
701.1
705.17
705.5
705.3
705.1
708.17
708.1
709.17
709.ai
709.ao
713.17
713.ai
713.ao
715.17
717.17
717.ai
717.ao
001.17
001.1
d00..d17
007.17
008.9
008.17
008.5
008.3
008.1
008.7
009.17
a00..a17

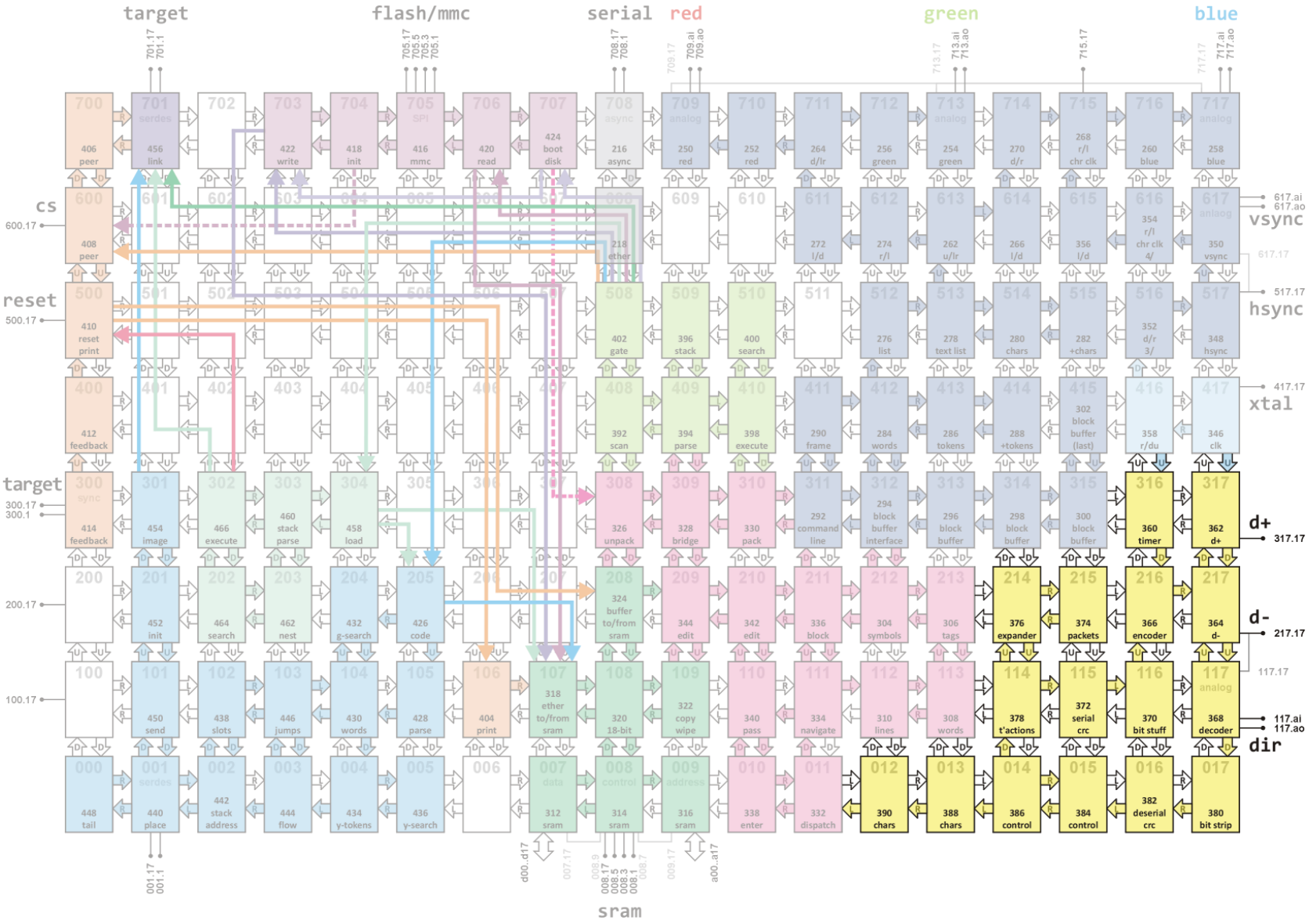
floorplan - clock



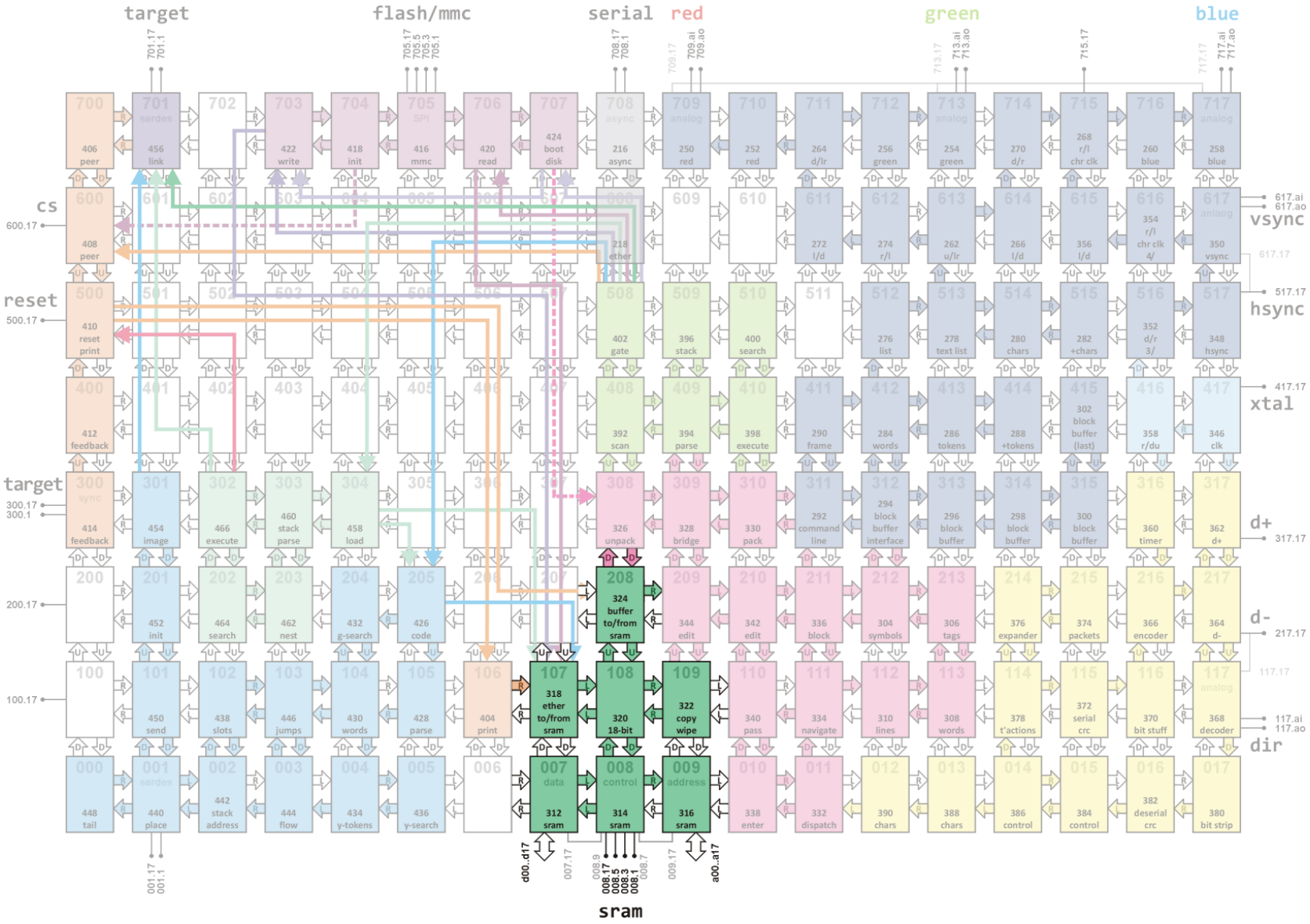
floorplan - video



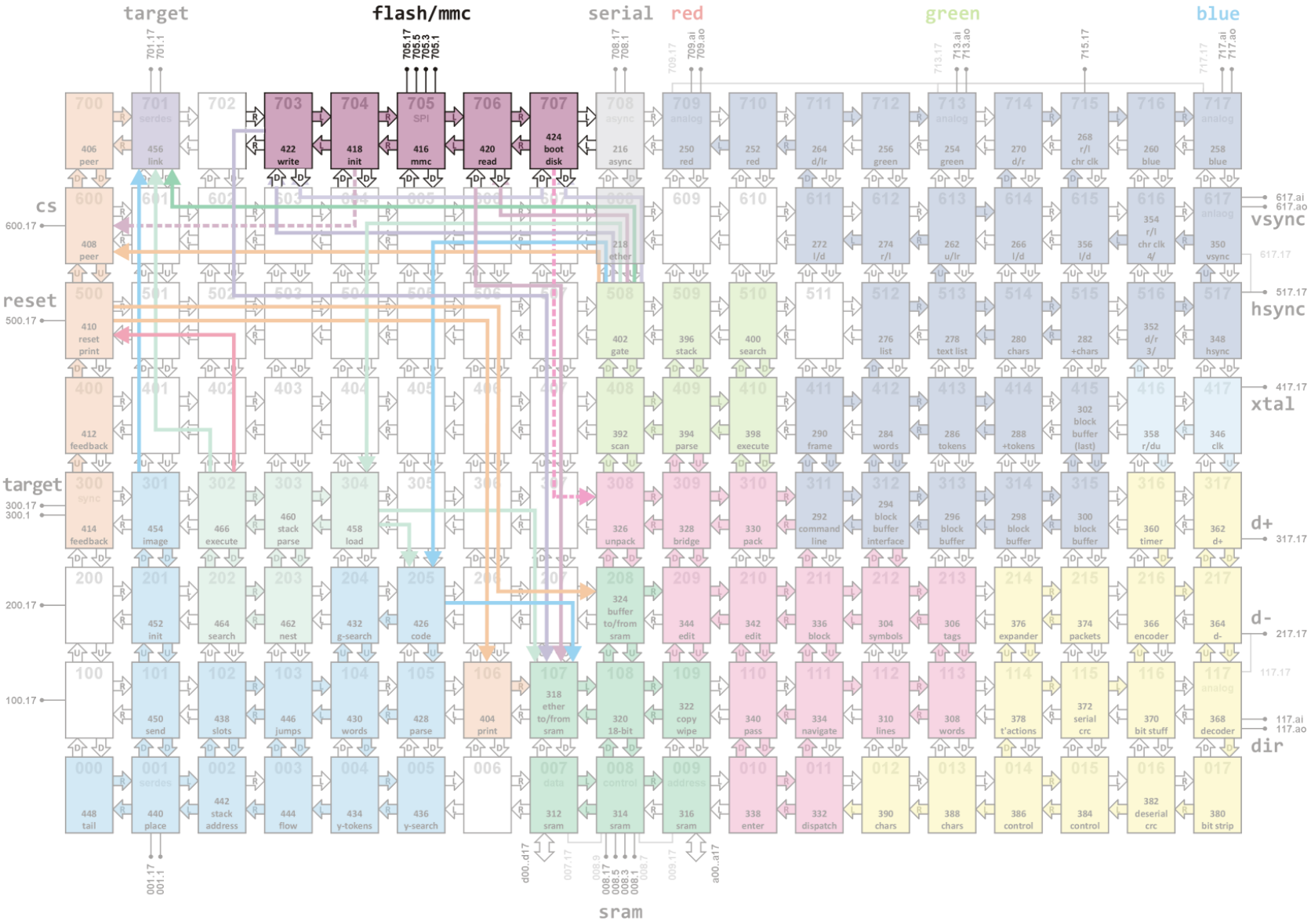
floorplan - keyboard



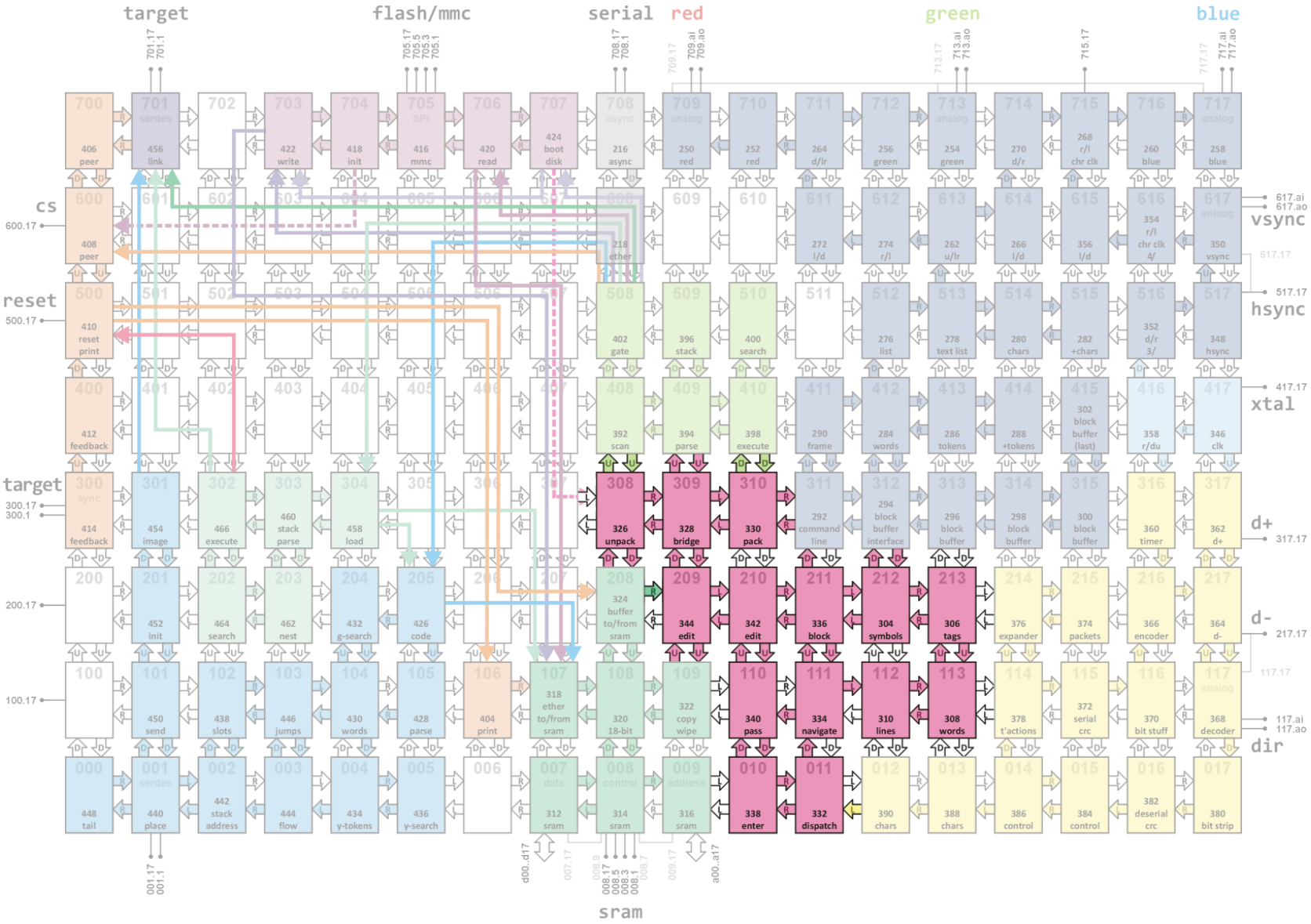
floorplan - sram



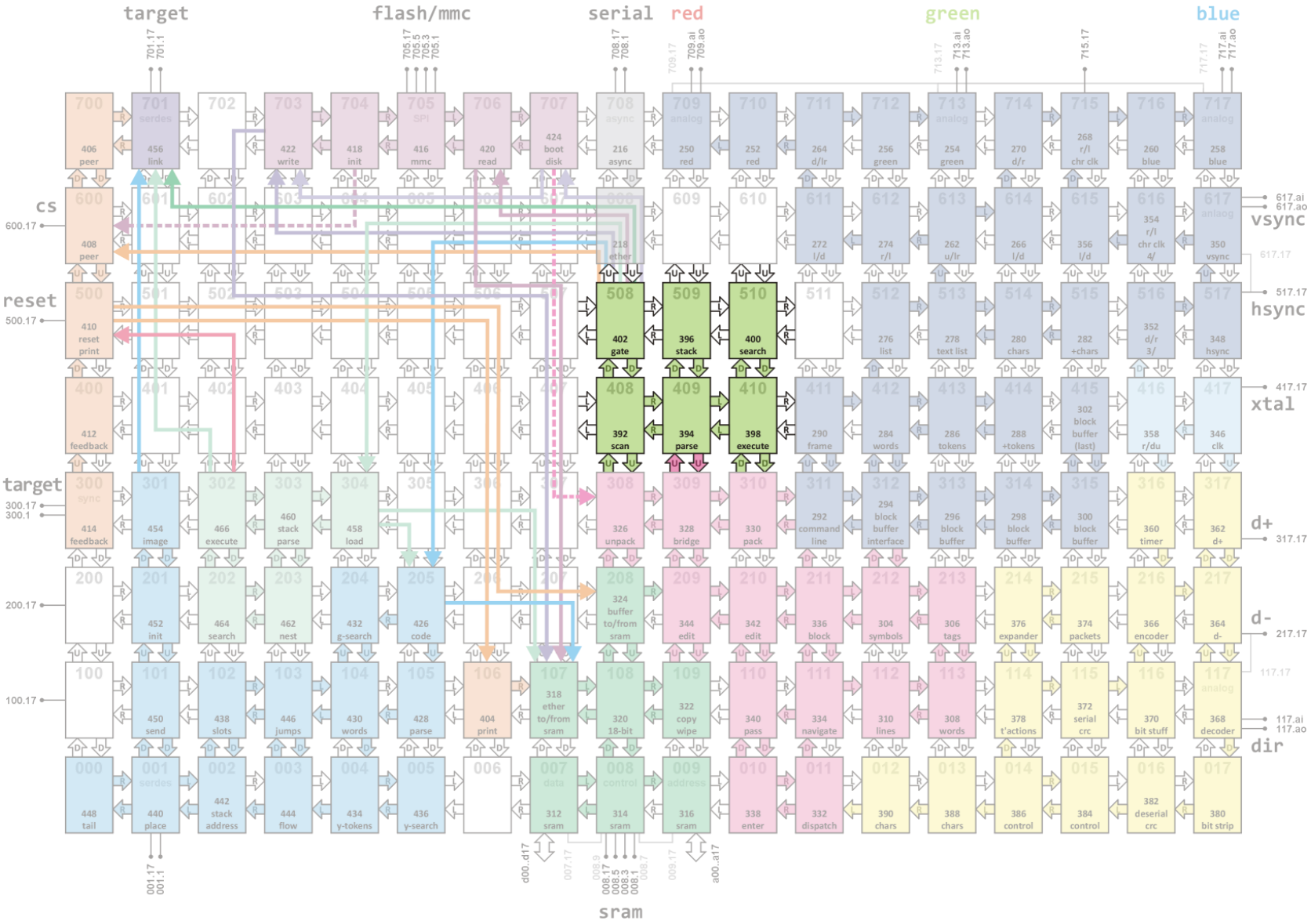
floorplan - mmc



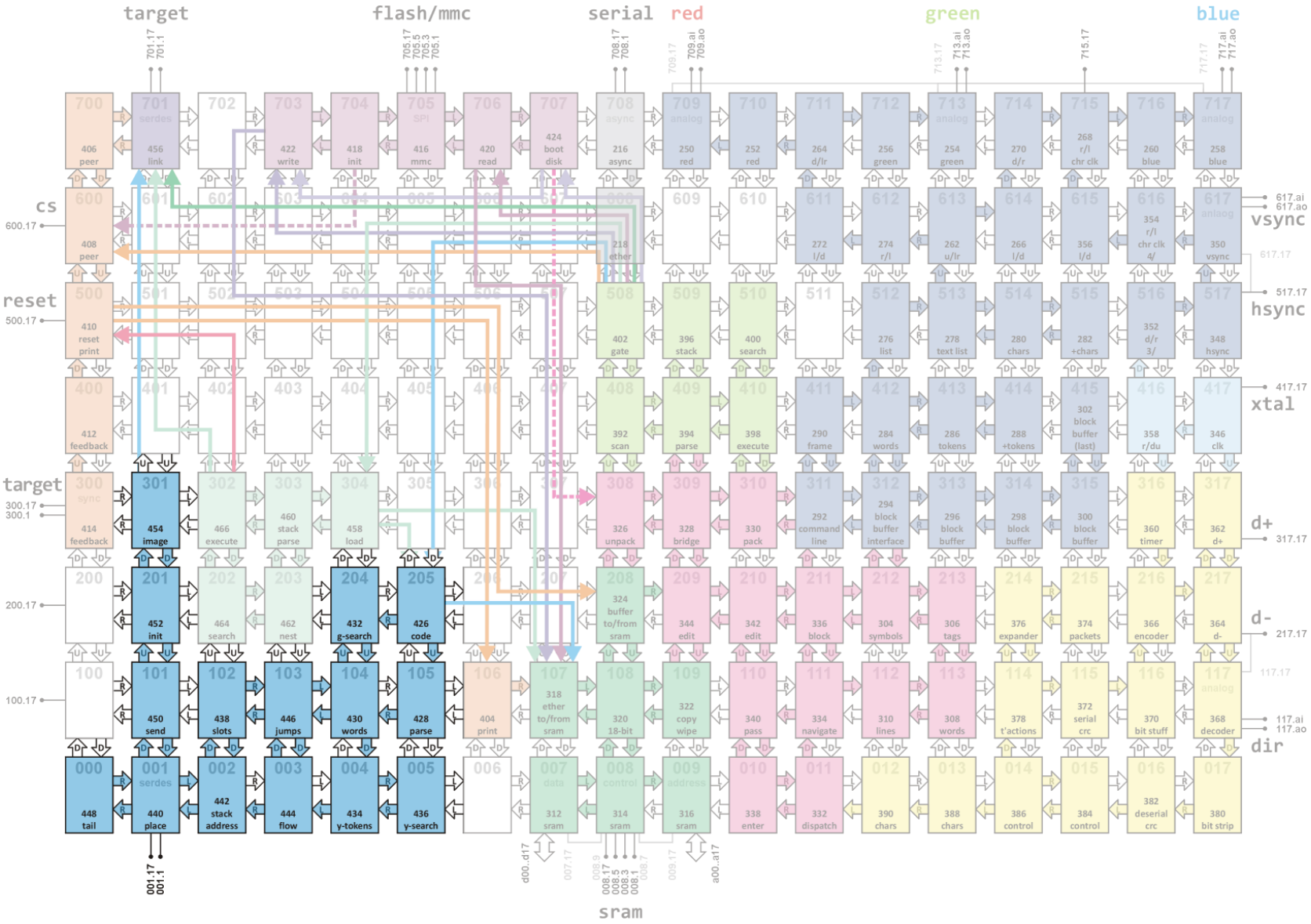
floorplan - editor



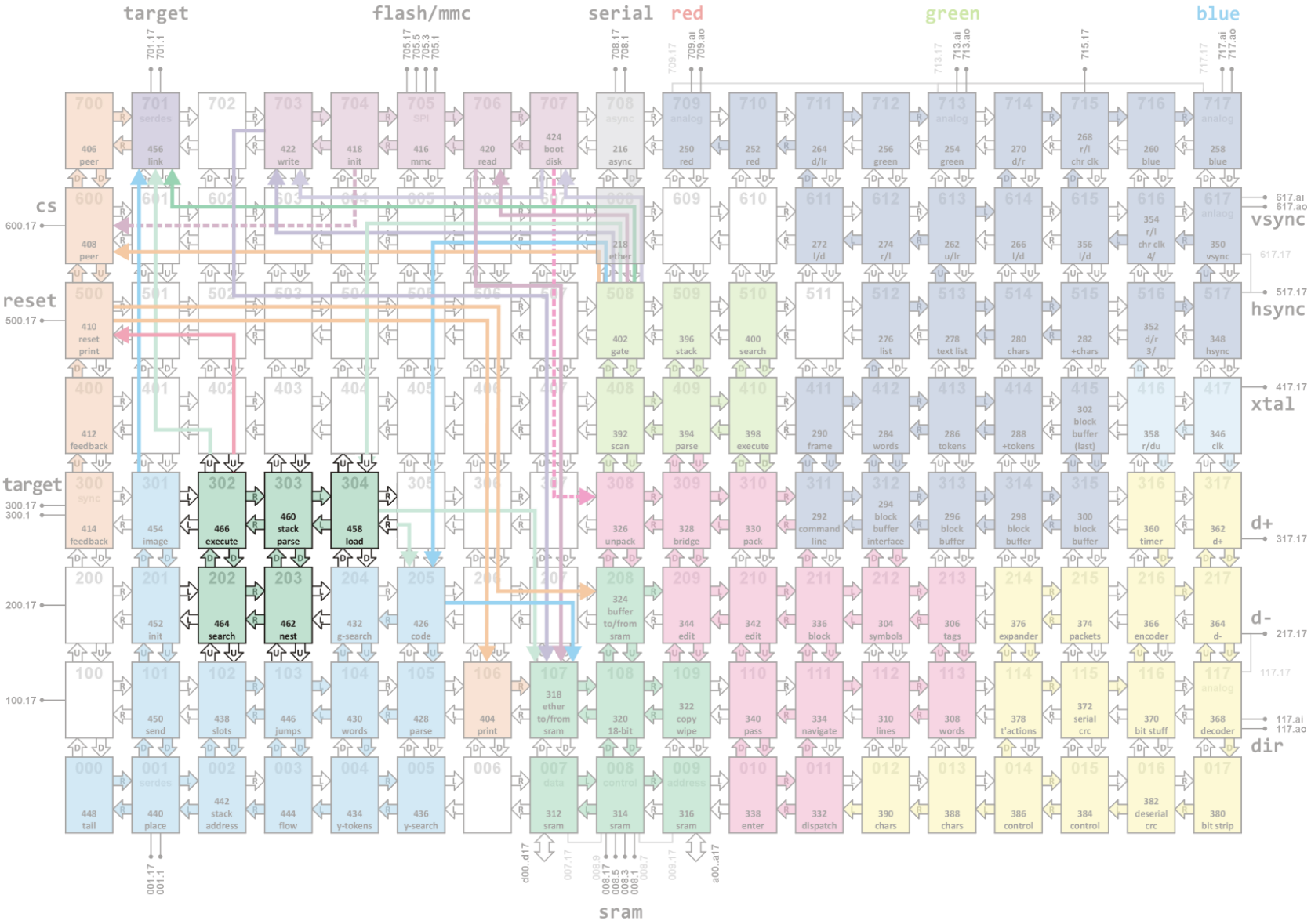
floorplan - interpreter



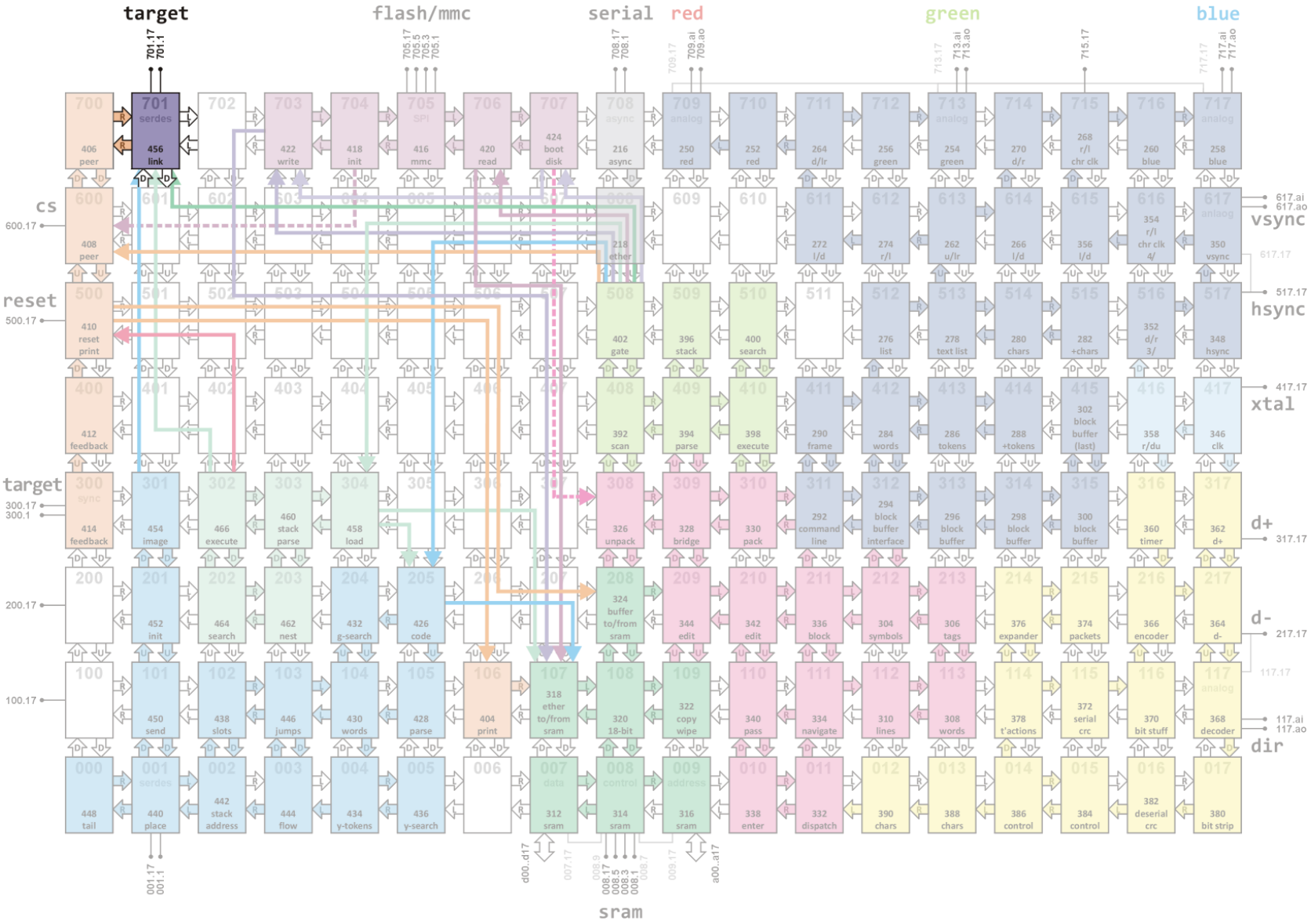
floorplan - compiler



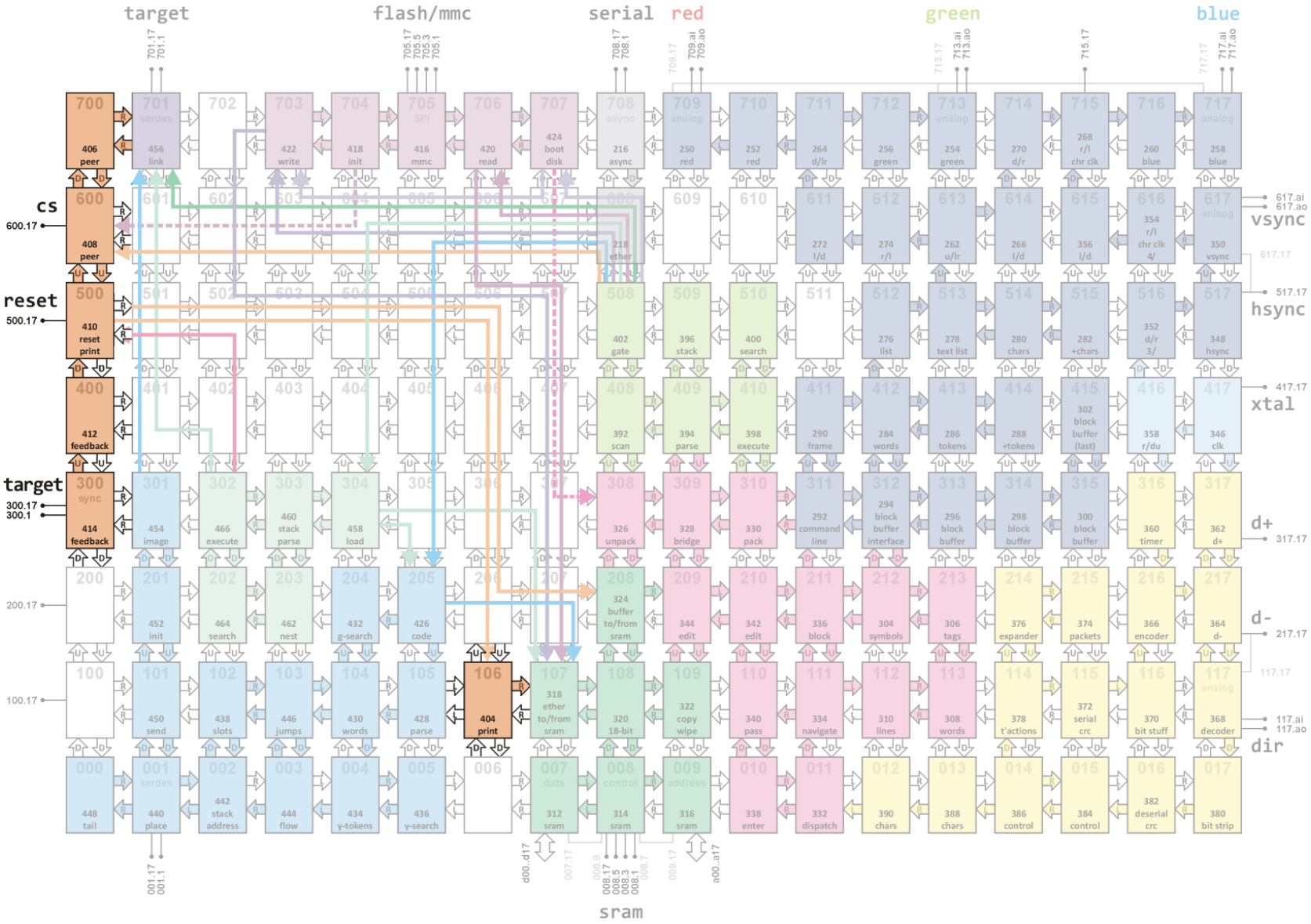
floorplan - loader



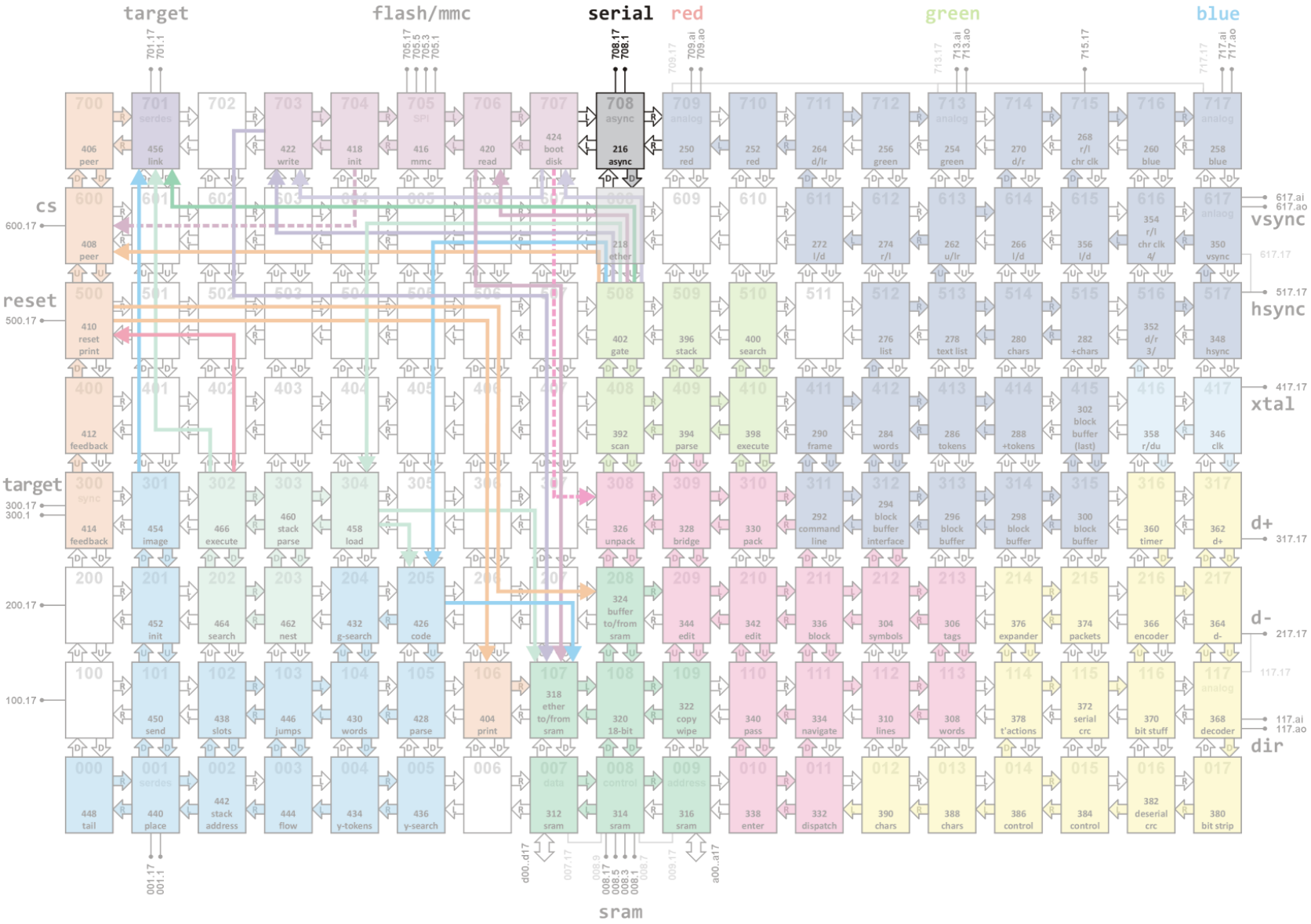
floorplan - link



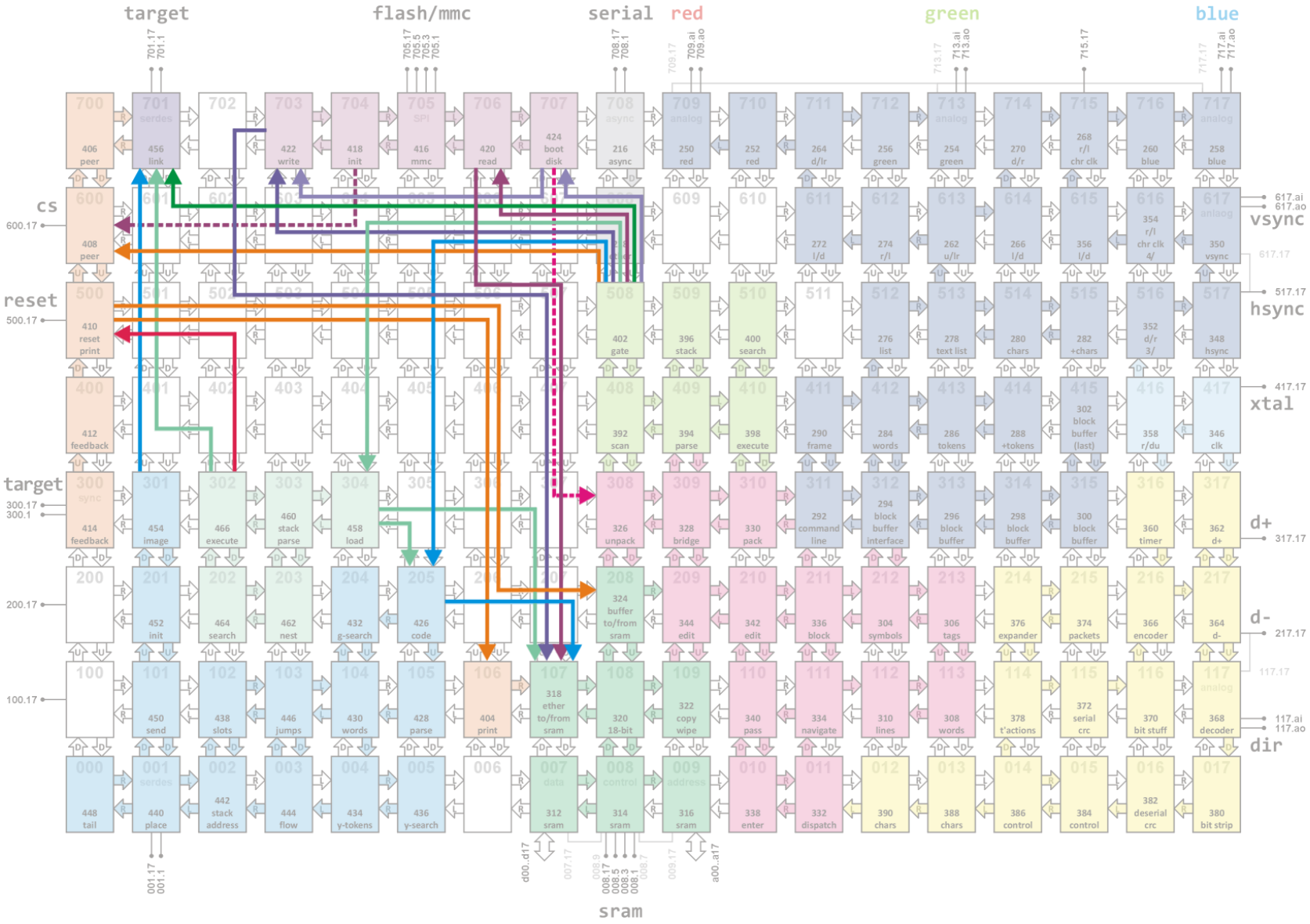
floorplan - utilities



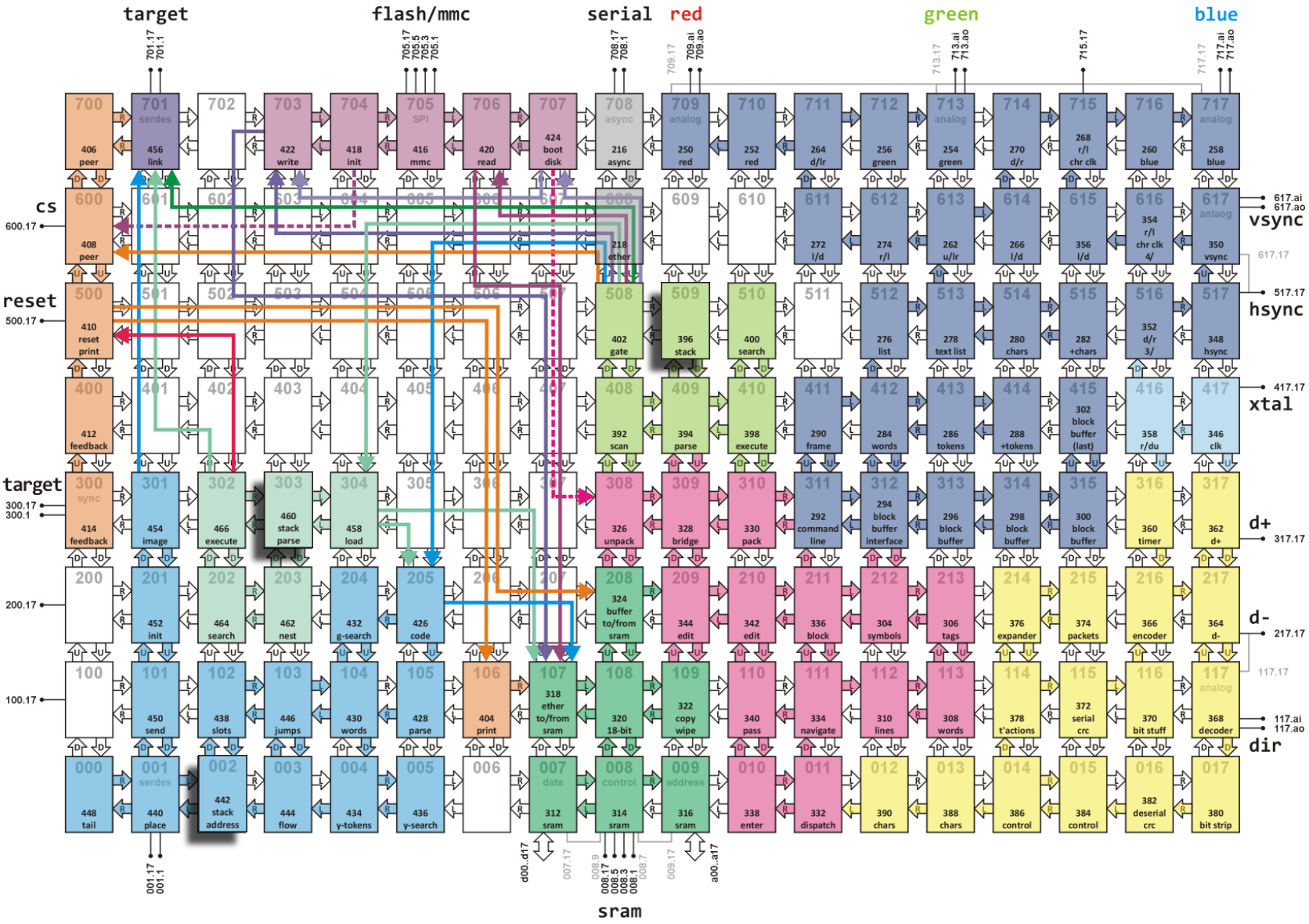
floorplan - async serial line



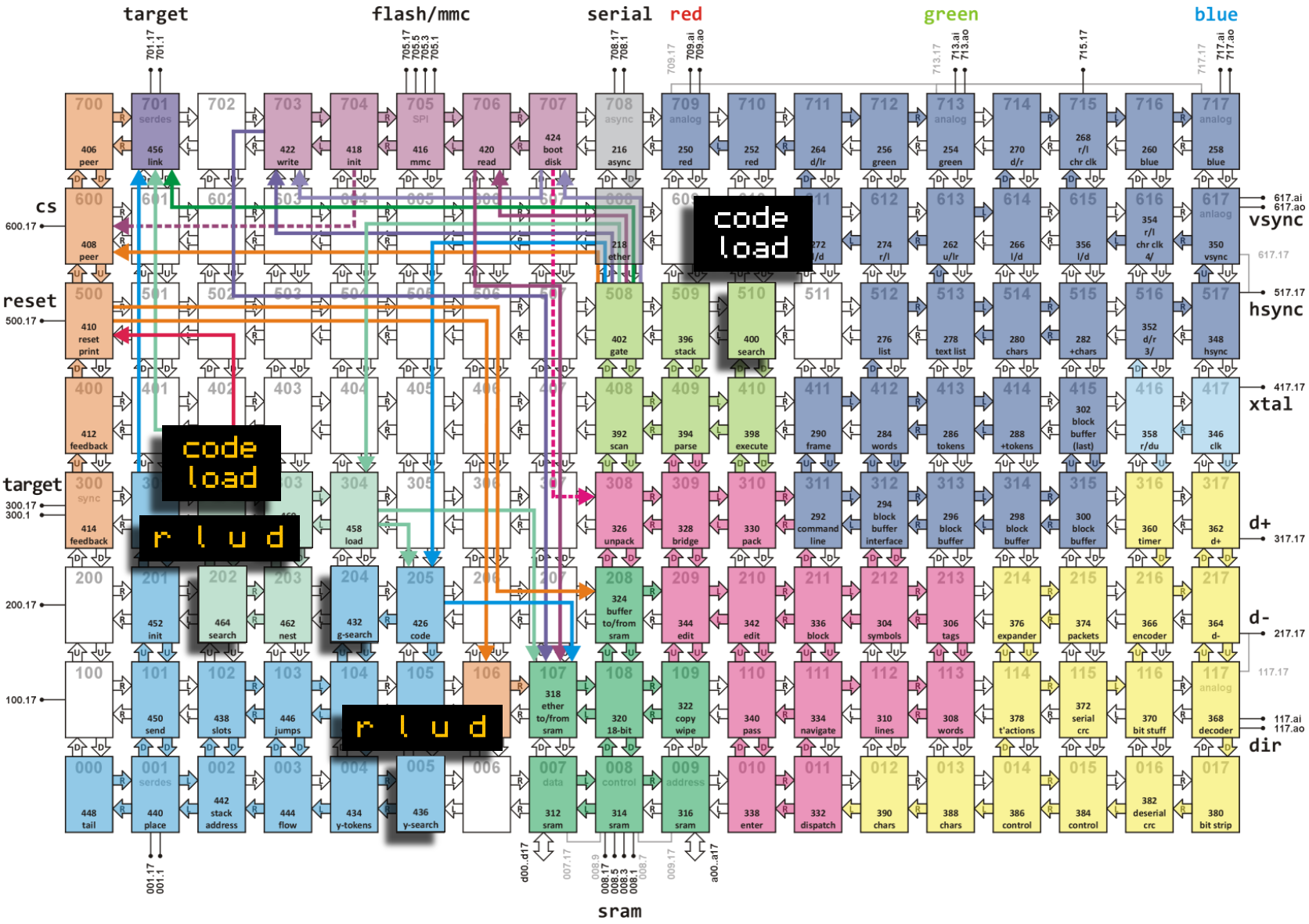
floorplan - ether paths



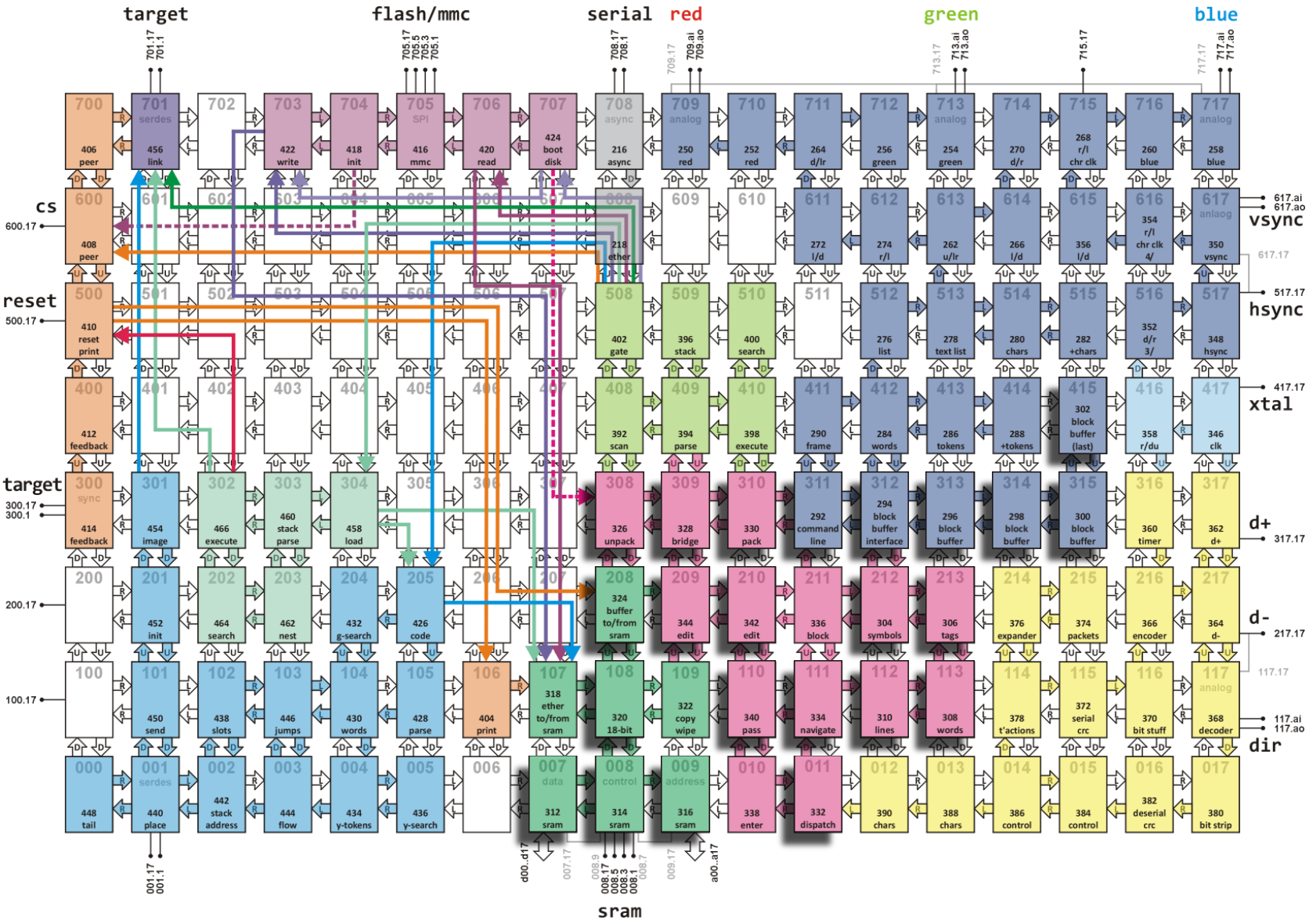
stack


target
flash/mmc
serial
red
green
blue
cs
reset
target
vsync
hsync
xtal
d+
d-
dir
sram
700 406 peer
701 serdes 456 link
702
703 422 write
704 418 init
705 SPI 416 mmc
706 420 read
707 424 boot disk
708 async 216 async
709 analog 250 red
710 252 red
711 264 d/lr
712 256 green
713 analog 254 green
714 270 d/r
715 268 r/l chr clk
716 260 blue
717 analog 258 blue
600 408 peer
601
602
603
604
605
606
607
608 218 ether
609
610
611 272 l/d
612 274 r/l
613 262 u/lr
614 266 l/d
615 356 l/d
616 354 r/l chr clk 4/
617 anlaog 350 vsync
500 410 reset print
501
502
503
504
505
506
507
508 402 gate
509 396 stack
510 400 search
511
512 276 list
513 278 text list
514 280 chars
515 282 +chars
516 352 d/r 3/
517 348 hsync
400 412 feedback
401
402
403
404
405
406
407
408 392 scan
409 394 parse
410 398 execute
411 290 frame
412 284 words
413 286 tokens
414 288 +tokens
415 302 block buffer (last)
416 358 r/du
417 346 clk
300 sync 414 feedback
301 454 image
302 466 execute
303 460 stack parse
304 458 load
305
306
307
308 326 unpack
309 328 bridge
310 330 pack
311 292 command line
312 294 block buffer interface
313 296 block buffer
314 298 block buffer
315 300 block buffer
316 360 timer
317 362 d+
200
201 452 init
202 464 search
203 462 nest
204 432 g-search
205 426 code
206
207
208 324 buffer to/from sram
209 344 edit
210 342 edit
211 336 block
212 304 symbols
213 306 tags
214 376 expander
215 374 packets
216 366 encoder
217 364 d-
100
101 450 send
102 438 slots
103 446 jumps
104 430 words
105 428 parse
106 404 print
107 318 ether to/from sram
108 320 18-bit
109 322 copy wipe
110 340 pass
111 334 navigate
112 310 lines
113 308 words
114 378 t'actions
115 372 serial crc
116 370 bit stuff
117 analog 368 decoder
000 448 tail
001 serdes 440 place
002 442 stack address
003 444 flow
004 434 y-tokens
005 436 y-search
006
007 data 312 sram
008 control 314 sram
009 address 316 sram
010 338 enter
011 332 dispatch
012 390 chars
013 388 chars
014 386 control
015 384 control
016 382 deserial crc
017 380 bit strip
600.17
500.17
300.17
300.1
200.17
100.17
617.ai
617.ao
617.17
517.17
417.17
317.17
217.17
117.17
117.ai
117.ao
701.17
701.1
705.17
705.5
705.3
705.1
708.17
708.1
709.17
709.ai
709.ao
713.17
713.ai
713.ao
715.17
717.17
717.ai
717.ao
001.17
001.1
d00..d17
007.17
008.9
008.17
008.5
008.3
008.1
008.7
009.17
a00..a17

dictionary


target
flash/mmc
serial
red
green
blue
cs
reset
target
vsync
hsync
xtal
d+
d-
dir
code load
r l u d
sram

deleting a word

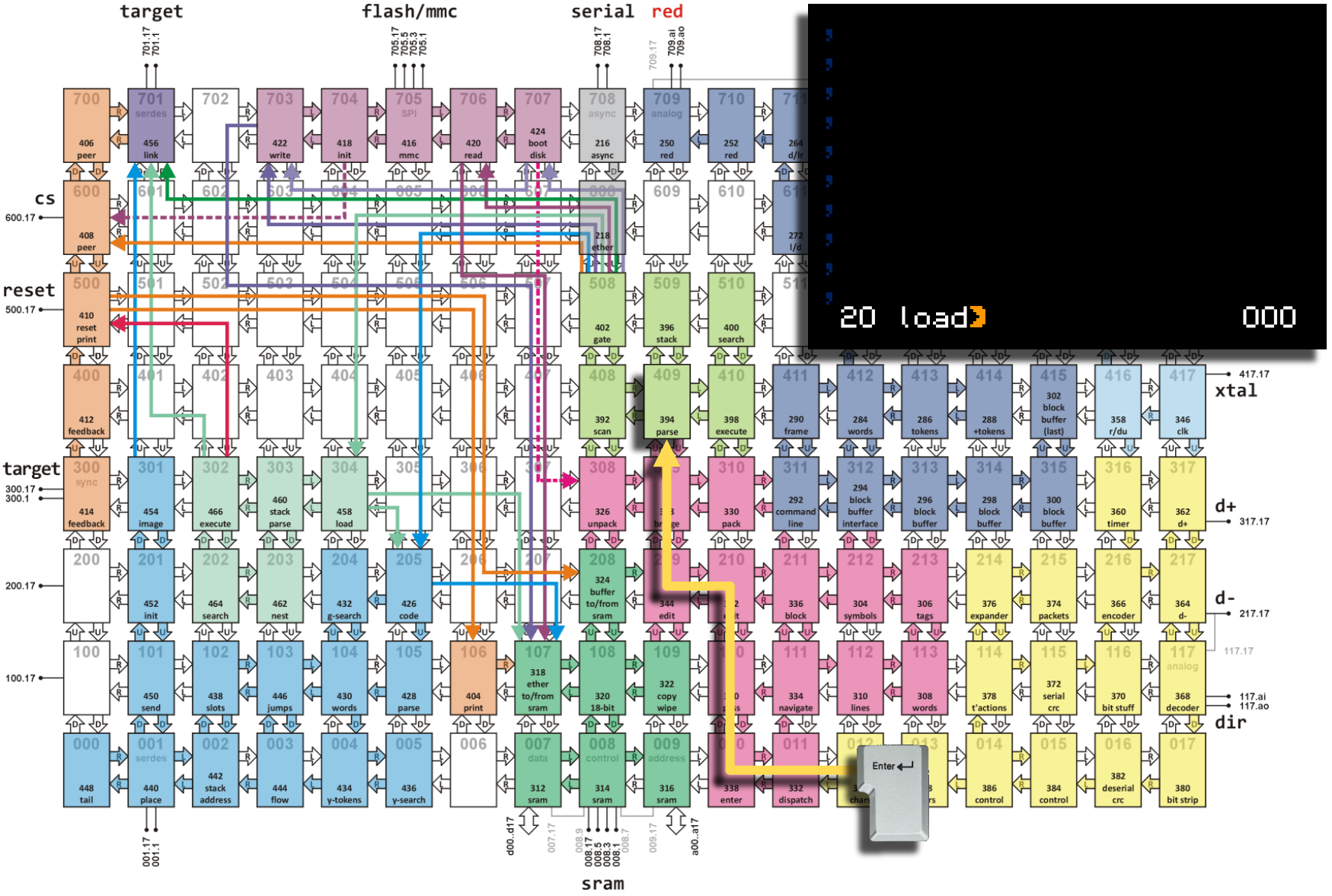


setup

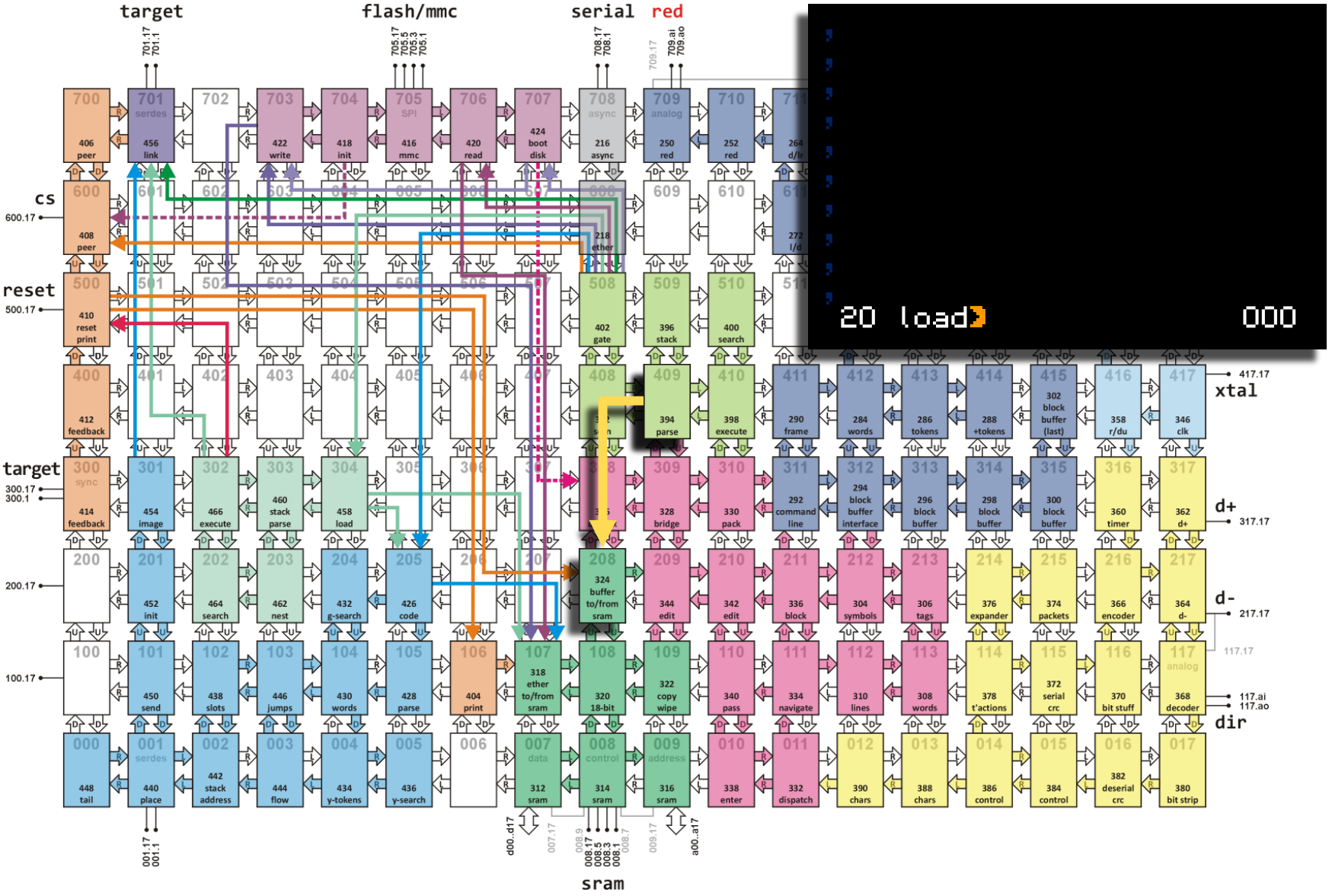
demo #1

let there be light

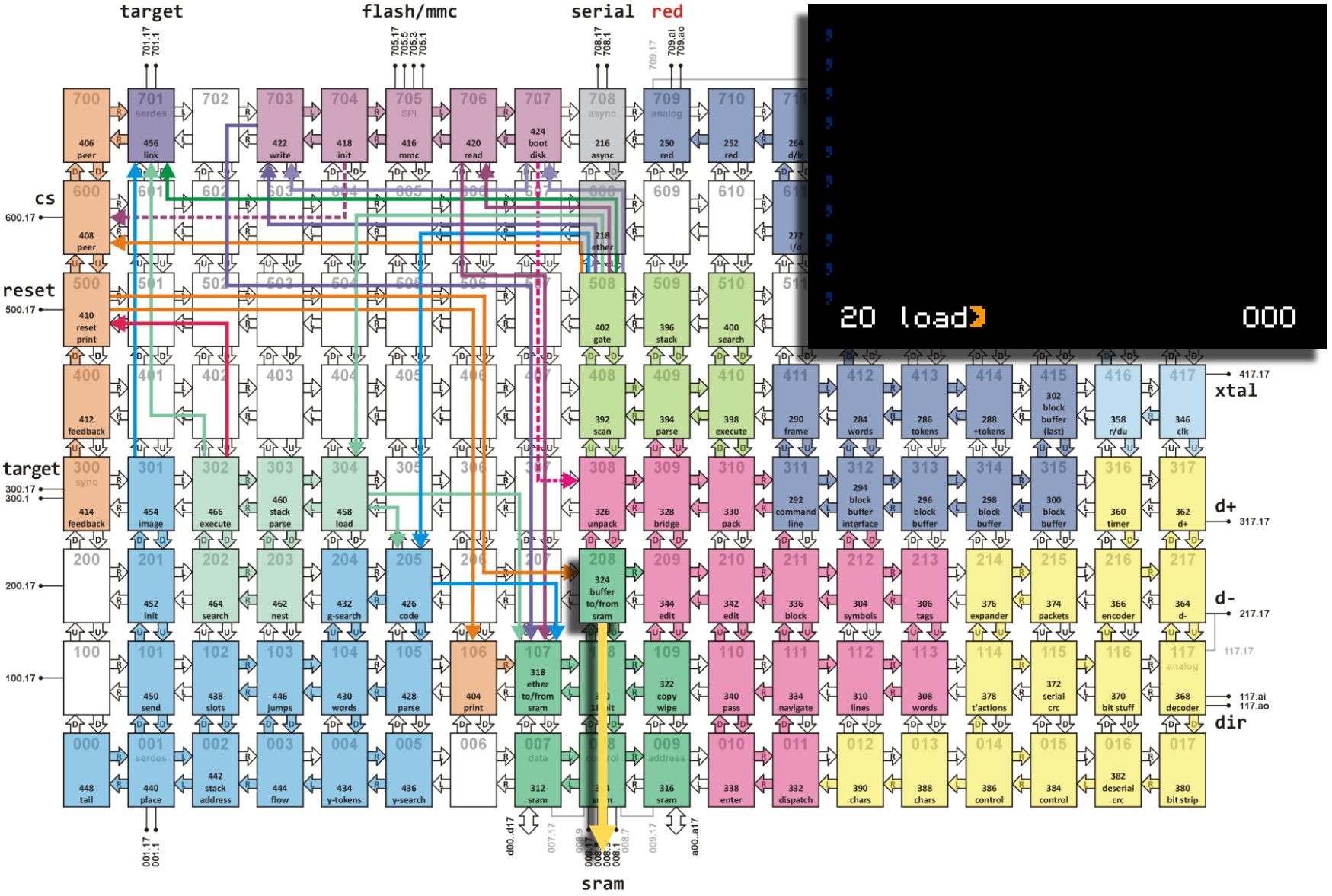
loading applications



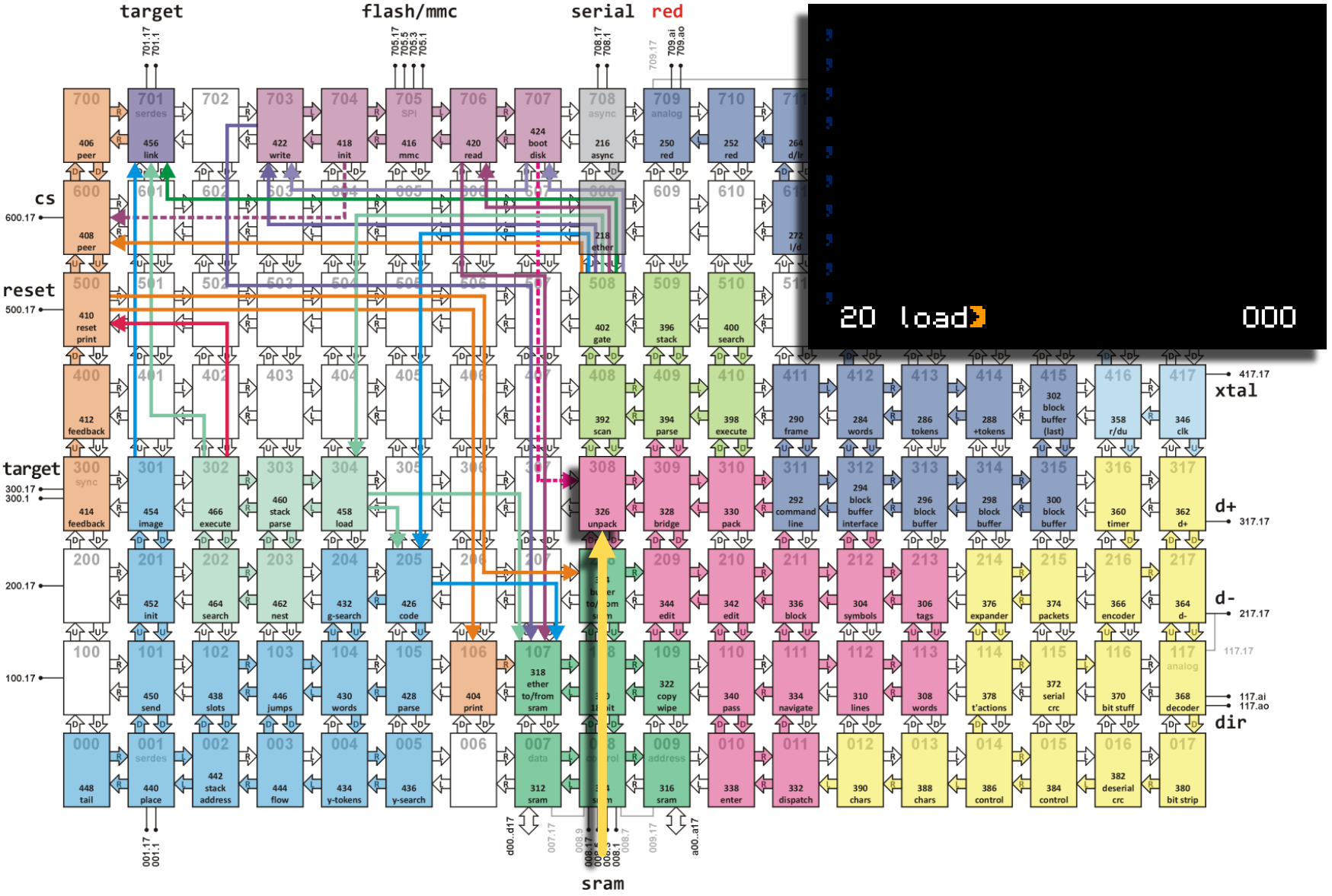
loading applications



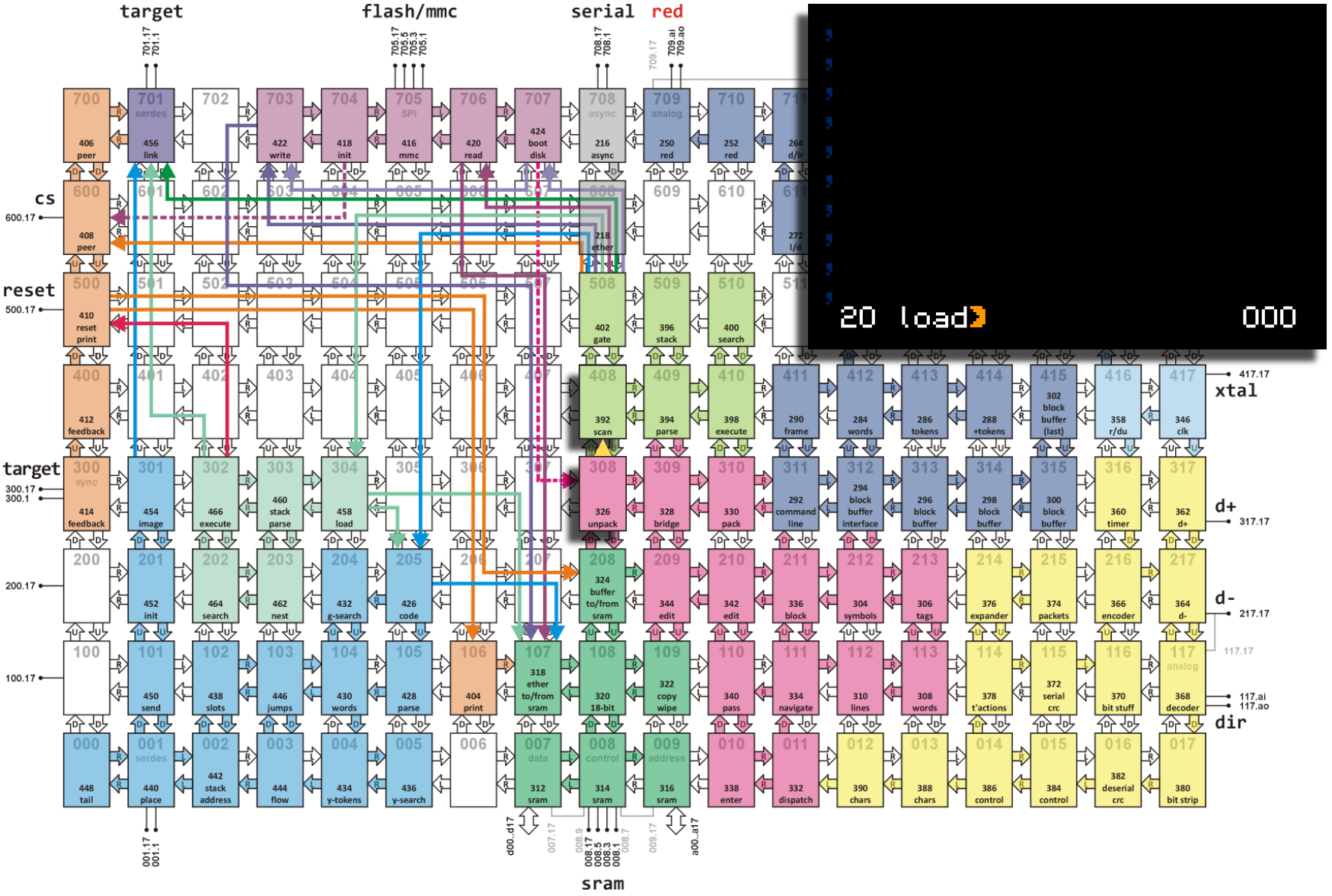
loading applications



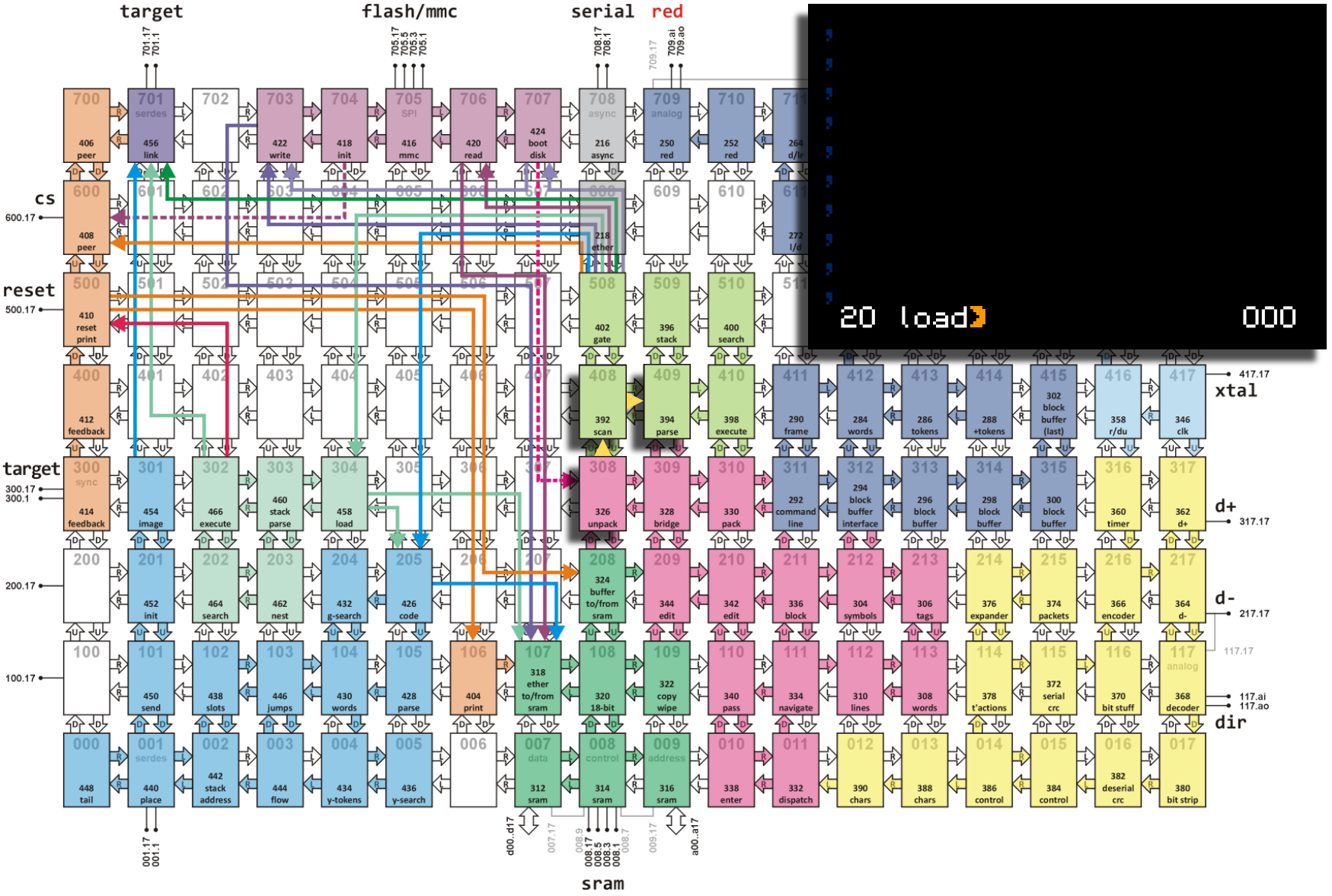
loading applications



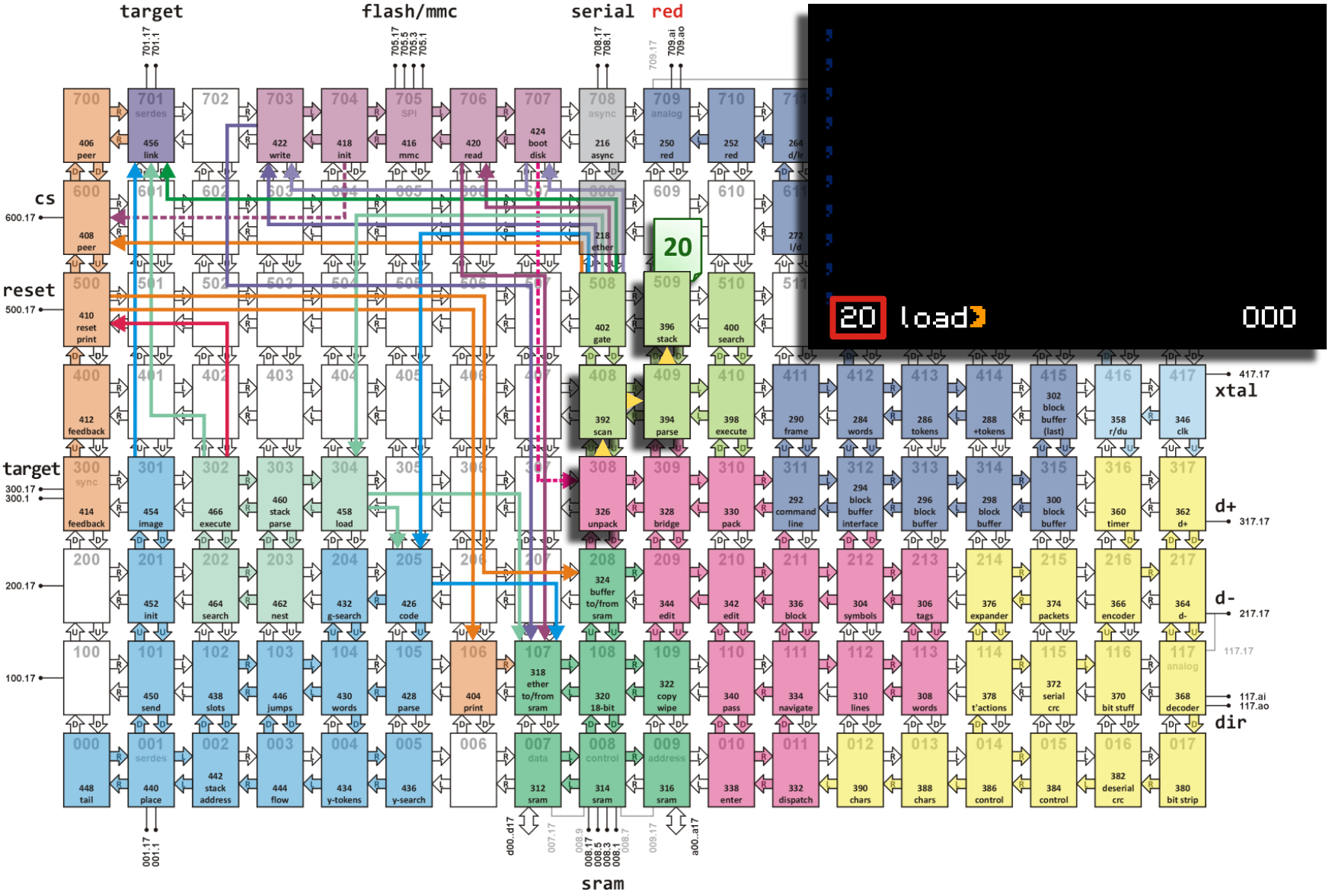
loading applications



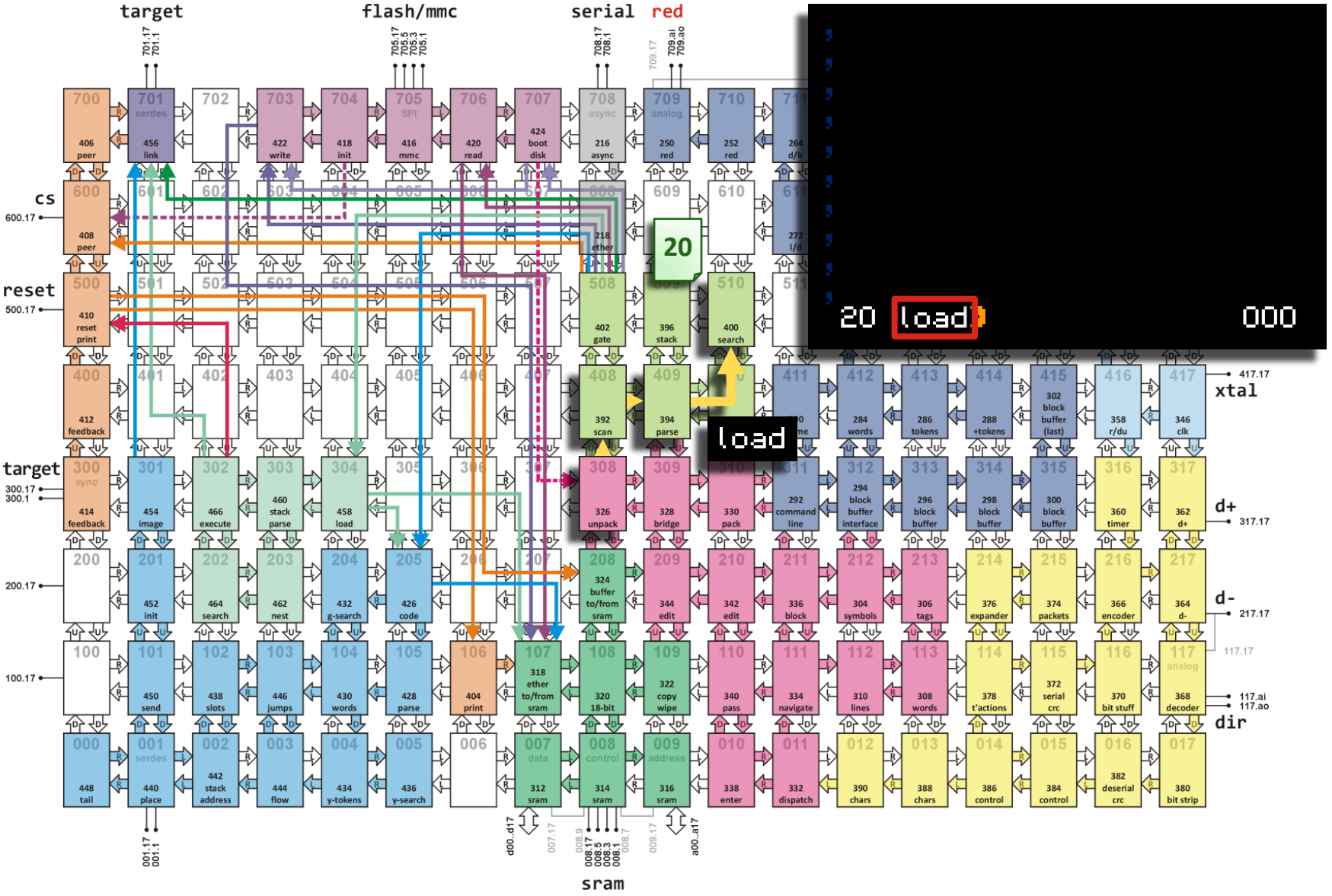
loading applications



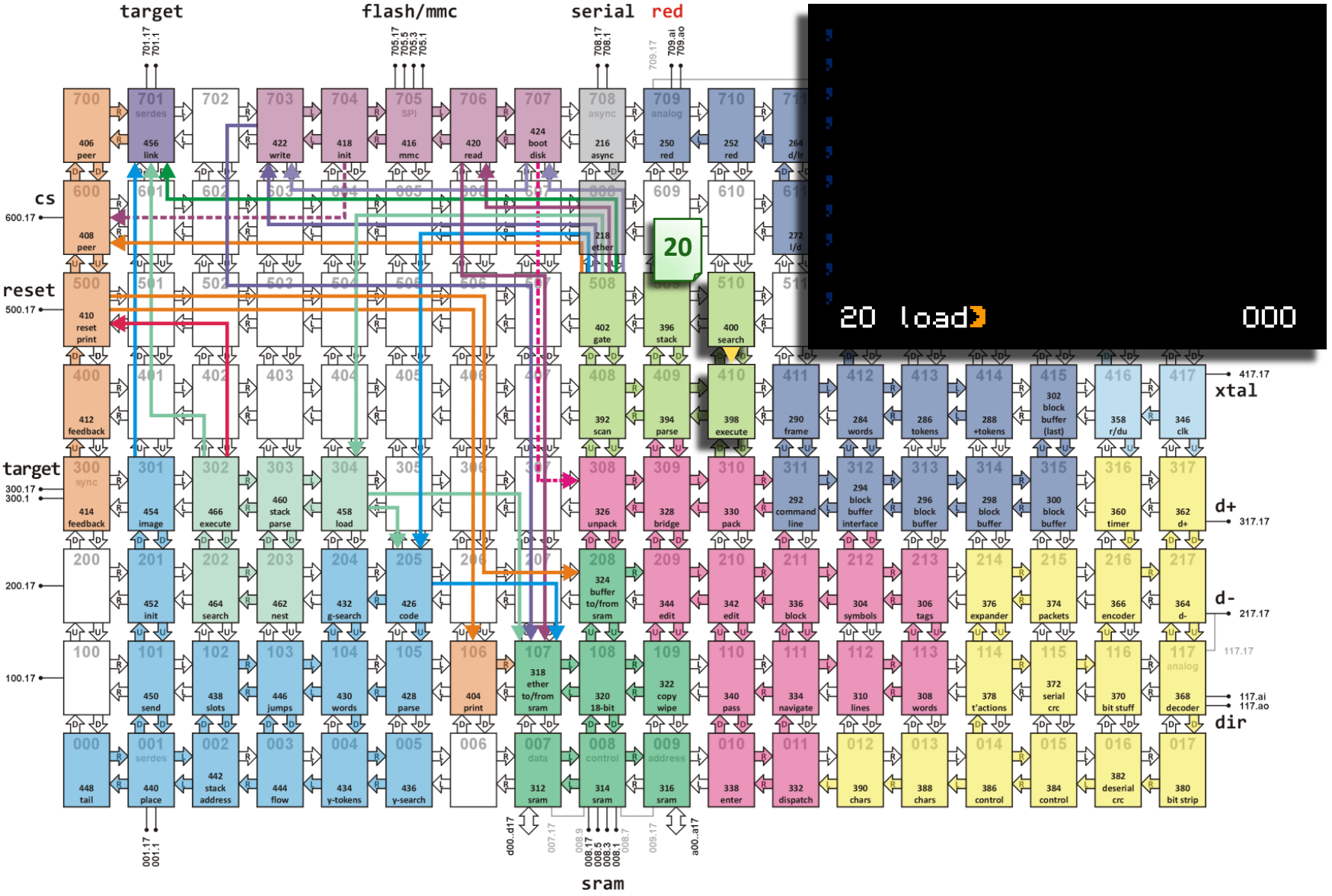
loading applications



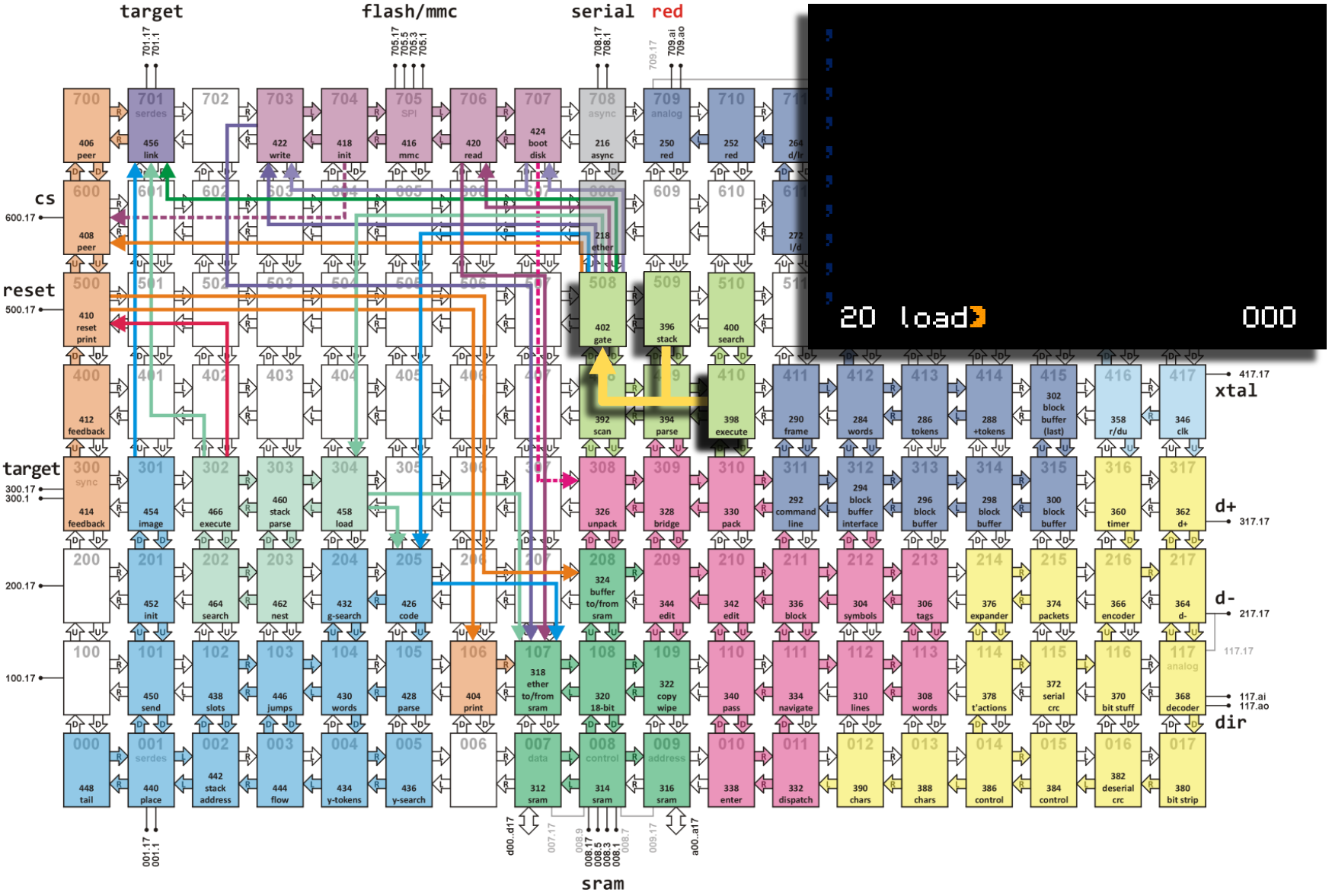
loading applications



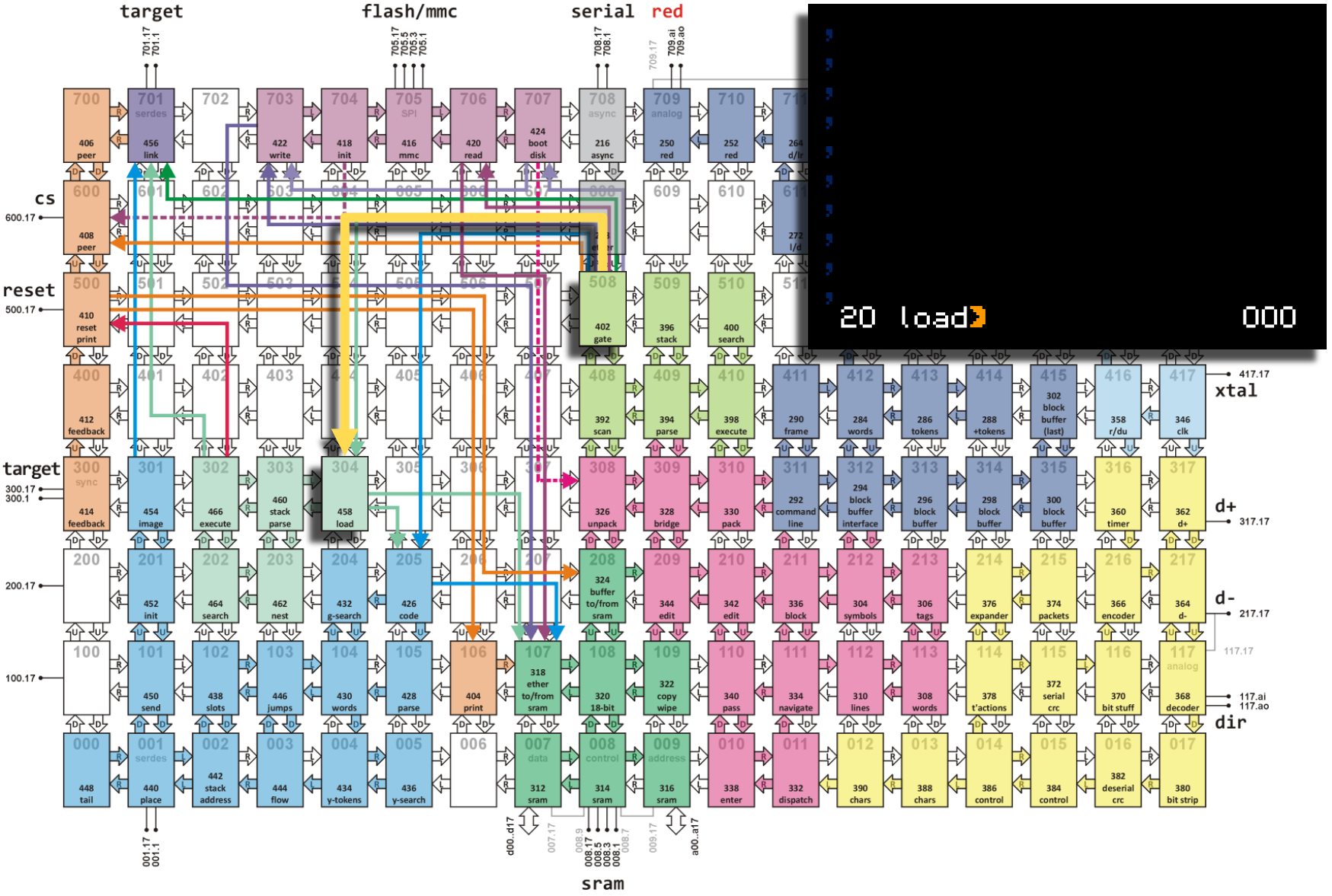
loading applications



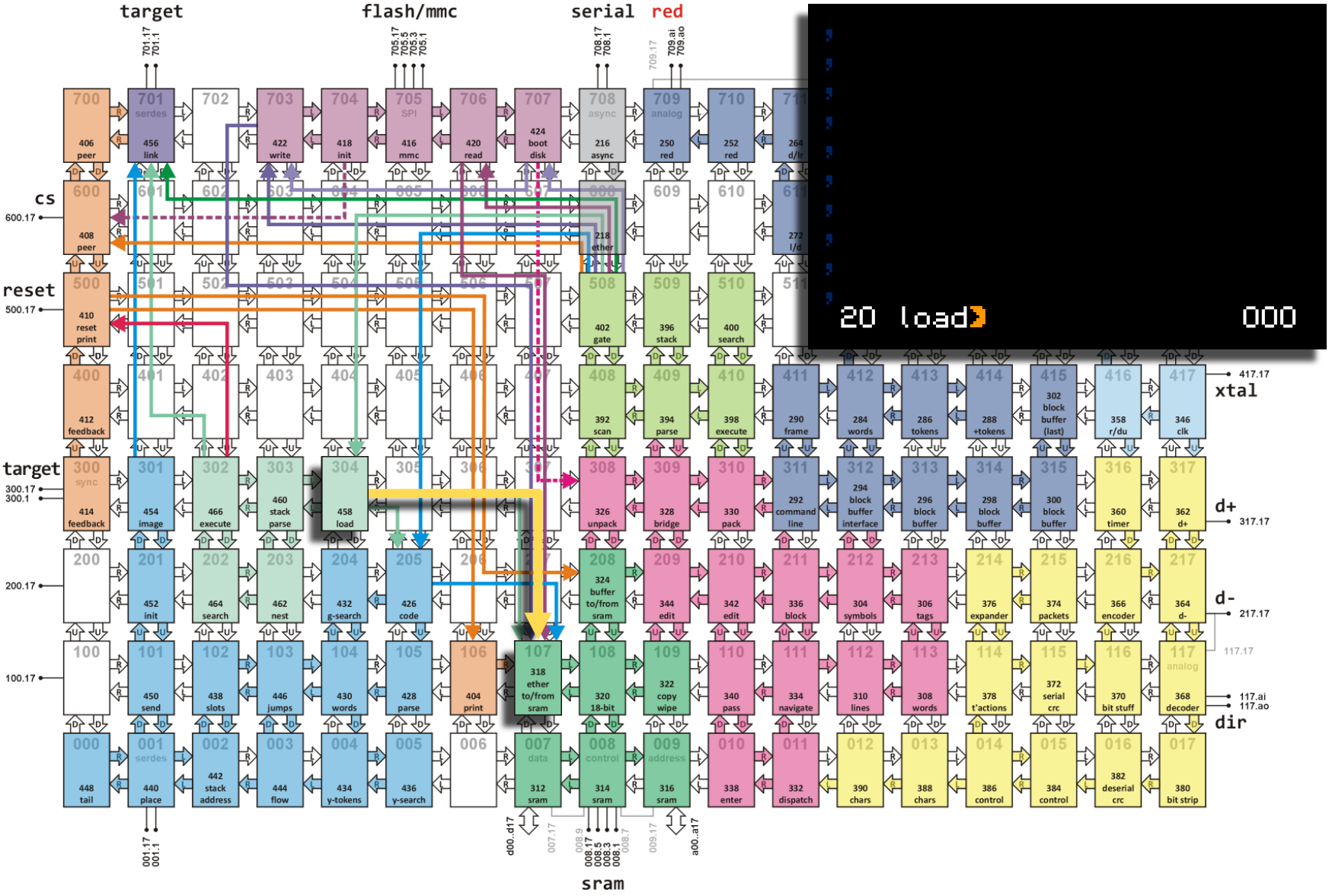
loading applications



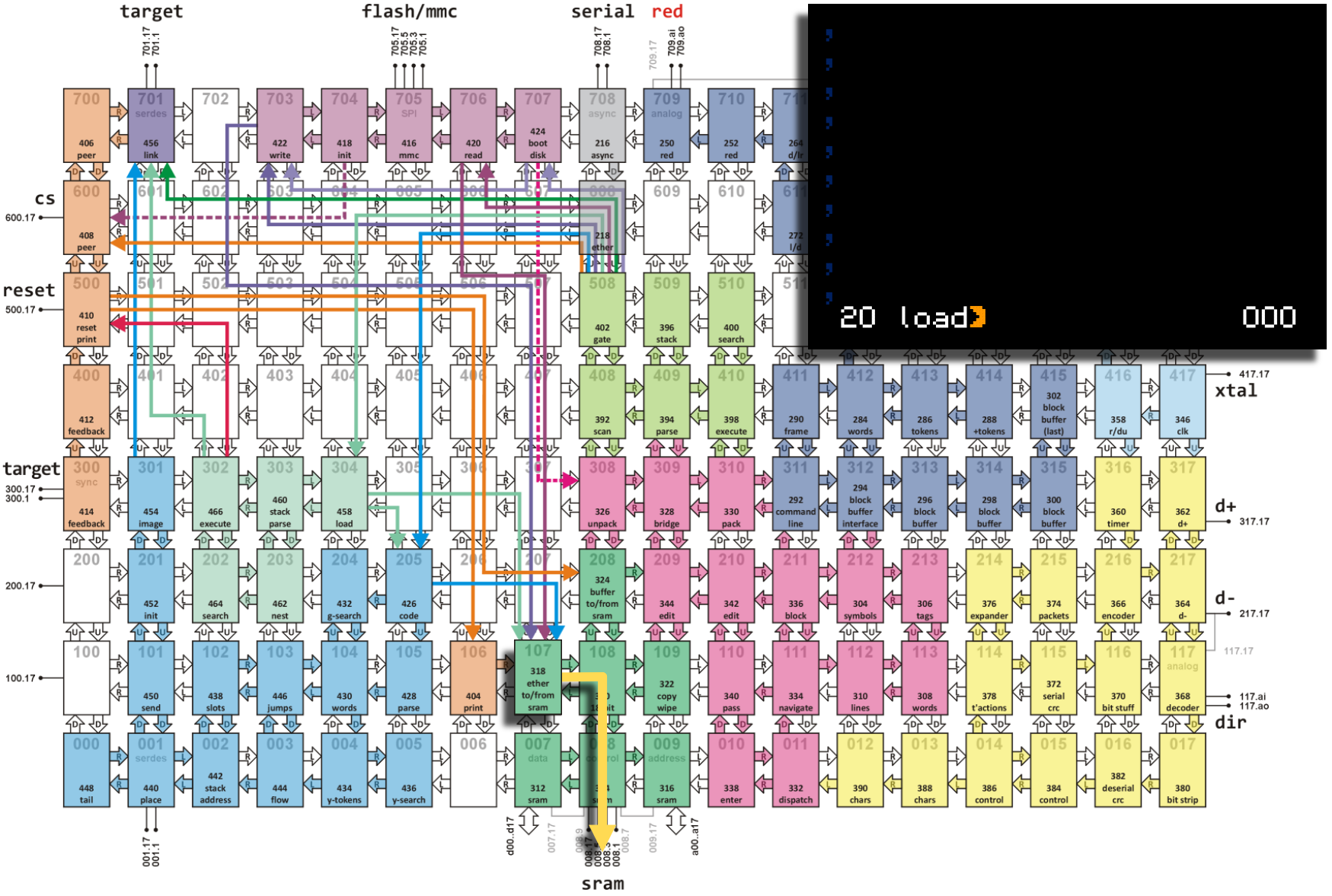
loading applications



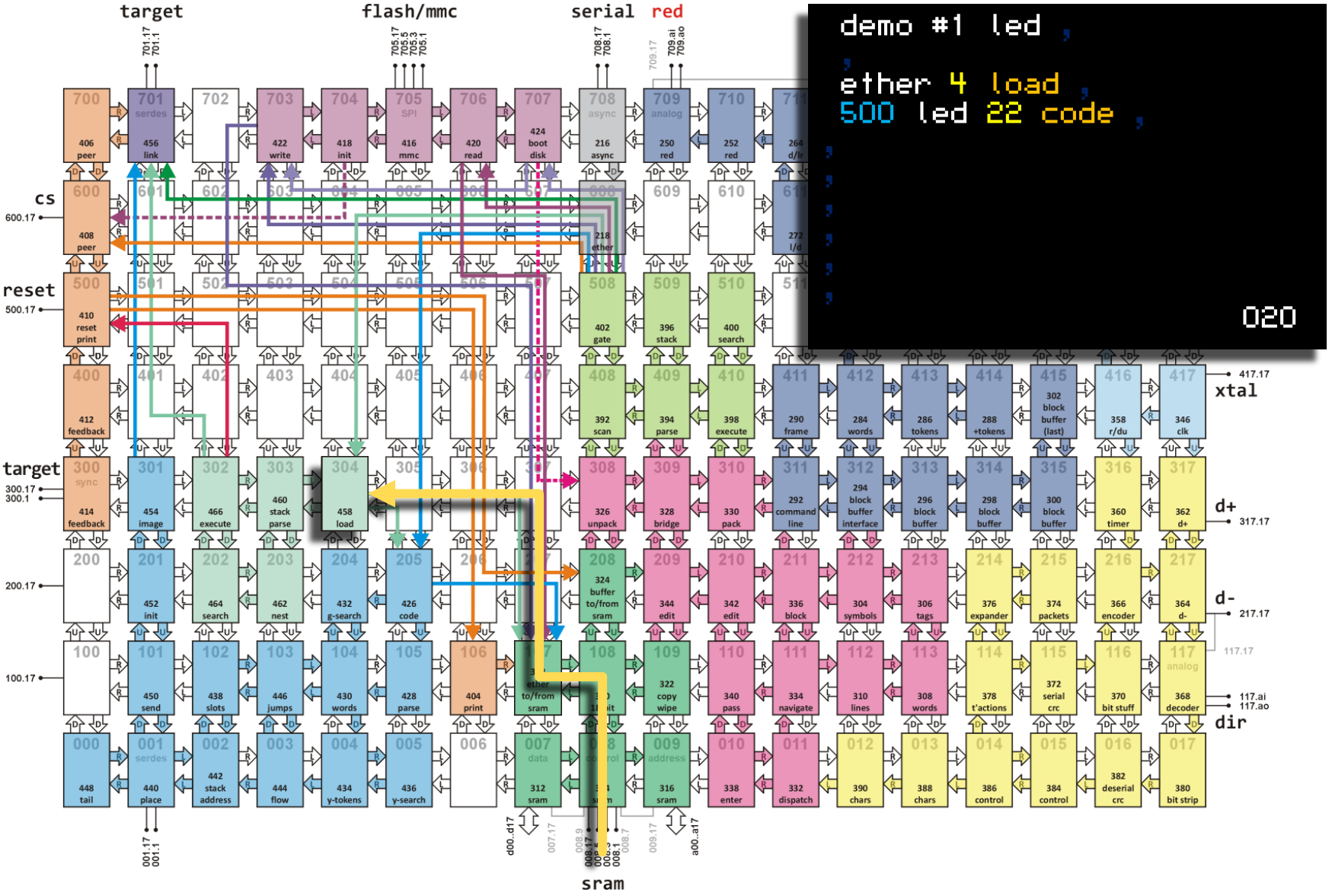
loading applications



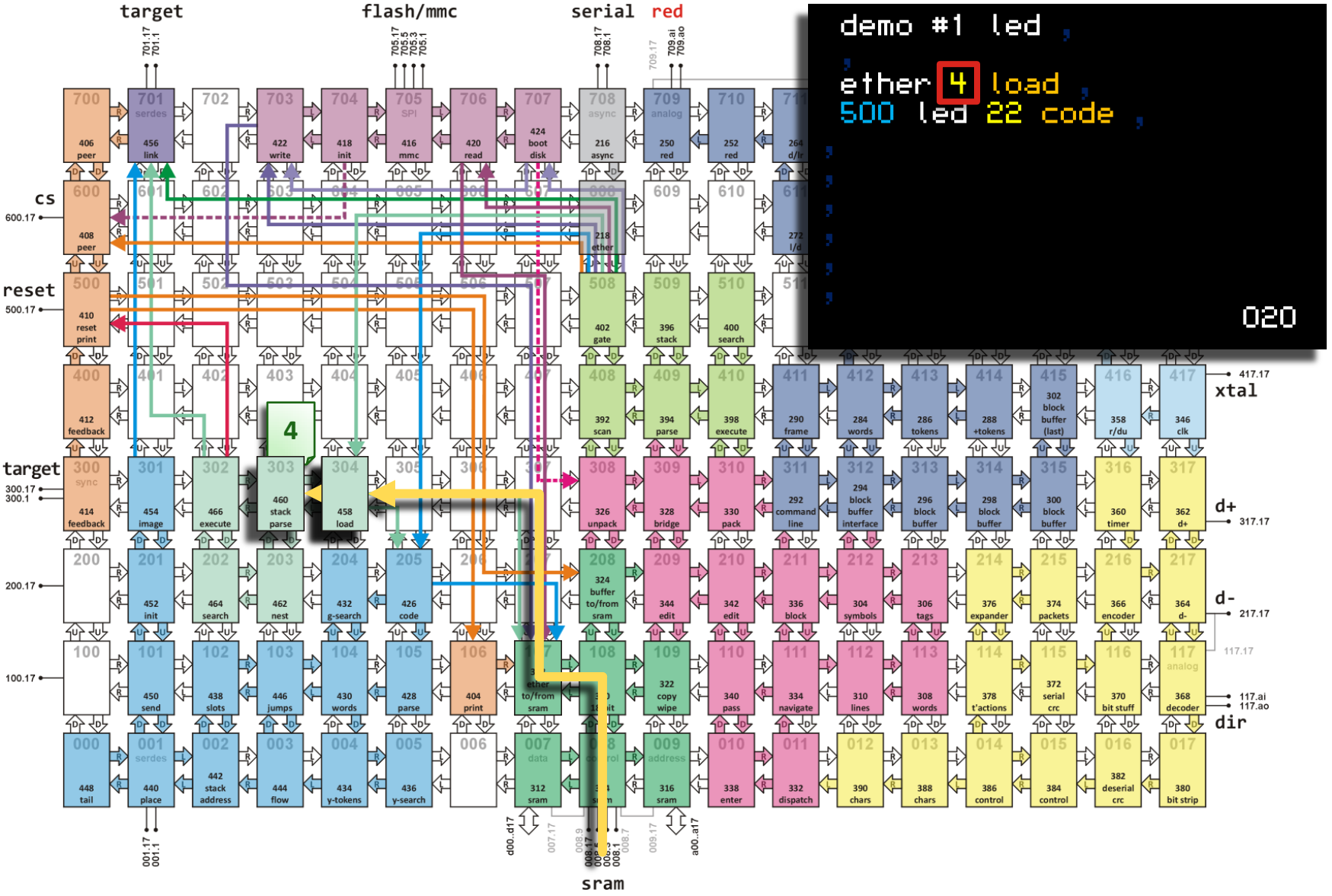
loading applications



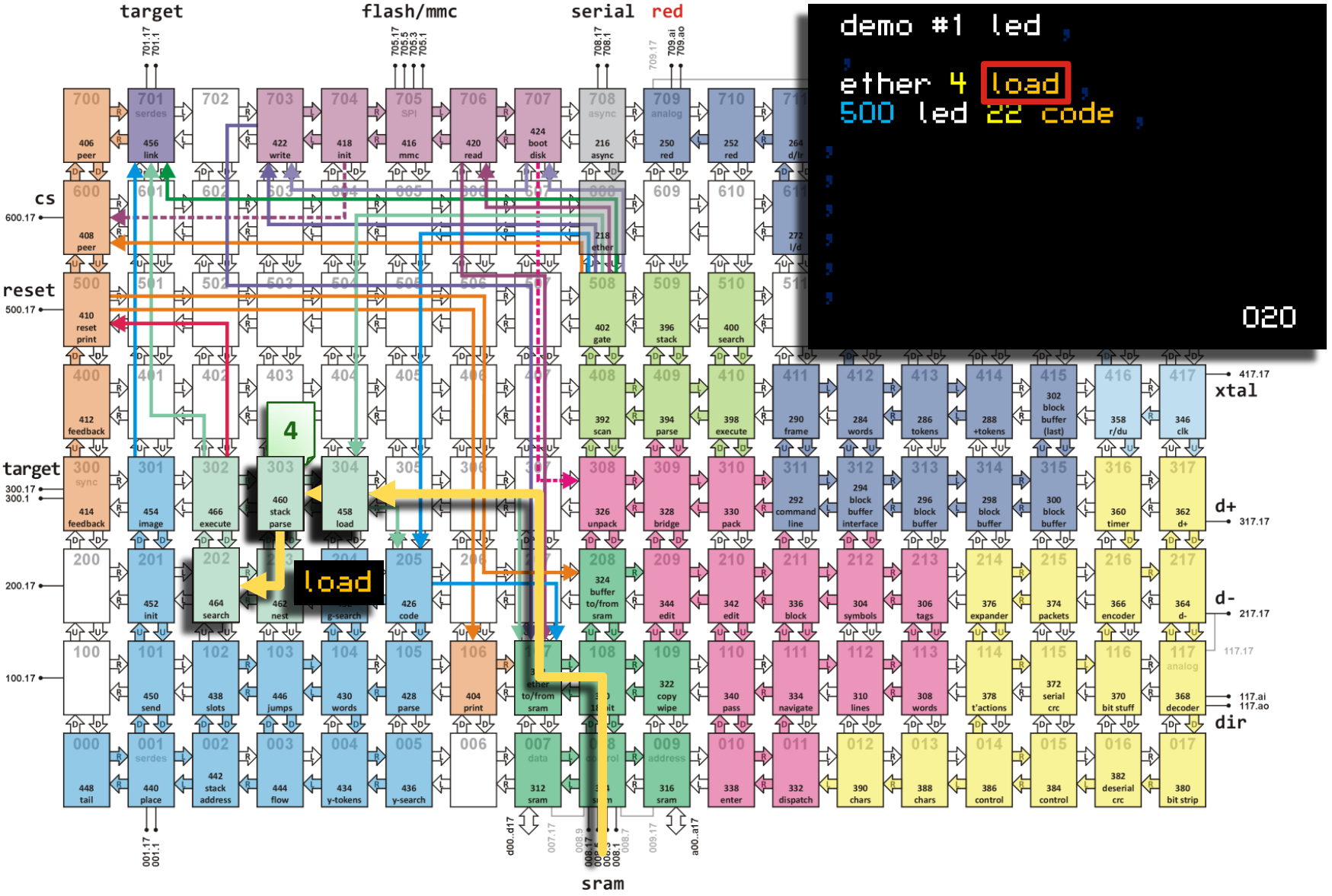
loading applications



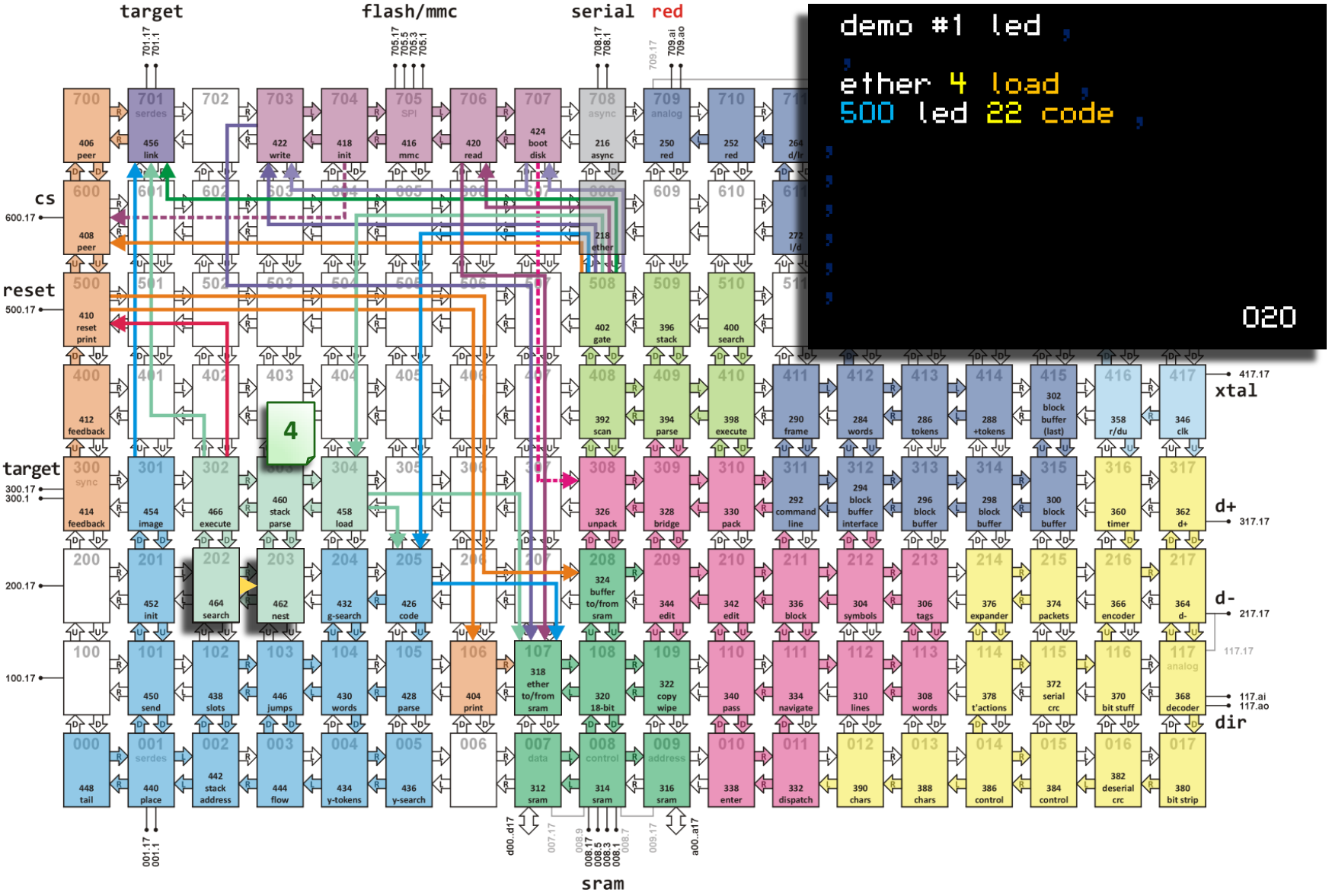
loading applications



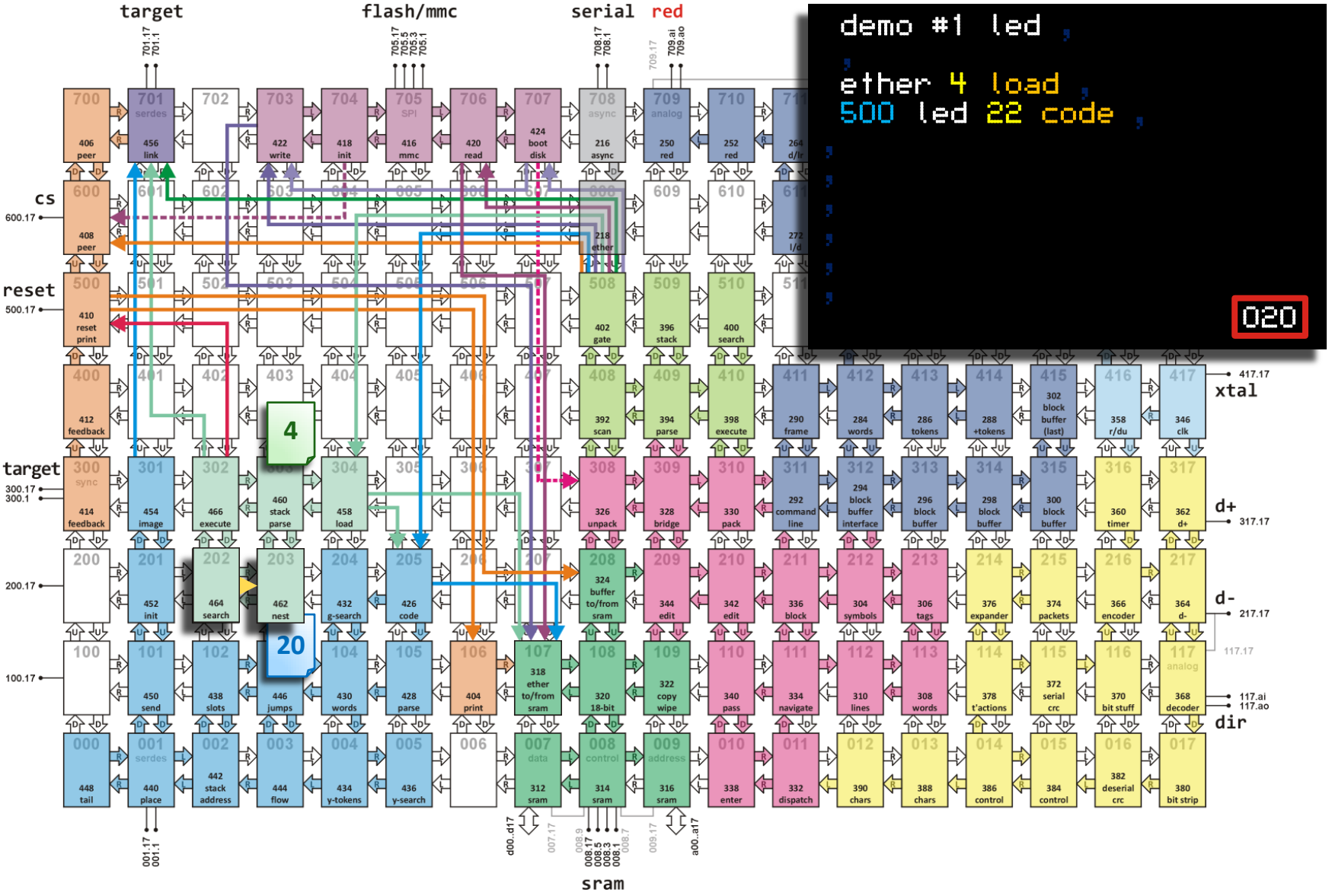
loading applications



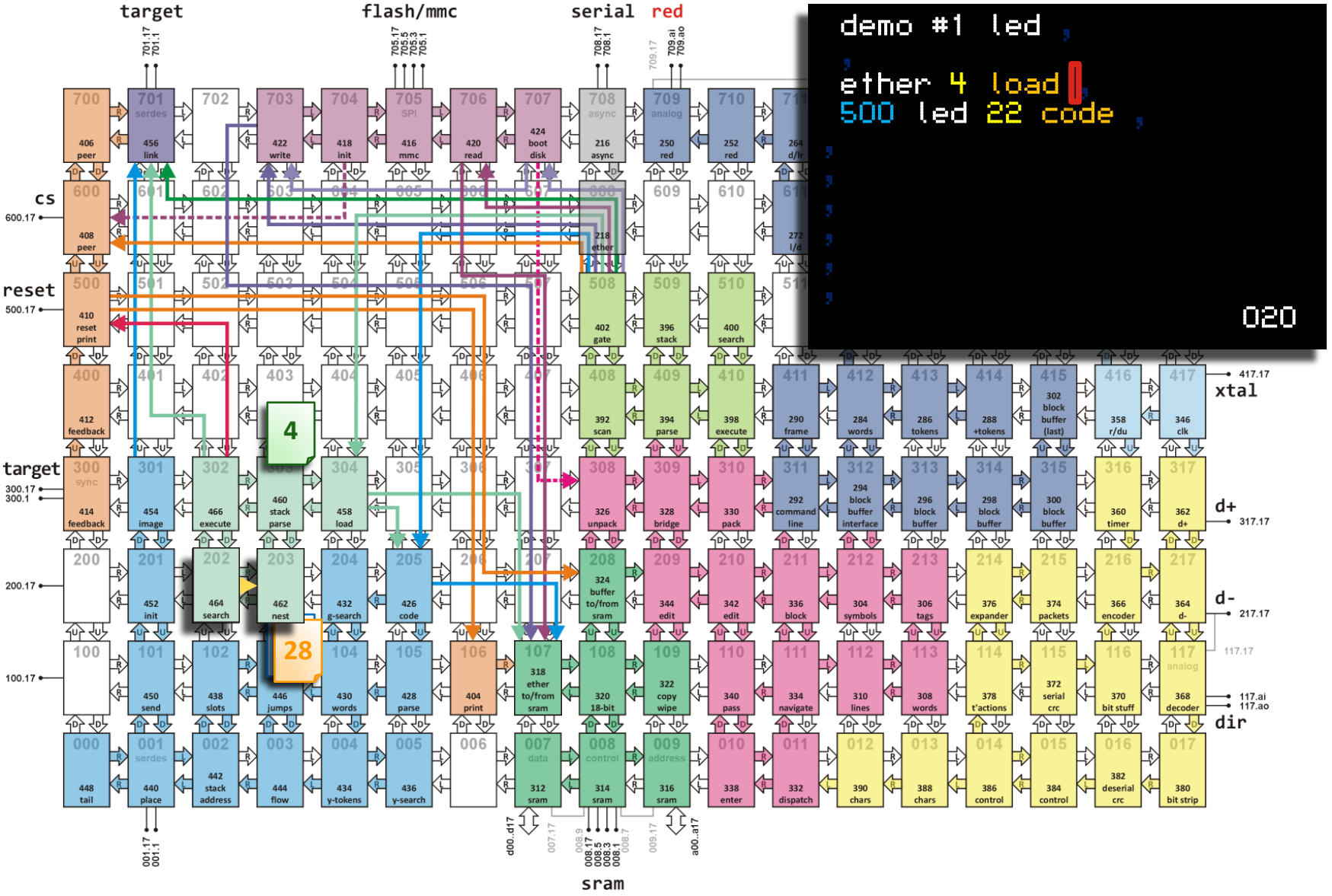
loading applications



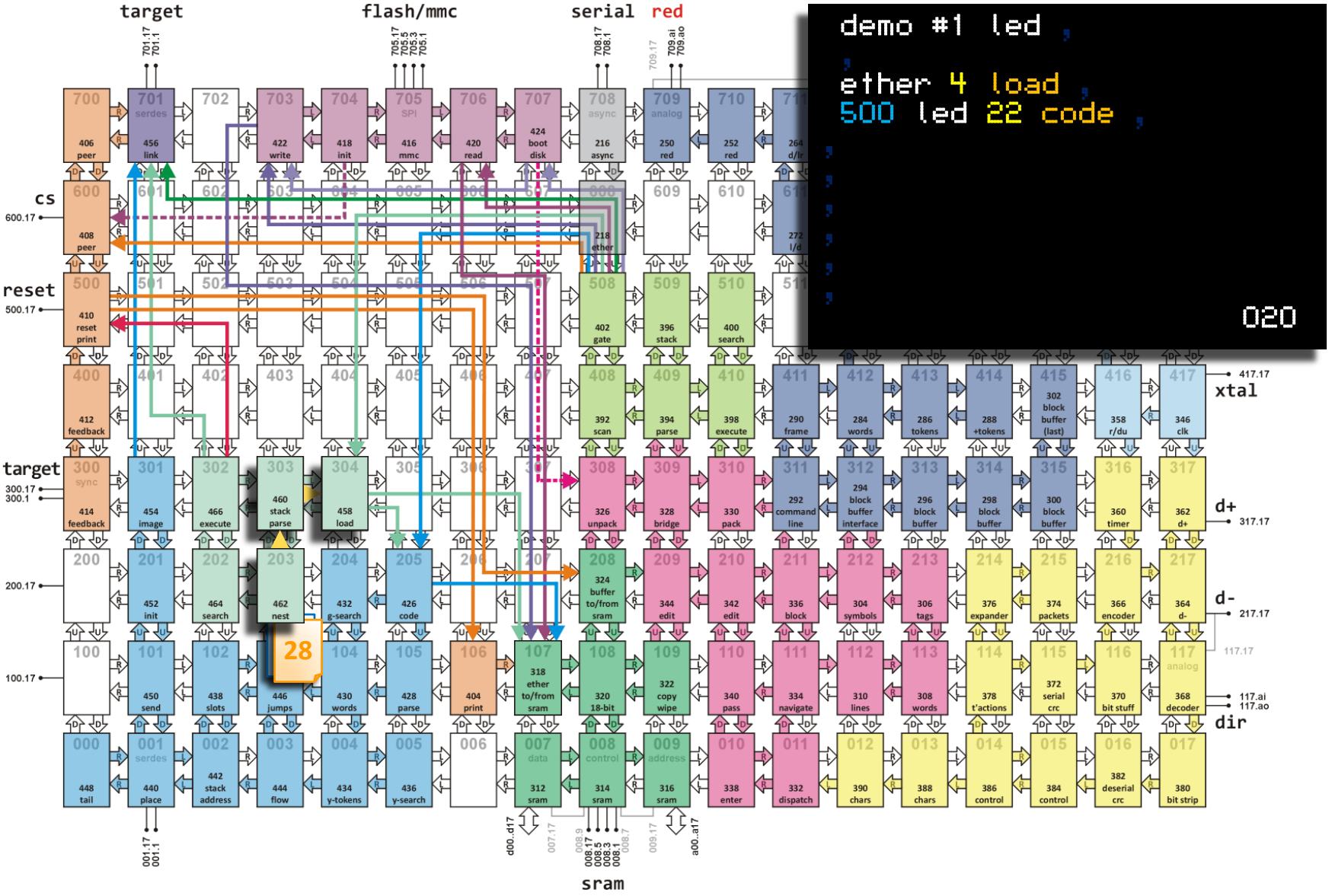
loading applications



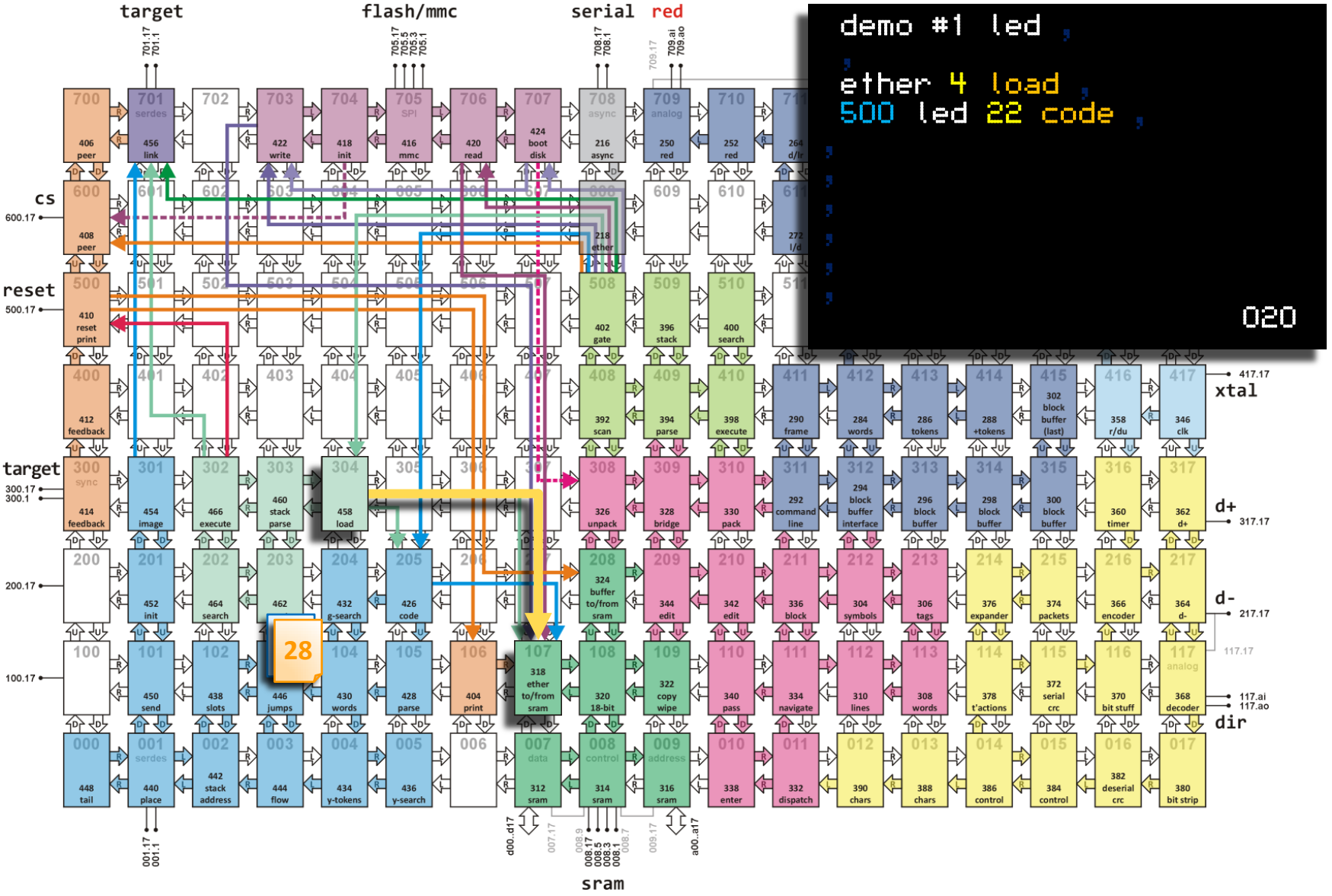
loading applications



loading applications



loading applications

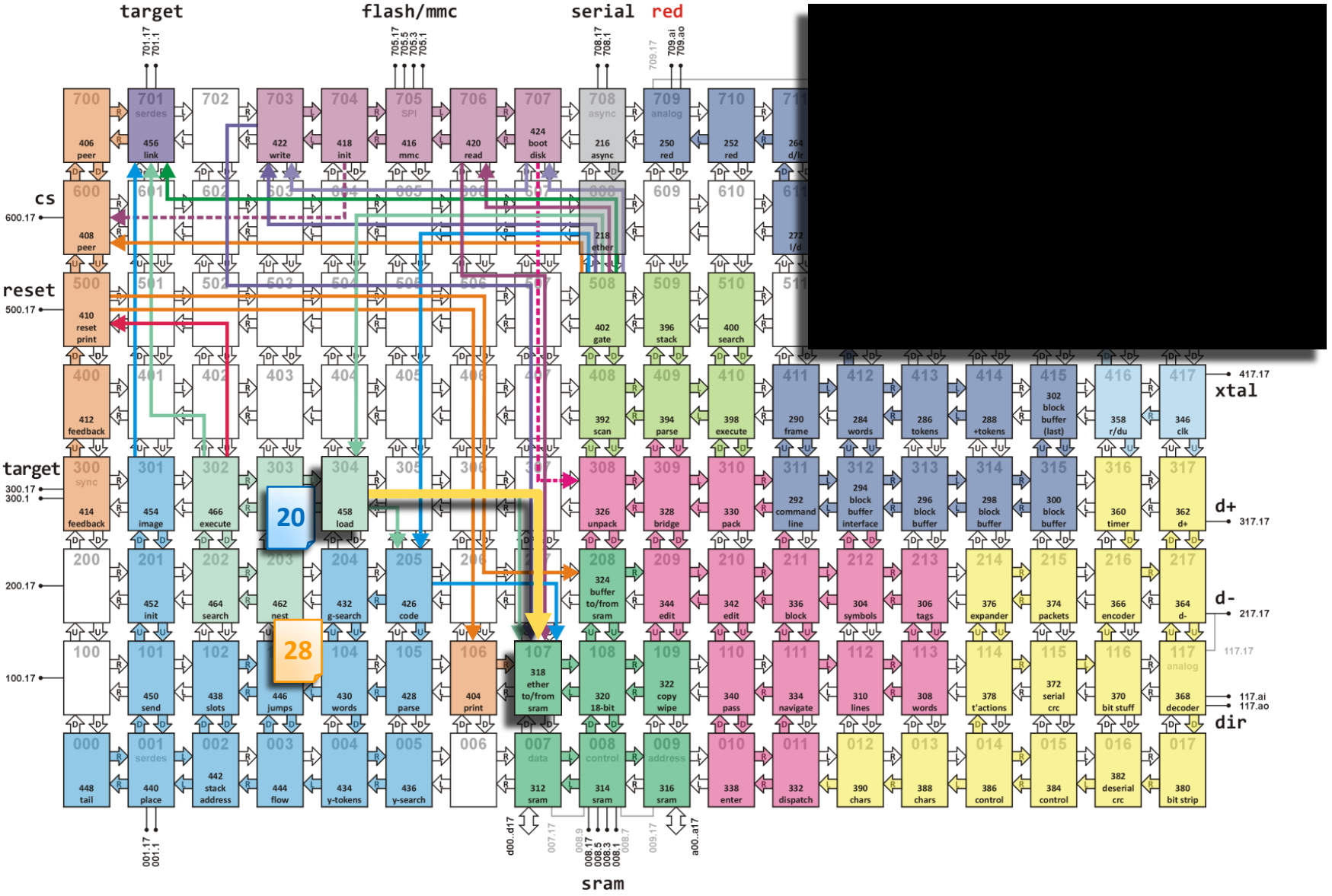


conditioning target chip

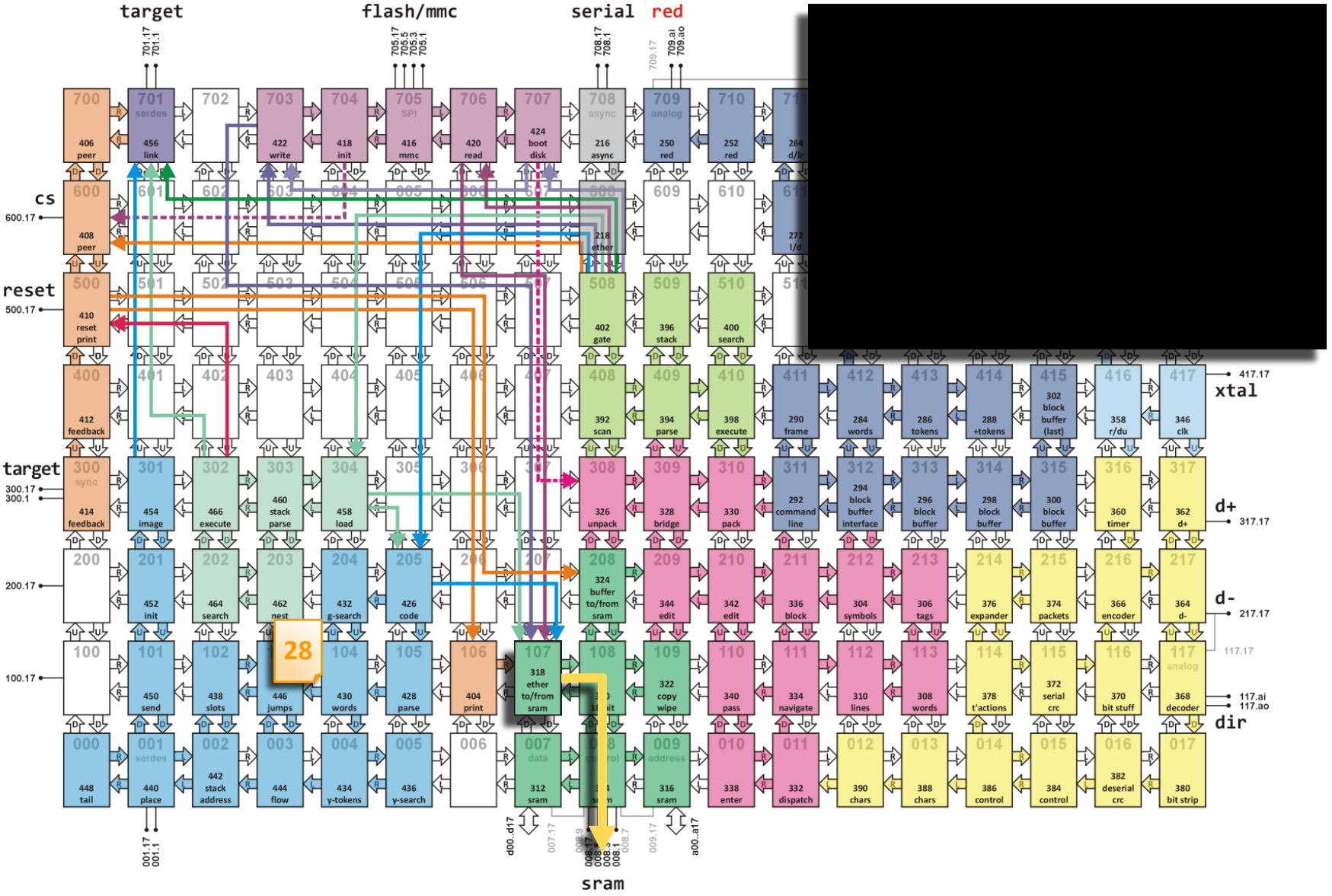
load block 004 to

- reset target chip
- wait till all pins set to their reset state
- create SERDES link – load code to the link node 001
- fill target chip with ether

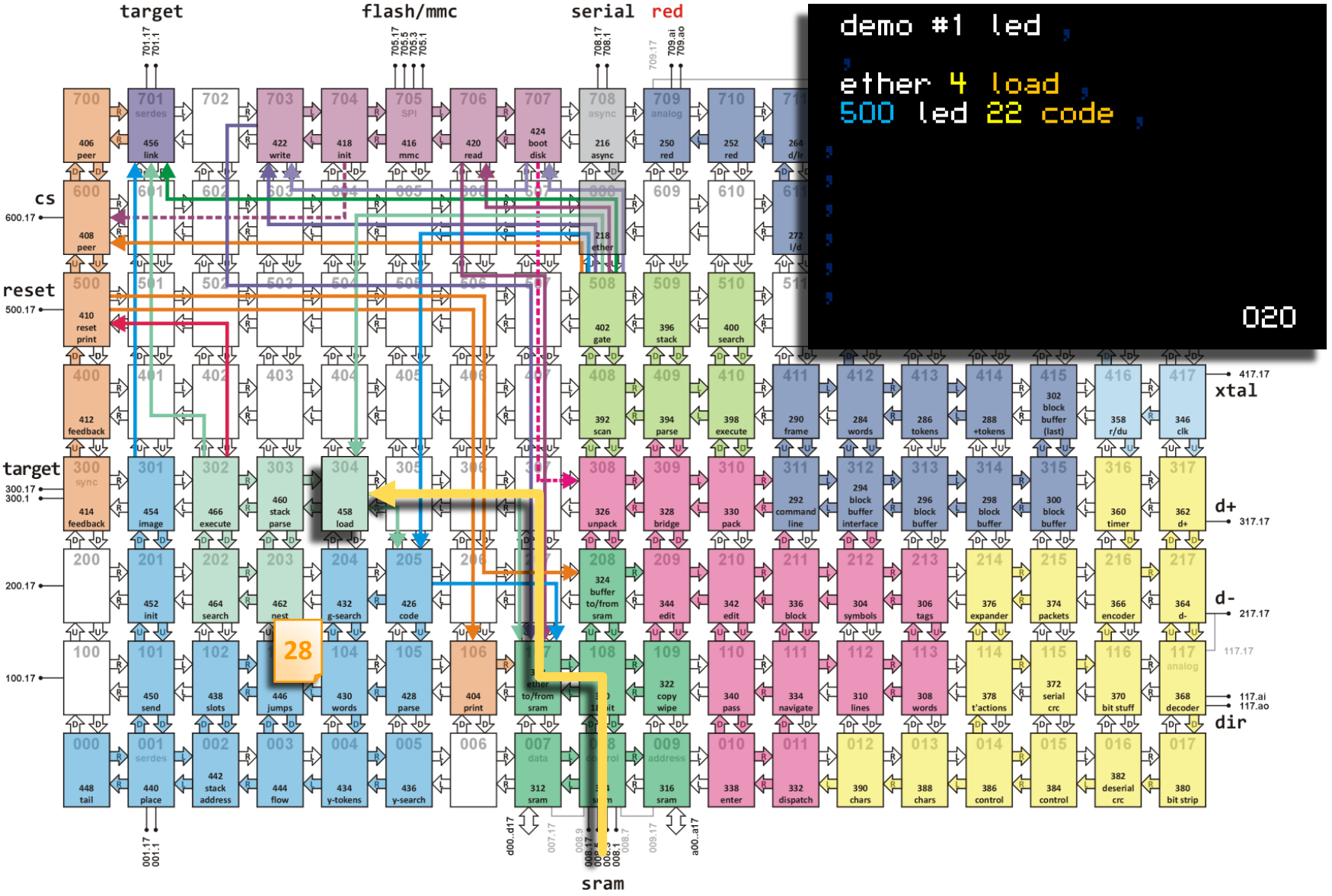
loading applications



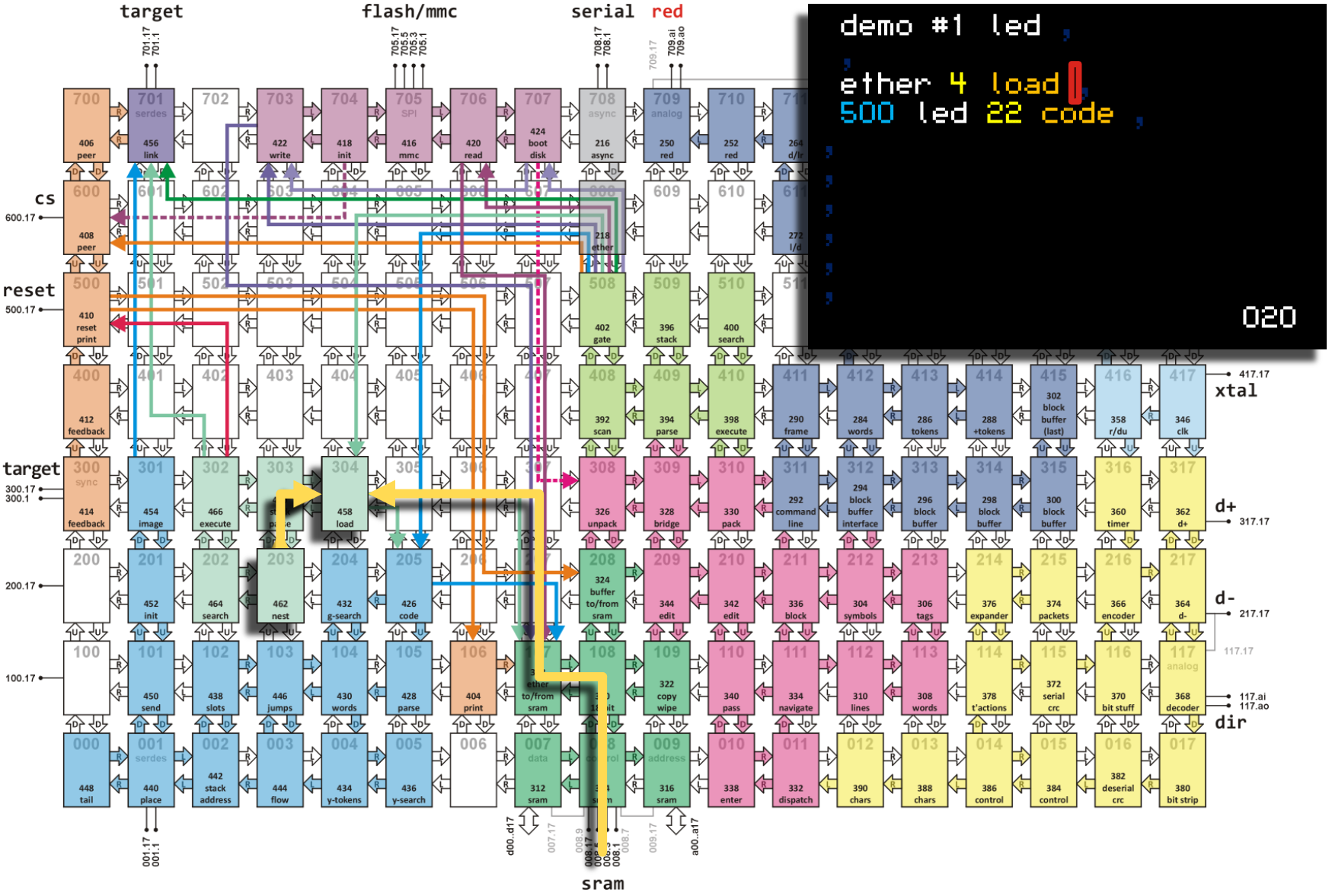
loading applications



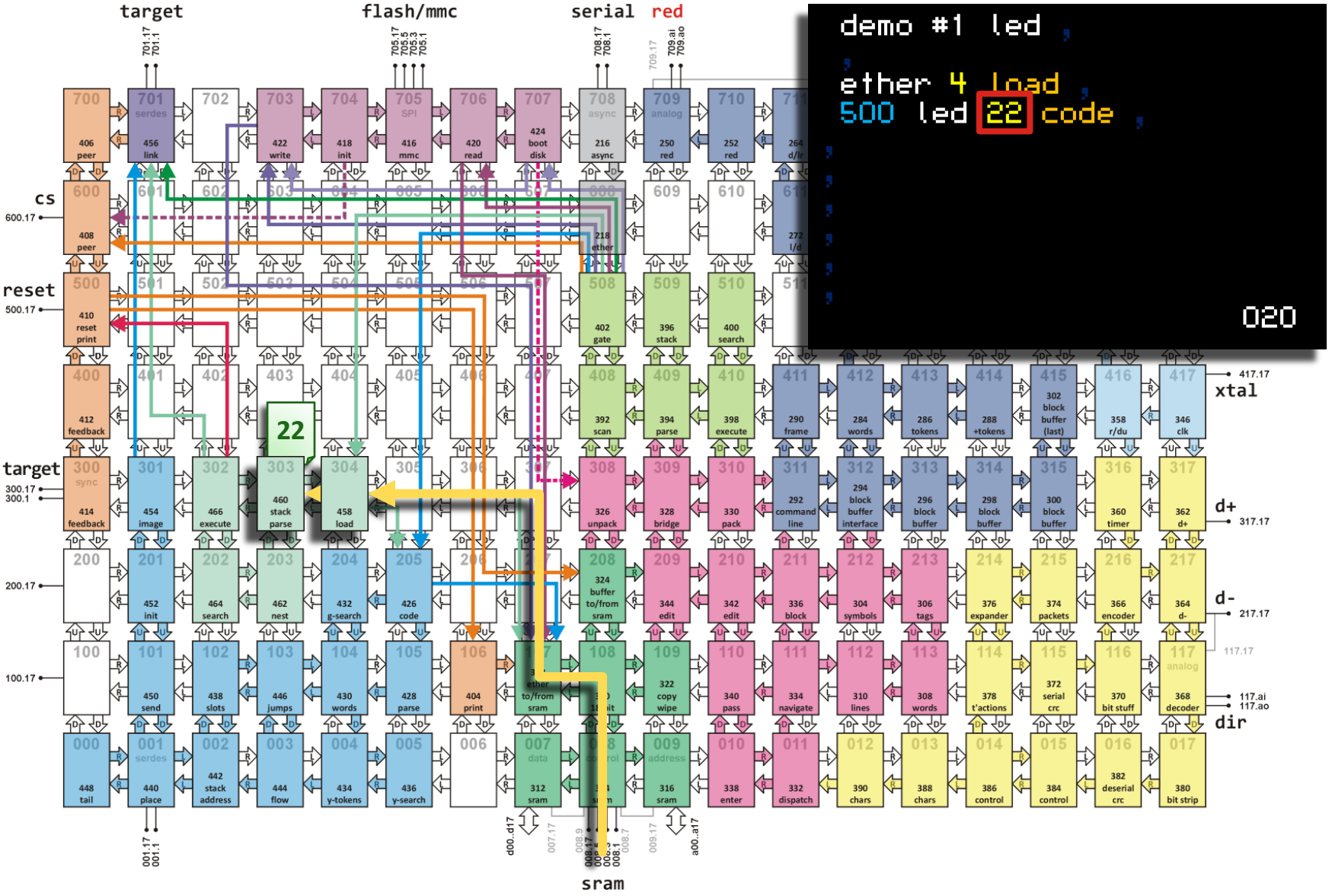
loading applications



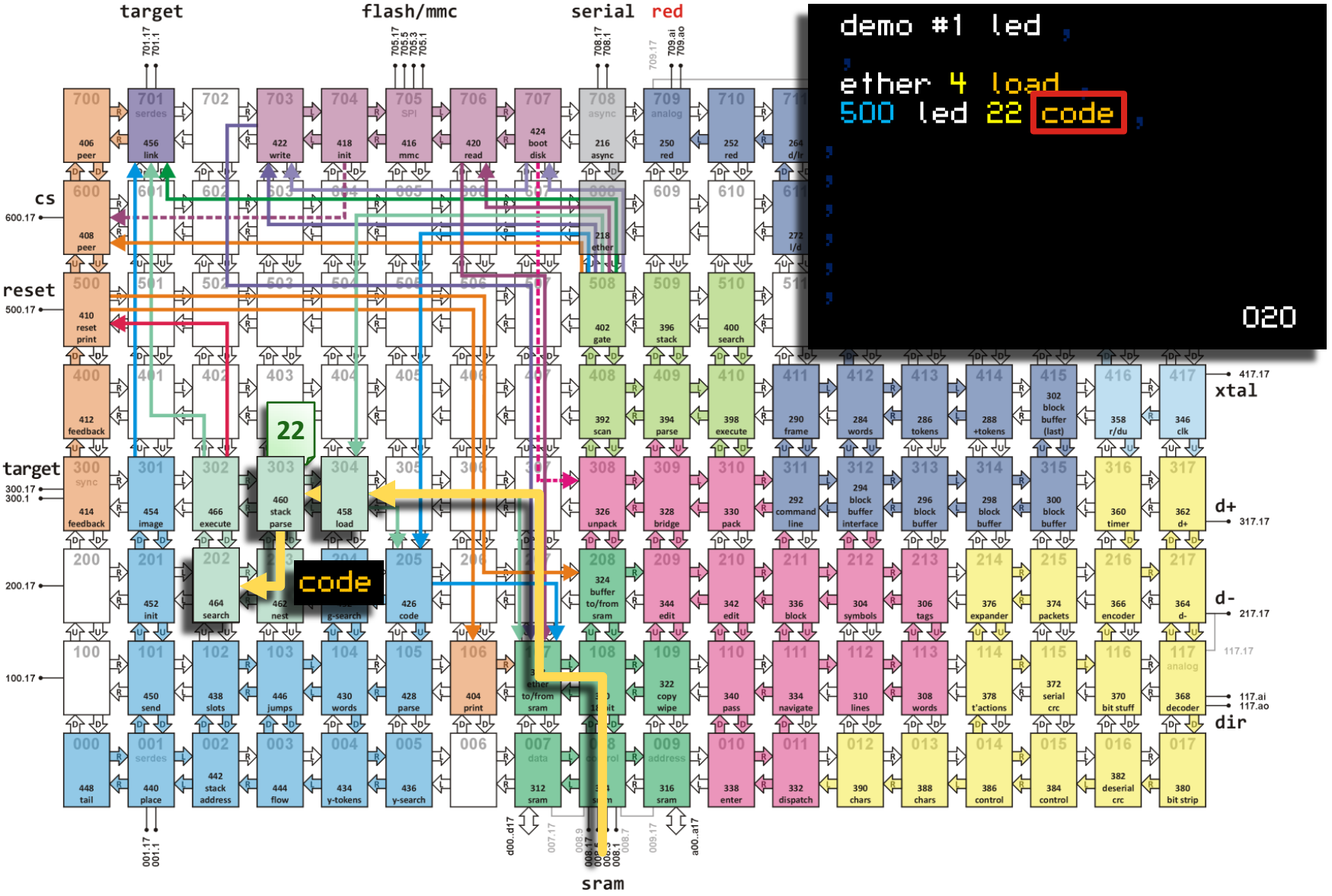
loading applications



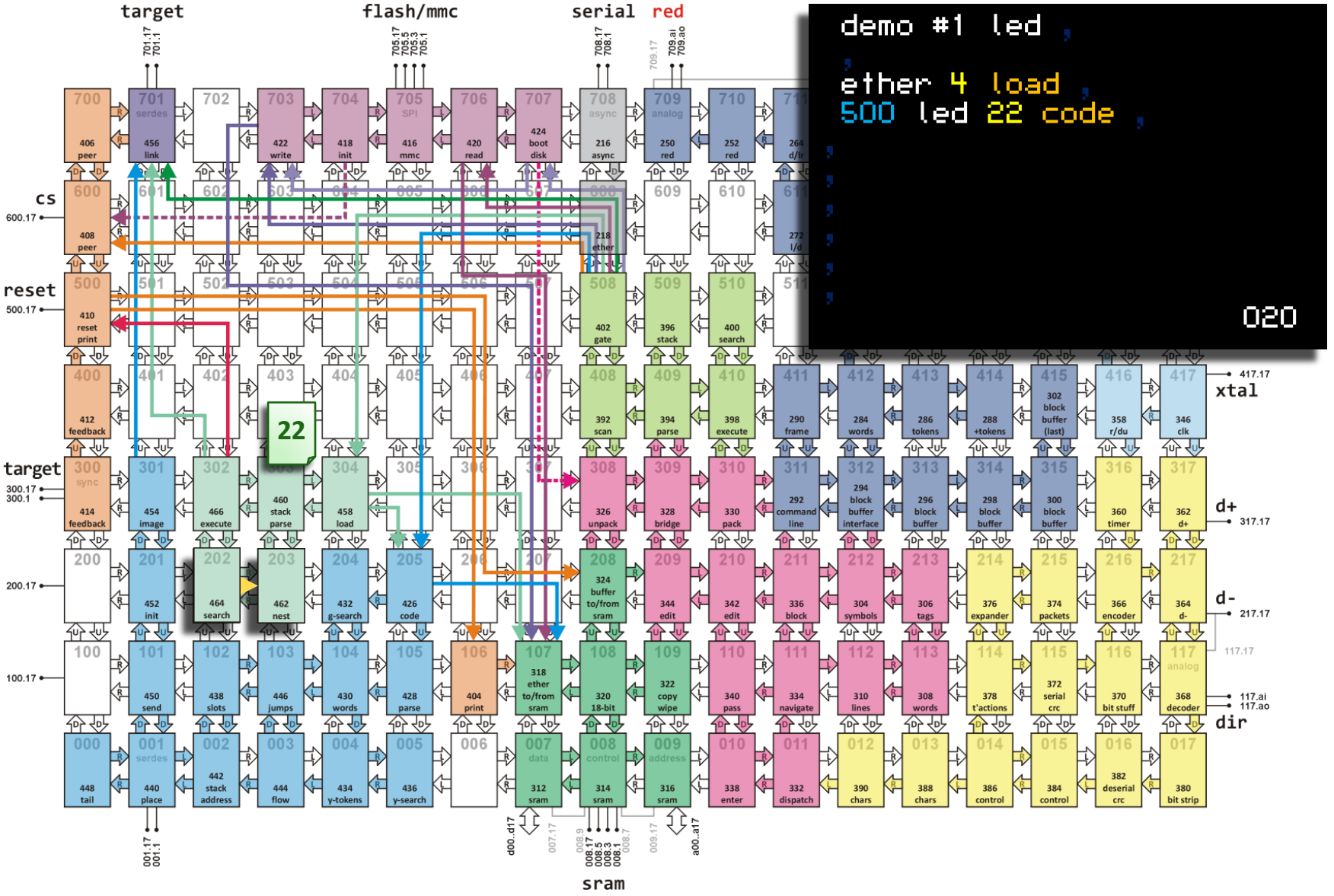
loading applications



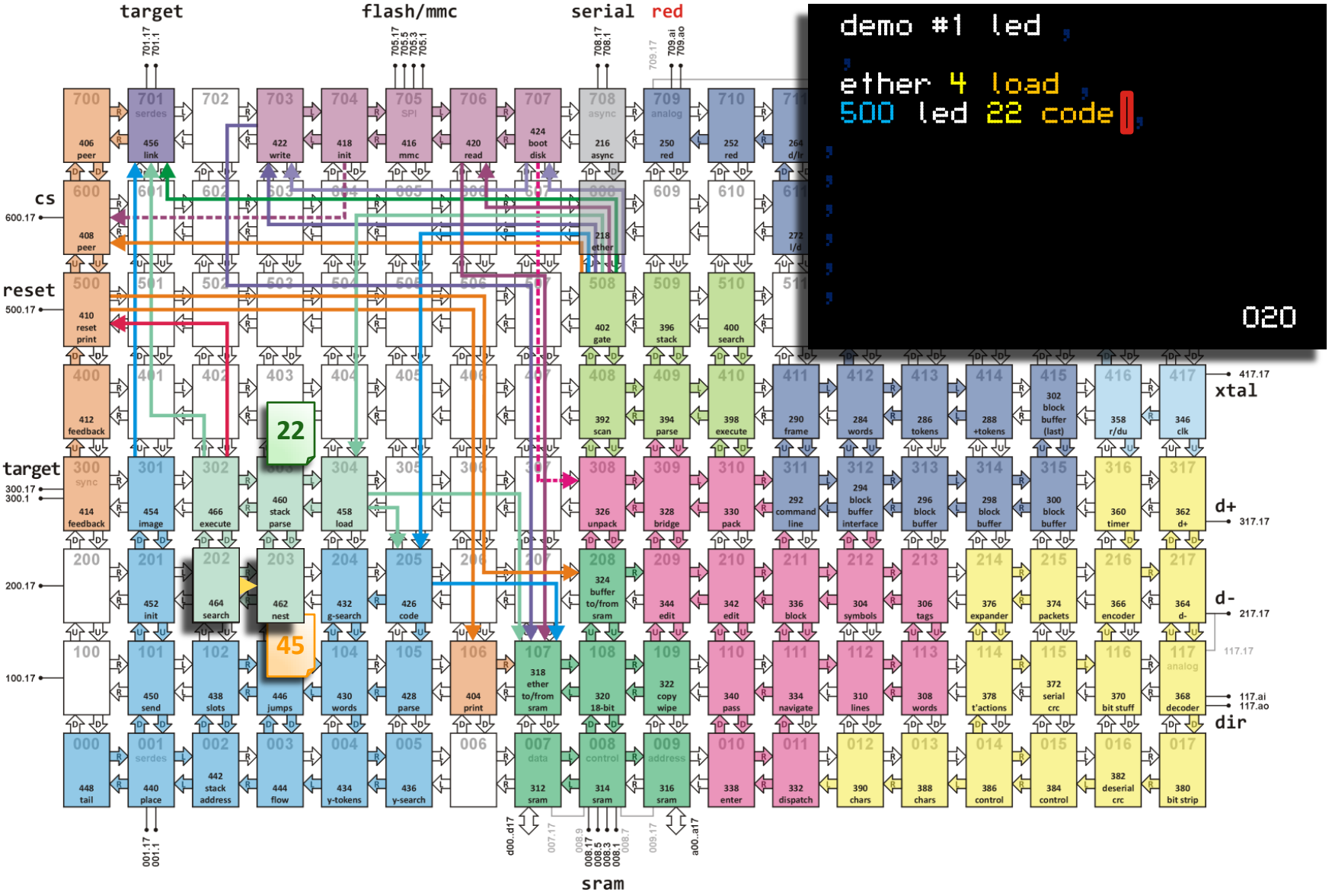
loading applications



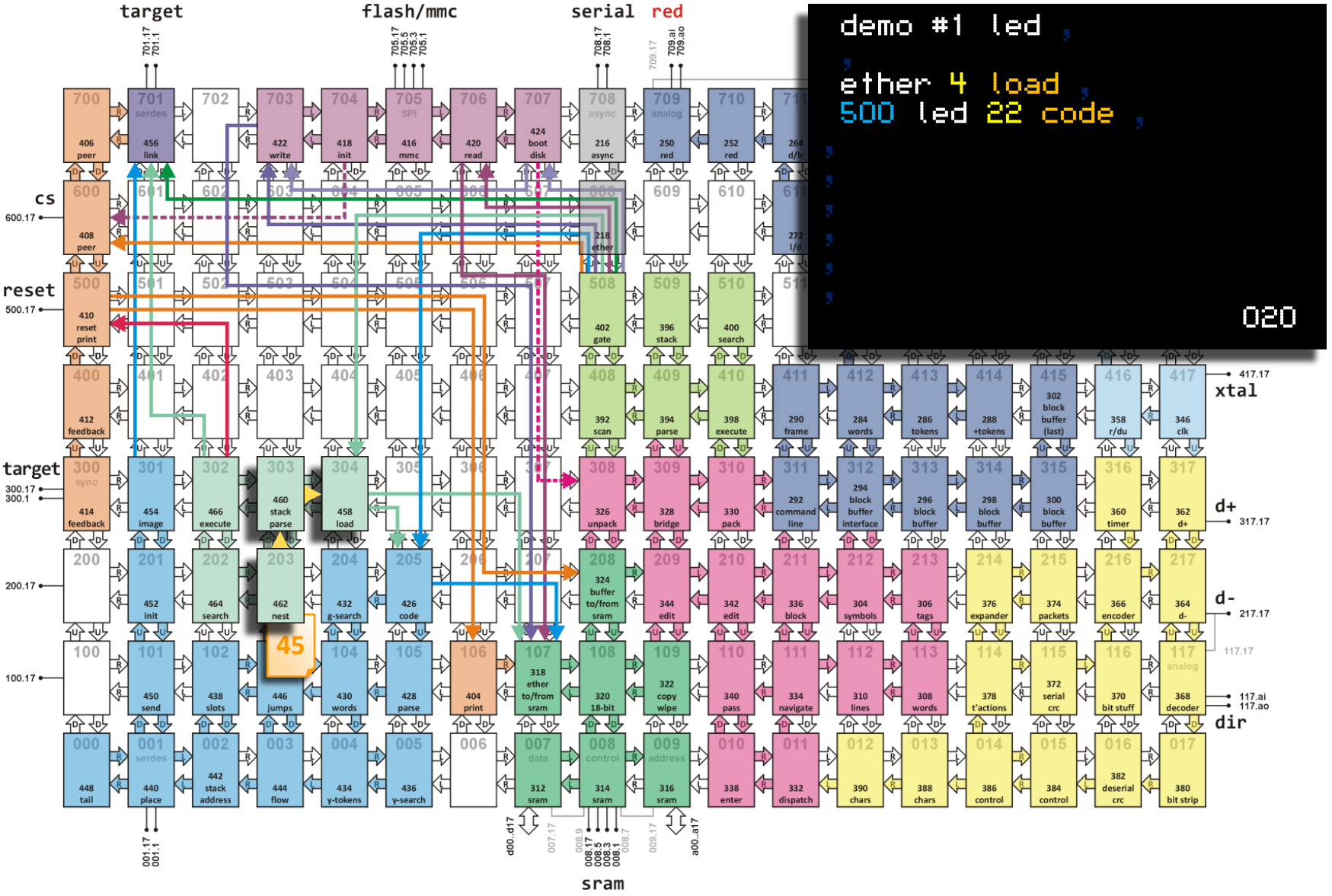
loading applications



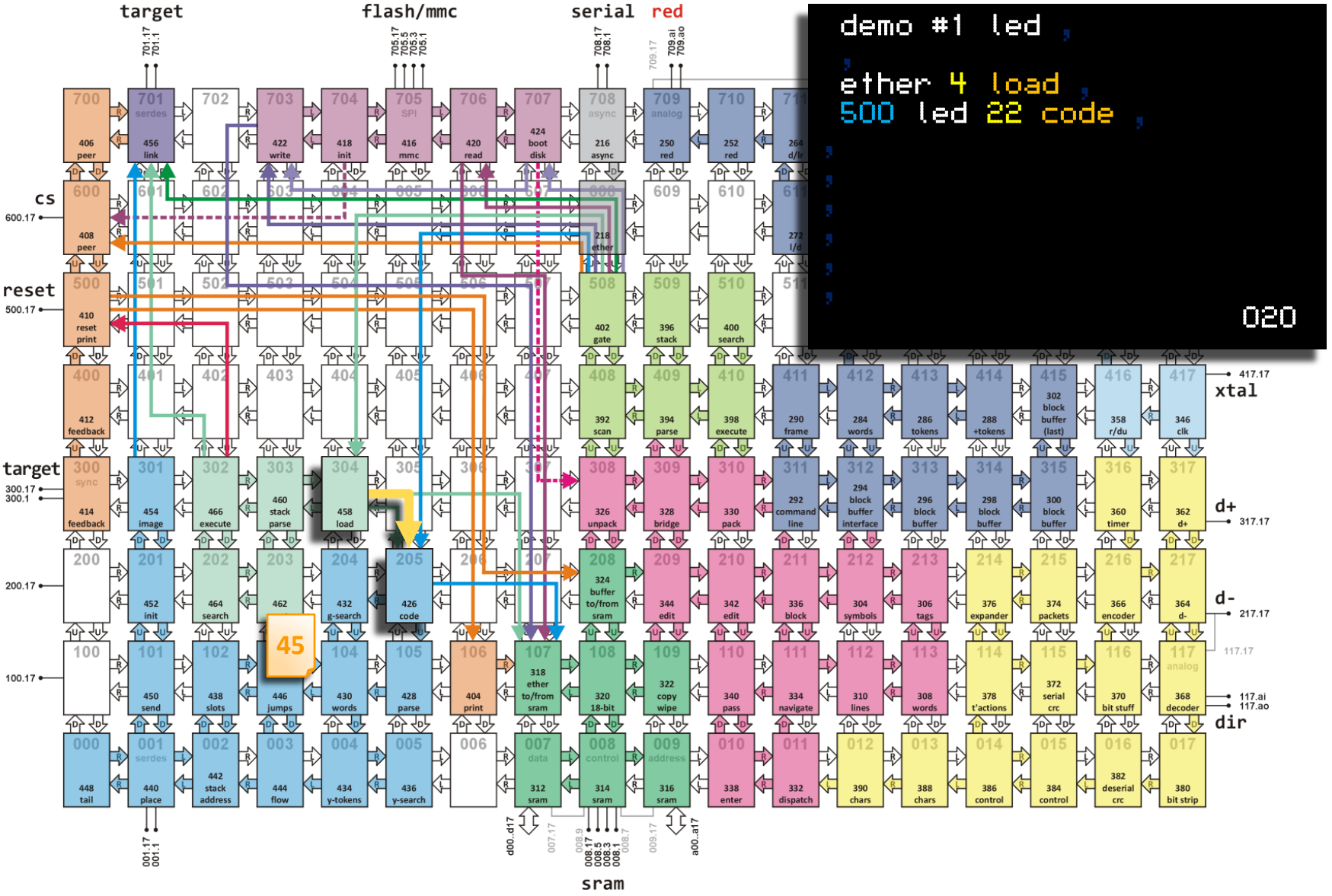
loading applications



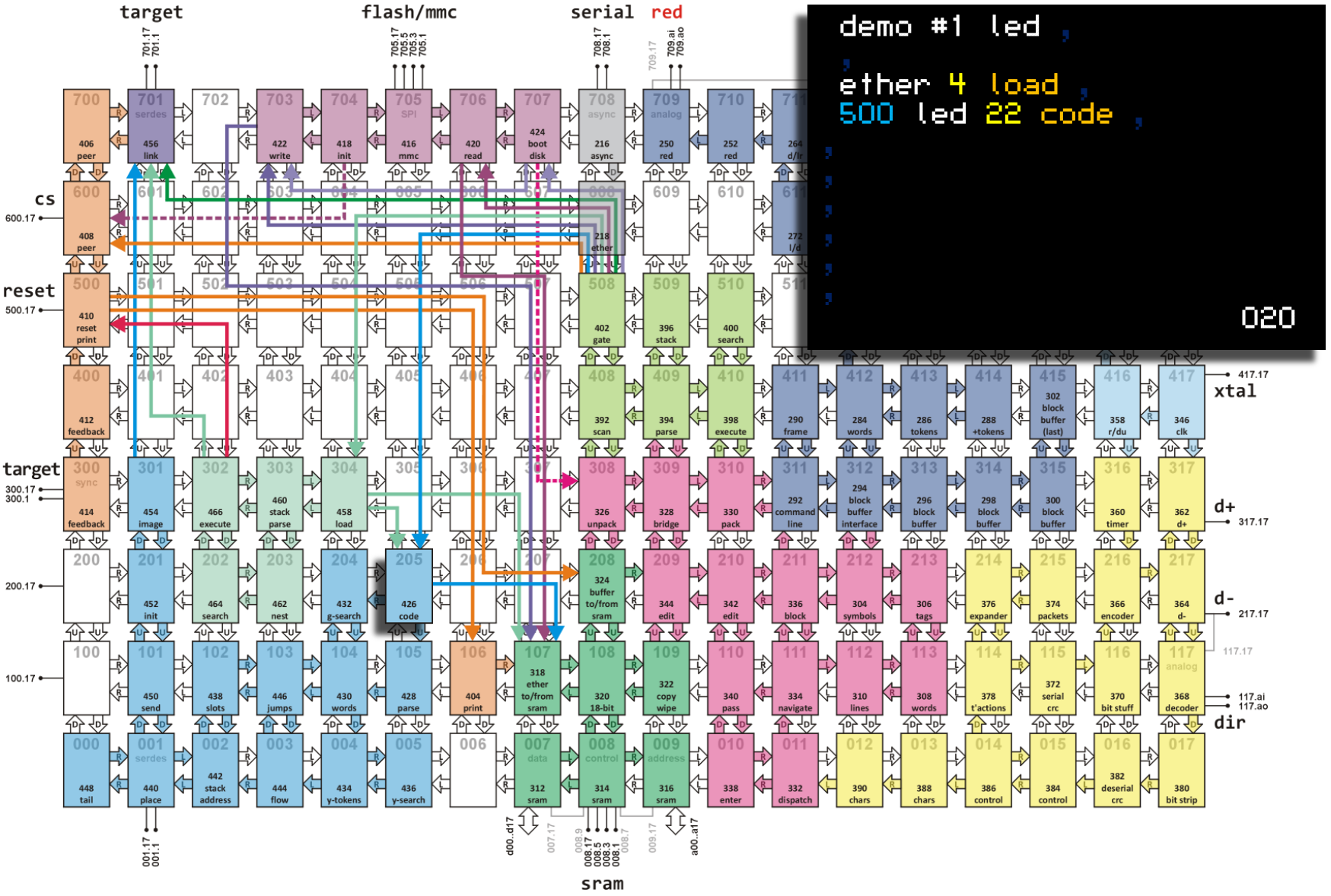
loading applications



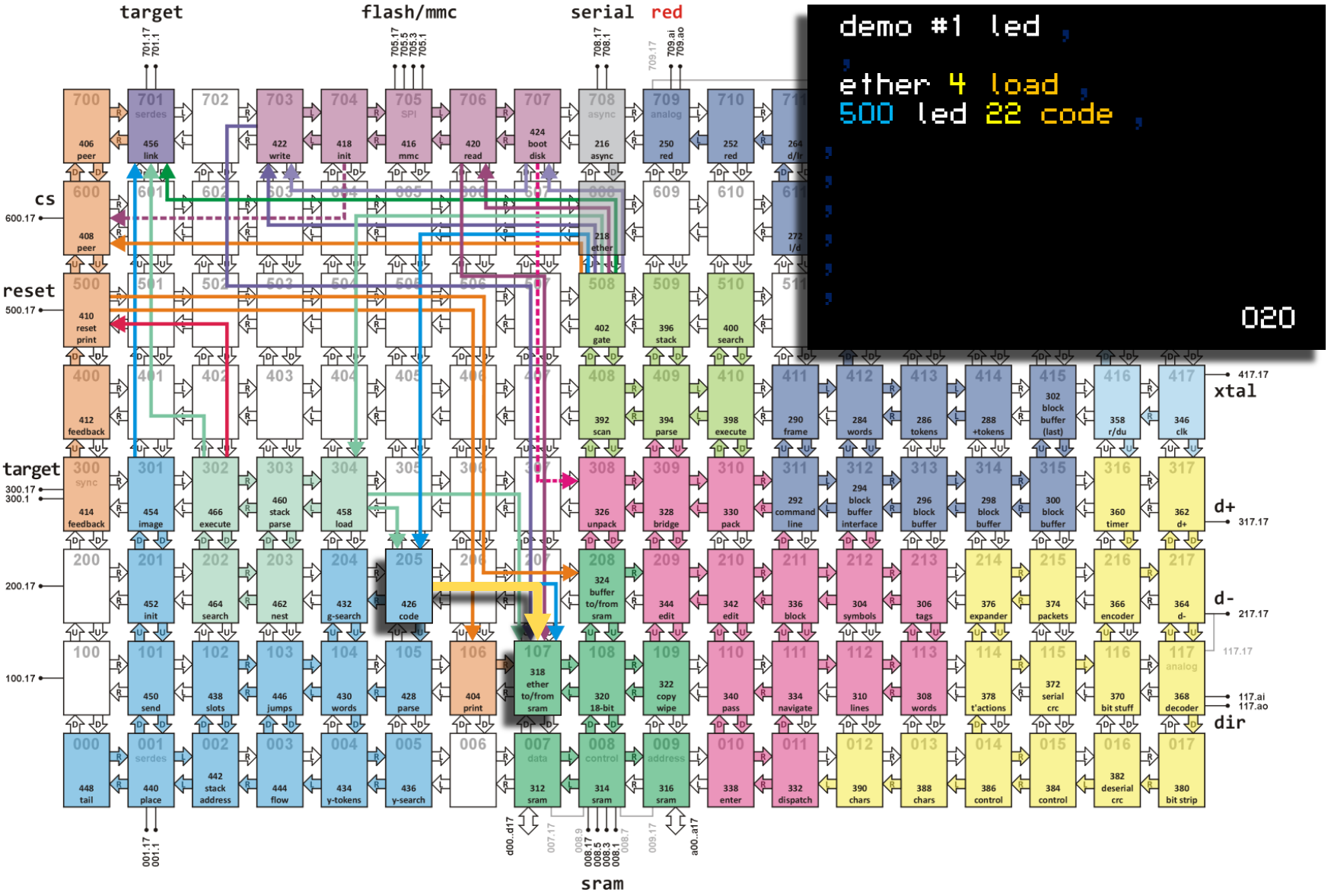
loading applications



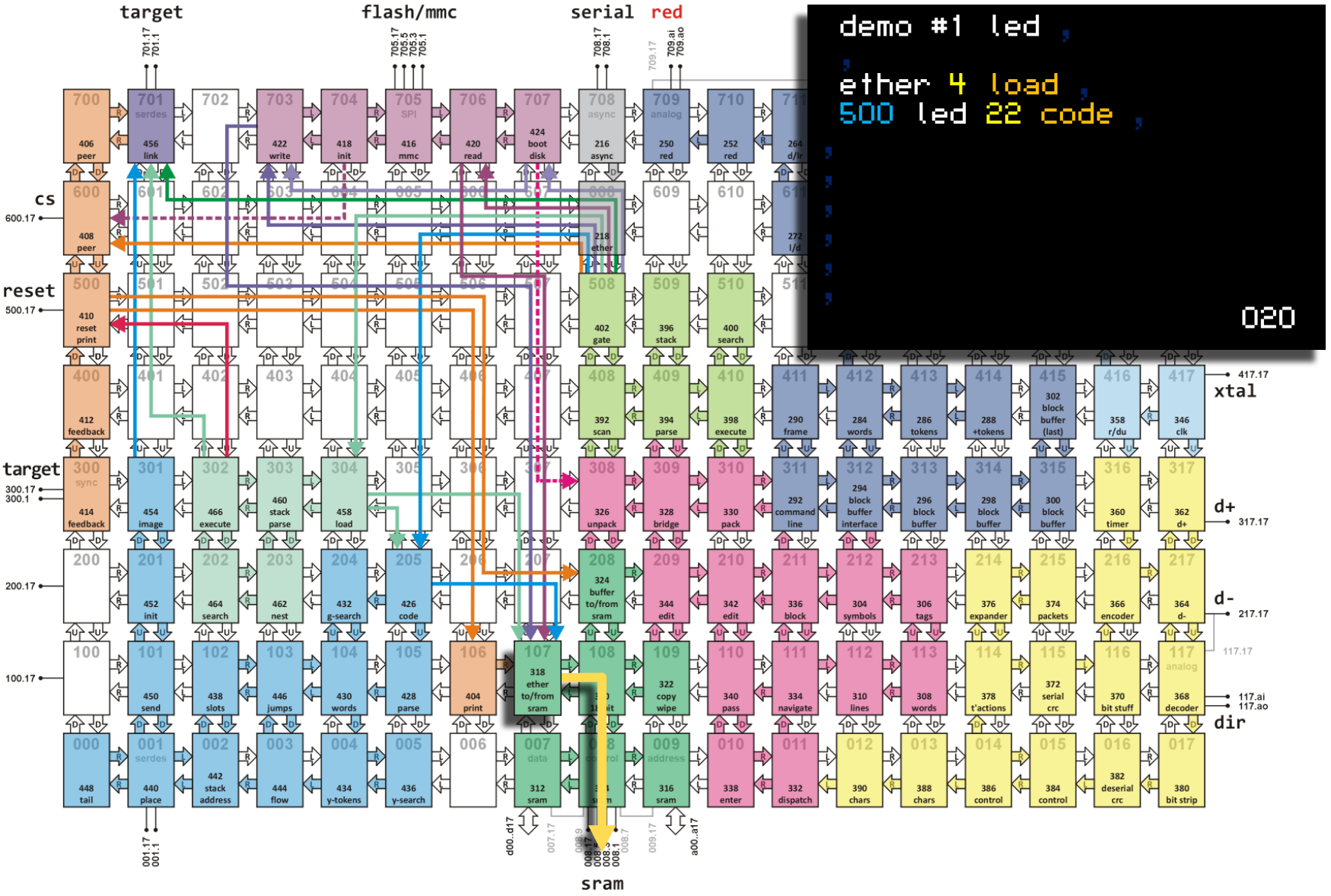
compiling source blocks



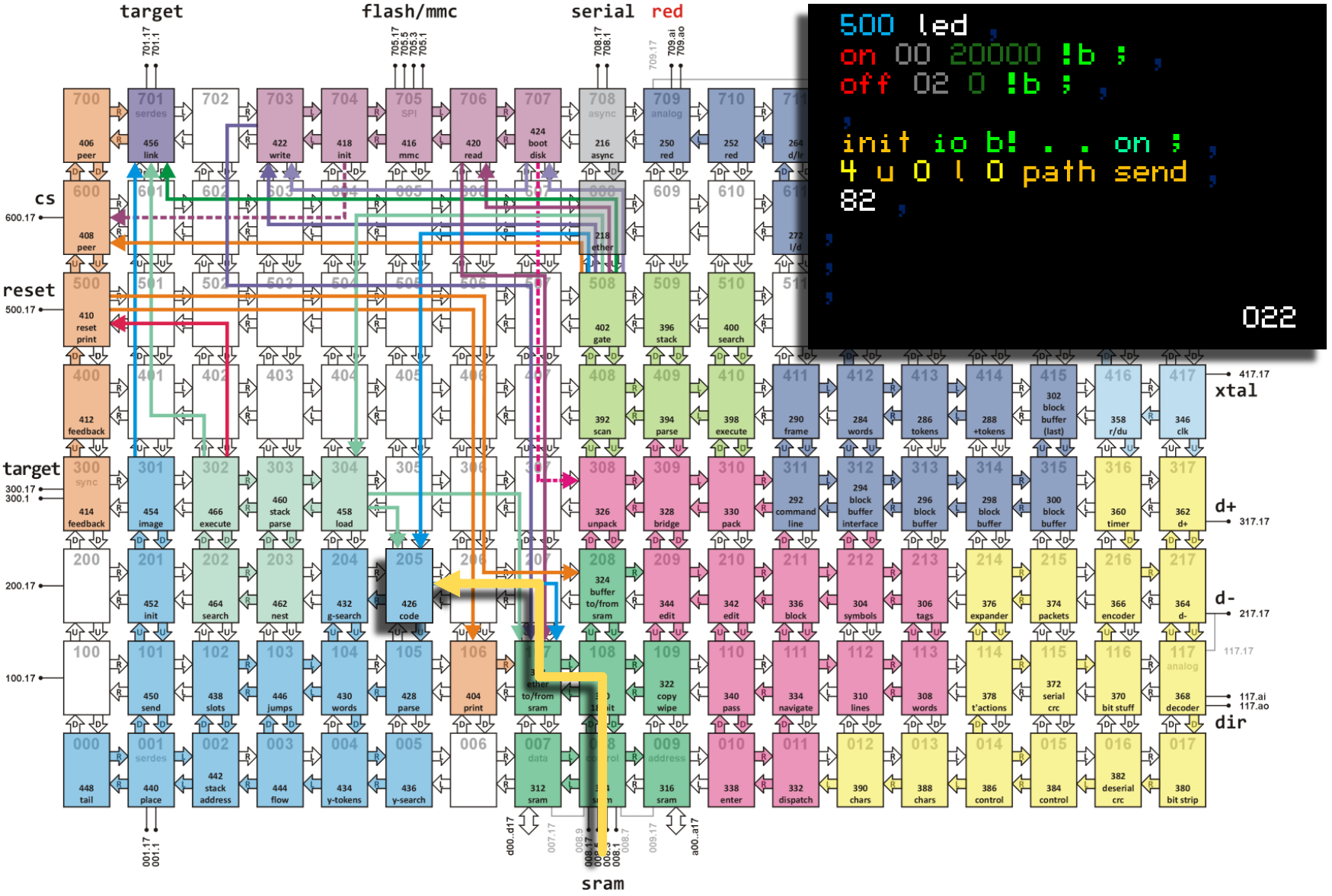
compiling source blocks



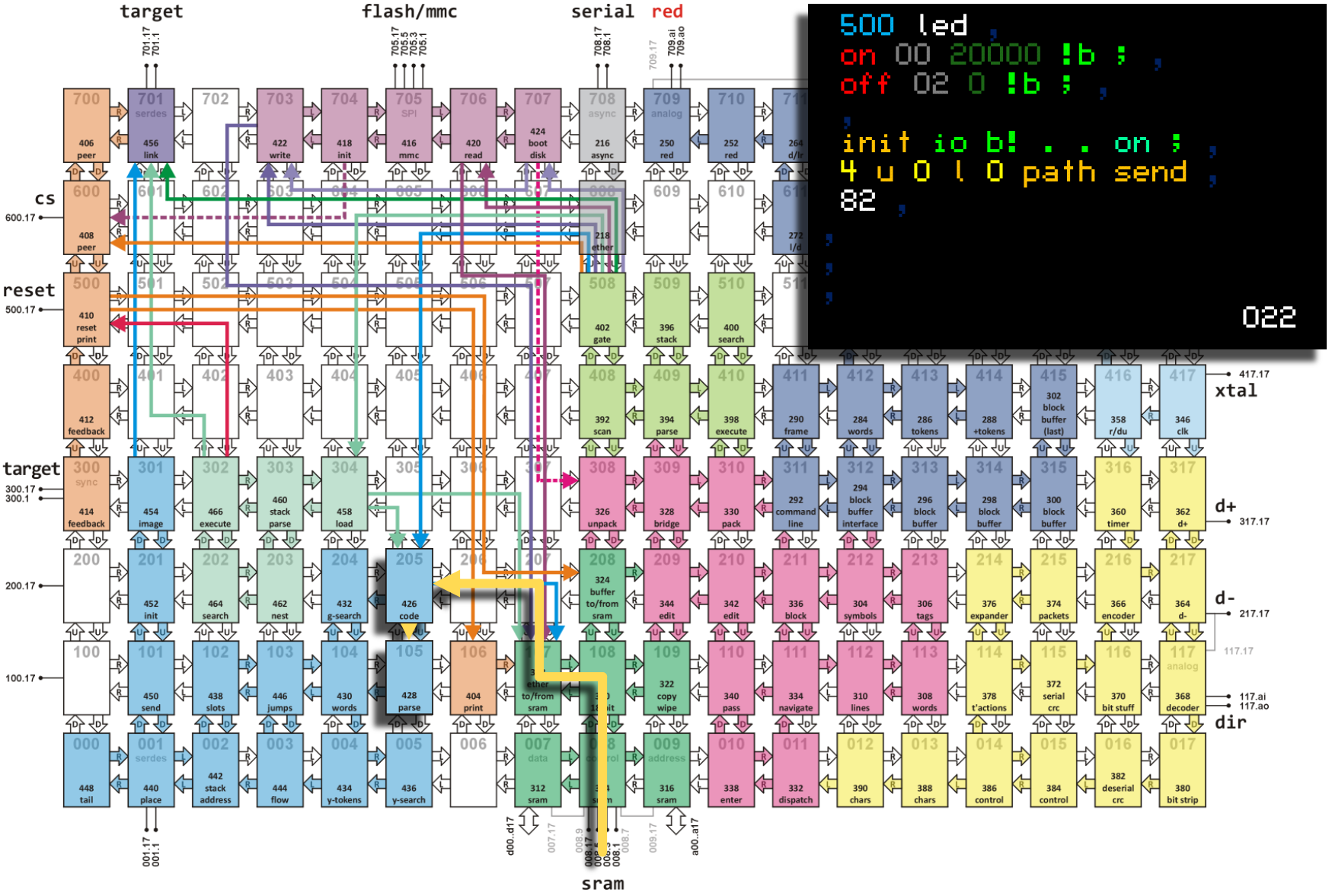
compiling source blocks



compiling source blocks



compiling source blocks

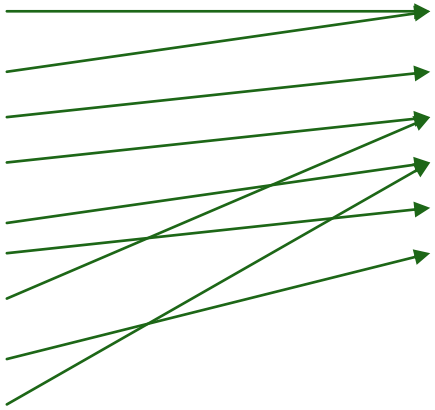


parsing tags

node 105 reduces number of tags

editor

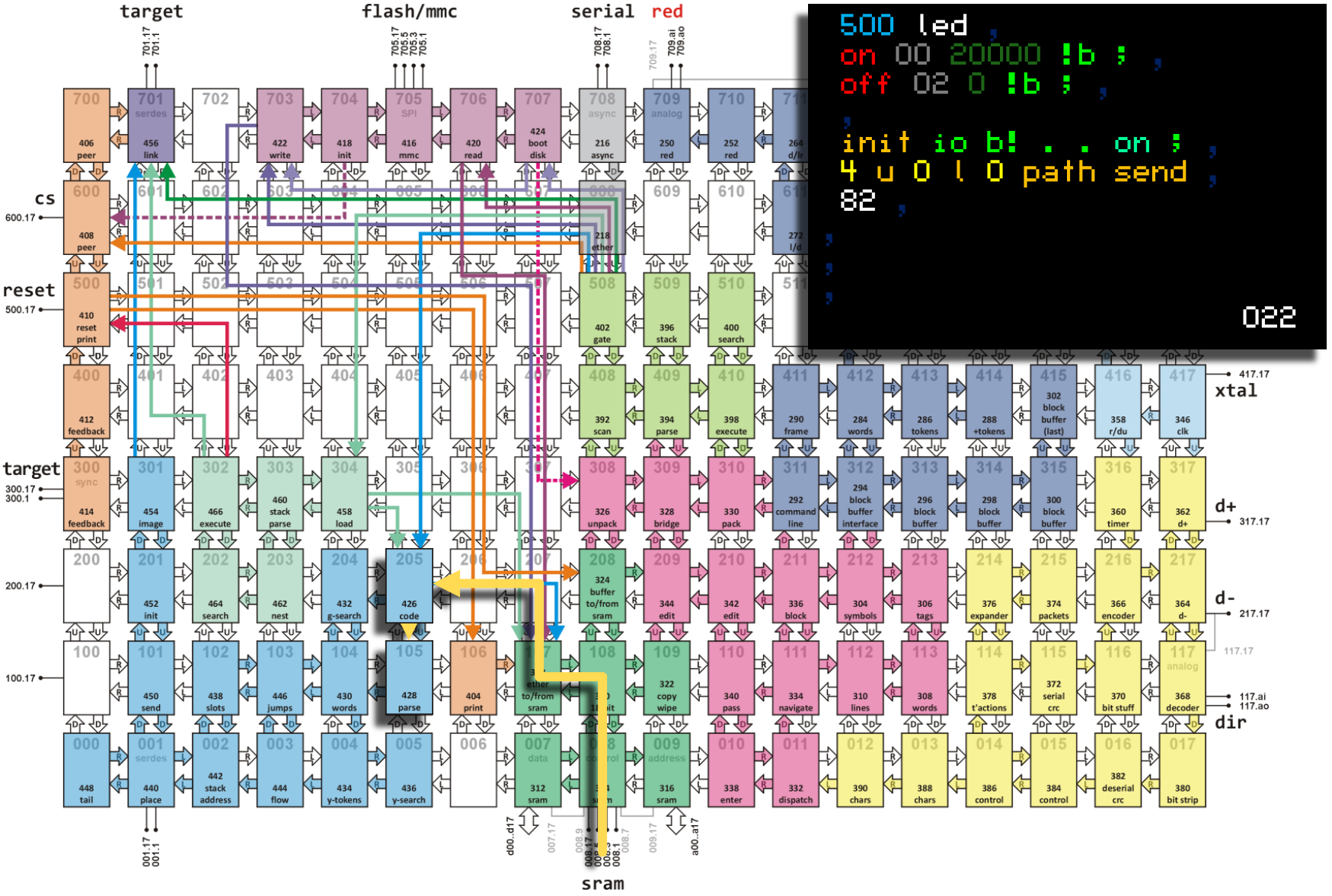
hex	tag	color
30	token	
31	token	
32	char	
33	hex	
34	hex	
35	char	
36	decimal	
37	char	
38	decimal	
39	eol	
3A	space	
3B	cursor	
3C	eob	
3D	char	
3E	char	
3F	char	



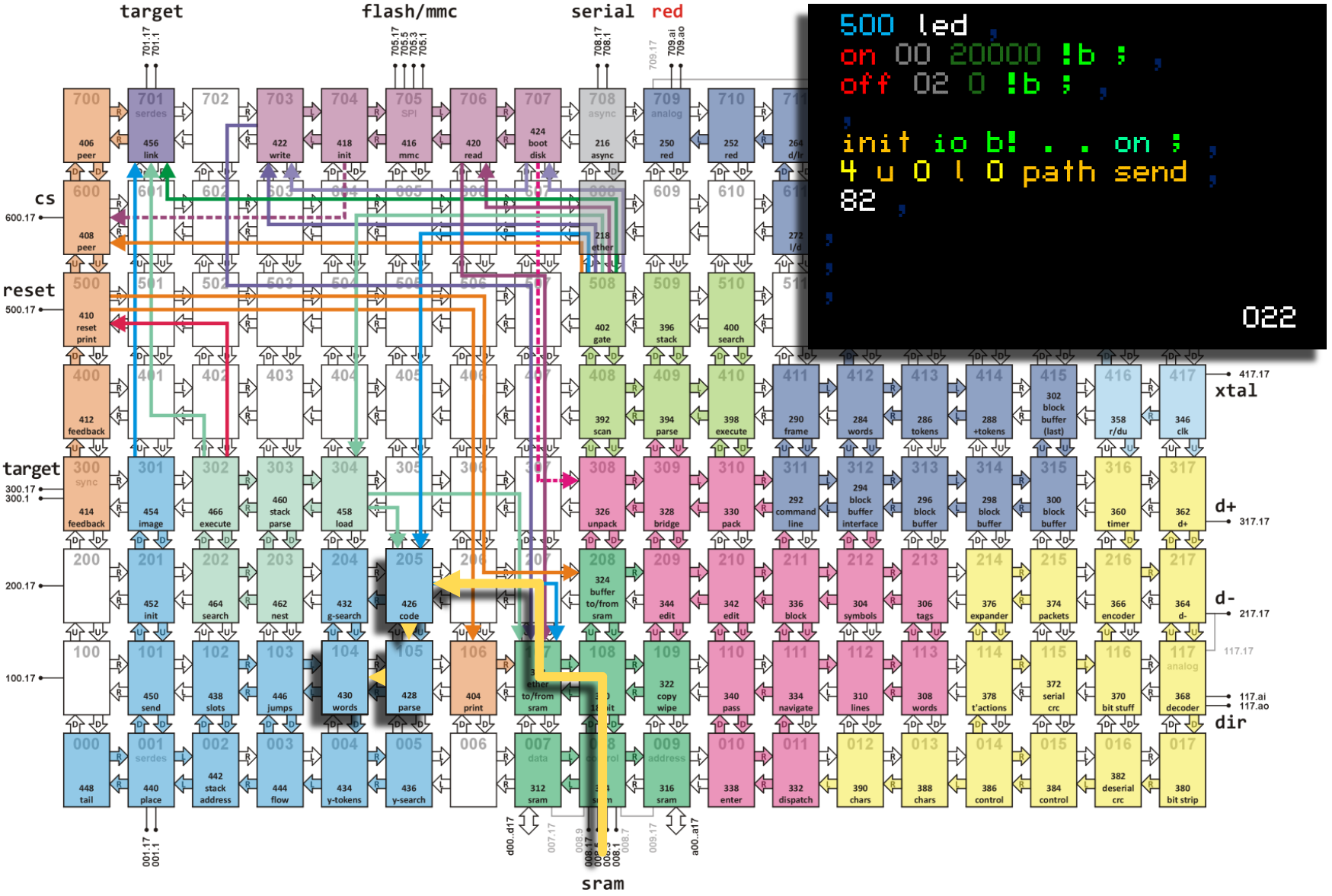
compiler

hex	tag	color
00	token	
02	char	
03	number	
04	number	
05	char	
07	char	

compiling source blocks



compiling source blocks



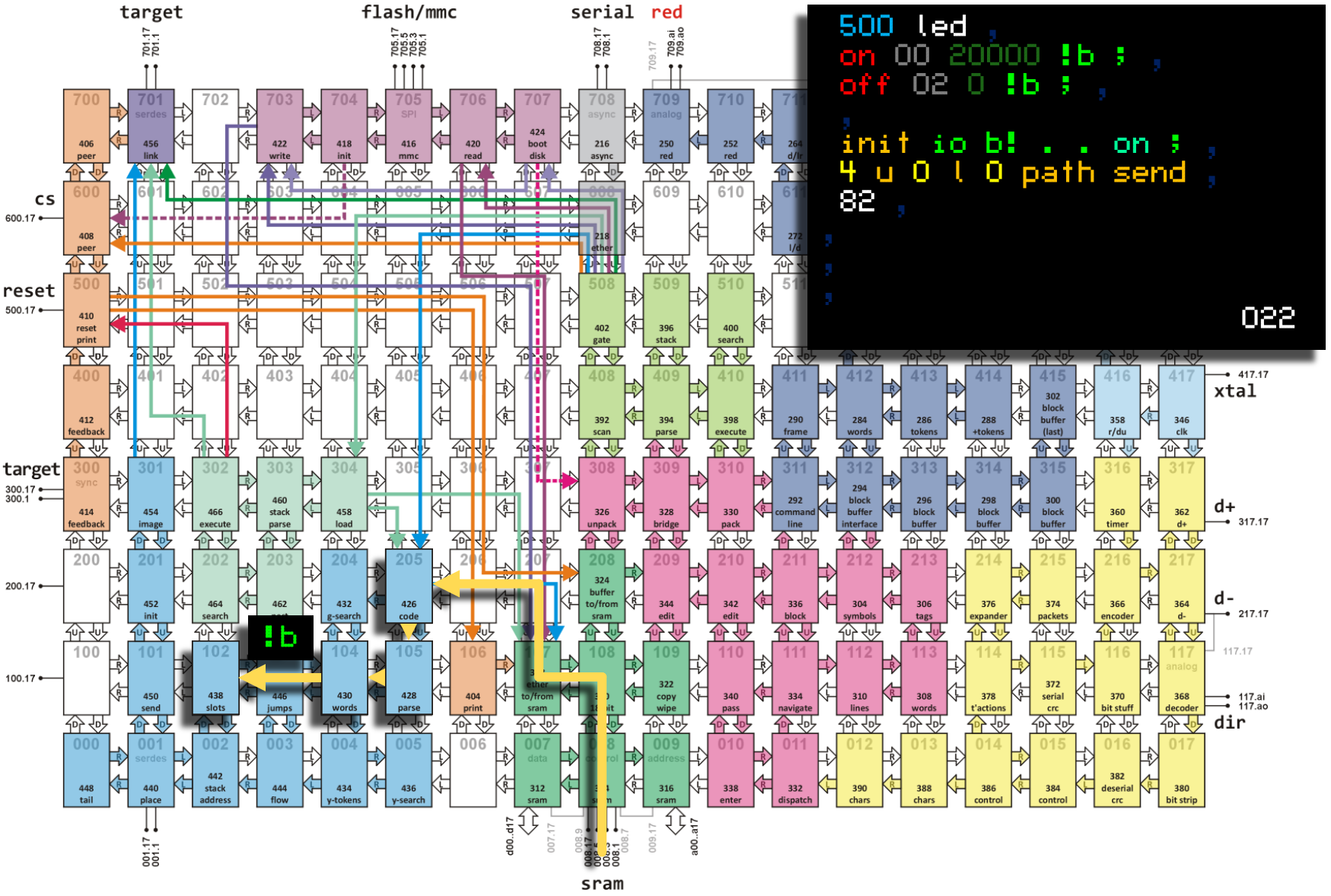
tokens

00, 01, 04, 08 – 1F
02, 03, 05 – 07, 20 – 29
2A – 2F

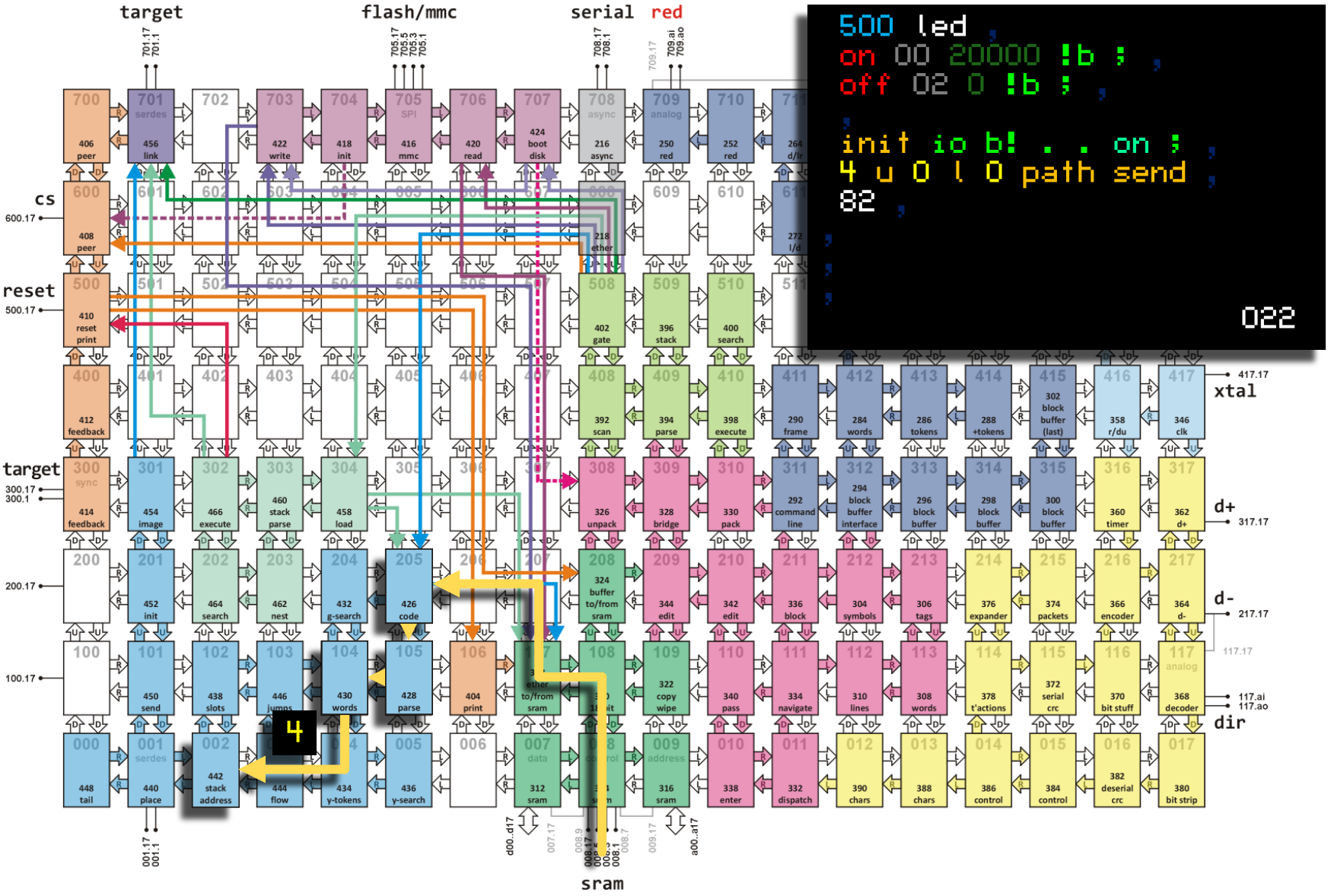
f18 instructions
jumps, loops, flow control
ports and registers

hex	token	hex	token	hex	token
00	;	10	+*	20	then
01	ex	11	2*	21	else
02	begin	12	2/	22	ahead
03	end	13	-	23	leap
04	unext	14	+	24	when
05	next	15	and	25	zif
06	if	16	or	26	till
07	-if	17	drop	27	-till
08	@p	18	dup	28	for
09	@+	19	pop	29	-when
0A	@b	1A	over	2A	left
0B	@	1B	a	2B	right
0C	!p	1C	.	2C	io
0D	!+	1D	push	2D	down
0E	!b	1E	b!	2E	up
0F	!	1F	a!	2F	data

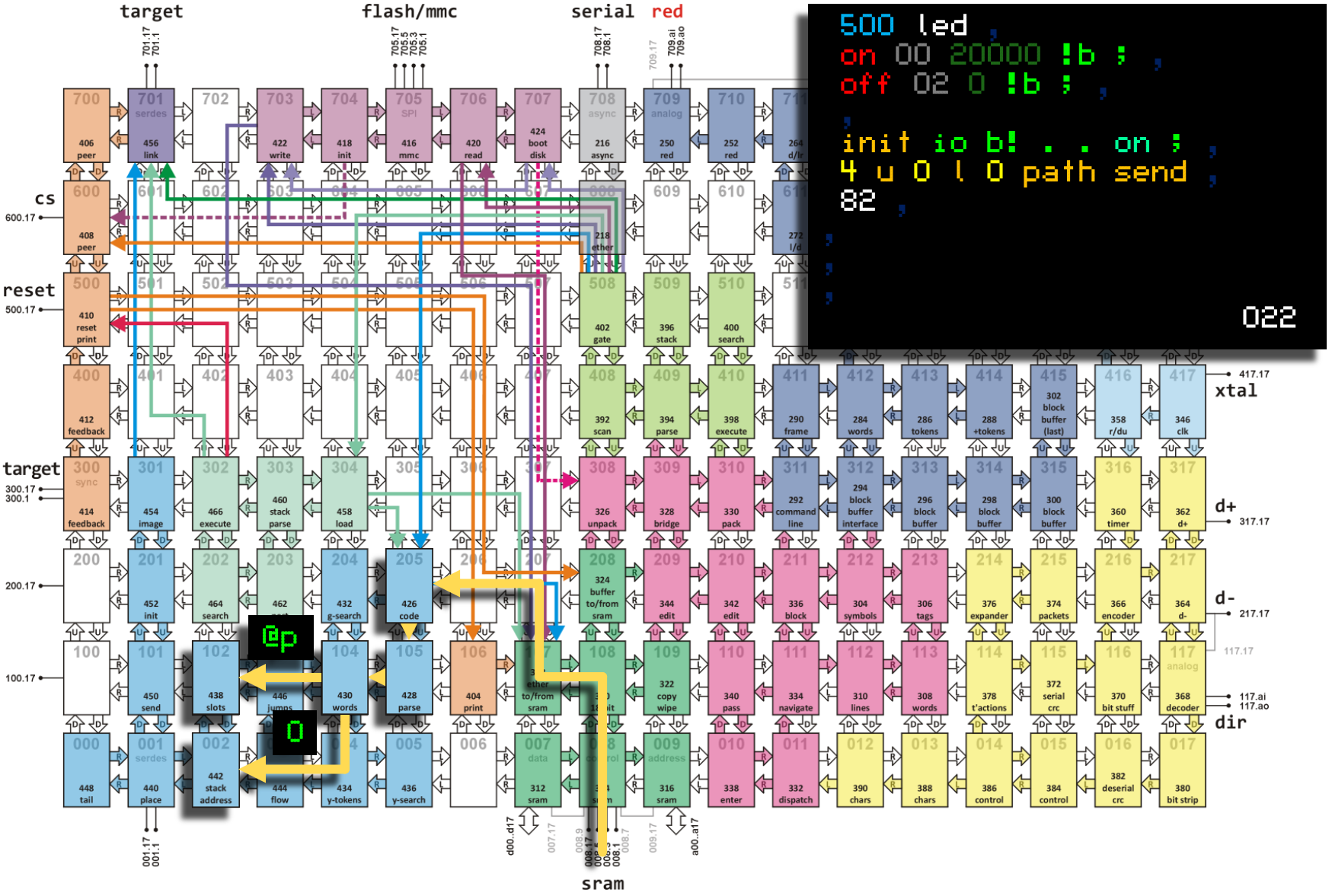
compiling source blocks



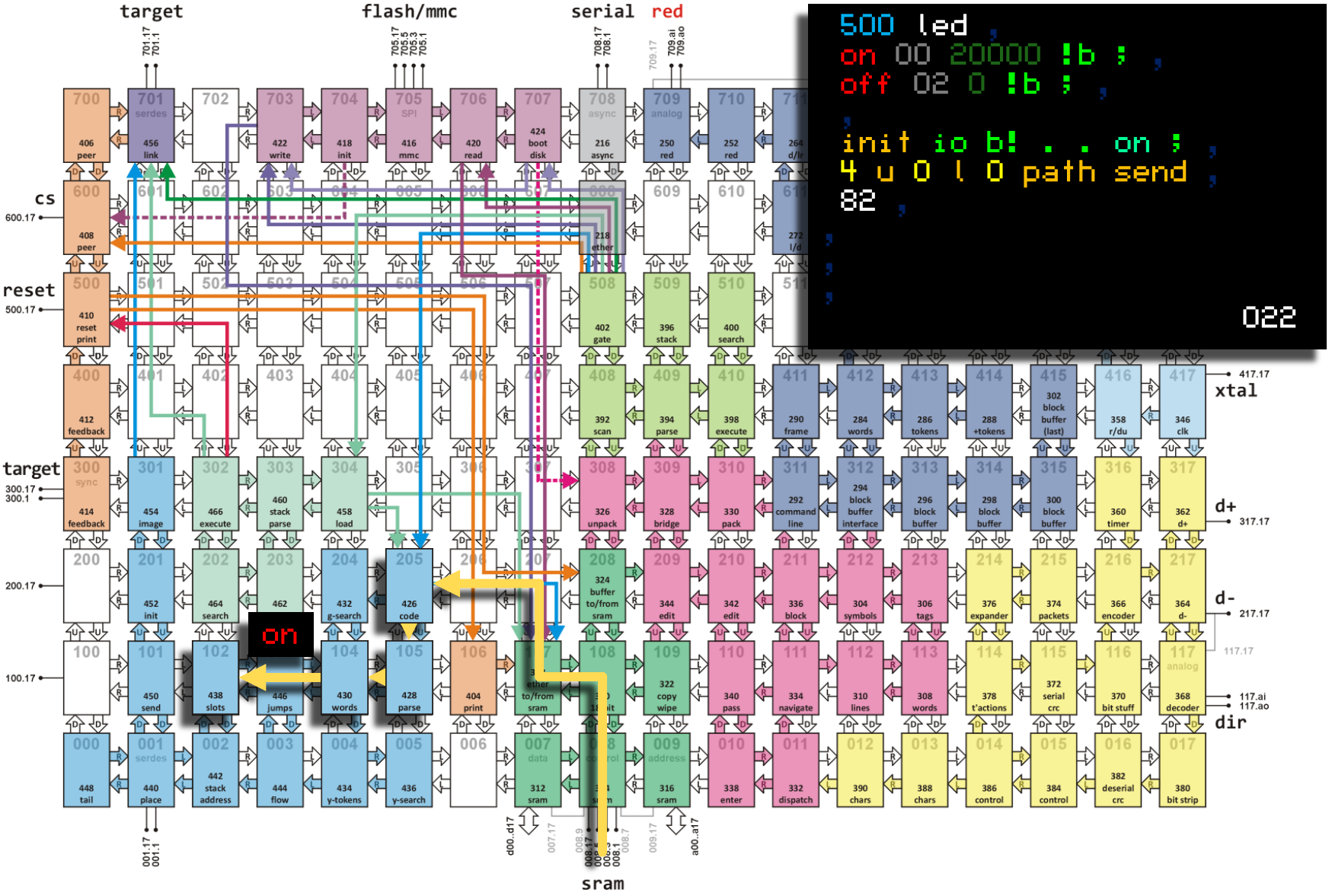
compiling source blocks



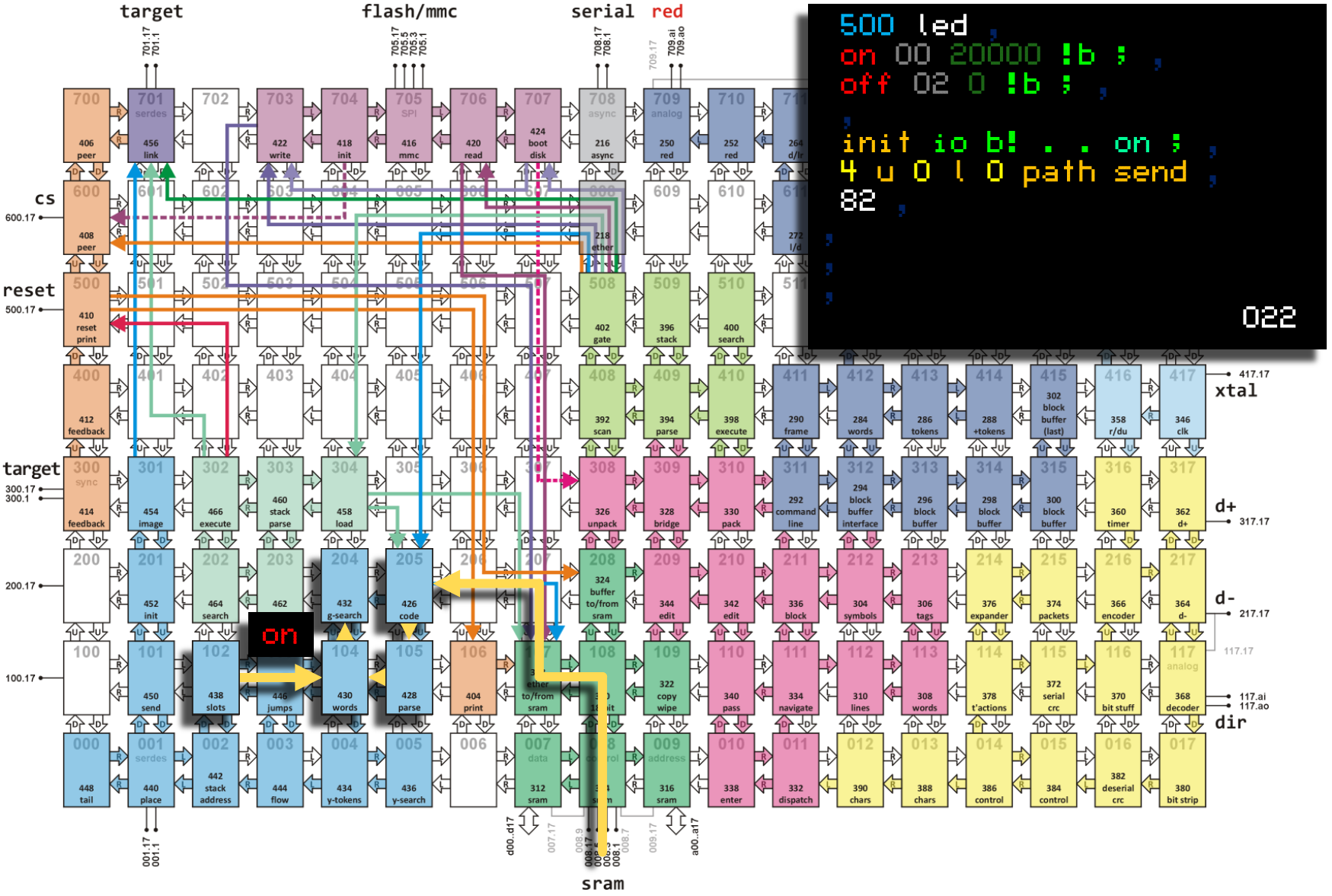
compiling source blocks



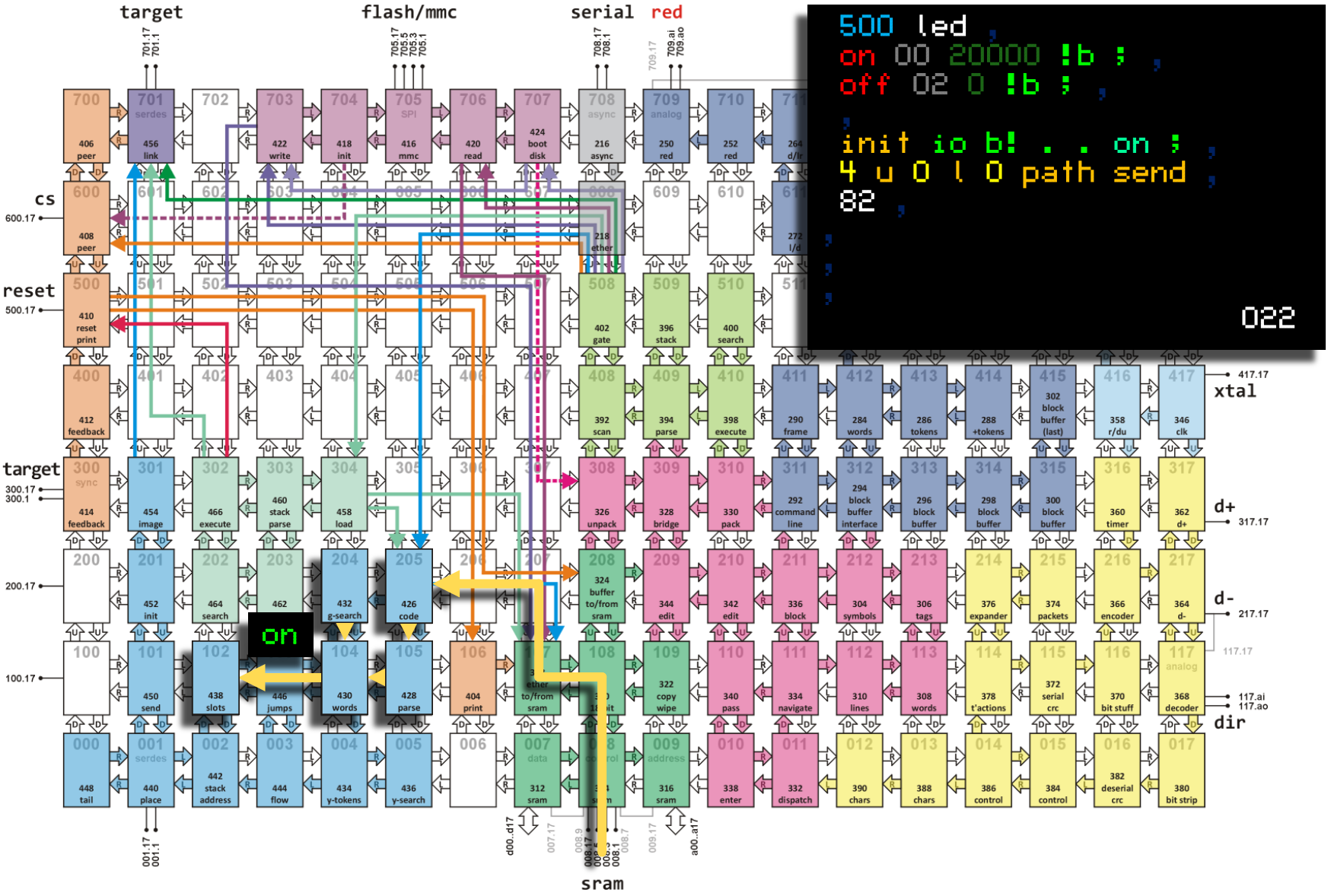
compiling source blocks



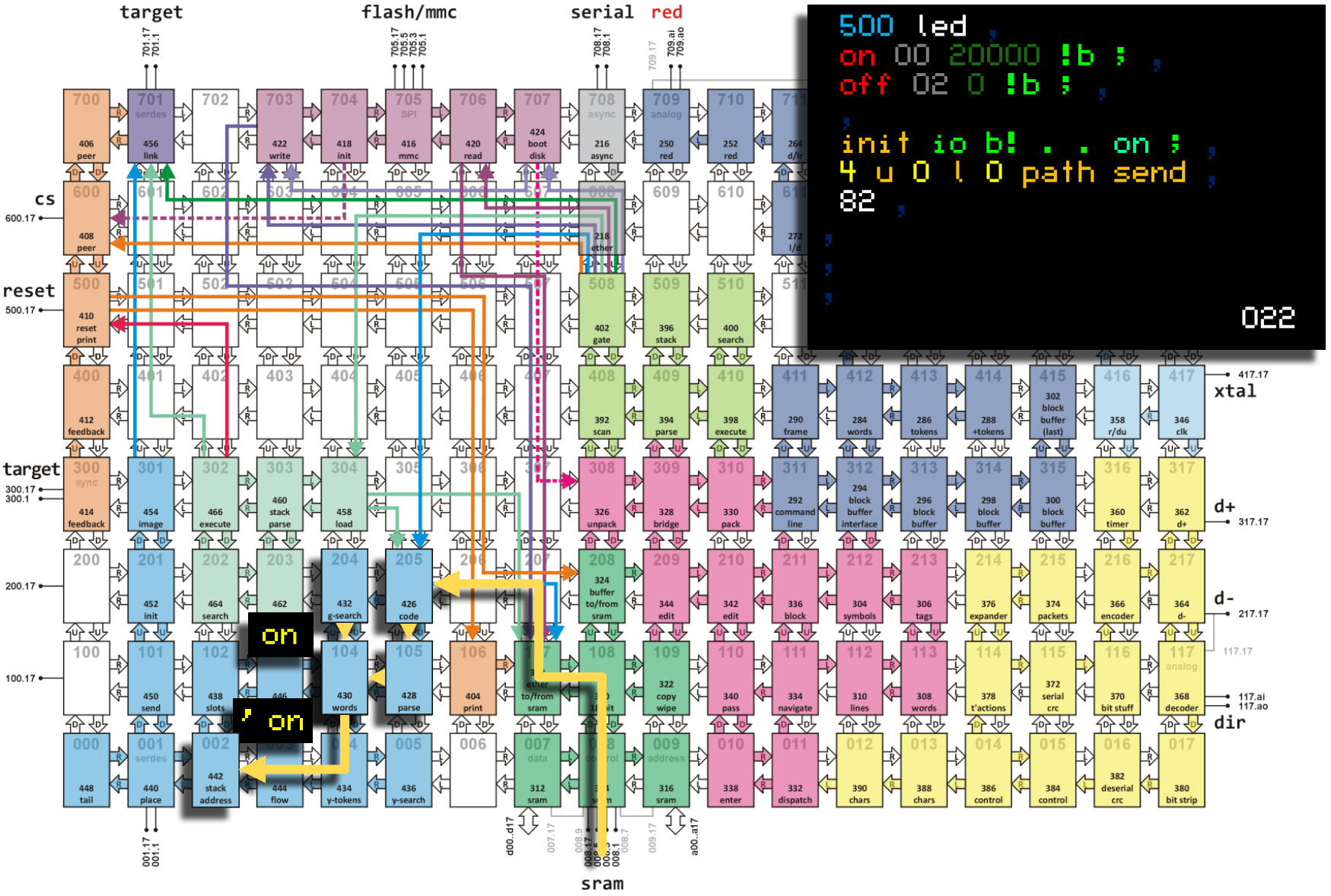
compiling source blocks



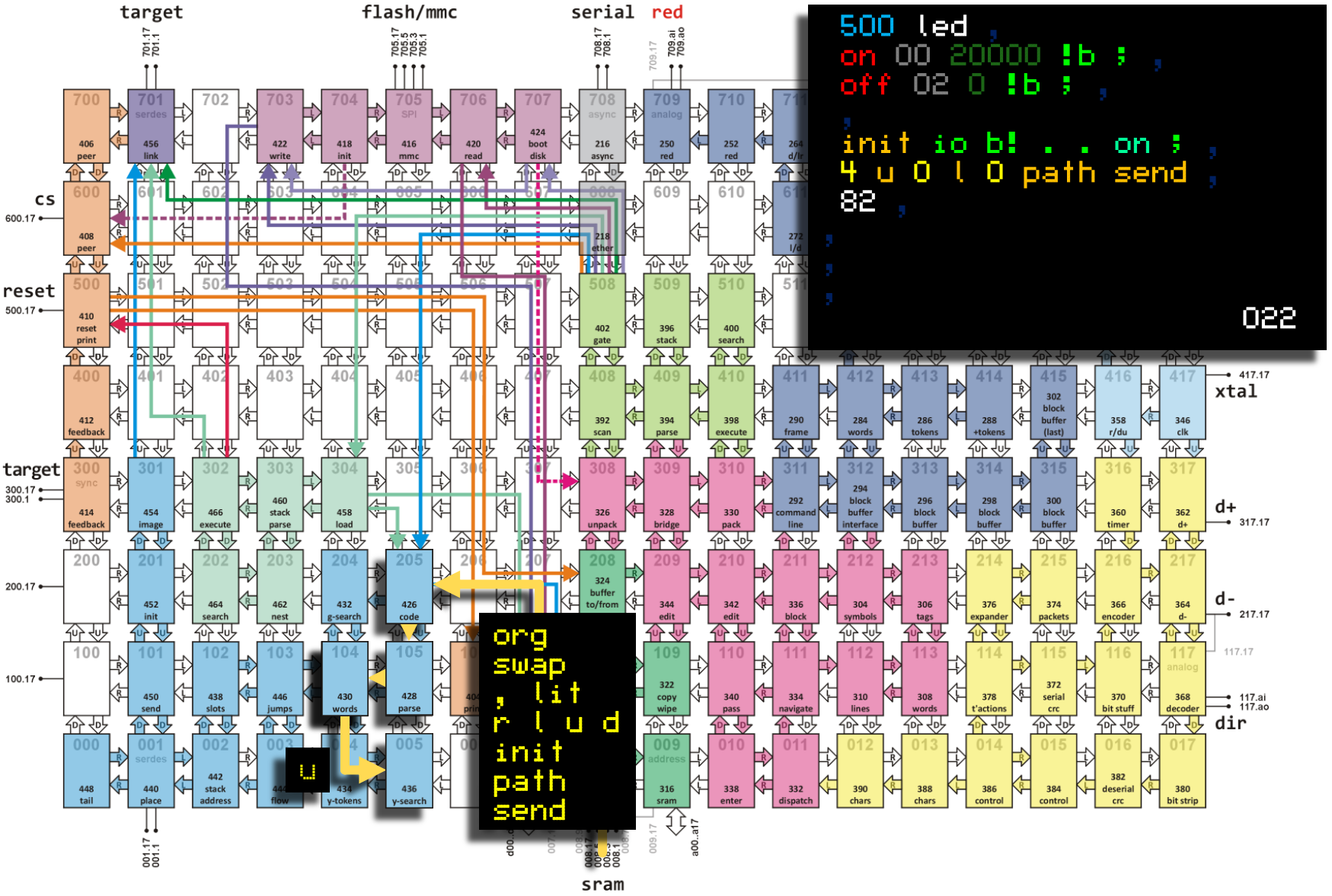
compiling source blocks



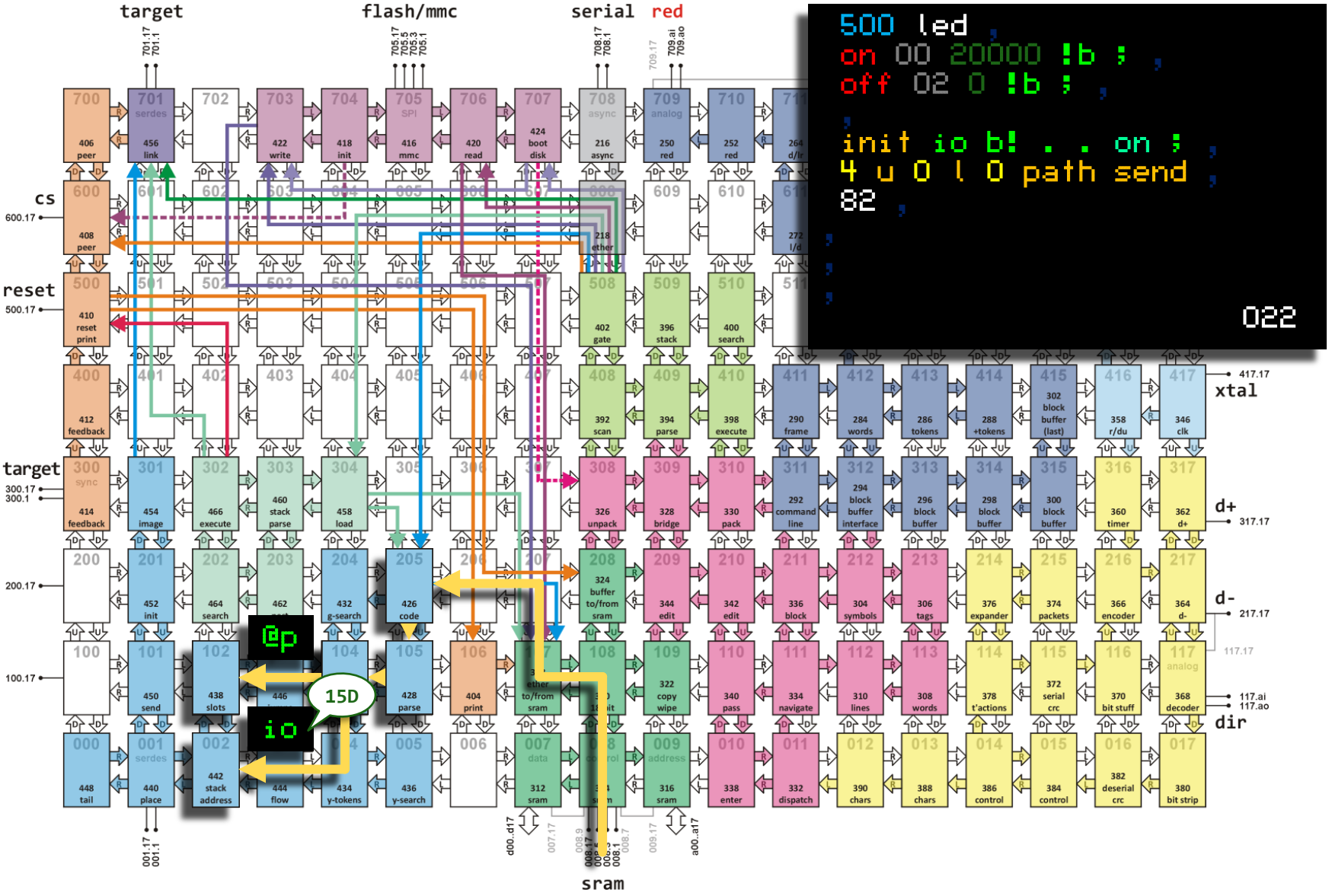
compiling source blocks



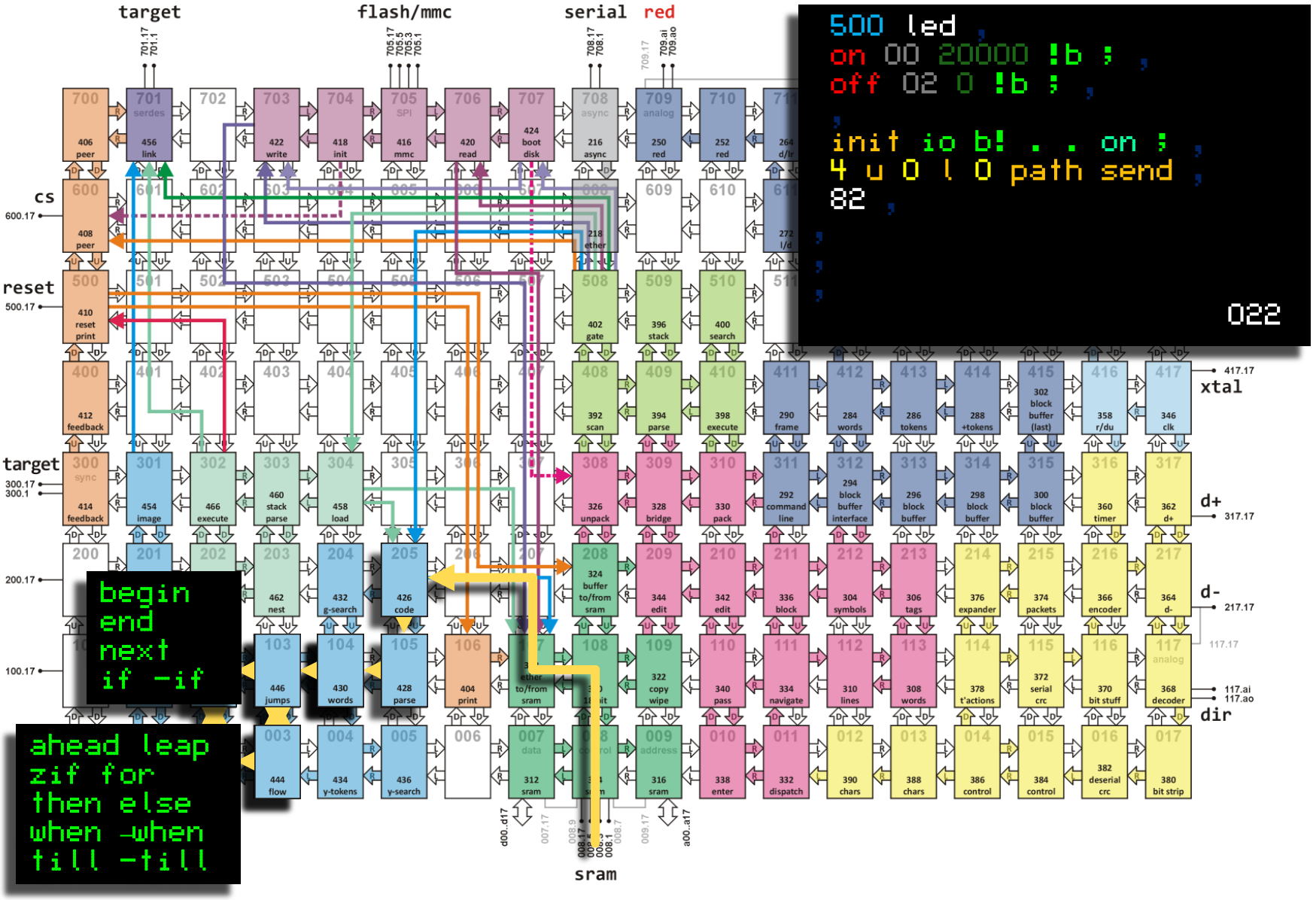
compiling source blocks



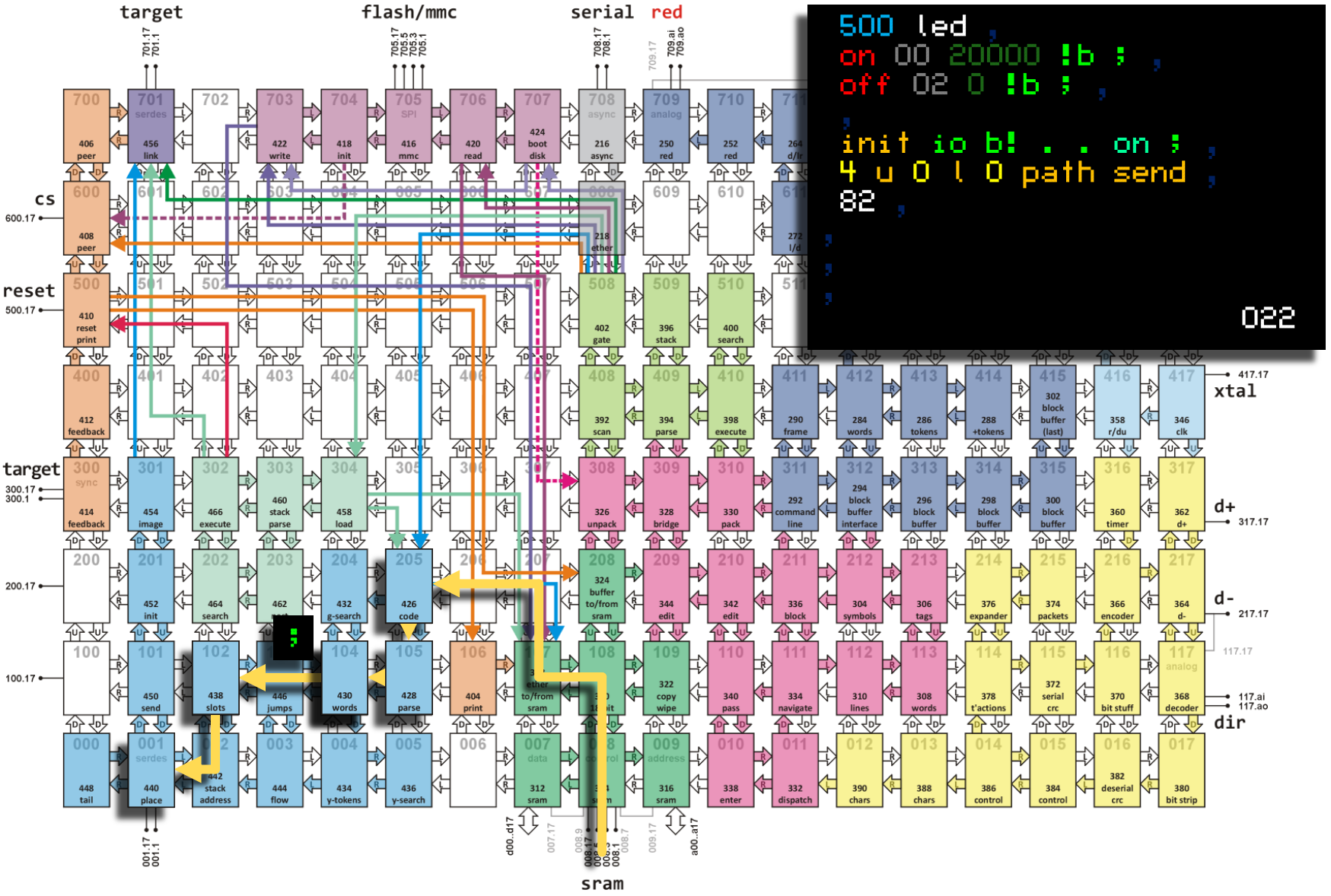
compiling source blocks



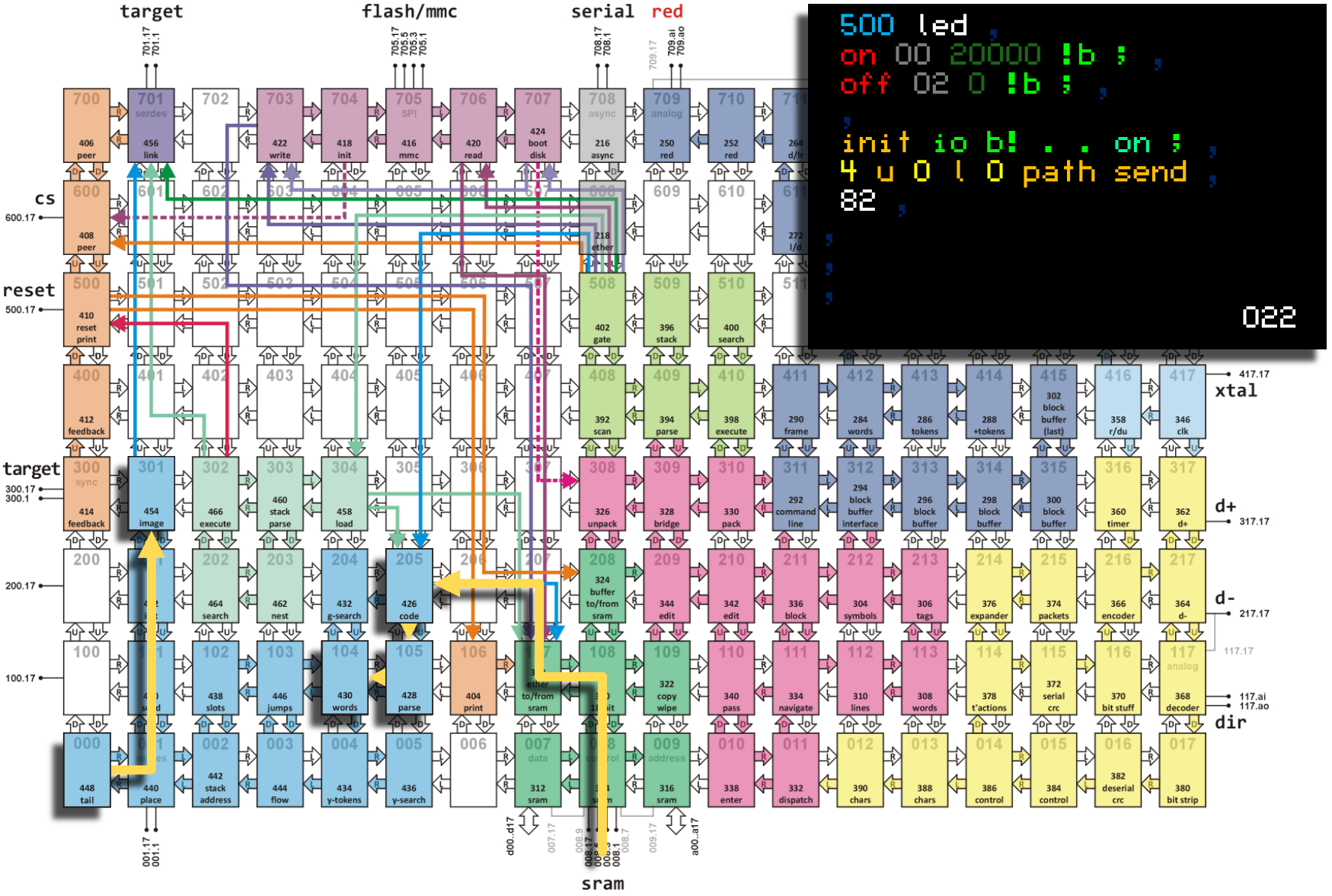
compiling source blocks



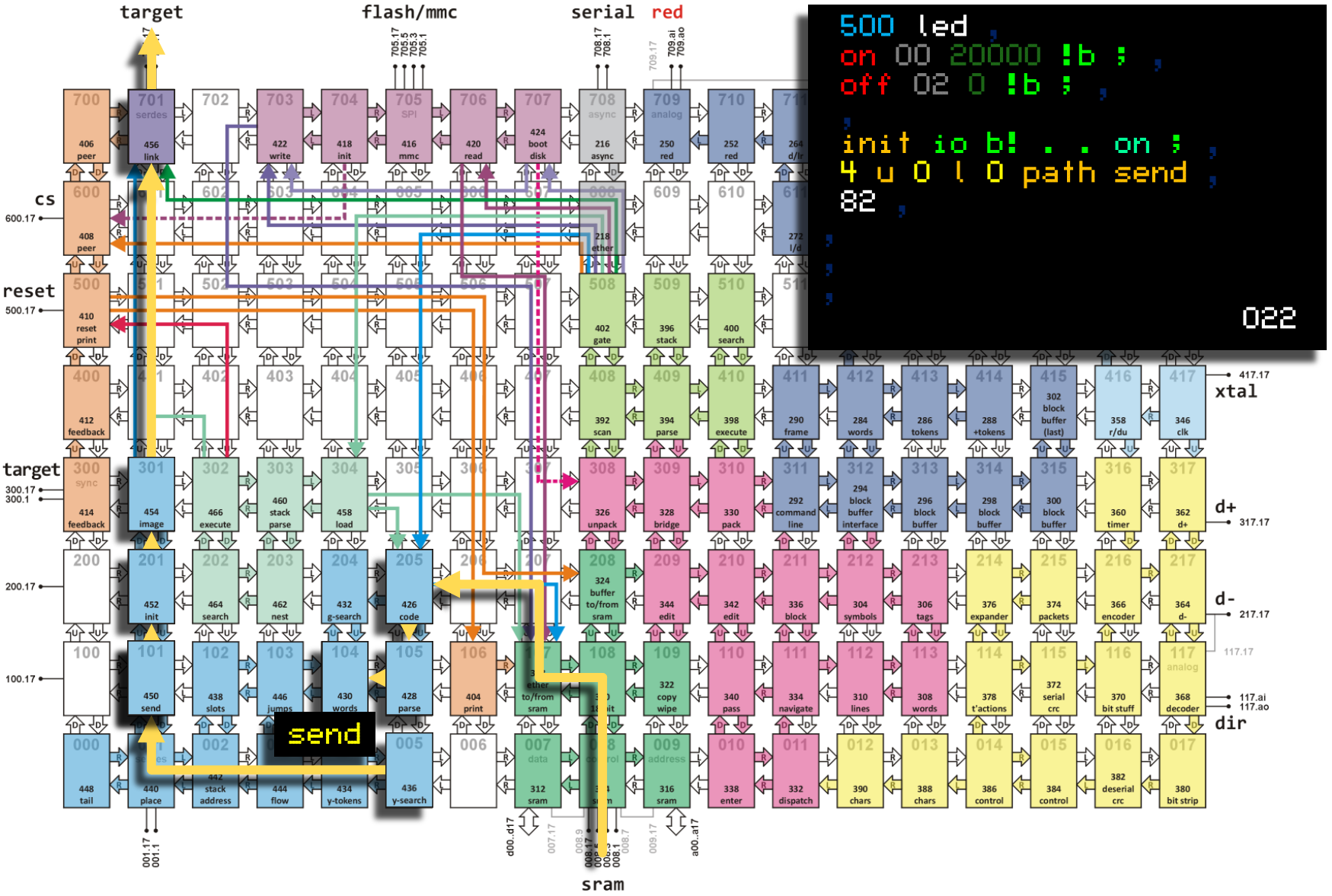
compiling source blocks



compiling source blocks



compiling source blocks



delivering object code

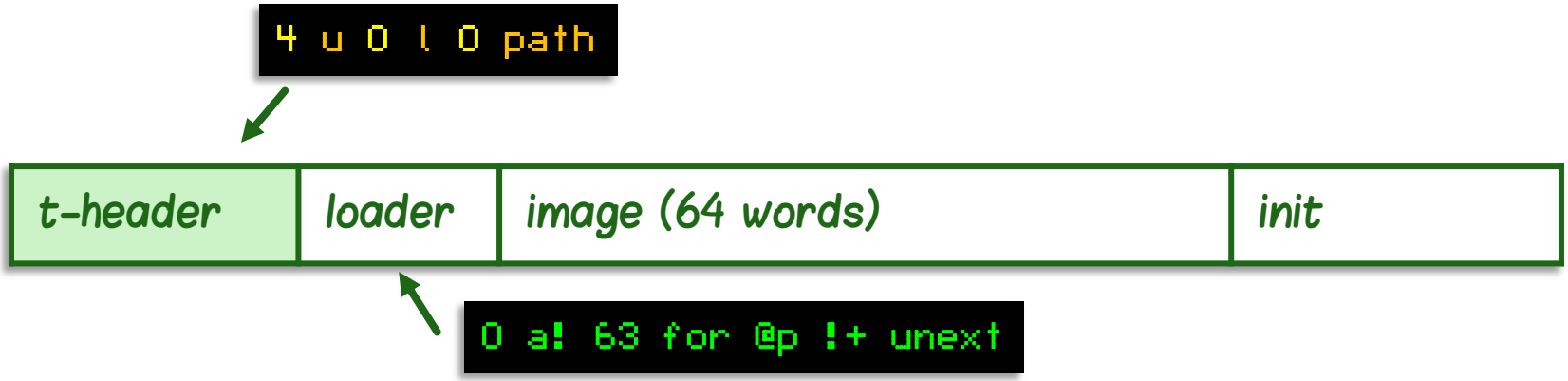
after compilation

- *deliver object code to the link node 701*
- *send over SERDES line to the link node 001*
- *deliver object code to the target node port*
- *load image into the target node RAM*
- *initialize the target node*

delivering object code

after compilation

- *deliver object code to the link node 701*
- *send over SERDES line to the link node 001*
- *deliver object code to the target node port*
- *load image into the target node RAM*
- *initialize the target node*



delivering object code

after compilation

- deliver object code to the link node 701
- send over SERDES line to the link node 001
- deliver object code to the target node port
- load image into the target node RAM
- initialize the target node

delivering object code

after compilation

- deliver object code to the link node 701
- send over SERDES line to the link node 001
- deliver object code to the target node port
- load image into the target node RAM
- initialize the target node

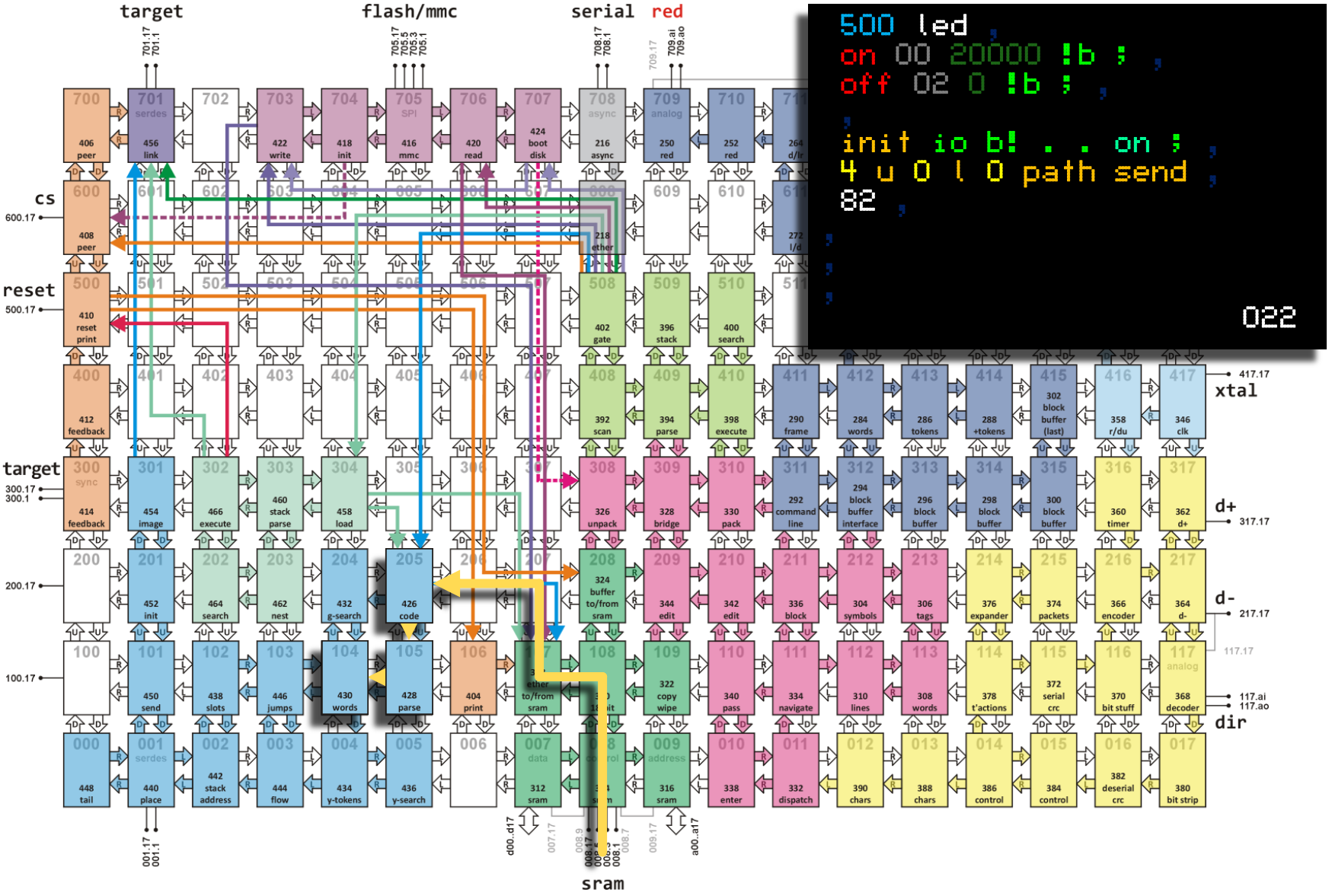
delivering object code

after compilation

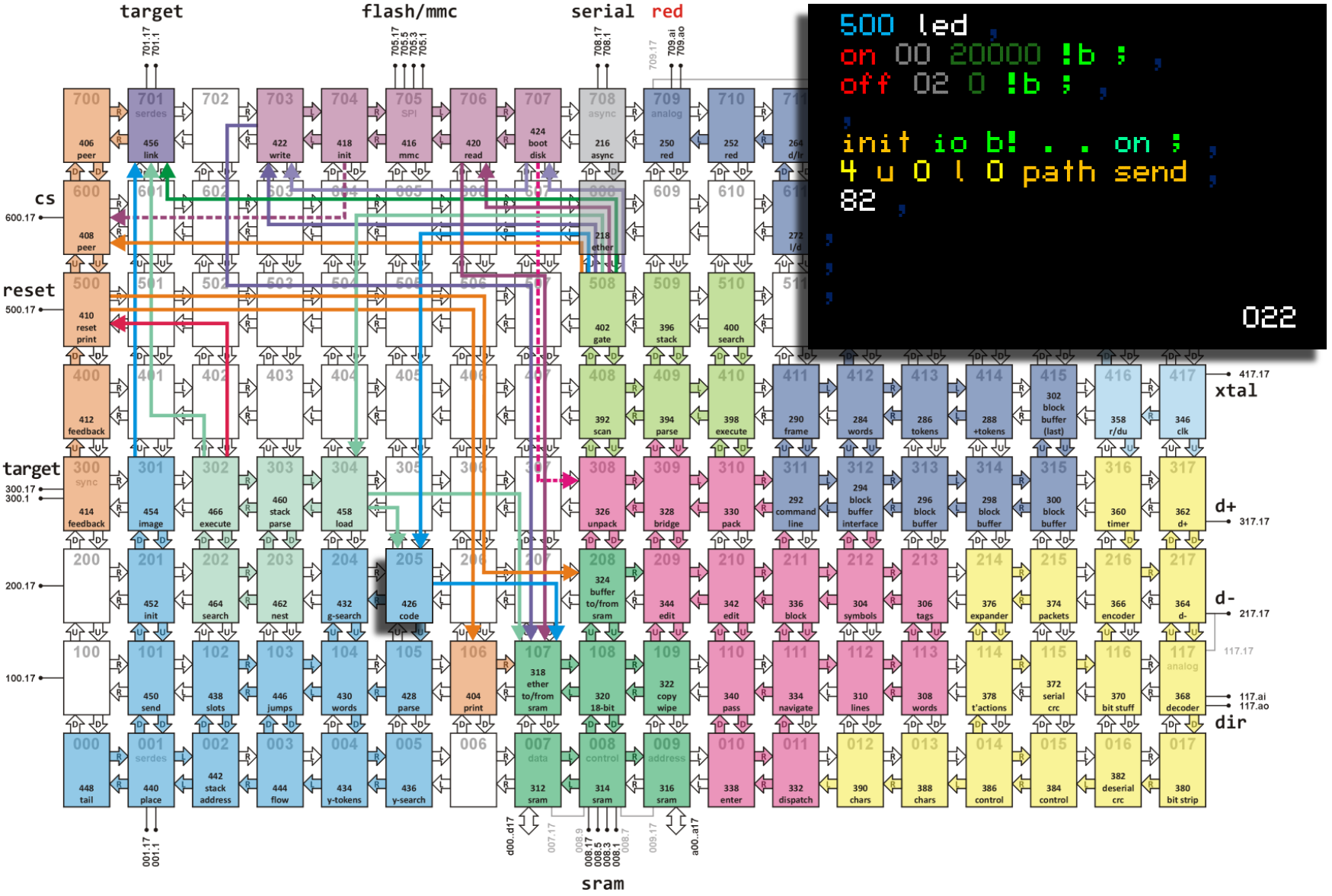
- deliver object code to the link node 701
- send over SERDES line to the link node 001
- deliver object code to the target node port
- load image into the target node RAM
- initialize the target node

init

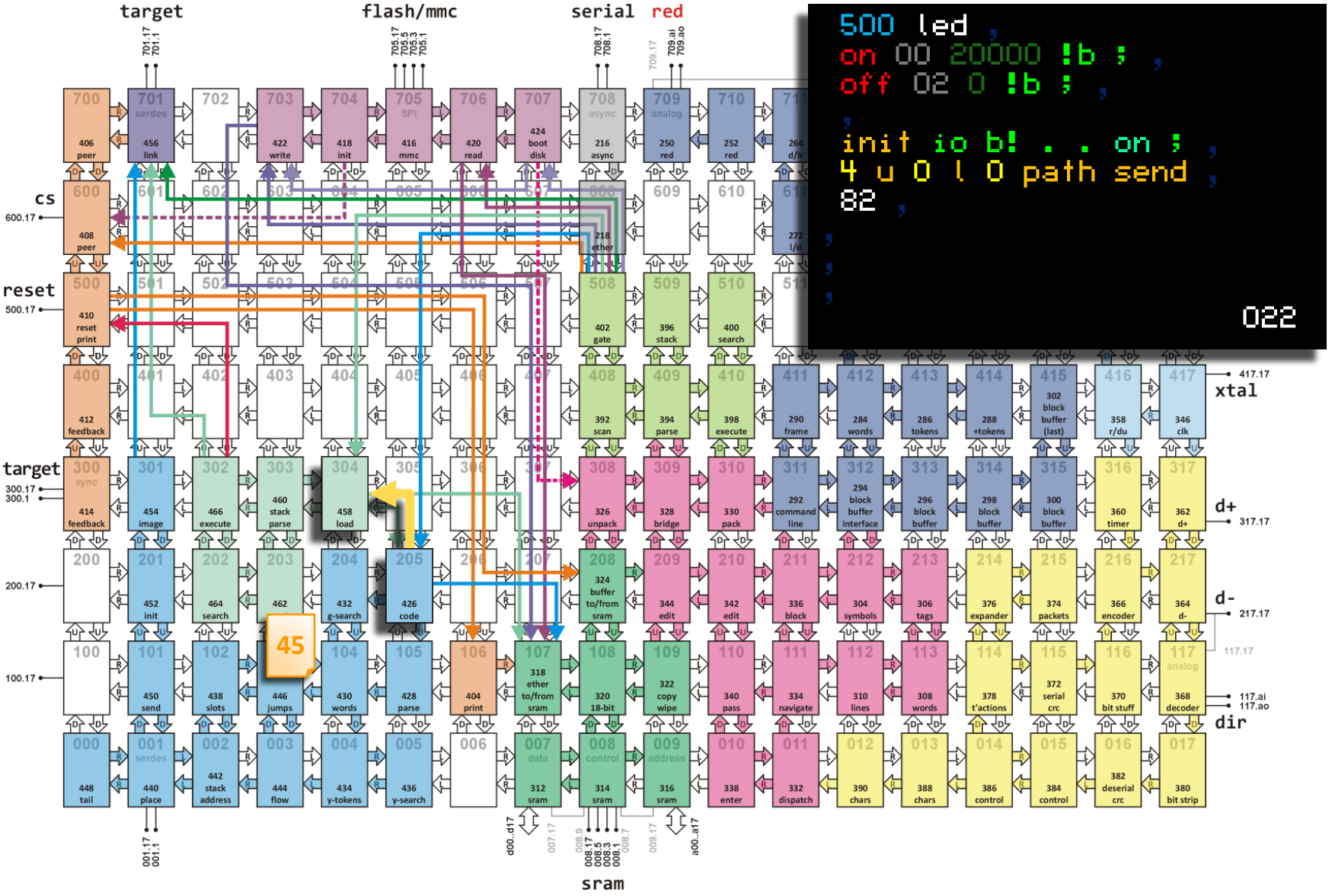
compiling source blocks



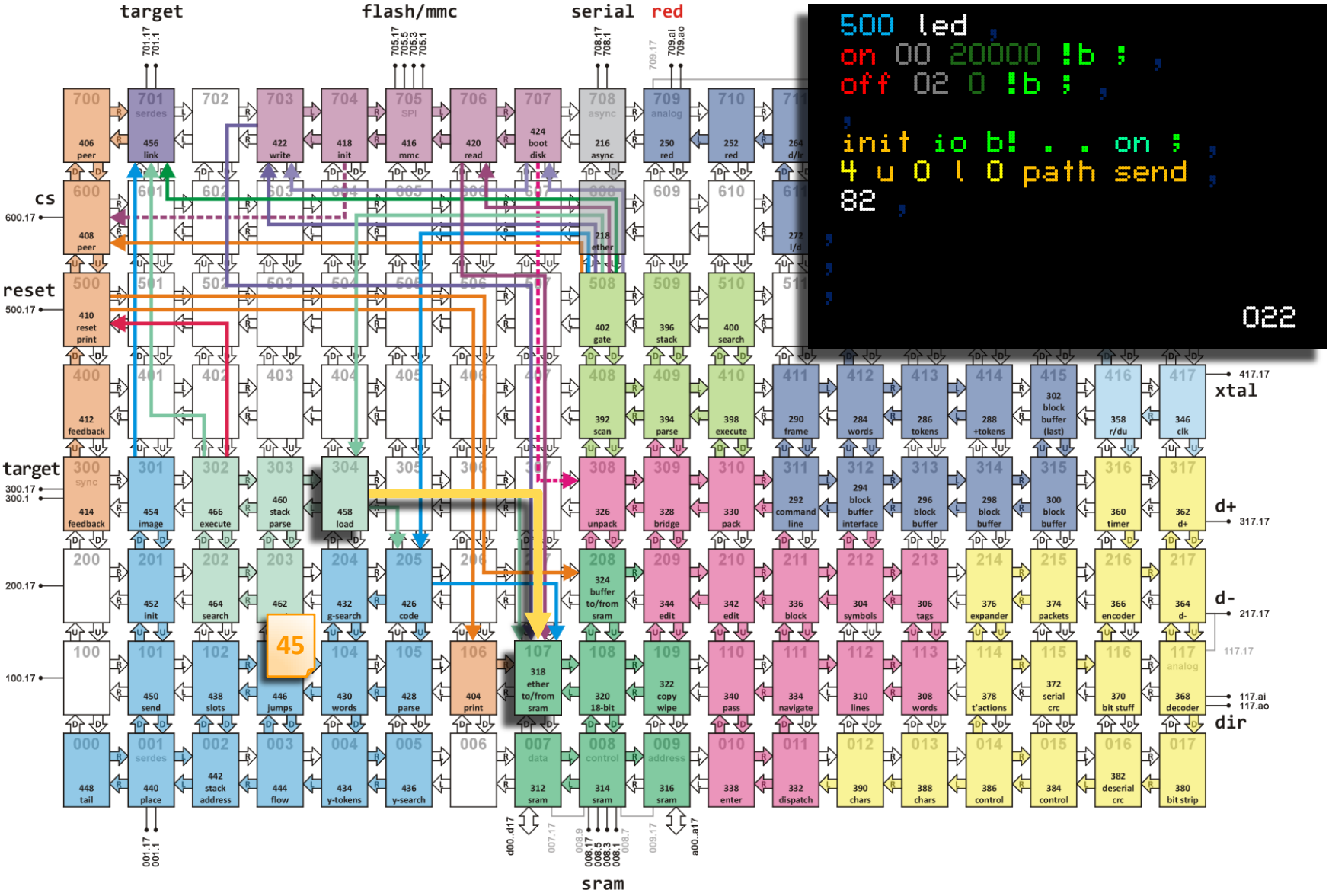
compiling source blocks



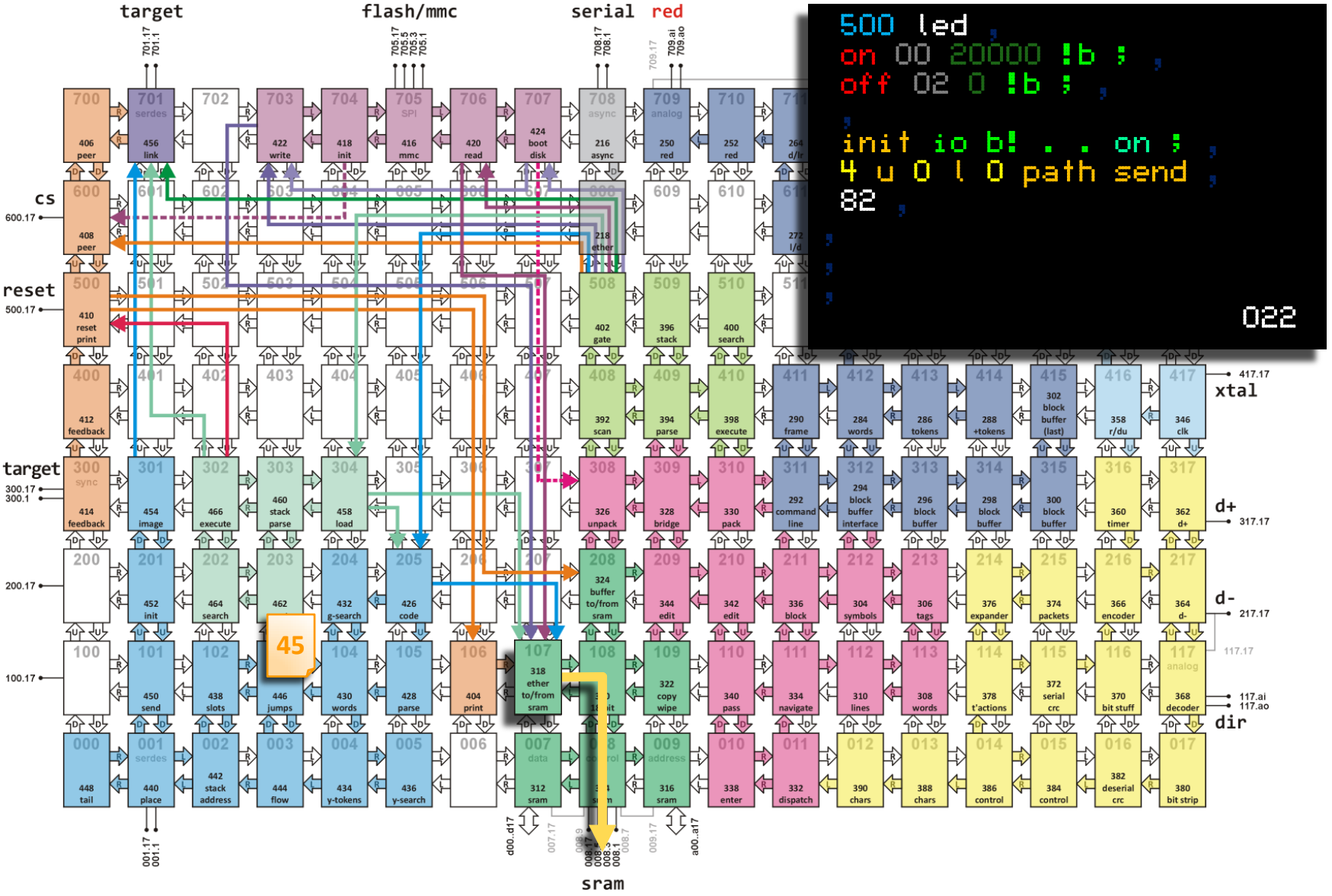
loading applications



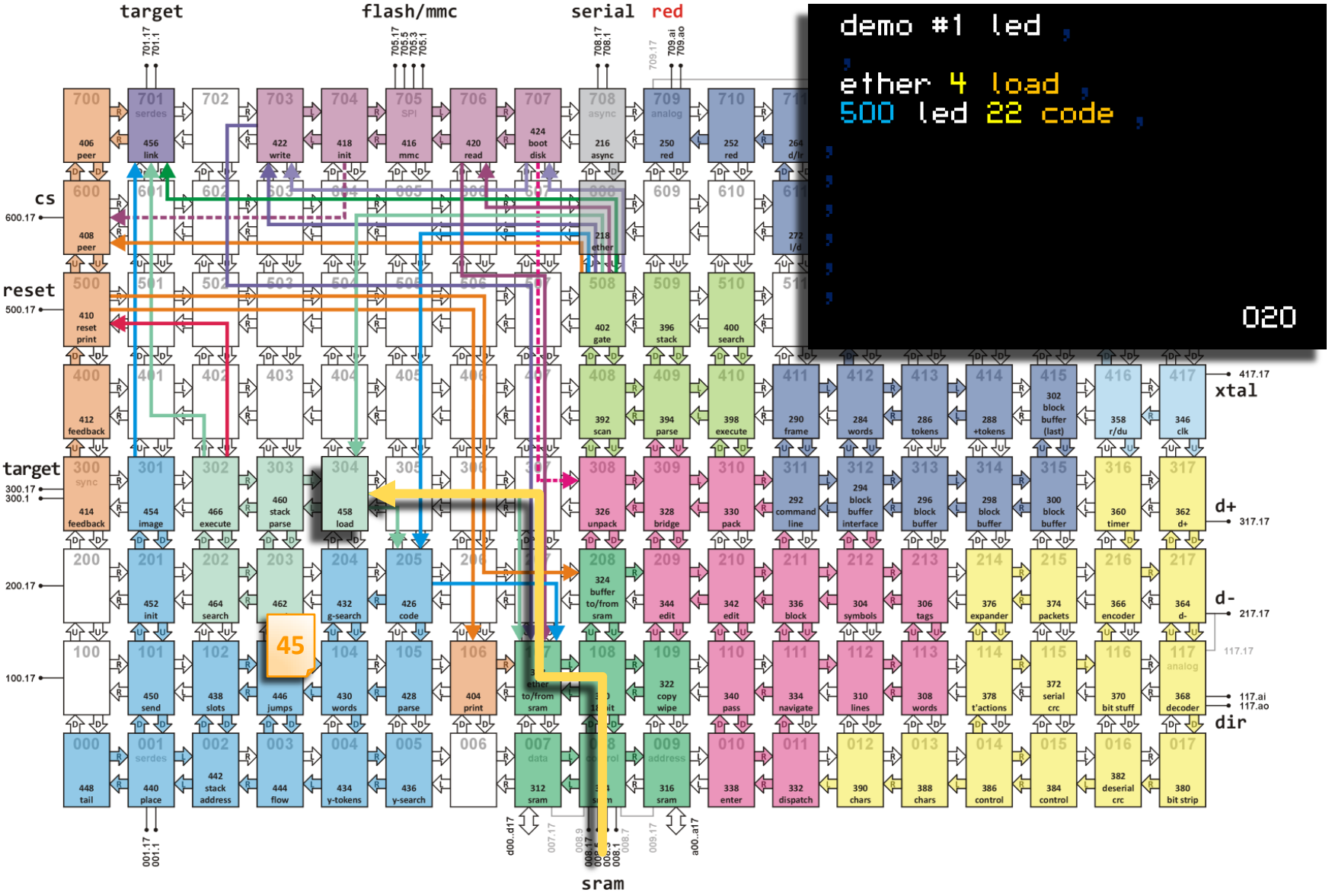
loading applications



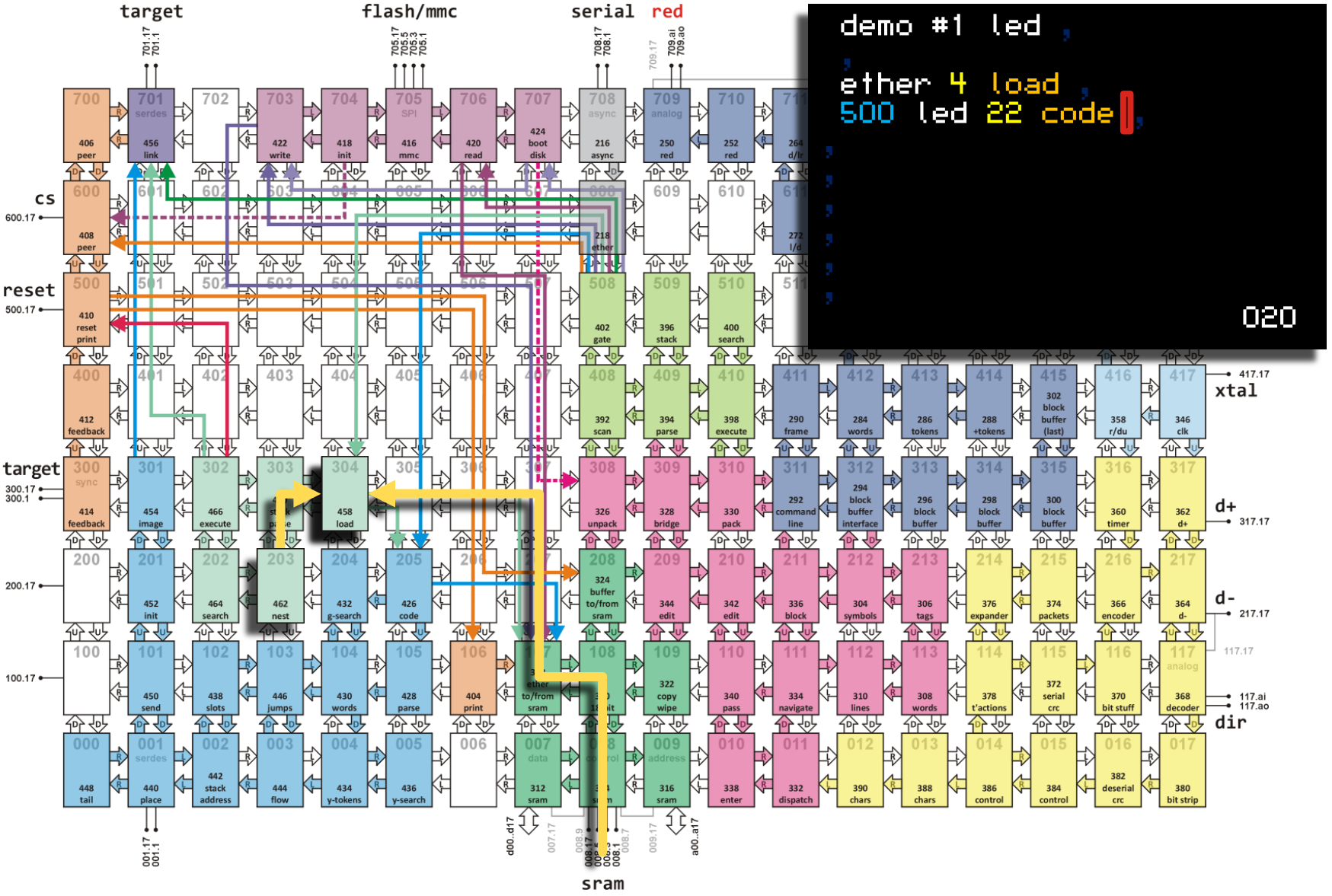
loading applications



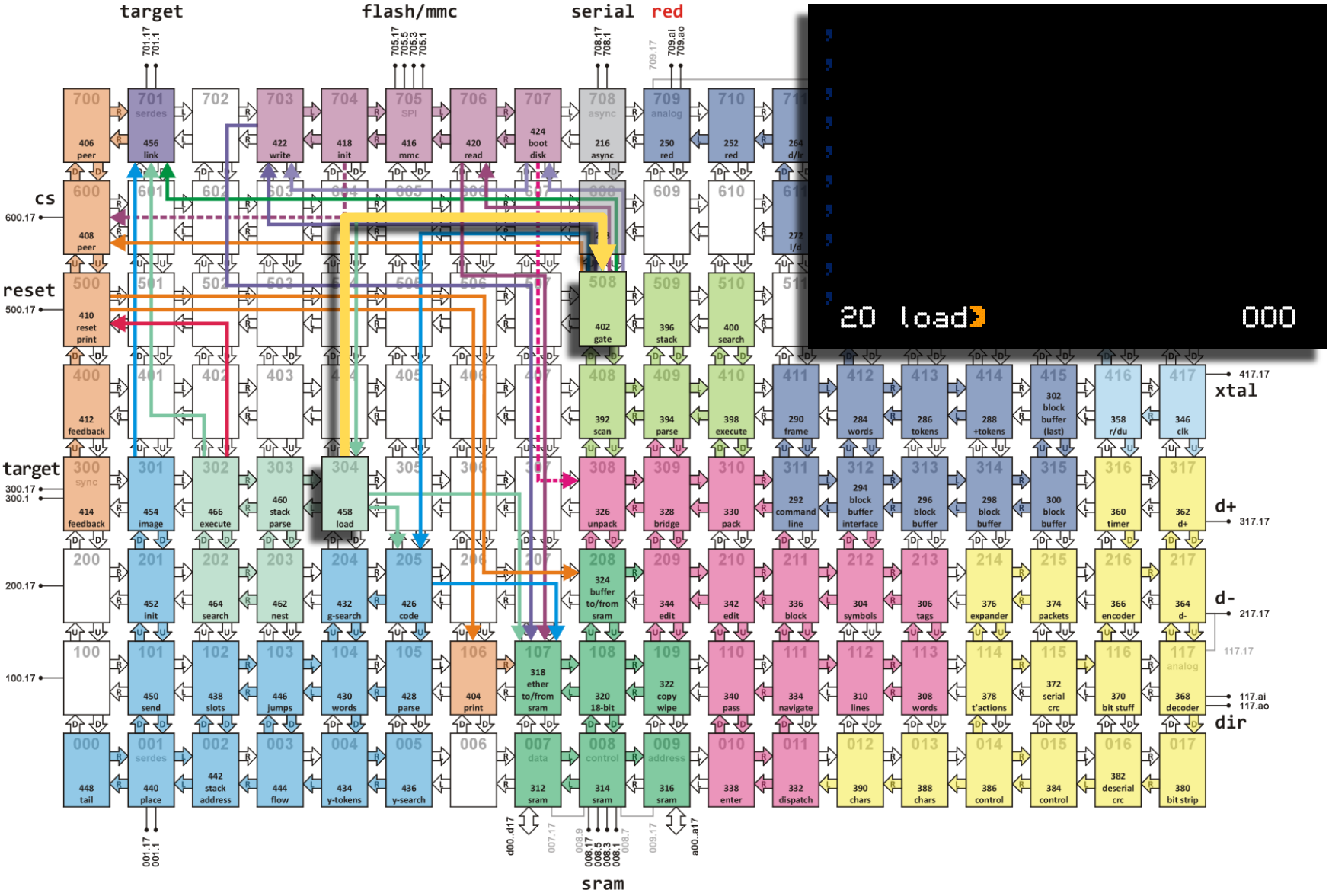
loading applications



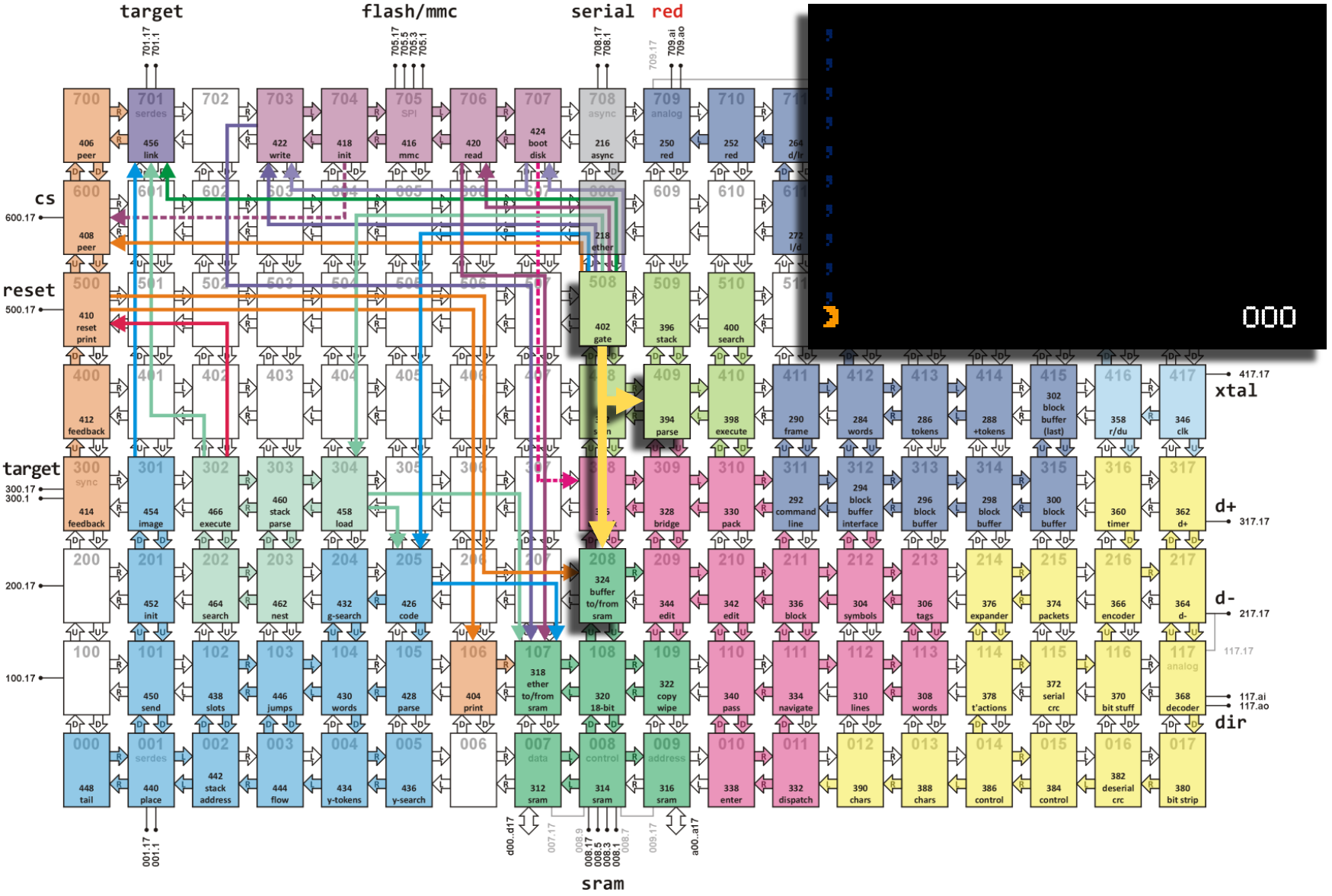
loading applications



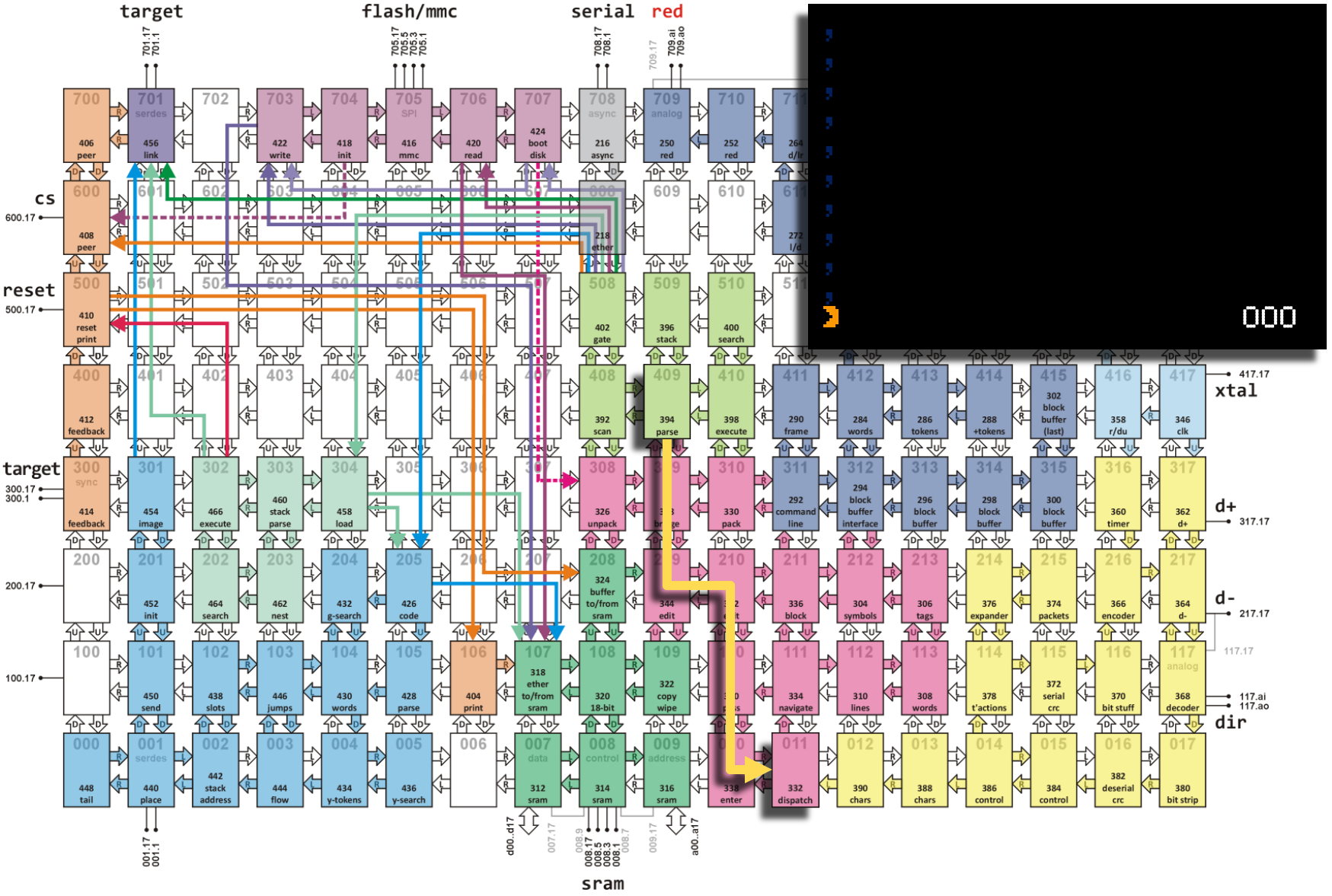
loading applications



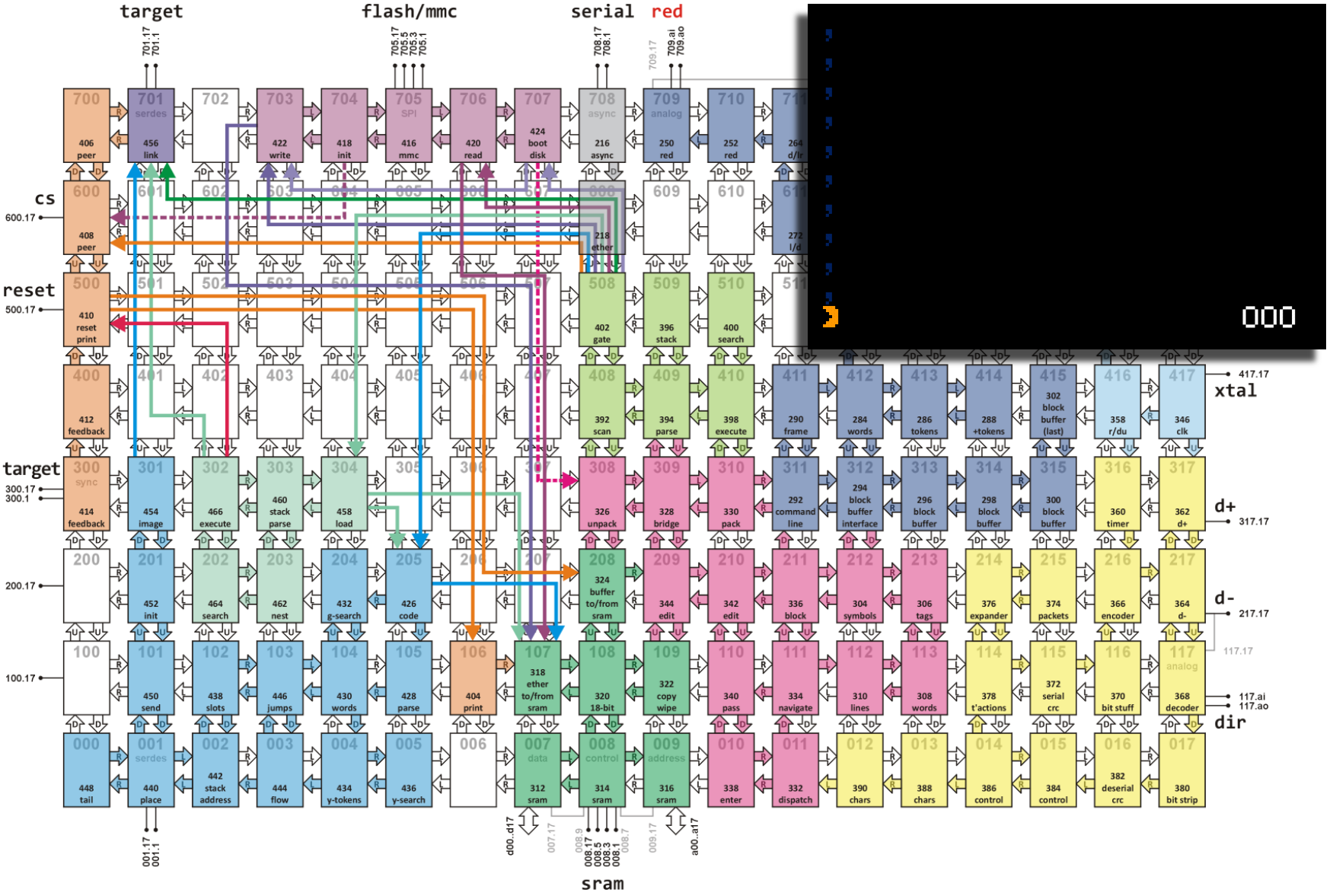
loading applications



loading applications



loading applications



development tools

exercising target node

- `hook` set a path to the target node
- `run` execute an instruction word in the target node

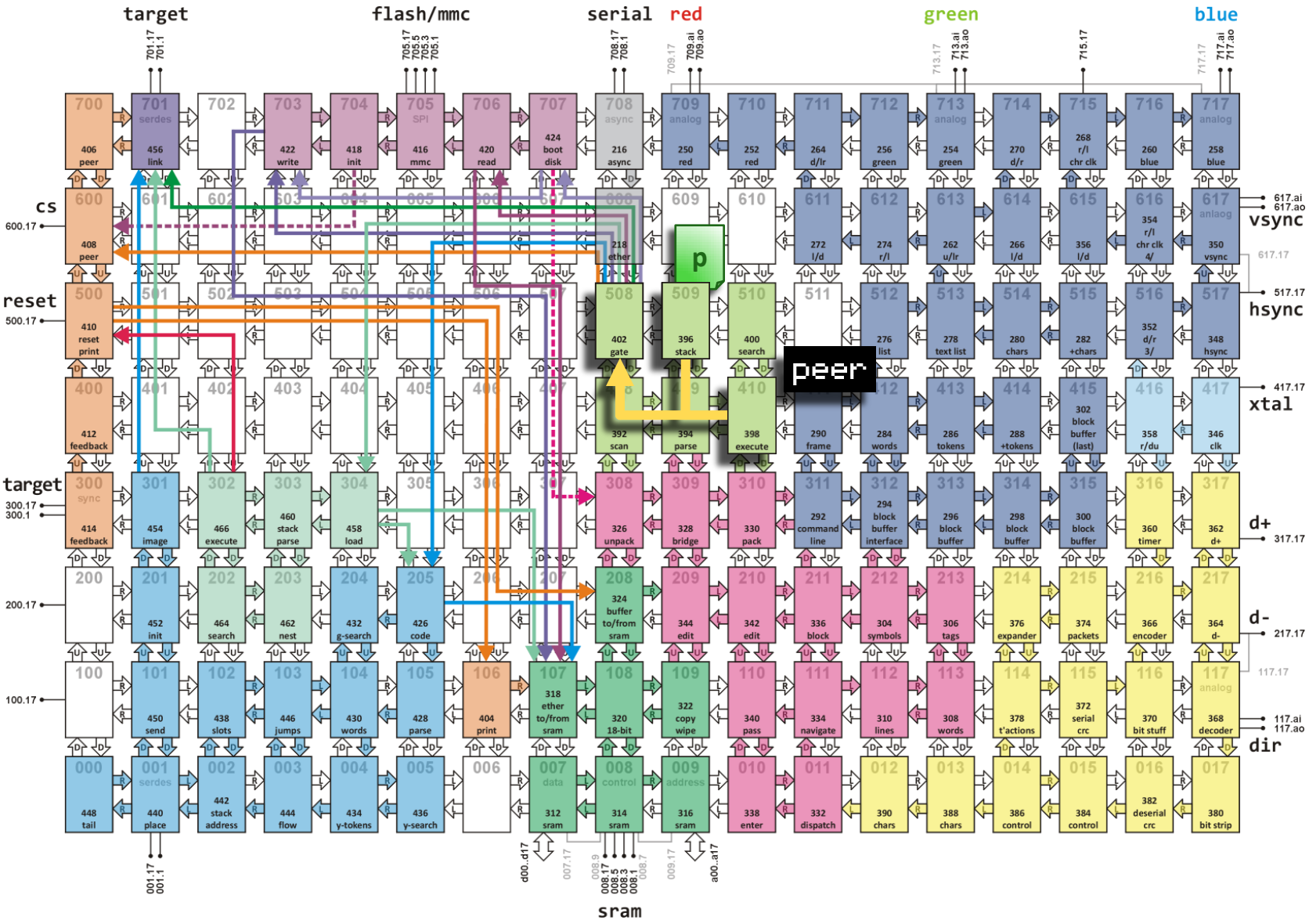
target node examination

- `peer` get content of the target node ram and stack

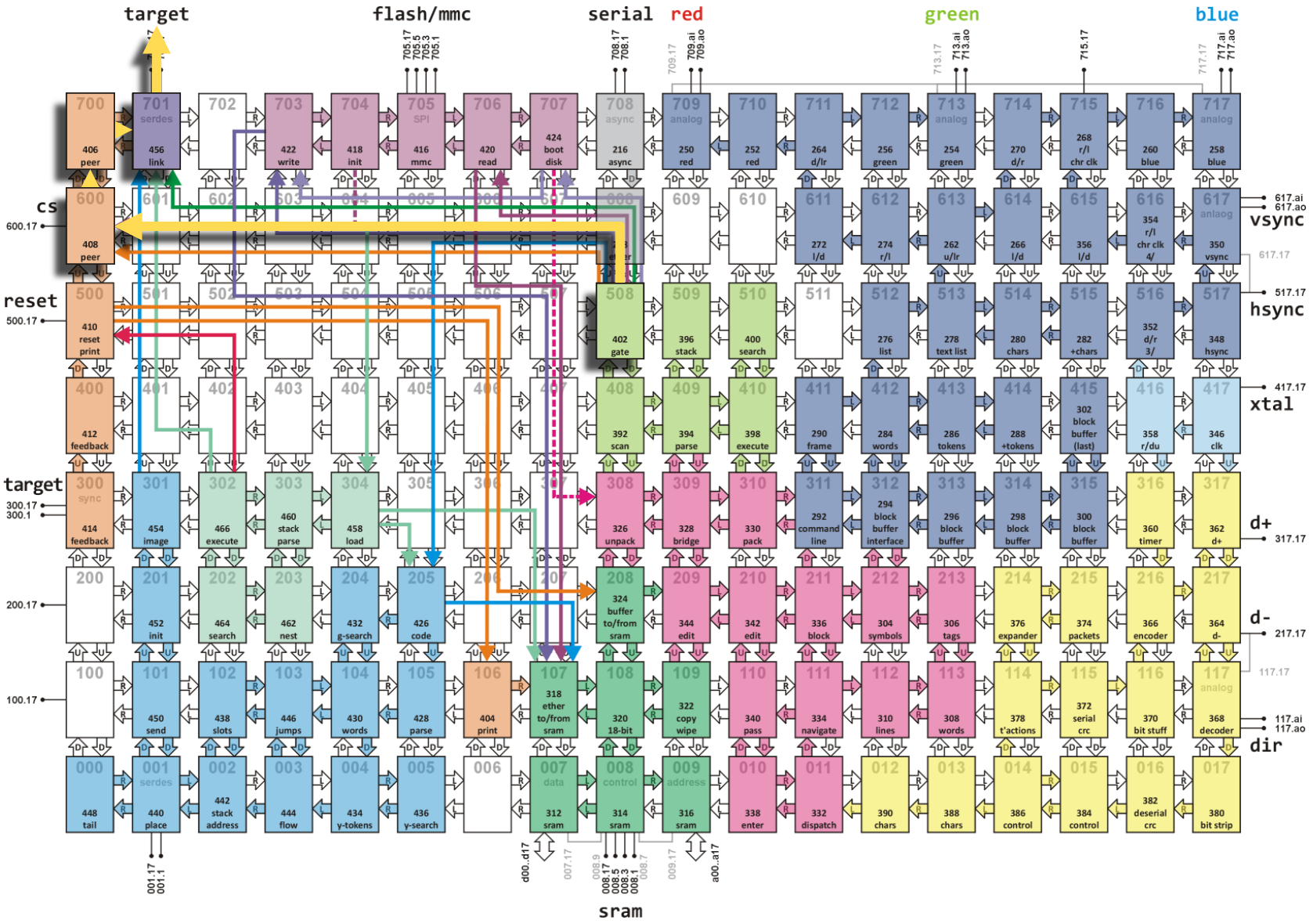
demo #2

exercising target node

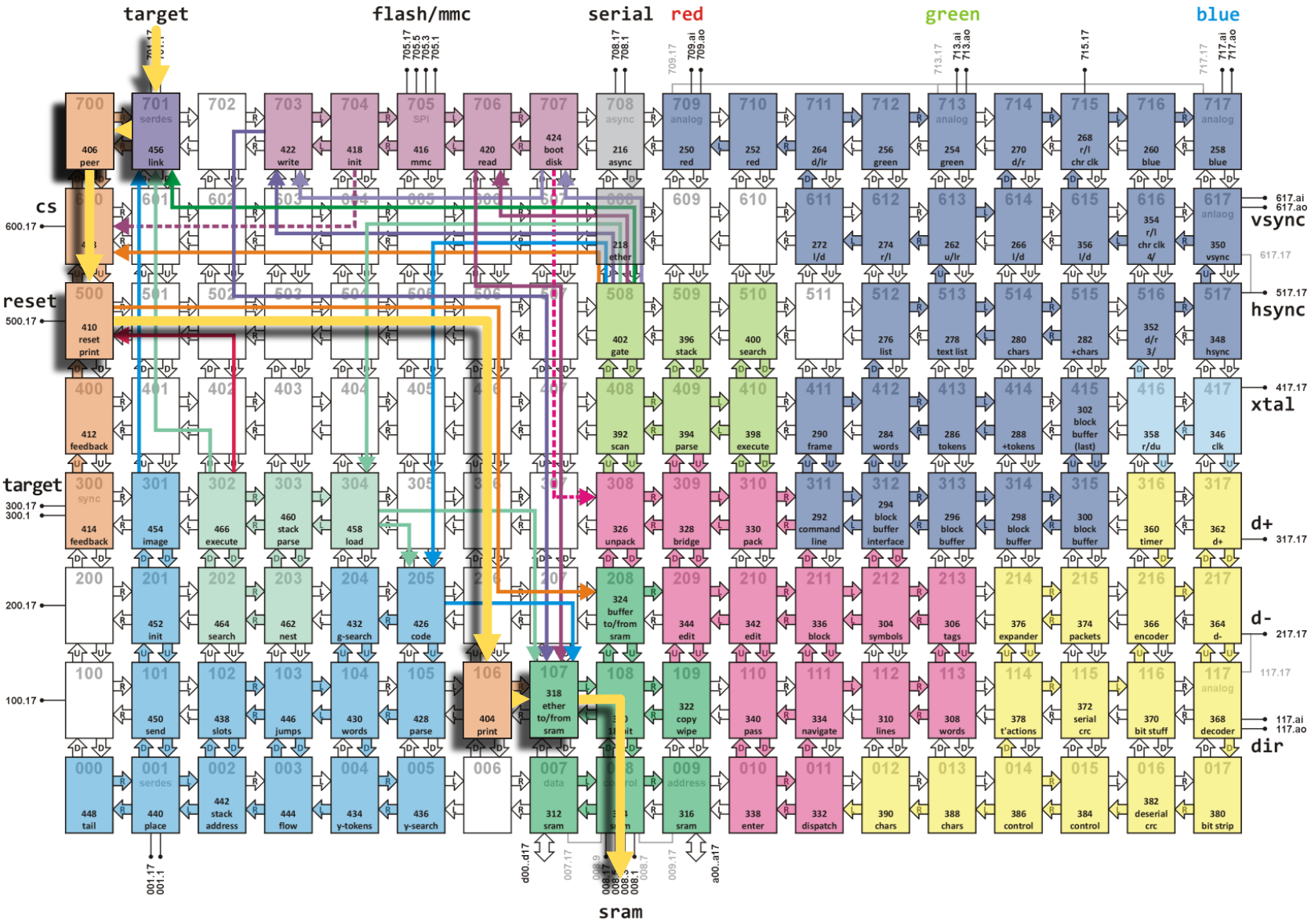
development tools



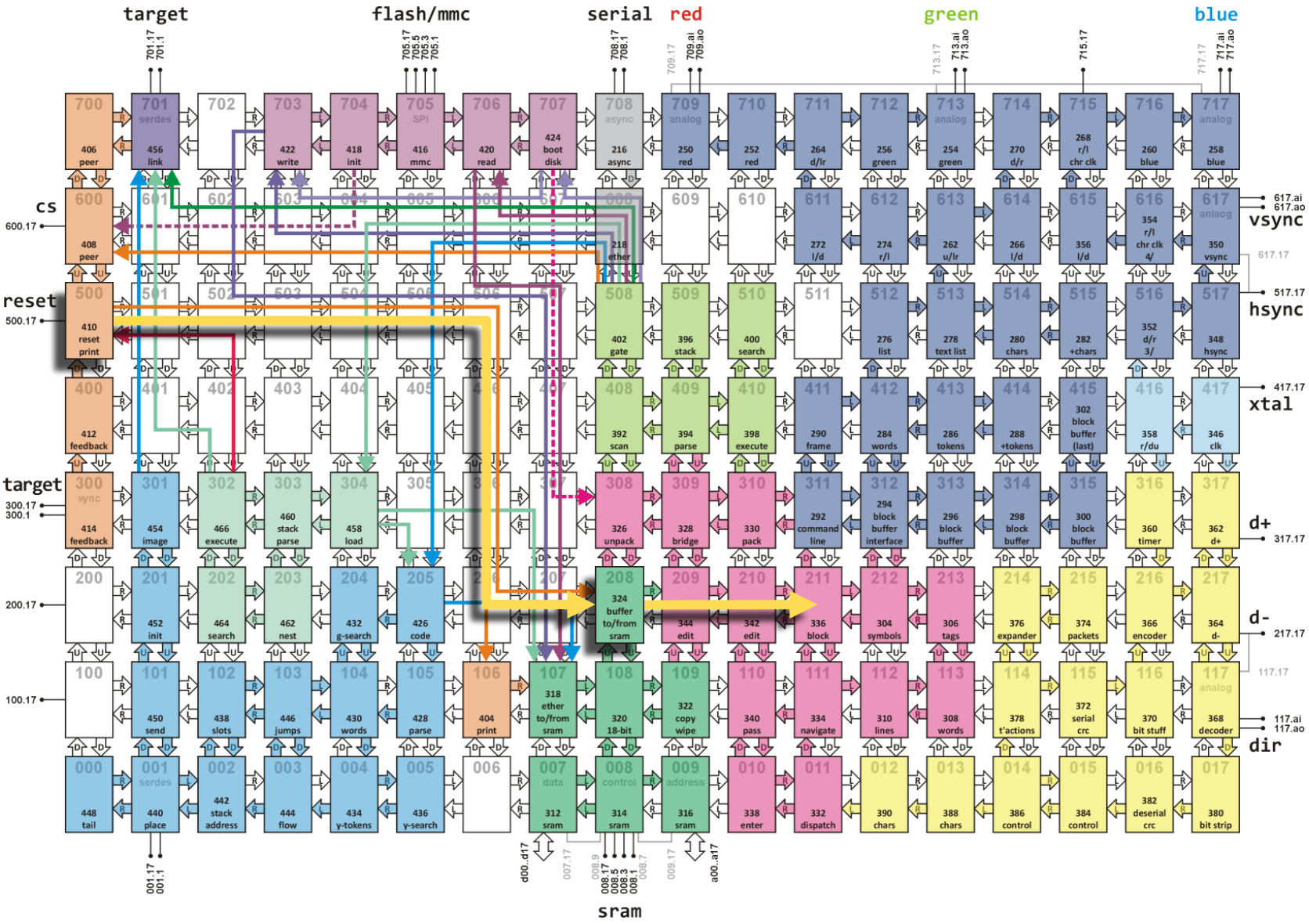
development tools



development tools



development tools



demo #3

target examination

data storage

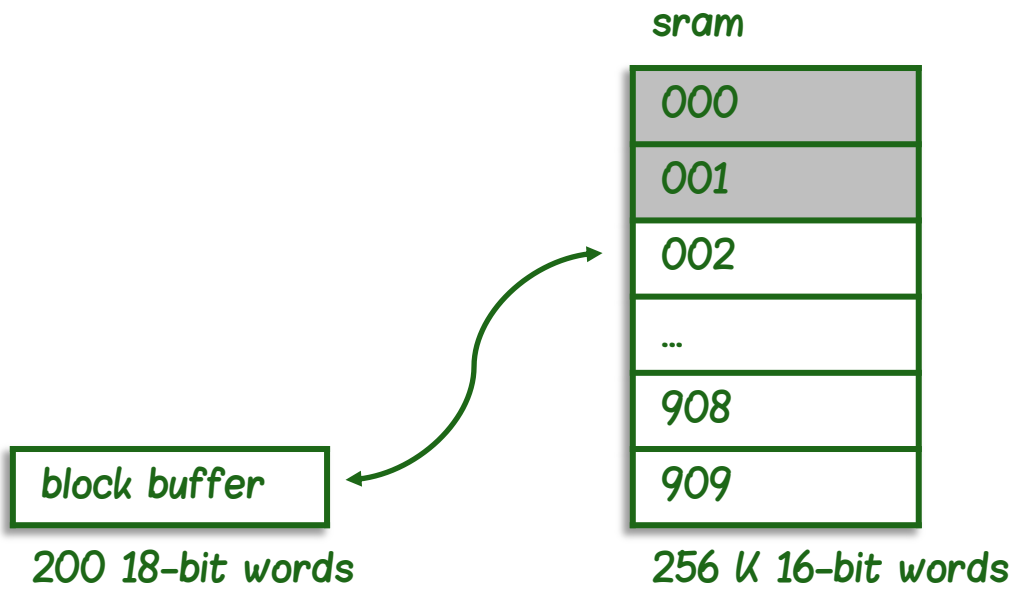
three levels of data storage

- block buffer – displayed on screen

data storage

three levels of data storage

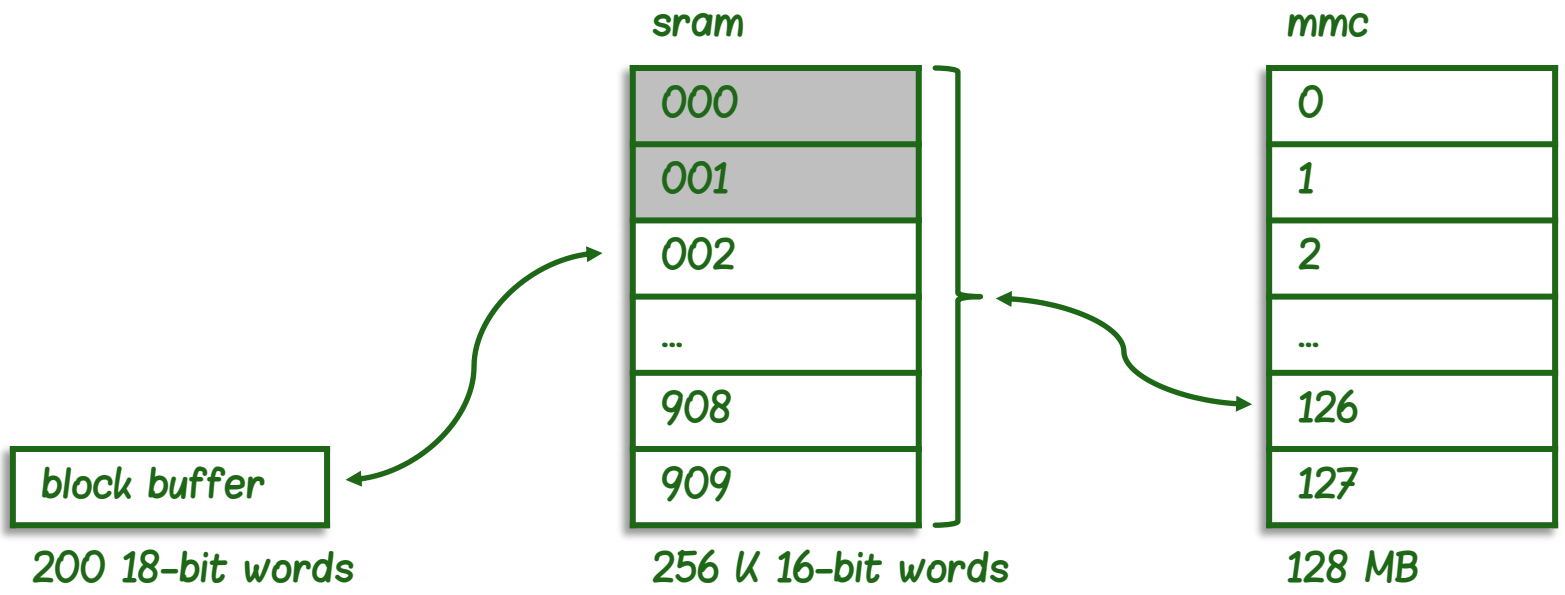
- block buffer – displayed on screen
- sram – all available blocks = one virtual disk



data storage

three levels of data storage

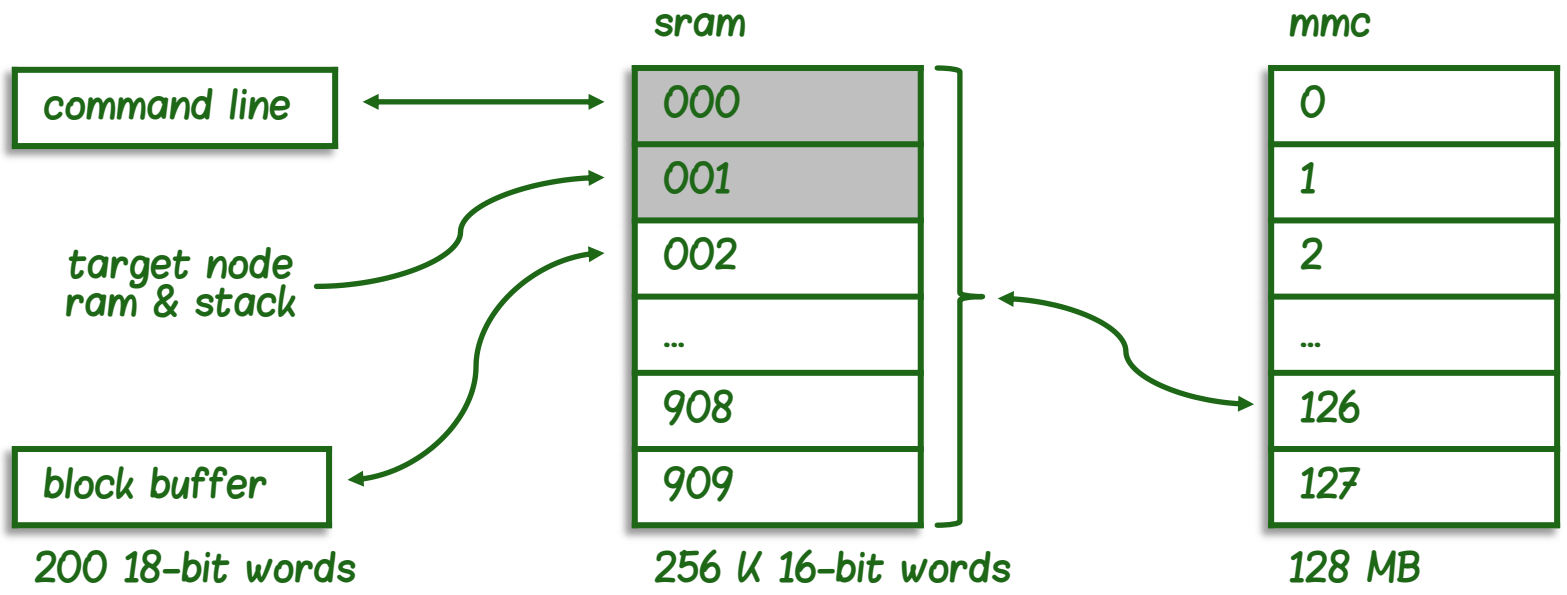
- block buffer – displayed on screen
- sram – all available blocks = one virtual disk
- mmc – permanent storage of all virtual disks



data storage

three levels of data storage

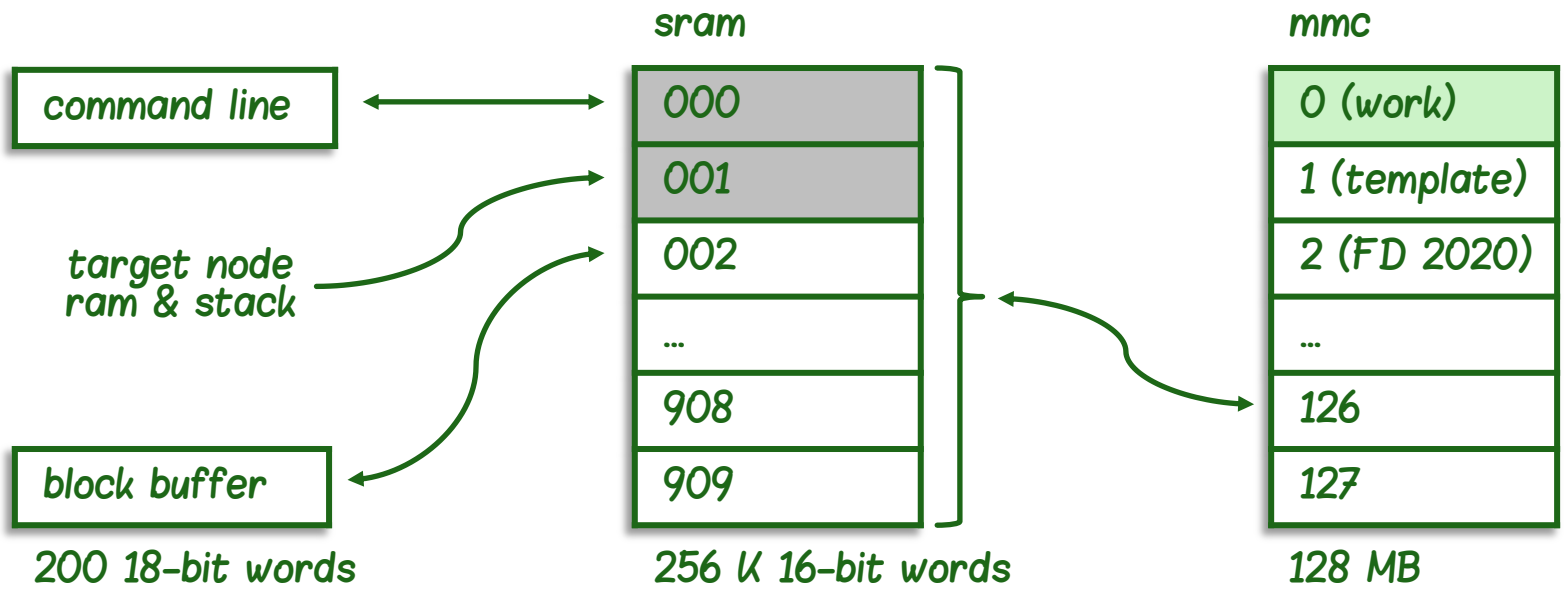
- block buffer – displayed on screen
- sram – all available blocks = one virtual disk
- mmc – permanent storage of all virtual disks



data storage

three levels of data storage

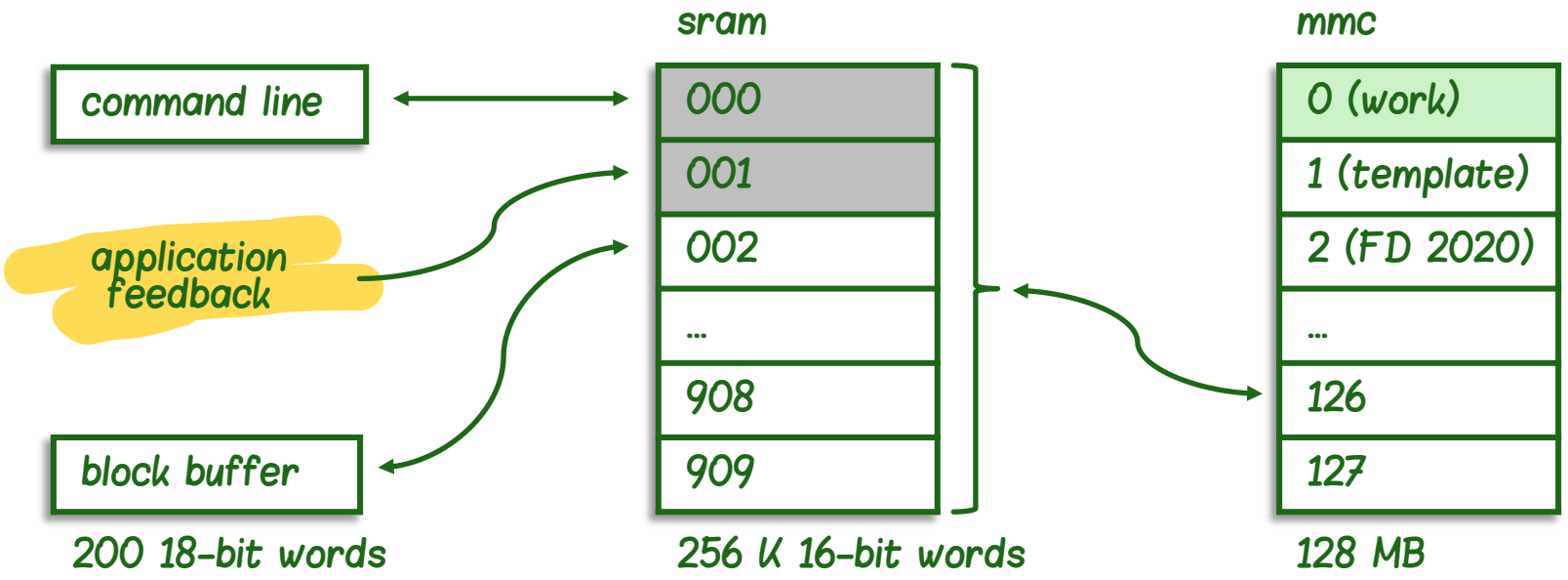
- block buffer – displayed on screen
- sram – all available blocks = one virtual disk
- mmc – permanent storage of all virtual disks



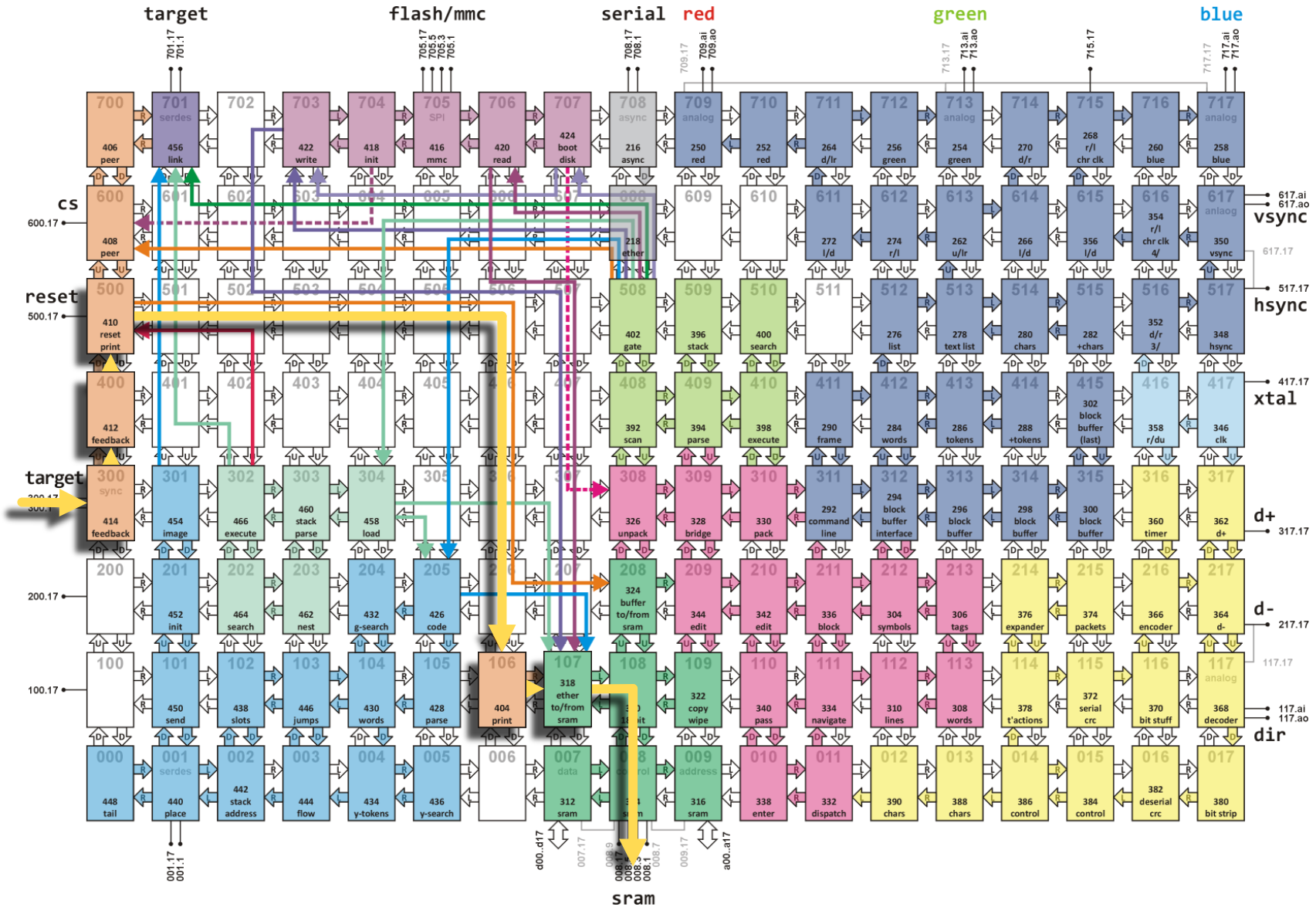
data storage

three levels of data storage

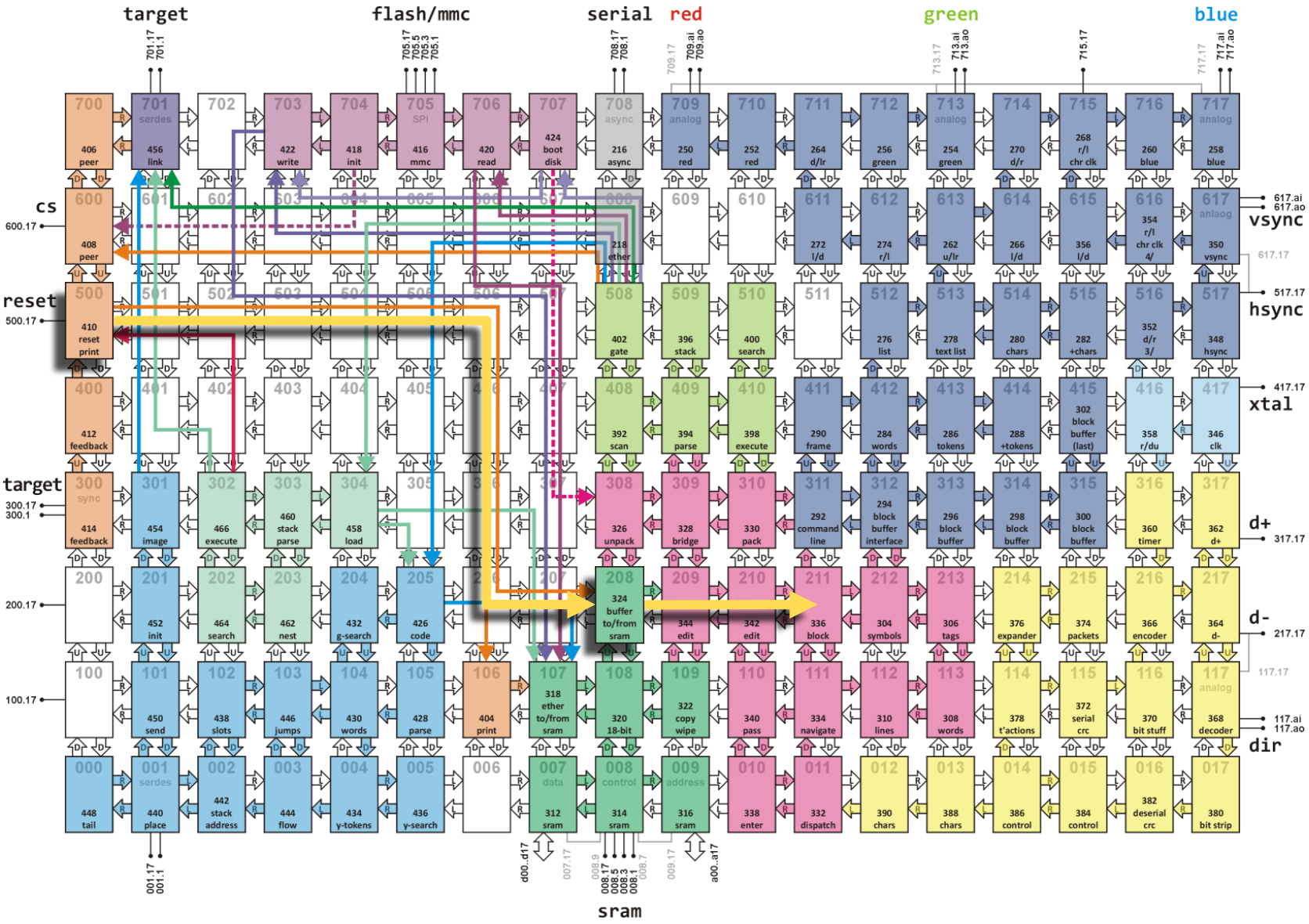
- block buffer – displayed on screen
- sram – all available blocks = one virtual disk
- mmc – permanent storage of all virtual disks



application feedback



application feedback



demo #4

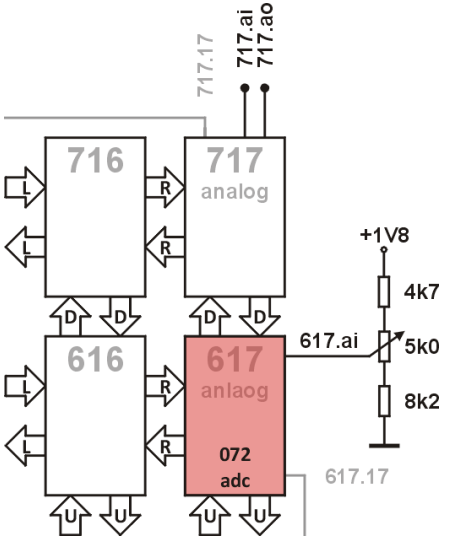
hello, world!

demo #5

big clock

demo #6 – sensor readout

potentiometer – voltage divider
analog to digital conversion
linear between 0.8 and 1.3 V



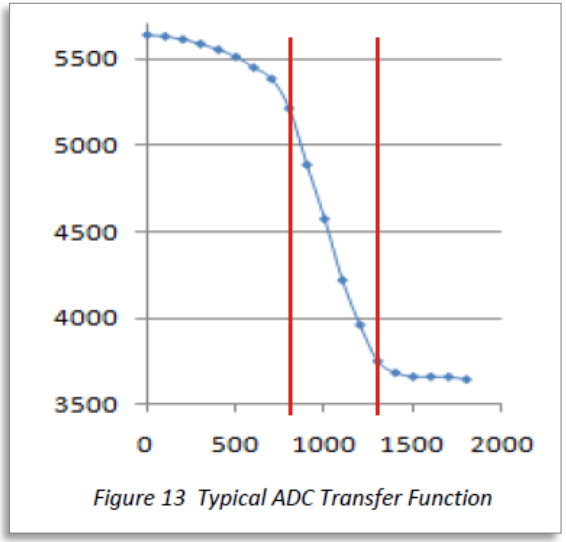
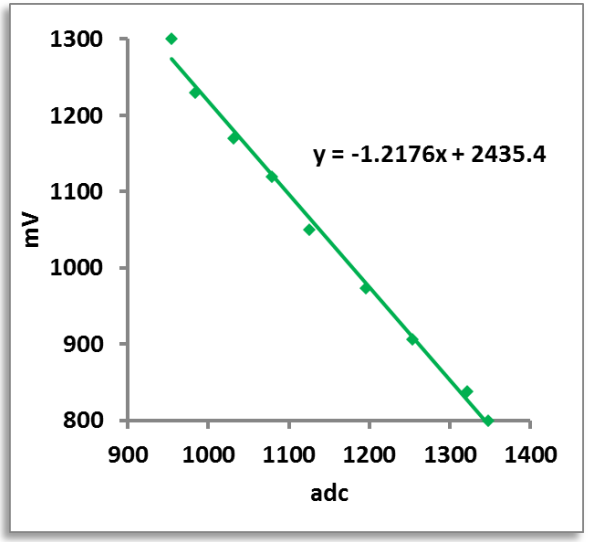


Figure 13 Typical ADC Transfer Function



demo #6

sensor readout

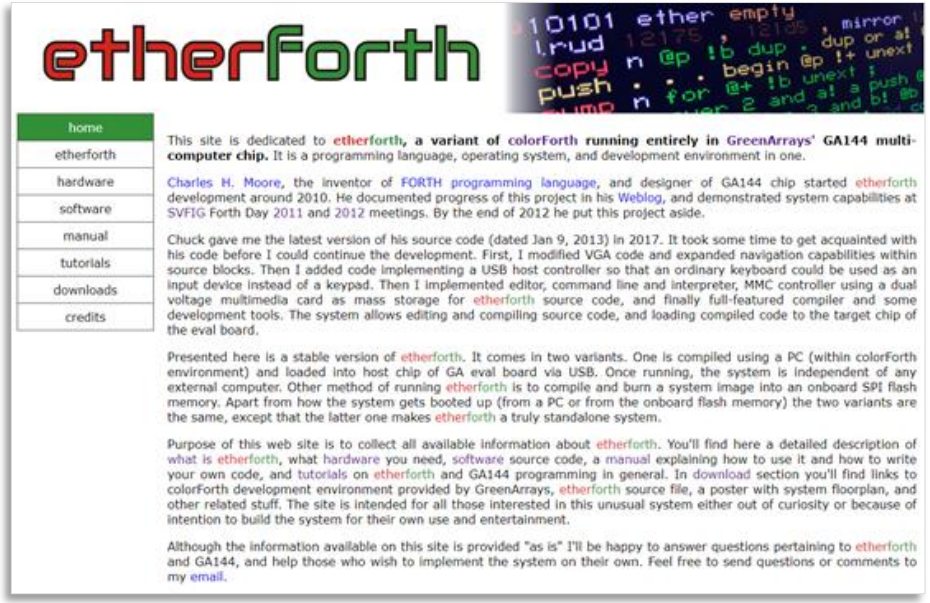
demo #7

single player pong game

resources

new web site www.etherforth.org

- detailed system description
- hardware section
- source code listing
- cF image file
- cF tools
- user's manual
- tutorials, examples, apps



acknowledgements

GreenArrays, Inc.

Chuck Moore

Greg Bailey

contact information
email: daniel@etherforth.org