I2C II

ECE 3710

I spilled spot remover on my dog...now he's gone.

- Steven Wright

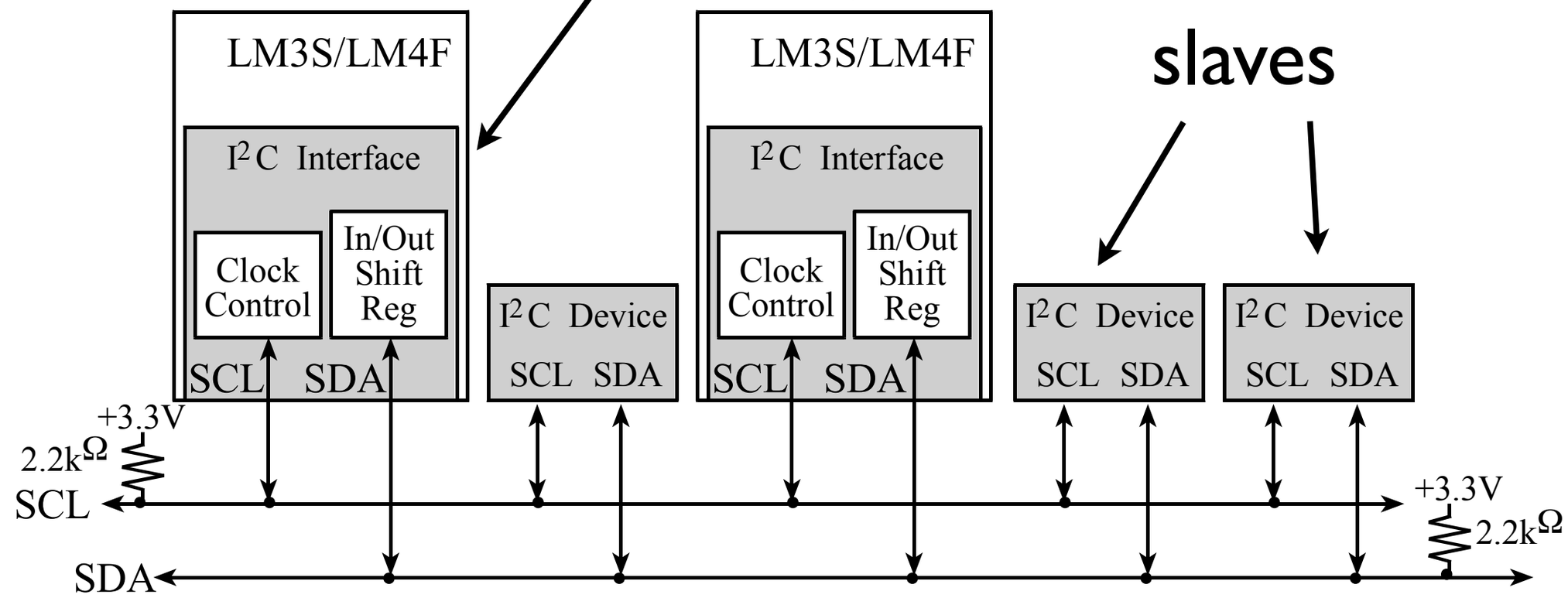
I2C architecture

multiple devices on same bus:

(no CS wires)

masters

slaves



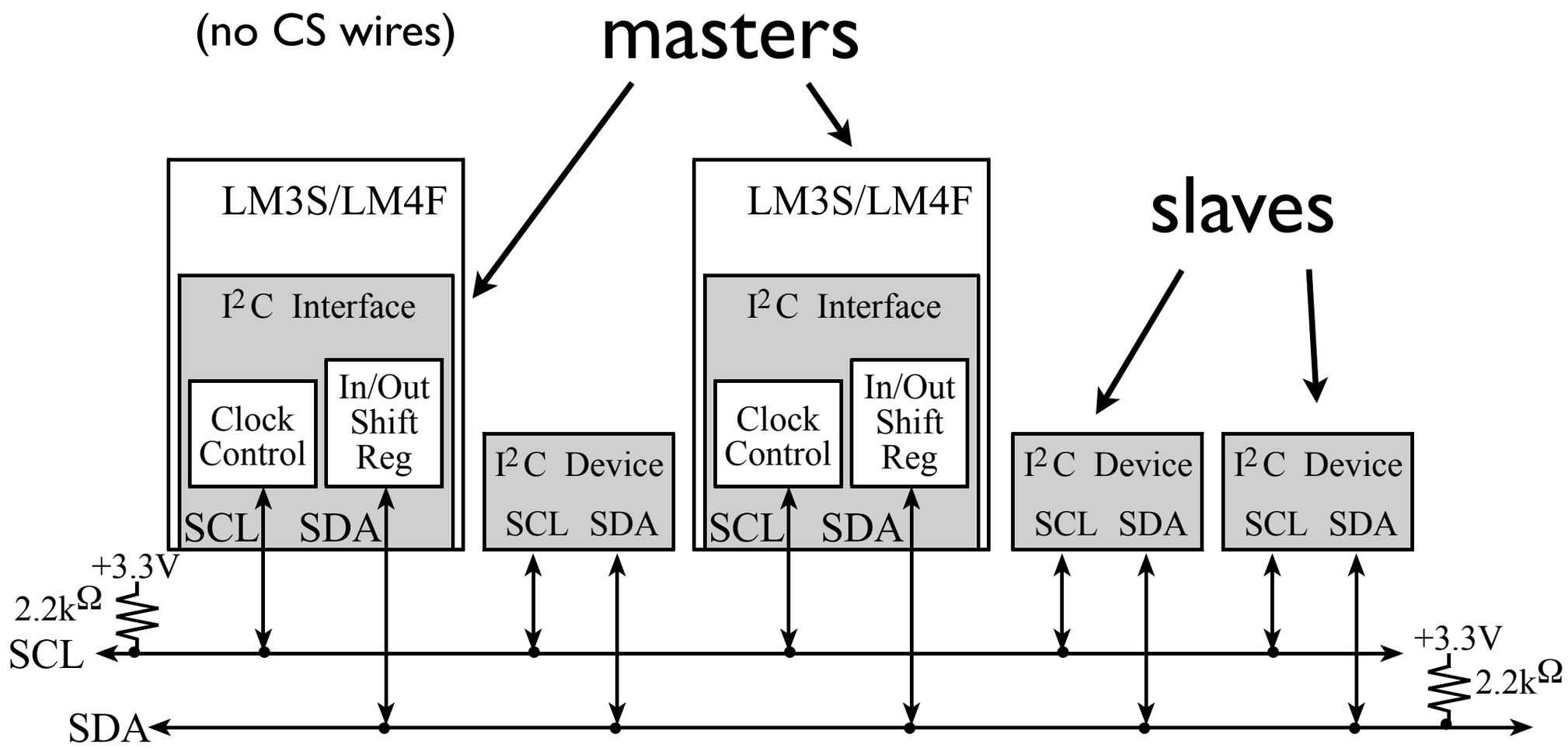
can have more than one → master(s):

1. control clock
2. control bus (initiate data TX/RX)

I2C architecture

multiple devices on same bus:

(no CS wires)



master-centric:

slaves:

1. do not initiate comm
2. no direct comm. between slaves

1. each has unique address
2. controlled by master

I2C architecture

shared bus:

open-drain

clock & data

1. default is logic one
2. any device (master or slave) can bring low

stays low until released

I2C protocol:

1. initiate comm
(master)
2a. send addr
(which device to comm with)
2b. read or write
(direction of comm: master2slave or slave2master [TX or RX])
3. acknowledge request
(slave)
4. TX/RX data
5. acknowledge data
(recipient of data: master or slave)
6. 4--5 until all data TX'd/RX'd
7. stop/restart
(master)

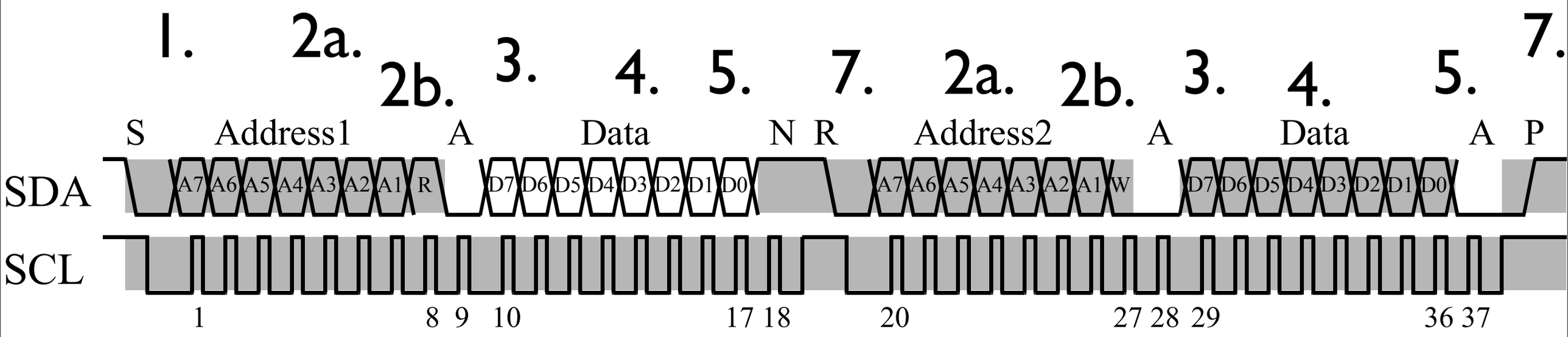
all communication:

1. 8-bit frames
2. MSB first

ex: slave2master then master2slave

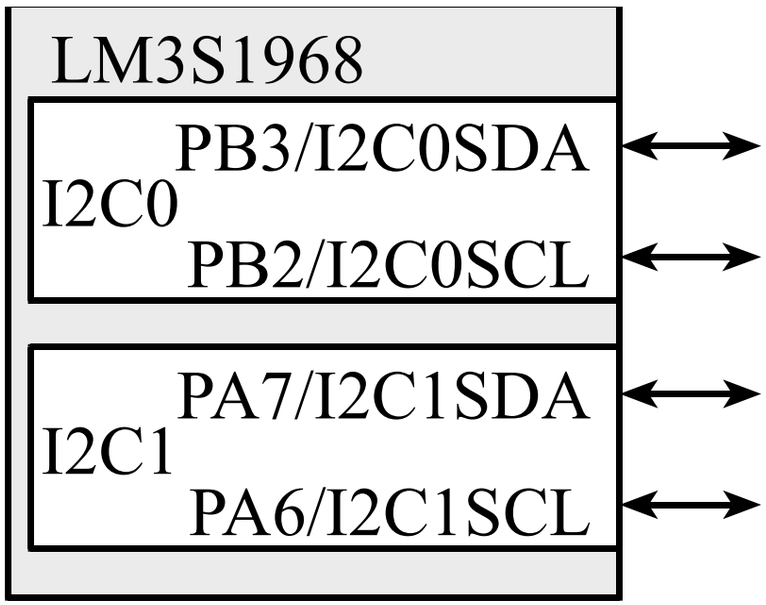
(master shaded, slave in white)

two bytes:



notice NACK and RESTART after byte one

LM3S1968 I2C modules



1. two, independent
2. master or slave
3. 100 Kbps (normal), 300 Kbps (fast) (others, too)
4. interrupts
5. multi-master support
6. rx/tx modes: single & burst

tx/rx single byte

tx/rx multiple bytes w/o releasing line

notice: share GPIO pins

LM3S1968 I2C configuration

for master

1. enable I2C peripheral clock:
(RCGC1, p218)
2. enable GPIO clock
(RCGC2, p227)
3. pin config: alt func, open drain, and digital enable
(AFSEL, p307; ODR, p312; DEN, p316)
4. set as master
(MCR, p538)
5. set I2C clock rate
(TPR, p532)

5. set I2C clock rate (TPR, p532)

use main osc (not internal)

fixed: 6 + 4

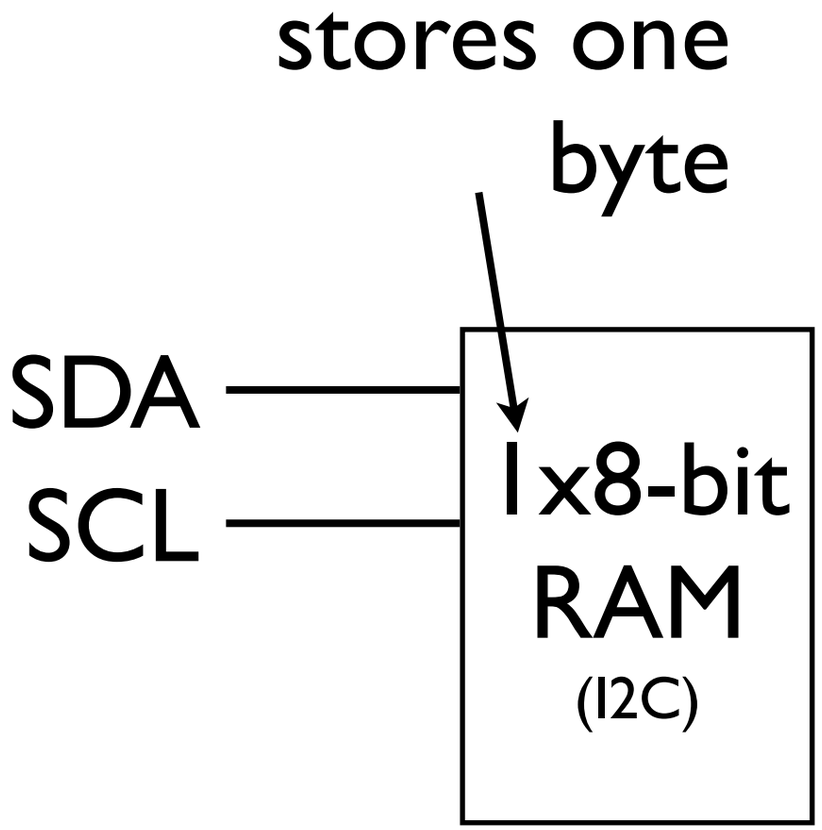
```
TPR = (System Clock / (2 * (SCL_LP + SCL_HP) * SCL_CLK)) - 1;
```

two data rates: 100 Kbps or 300 Kbps (100e3 or 300e3)

e.g.: SysClk = 12 MHz

=> TPR = 5

LM3S1968 TX/RX example



if direction bit:

0: store byte sent

1:return stored byte

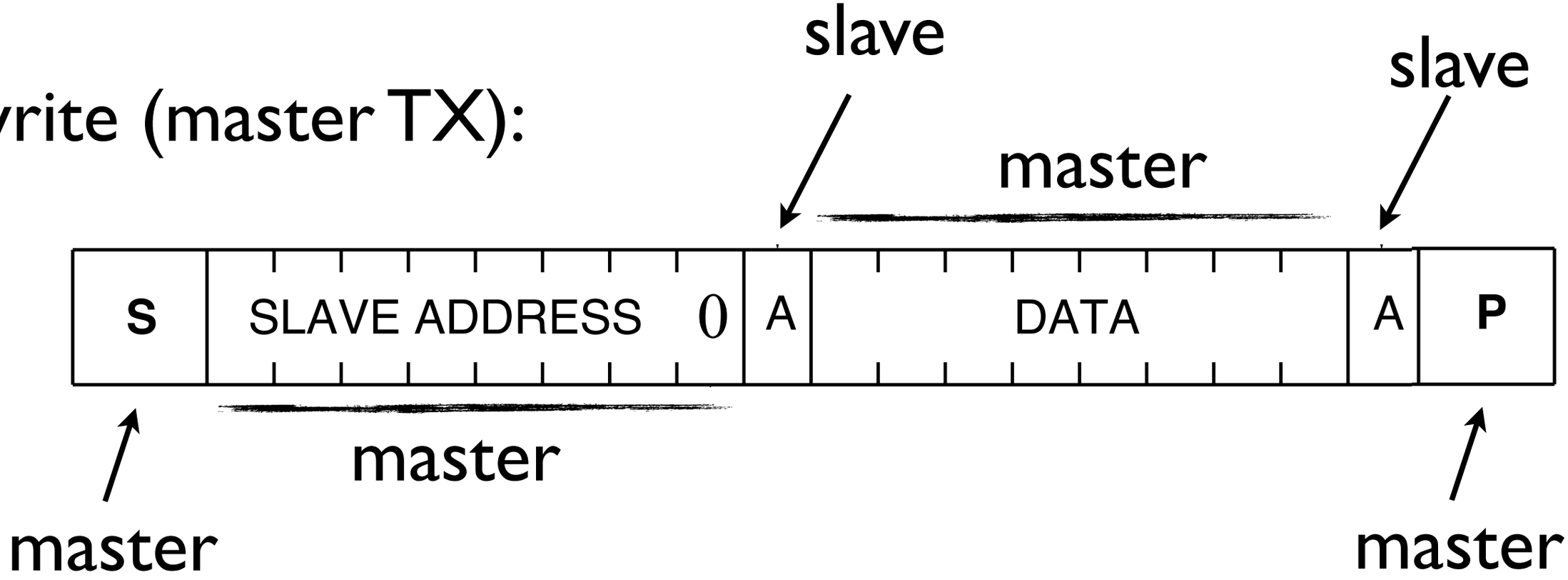
single RX/TX need to
read/write to device

one byte of RAM?

jokes on us: still have to interface it

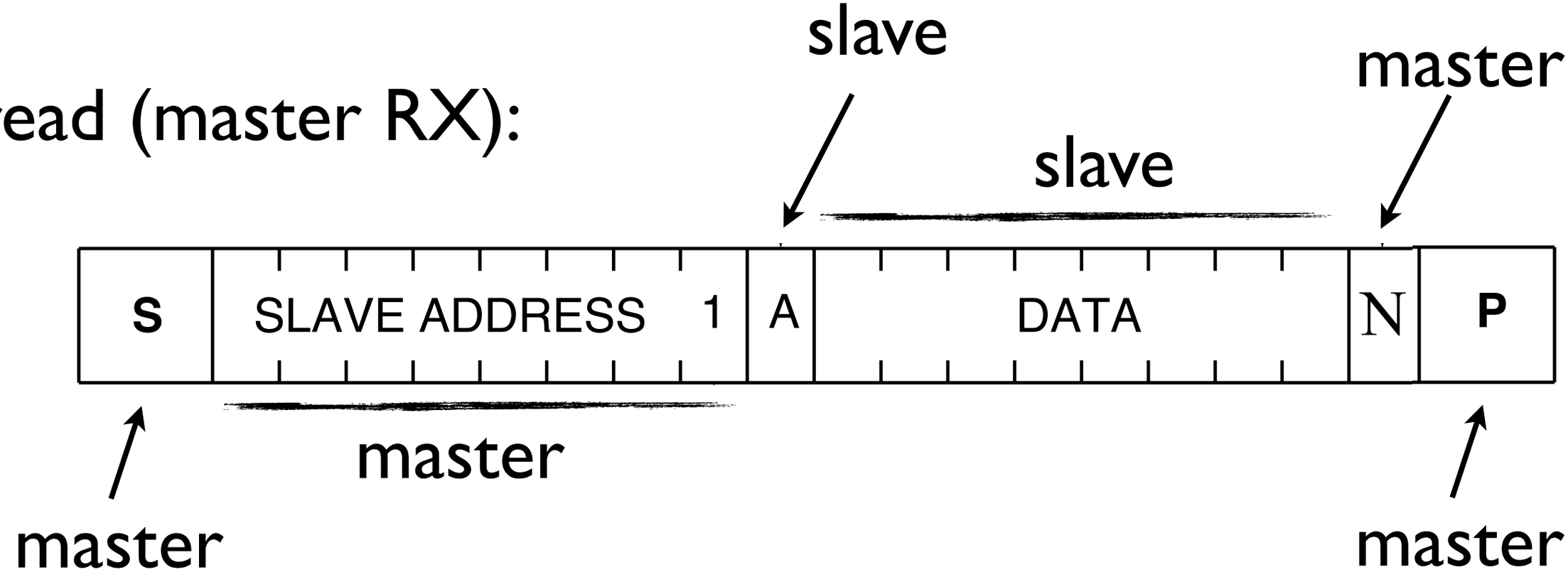
LM3S1968 TX/RX example

write (master TX):



LM3S1968 TX/RX example

read (master RX):



LM3S1968 I2C single TX/RX

using polling:

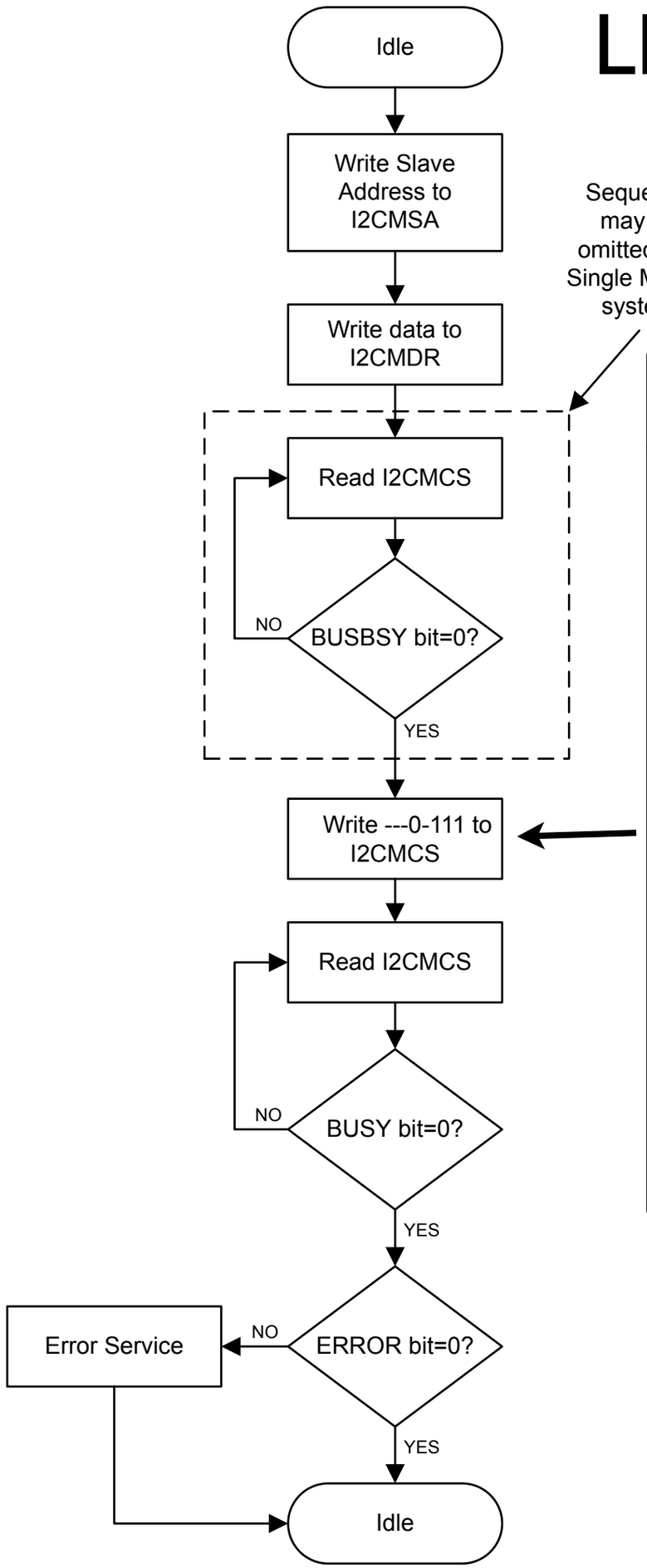
1. put data in data register (MDR, p5232)
2. select RX/TX single mode (MCS, p529--30)
3. poll busy bit (if ==0 then RX/TX finished) (MCS, p528)

half-duplex, only

bit zero of MCS (read)

MCS: bits represent diff things depending on read or write

LM3S1968 I2C single TX



Repeated START: master takes control of bus via START

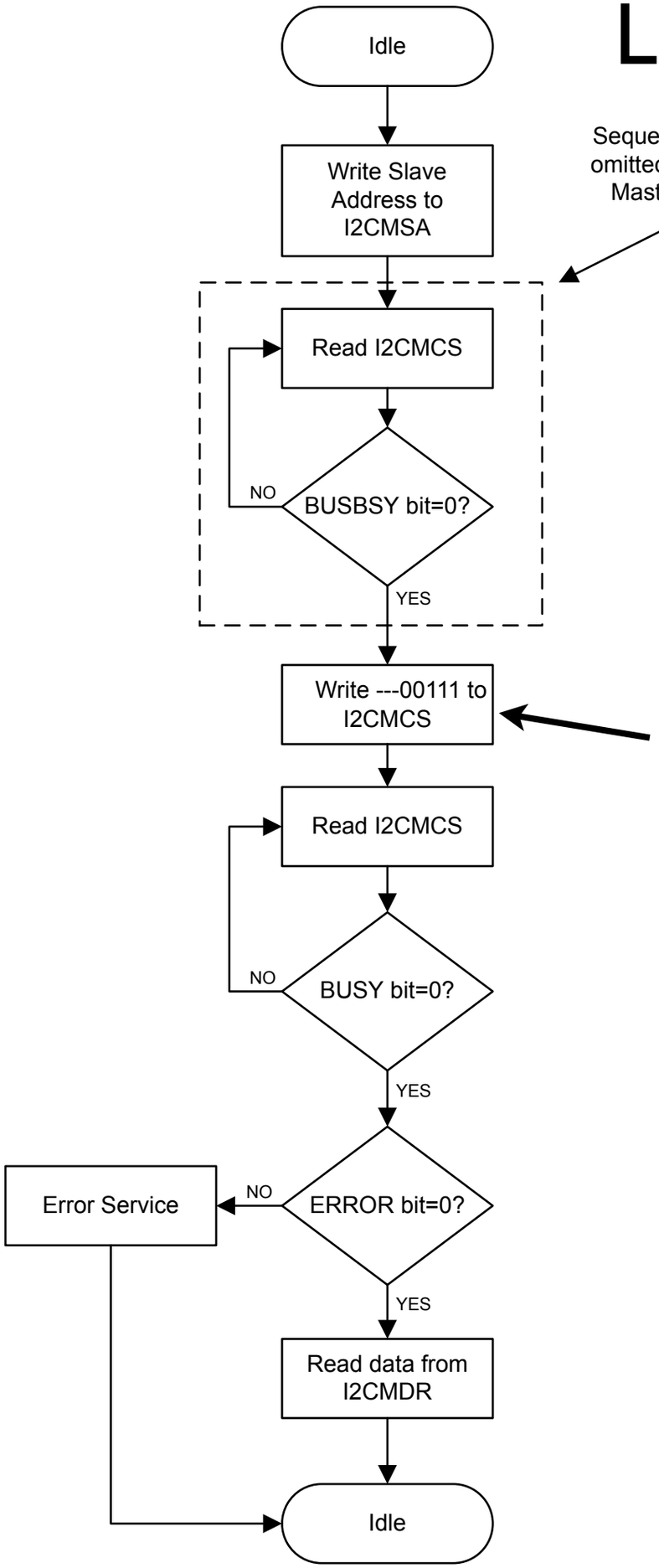
Current State	I2CMSA[0]	I2CMCS[3:0]				Description
	R/S	ACK	STOP	START	RUN	
Master Transmit	X	X	0	0	1	SEND operation (master remains in Master Transmit state).
	X	X	1	0	0	STOP condition (master goes to Idle state).
	X	X	1	0	1	SEND followed by STOP condition (master goes to Idl state).
	0	X	0	1	1	Repeated START condition followed by a SEND (maste remains in Master Transmit state).
	0	X	1	1	1	Repeated START condition followed by SEND and STO condition (master goes to Idle state).
	1	0	0	1	1	Repeated START condition followed by a RECEIVE operation with a negative ACK (master goes to Maste Receive state).
	1	0	1	1	1	Repeated START condition followed by a SEND and STOP condition (master goes to Idle state).
	1	1	0	1	1	Repeated START condition followed by RECEIVE (master goes to Master Receive state).
	1	1	1	1	1	Illegal.
	All other combinations not listed are non-operations.					NOP.

MCS:

1. sets TX parameters
2. what master does after TX

(e.g. repeated TX w/o resending addr, RX, etc.)

LM3S1968 I2C single RX



Sequence may be omitted in a Single Master system

Current State	I2CMSA[0]	I2CMCS[3:0]				Description
	R/S	ACK	STOP	START	RUN	
Master Receive	X	0	0	0	1	RECEIVE operation with negative ACK (master remains in Master Receive state).
	X	X	1	0	0	STOP condition (master goes to Idle state).[b]
	X	0	1	0	1	RECEIVE followed by STOP condition (master goes to Idle state).
	X	1	0	0	1	RECEIVE operation (master remains in Master Receive state).
	X	1	1	0	1	Illegal.
	1	0	0	1	1	Repeated START condition followed by RECEIVE operation with a negative ACK (master remains in Master Receive state).
	1	0	1	1	1	Repeated START condition followed by RECEIVE and STOP condition (master goes to Idle state).
	1	1	0	1	1	Repeated START condition followed by RECEIVE (master remains in Master Receive state).
	0	X	0	1	1	Repeated START condition followed by SEND (master goes to Master Transmit state).
	0	X	1	1	1	Repeated START condition followed by SEND and STOP condition (master goes to Idle state).
	All other combinations not listed are non-operations.					NOP.

LM3S1968 I2C single TX

```
I2C_TX
    ; 0. save return addr
  push {LR}

  ; 1. set slave addr and direction
  ldr R1,=I2C0
  lsl R0,R12,#0x1
  str R0,[R1,#0x0]

  ; 2. tx byte
  str R10,[R1,#0x8]

; 3. set master to single tx mode
mov R0,#0x7 ;0x7=0b111 (master idles after tx)
str R0,[R1,#0x4]

; 4. poll busy bit (busy=1 then still tx)
bl I2C0_POLL

; 5. we're done: restore LR and return to main
pop {LR}
bx LR
```

this routine performs single write procedure
R10: byte to write
R12: slave addr

LM3S1968 I2C single RX

```
I2C_RX
  ; 0. save return addr
  push {LR}

  ; 1. set slave addr and direction
  ldr R1,=I2C0
  str R0,[R1,#0x8]
  lsl R0,R12,#0x1
  orr R0,#1
  str R0,[R1,#0x0]

  ; 3. set master to single rx mode
  mov R0,#0x7 ;0x7=0b111 (master idles after rx)
  str R0,[R1,#0x4]

  ; 4. poll busy bit (busy=1 then still tx)
  bl I2C0_POLL

  ; 5. rx byte
  ldr R10,[R1,#0x8]

  ; 5. we're done: restore LR and return to main
  pop {LR}
  bx LR
```

this routine performs single read procedure

R10: where byte is put

R12: slave addr

LM3S1968 I2C busy bit

same for RX/TX

(i.e. =1 if RXing or TXing)

one polling routine
for both RX/TX:

```
I2C0_POLL
   ; R0: tmp
   ; R1: base addr of I2C0
ldr R0,[R1,#0x4] ;busy bit
ands R0,#1 ;(Z=1 if result is zero)
bne I2C0_POLL ;(branch if Z=0)
bx LR
```

interrupts (master):

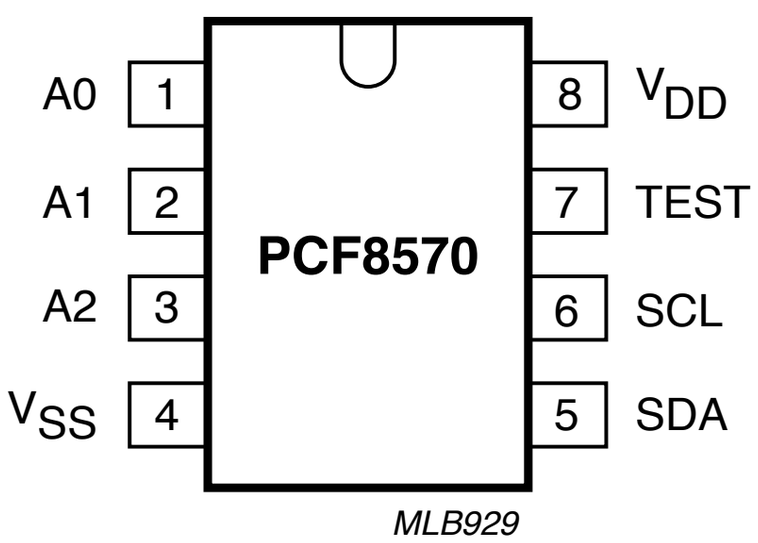
1. RX/TX complete
2. arbitration lost
3. error in RX/TX

In your own words. Do you have your own words? Personally, I'm using the ones that everyone else has been using.

- George Carlin

LM3S1968 burst TX example

256x8-bit
RAM
(I2C)



MLB929

to read memory:
1. specify address
(master TX)
2. get byte
(master RX)

to write memory:
1. specify address
(master TX)
2. send byte
(master TX)

multiple RX/TX need to
read/write to device

Just another I2C device, right?

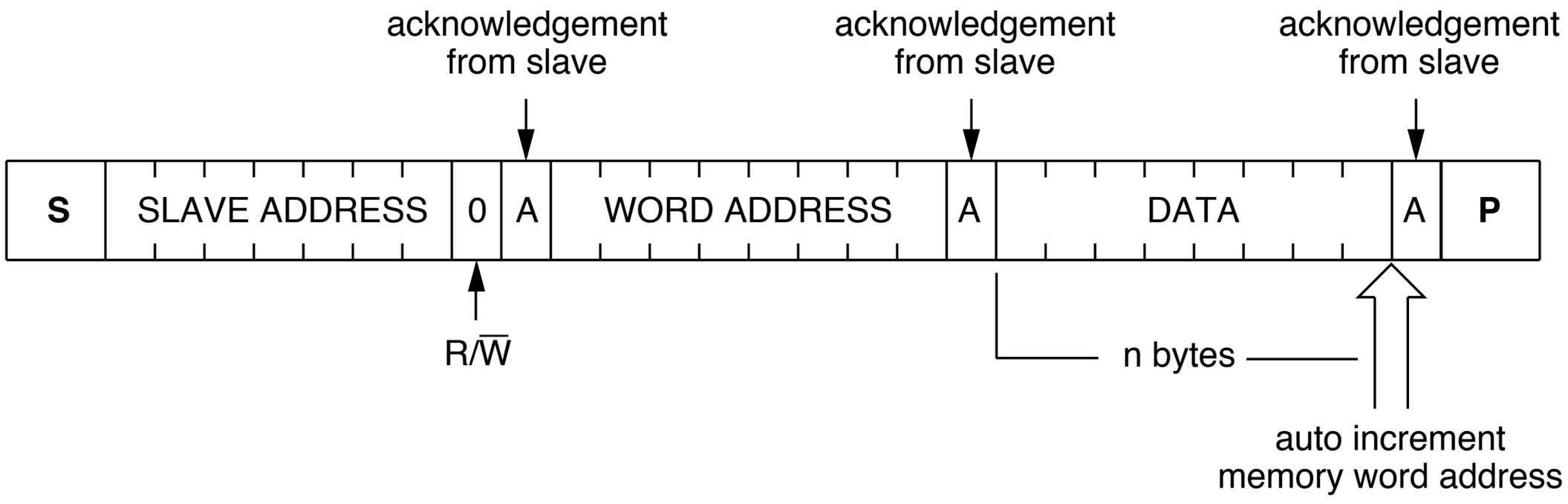
Config no problem...

LM3S1968 burst TX example

requires burst mode

(master doesn't give up line
or issue STOP/START)

write (master TX/TX):



last addr specified +1
remembered by device

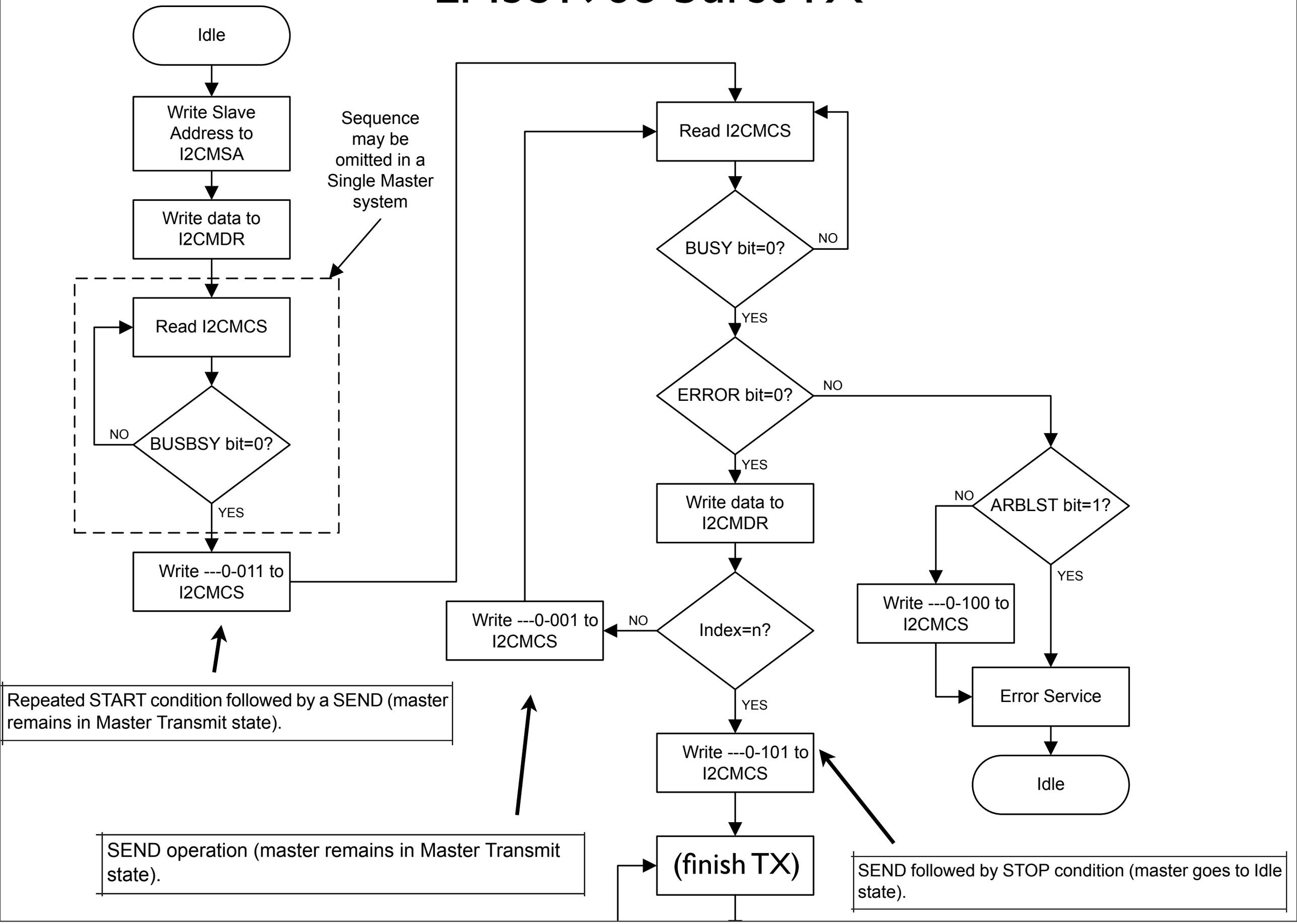
(remembers current addr for a special no-addr read)

master could TX another byte
w/o RESTART

Is it I2C?

we need more config options

LM3S1968 burst TX



write to PCF8570-style RAM

```
I2C_MEM_WR
  ; 0. save return addr
  push {LR}

  ; 1. set slave addr and direction
  ldr R1,=I2C0
  lsl R0,R12,#0x1
  str R0,[R1,#0x0]

  ; 2. tx byte addr first (put data in tx reg)
  str R11,[R1,#0x8]

  ; 3. set master to burst mode
  mov R0,#0x3 ;0x3=0b11 (master remains in tx mode after tx)
  str R0,[R1,#0x4]

  ; 4. poll busy bit (busy=1 then still tx)
  bl I2C0_POLL

  ; 5. now tx byte
  str R10,[R1,#0x8]

  ; 6. set master to single send mode
  mov R0,#0x5 ;0x5=0b101 (master goes idle after tx)
  str R0,[R1,#0x4]

  ; 7. poll busy bit
  bl I2C0_POLL

  ; 8. we're done: restore LR and return to main
  pop {LR}
  bx LR
```

this routine performs burst write procedure for writing to PCF8570-style RAM

R10: byte to write

R11: destination of byte

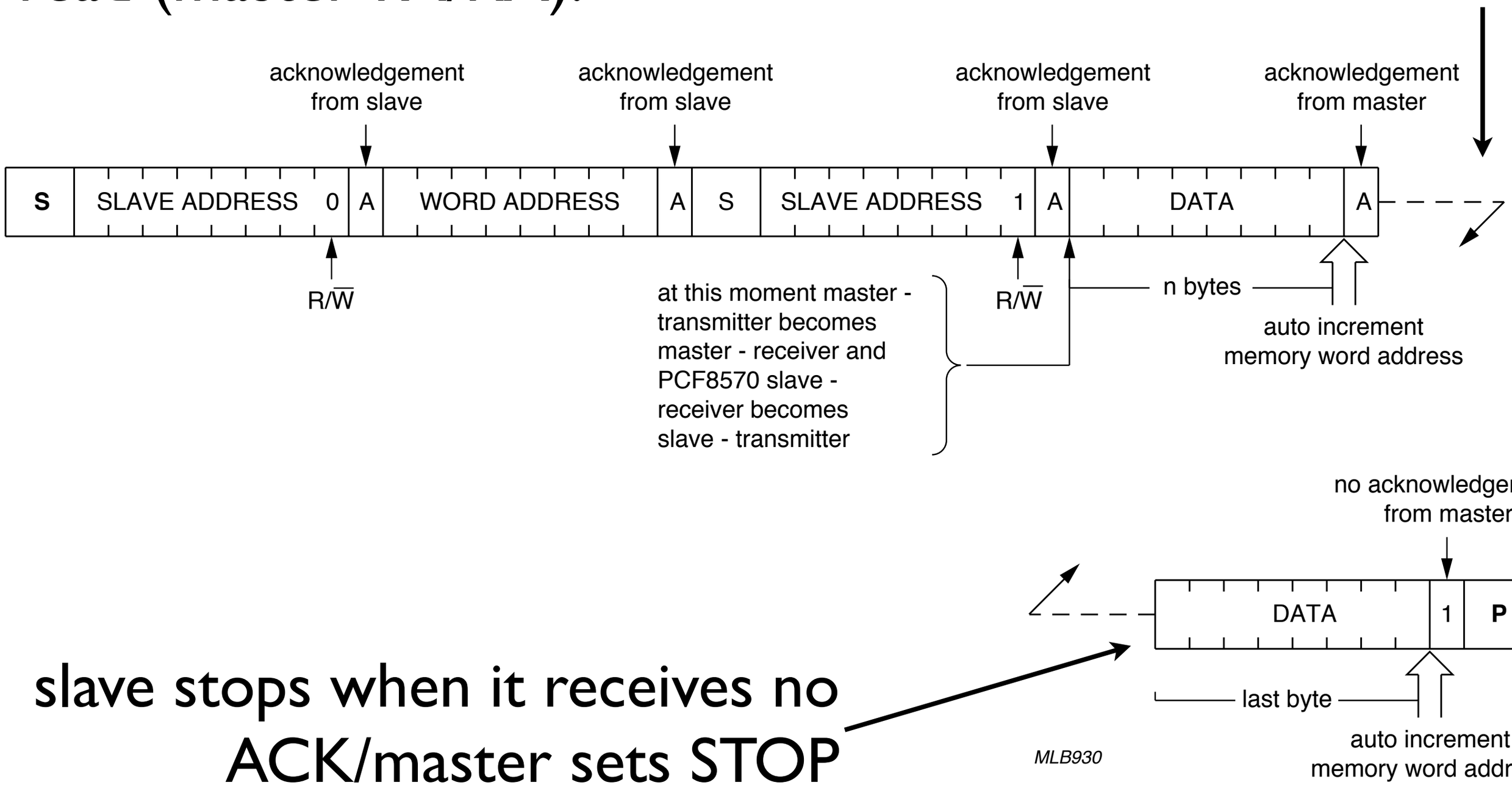
R12: slave addr

essentially: str R10,[R11]
(where memory is external)

LM3S1968 burst RX example

really only have to give start addr; can read consecutive bytes

read (master TX/RX):



slave stops when it receives no ACK/master sets STOP

read from PCF8570-style RAM

this routine performs read procedure
for PCF8570-style RAM
R10: where we put
R11: addr of byte
R12: slave addr
essentially: ldr R10,[R11]
(where memory is external)

```
I2C_MEM_RD
  ; 0. save return addr
  push {LR}

; 1. tx addr of byte to grab
mov R10,R11 ;TX addr of byte we want
bl I2C0_TX

; 2. rx byte at addr in R11 from slave
bl I2C0_RX

; 3. we're done: restore LR and return to main
pop {LR}
bx LR
```

re-use I2C0_TX:
R10: byte to write
R12: slave addr

re-use I2C0_RX:
R10: where to put byte
R12: slave addr