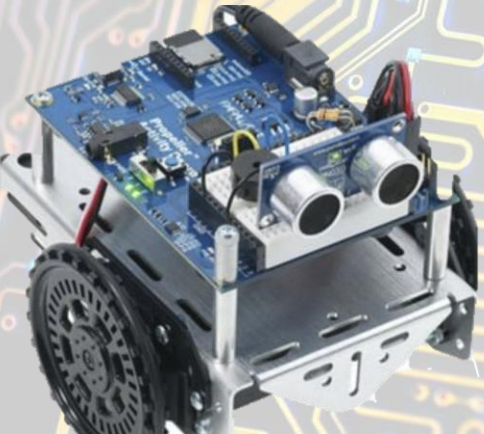
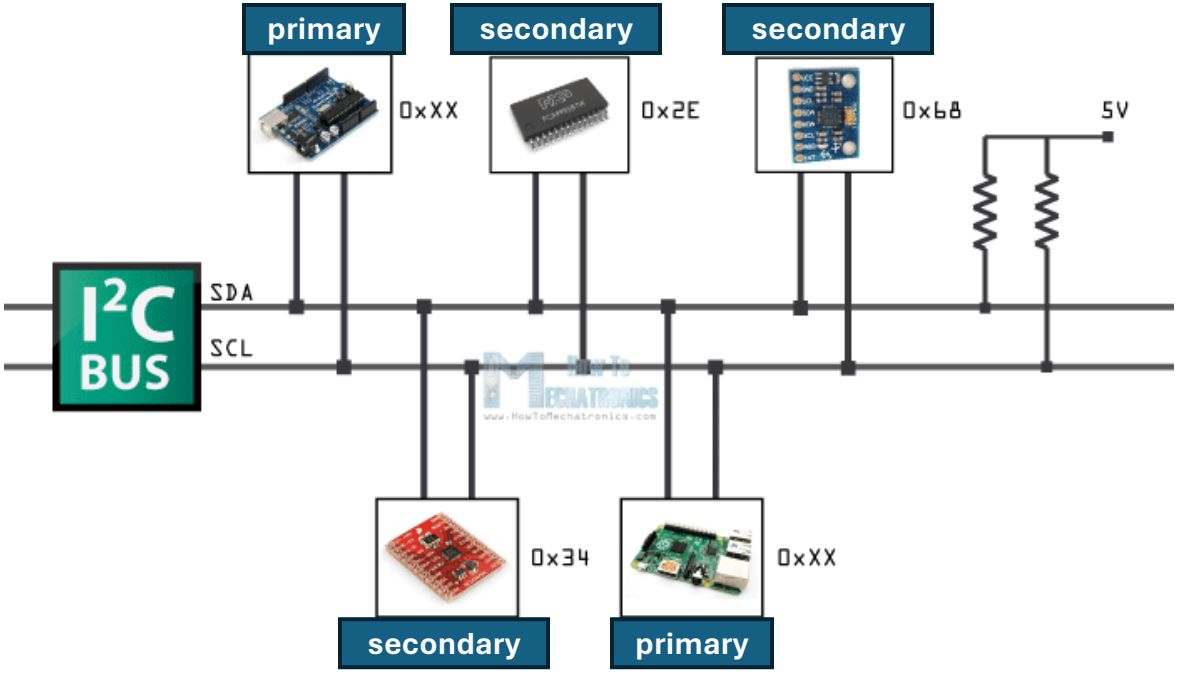
EC535 Introduction to Embedded Systems



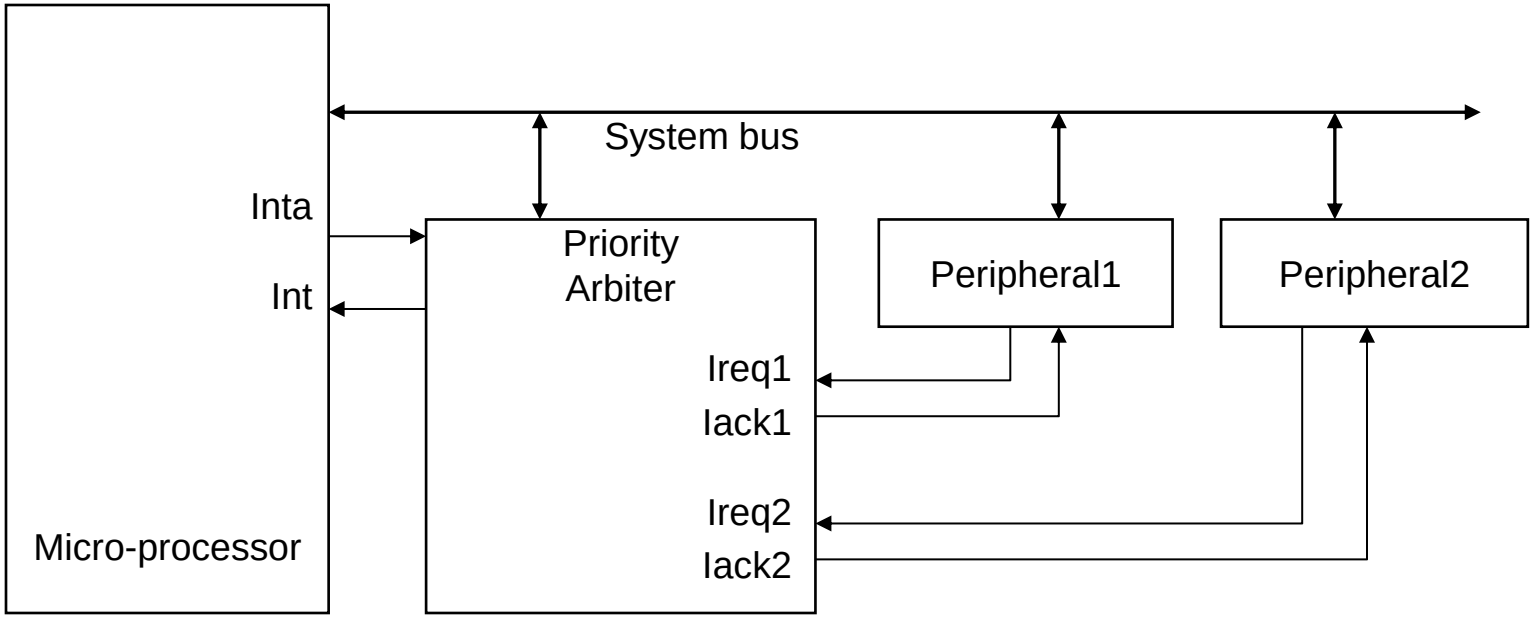
Serial protocols: I²C

- I²C (Inter-IC)
  - Two-wire serial bus protocol originally developed by Philips
  - Data transfer rates up to 100 kbits/s and 7-bit addressing
  - Common devices capable of interfacing to I²C bus:
    - EPROMS, Flash, LCD controller, some RAM memory, real-time clocks, watchdog timers, and microcontrollers



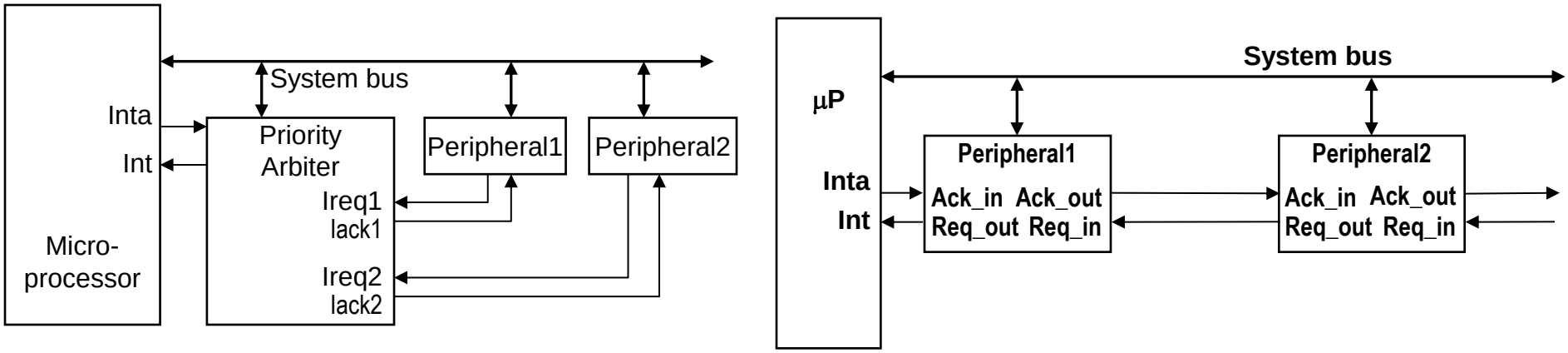
Arbitration: Priority arbiter

- Multiple peripherals request service from single resource (e.g., microprocessor, memory access controller) simultaneously.
- **Priority arbiter**
  - Single-purpose processor
  - Peripherals make requests to arbiter, arbiter makes requests to resource
  - Arbiter connected to system bus for ***configuration only***

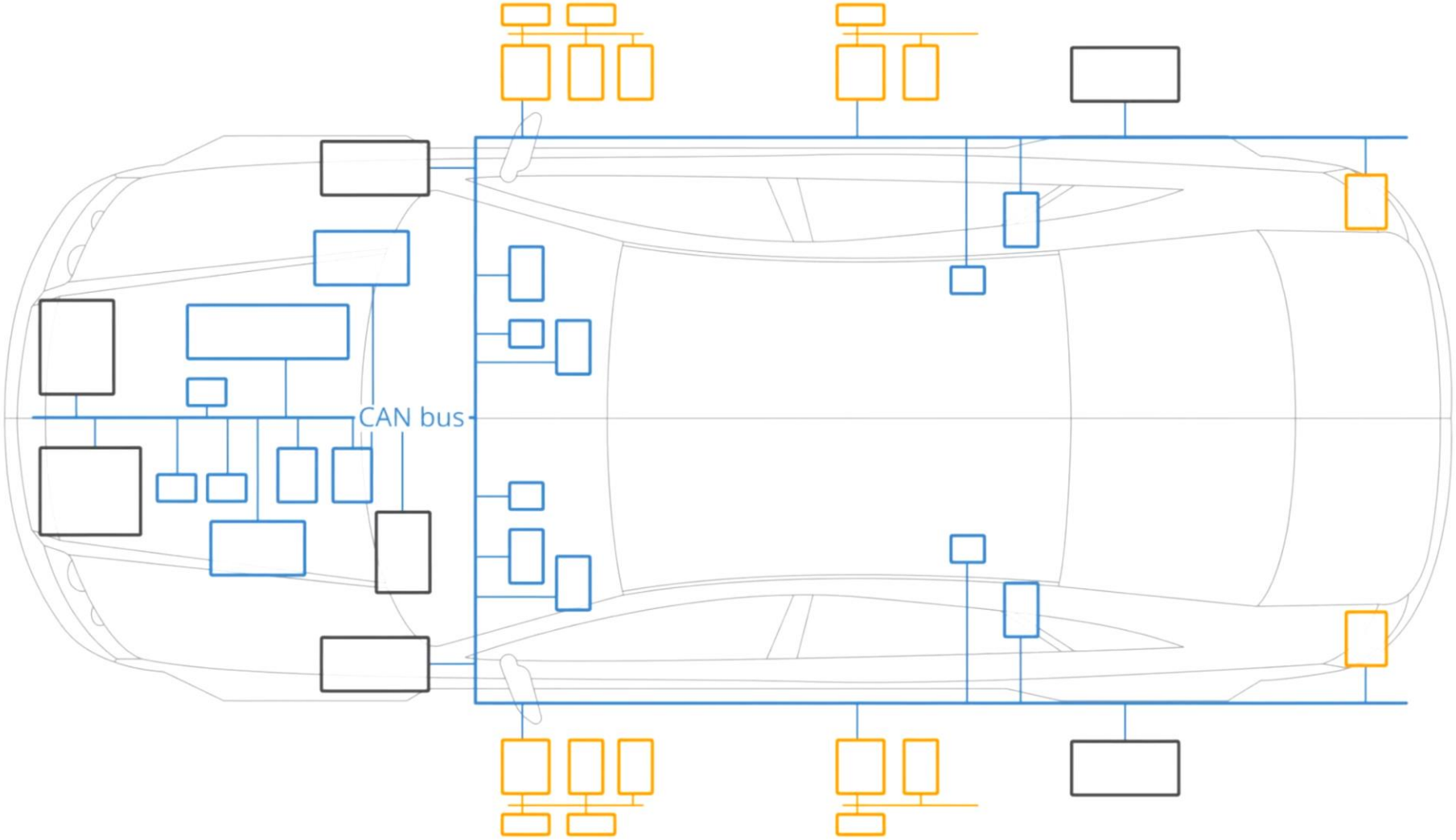


Arbitration: Daisy-chain arbitration

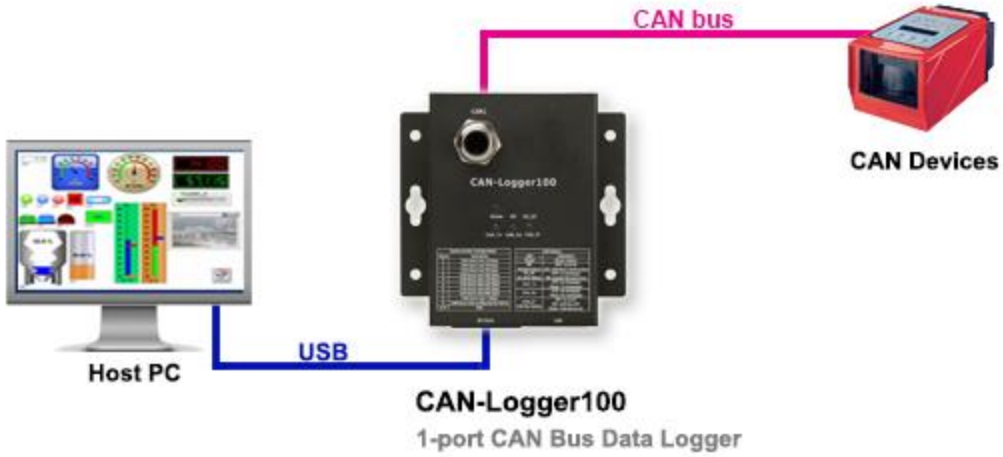
- Pros/cons
  - Easy to add/remove peripheral - no system redesign needed
  - Does not support rotating priority
  - One broken peripheral can cause loss of access to other peripherals

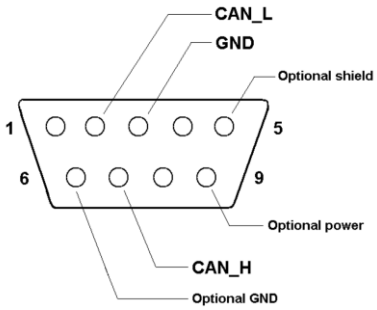


Another Serial Protocol: CAN



CAN Bus Logger





How to log CAN bus data

timestamps	ID [Hex]	DataBytes	BusChannel [He	IDE [Hex]	DLC [Hex]	DataLength
2020-01-13 16:30:50.712699890+02:00	14FF0121	FF FF FF FF FF FF FC FF	1	1	8	8
2020-01-13 16:30:50.714050055+02:00	C00000B	FC FF FA FA FF FF FF FF	1	1	8	8
2020-01-13 16:30:50.718549967+02:00	14FF3131	00 30 03 00 00 FF FF F0	1	1	8	8
2020-01-13 16:30:50.719899893+02:00	14FDA421	FF FF FF FF FF FF FF FF	1	1	8	8
2020-01-13 16:30:50.720599890+02:00	CF00400	01 7D 7D 3F 21 00 F4 7D	1	1	8	8
2020-01-13 16:30:50.721199989+02:00	18FCDC00	E0 FE FE FF FF FF FF FF	1	1	8	8
2020-01-13 16:30:50.722549915+02:00	14FF0221	FF FF FF FF FF FF FF FF	1	1	8	8
2020-01-13 16:30:50.725250006+02:00	CFF0281	44 03 7D 01 D2 00 61 00	1	1	8	8
2020-01-13 16:30:50.729899883+02:00	14FEF031	FF FF FF FF FF FC 00 FF	1	1	8	8
2020-01-13 16:30:50.730550051+02:00	CF00400	01 7D 7D 38 21 00 F4 7D	1	1	8	8
2020-01-13 16:30:50.731149912+02:00	18F0000F	00 7D FF FF 0F 7D FF FF	1	1	8	8
2020-01-13 16:30:50.734400034+02:00	CF00A01	D9 0E E8 1E FF FF FF FF	1	1	8	8
2020-01-13 16:30:50.734950066+02:00	10F01A01	C0 8F FF FF FF FF FF FF	1	1	8	8
2020-01-13 16:30:50.735549927+02:00	10FCFD01	FF FF FF FF D6 26 FF FF	1	1	8	8
2020-01-13 16:30:50.736099958+02:00	18FD9401	C4 07 FF FF E7 1F 02 02	1	1	8	8
2020-01-13 16:30:50.736700058+02:00	18FDA101	60 19 FF FF FF FF 00 FF	1	1	8	8
2020-01-13 16:30:50.737250090+02:00	10FDA301	FF FF FF FF FF FF FF FF	1	1	8	8

FF FF FF 68 13 …

How to log CAN bus data

timestamps	ID [Hex]	DataBytes	BusChannel [He	IDE [Hex]	DLC [Hex]	DataLength
2020-01-13 16:30:50.712699890+02:00	14FF0121	FF FF FF FF FF FF FC FF	1	1	8	8
2020-01-13 16:30:50.714050055+02:00	C00000B	FC FF FA FA FF FF FF FF	1	1	8	8
2020-01-13 16:30:50.718549967+02:00	14FF3131	00 30 03 00 00 FF FF F0	1	1	8	8
2020-01-13 16:30:50.719899893+02:00	14FDA421	FF FF FF FF FF FF FF FF	1	1	8	8
2020-01-13 16:30:50.720599890+02:00	CF00400	01 7D 7D 3F 21 00 F4 7D	1	1	8	8
2020-01-13 16:30:50.721199989+02:00	18FCDC00	E0 FE FE FF FF FF FF FF	1	1	8	8
2020-01-13 16:30:50.722549915+02:00	14FF0221	FF FF FF FF FF FF FF FF	1	1	8	8
2020-01-13 16:30:50.725250006+02:00	CFF0281	44 03 7D 01 D2 00 61 00	1	1	8	8
2020-01-13 16:30:50.729899883+02:00	14FEF031	FF FF FF FF FF FC 00 FF	1	1	8	8
2020-01-13 16:30:50.730550051+02:00	CF00400	01 7D 7D 38 21 00 F4 7D	1	1	8	8
2020-01-13 16:30:50.731149912+02:00	18F0000F	00 7D FF FF 0F 7D FF FF	1	1	8	8
2020-01-13 16:30:50.734400034+02:00	CF00A01	D9 0E E8 1E FF FF FF FF	1	1	8	8
2020-01-13 16:30:50.734950066+02:00	10F01A01	C0 8F FF FF FF FF FF FF	1	1	8	8
2020-01-13 16:30:50.735549927+02:00	10FCFD01	FF FF FF FF D6 26 FF FF	1	1	8	8
2020-01-13 16:30:50.736099958+02:00	18FD9401	C4 07 FF FF E7 1F 02 02	1	1	8	8
2020-01-13 16:30:50.736700058+02:00	18FDA101	60 19 FF FF FF FF 00 FF	1	1	8	8
2020-01-13 16:30:50.737250090+02:00	10FDA301	FF FF FF FF FF FF FF FF	1	1	8	8

FF FF FF 68 13 ...

Could be one value out of multiple,
e.g., right wheel speed, engine speed

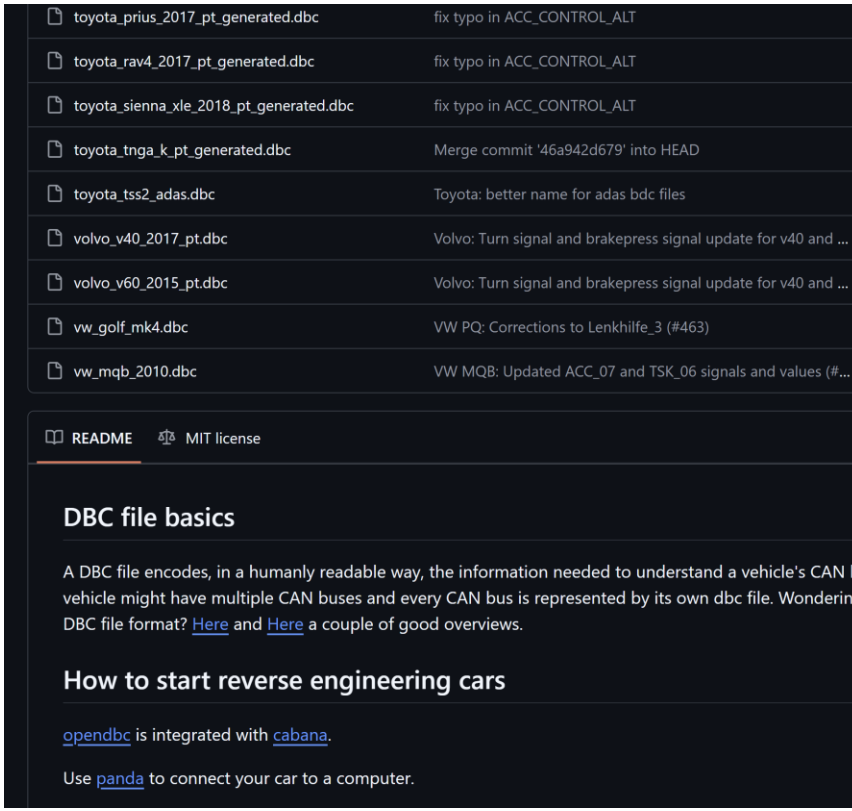
How to log CAN bus data

timestamps	ID [Hex]	DataBytes	BusChannel [He	IDE [Hex]	DLC [Hex]	DataLength
2020-01-13 16:30:50.712699890+02:00	14FF0121	FF FF FF FF FF FF FC FF	1	1	8	8
2020-01-13 16:30:50.714050055+02:00	C00000B	FC FF FA FA FF FF FF FF	1	1	8	8
2020-01-13 16:30:50.718549967+02:00	14FF3131	00 30 03 00 00 FF FF F0	1	1	8	8
2020-01-13 16:30:50.719899893+02:00	14FDA421	FF FF FF FF FF FF FF FF	1	1	8	8
2020-01-13 16:30:50.720599890+02:00	CF00400	01 7D 7D 3F 21 00 F4 7D	1	1	8	8
2020-01-13 16:30:50.721199989+02:00	18FCDC00	E0 FE FE FF FF FF FF FF	1	1	8	8
2020-01-13 16:30:50.722549915+02:00	14FF0221	FF FF FF FF FF FF FF FF	1	1	8	8
2020-01-13 16:30:50.725250006+02:00	CFF0281	44 03 7D 01 D2 00 61 00	1	1	8	8
2020-01-13 16:30:50.729899883+02:00	14FEF031	FF FF FF FF FF FC 00 FF	1	1	8	8
2020-01-13 16:30:50.730550051+02:00	CF00400	01 7D 7D 38 21 00 F4 7D	1	1	8	8
2020-01-13 16:30:50.731149912+02:00	18F0000F	00 7D FF FF 0F 7D FF FF	1	1	8	8
2020-01-13 16:30:50.734400034+02:00	CF00A01	D9 0E E8 1E FF FF FF FF	1	1	8	8
2020-01-13 16:30:50.734950066+02:00	10F01A01	C0 8F FF FF FF FF FF FF	1	1	8	8
2020-01-13 16:30:50.735549927+02:00	10FCFD01	FF FF FF FF D6 26 FF FF	1	1	8	8
2020-01-13 16:30:50.736099958+02:00	18FD9401	C4 07 FF FF E7 1F 02 02	1	1	8	8
2020-01-13 16:30:50.736700058+02:00	18FDA101	60 19 FF FF FF FF 00 FF	1	1	8	8
2020-01-13 16:30:50.737250090+02:00	10FDA301	FF FF FF FF FF FF FF FF	1	1	8	8

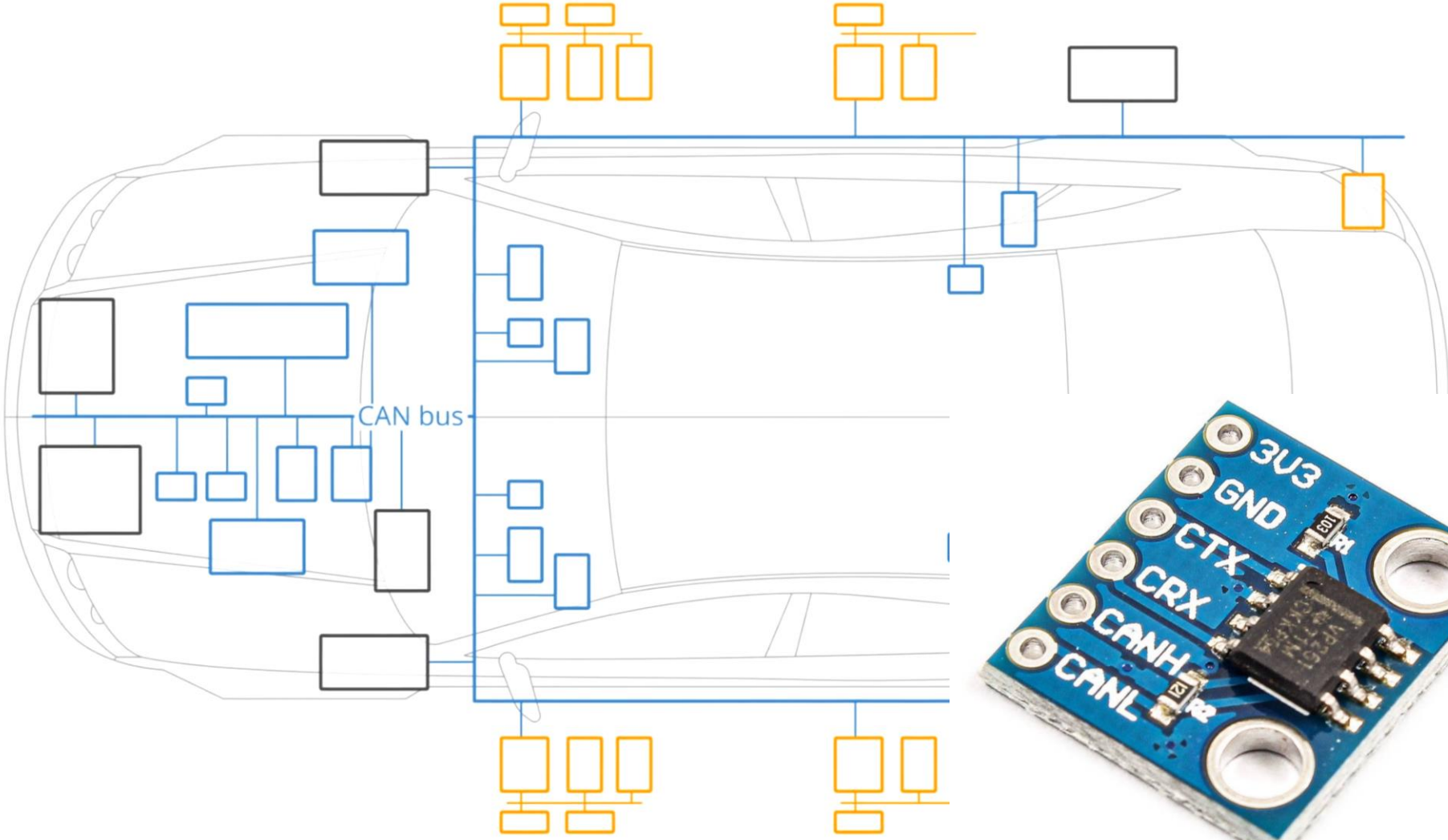
FF FF FF 68 13 …

Could be one value out of multiple, e.g., right wheel speed, engine speed

https://github.com/BogGyver/opendbc

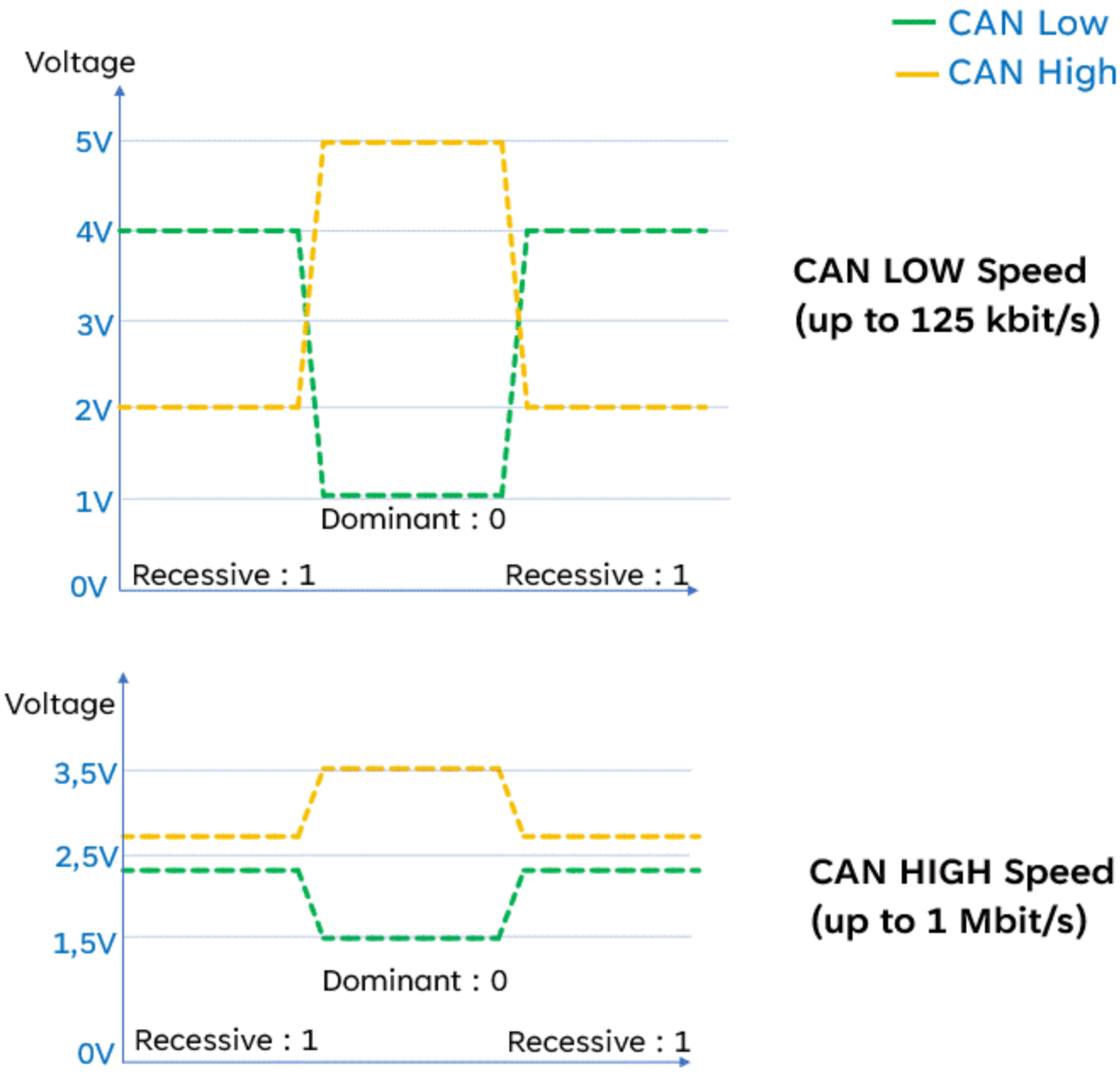


Another Serial Protocol: CAN



CAN uses a wired-AND bus with open-drain (or open-collector) outputs:

- Each node can pull the line low, but not drive it high
- If no one pulls low, the bus floats high using resistors
- If any node pulls low, the bus is low — regardless of what others do


CAN Low
CAN High
Voltage
5V
4V
3V
2V
1V
0V
Dominant : 0
Recessive : 1
Recessive : 1
CAN LOW Speed
(up to 125 kbit/s)
Voltage
3,5V
2,5V
1,5V
Dominant : 0
0V
Recessive : 1
Recessive : 1
CAN HIGH Speed
(up to 1 Mbit/s)

Physical Layer Synchronization

- Nodes synchronize to each other by watching the **edges** (transitions) on the CAN bus.

- The recessive-to-dominant edge at the **Start of Frame (SOF)** is used to trigger bit timing alignment.

Physical Layer Synchronization

- Nodes synchronize to each other by watching the **edges** (transitions) on the CAN bus.

- The recessive-to-dominant edge at the **Start of Frame (SOF)** is used to trigger bit timing alignment.

In contrast to I^2C:

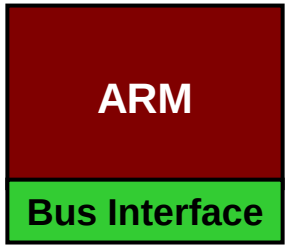
- Assumes **one device controls timing**

- Doesn't use **differential signaling**, so it's more sensitive to noise (can't reliably extract timing just from edges like CAN does)

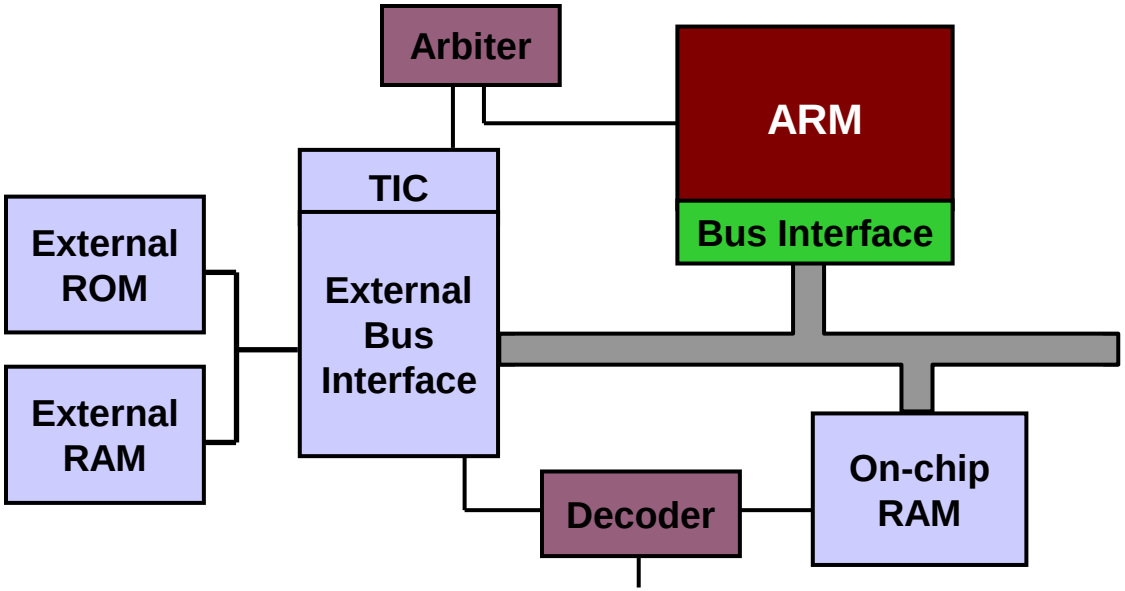
A Parallel Protocol: AMBA Bus

Typical ARM-based System

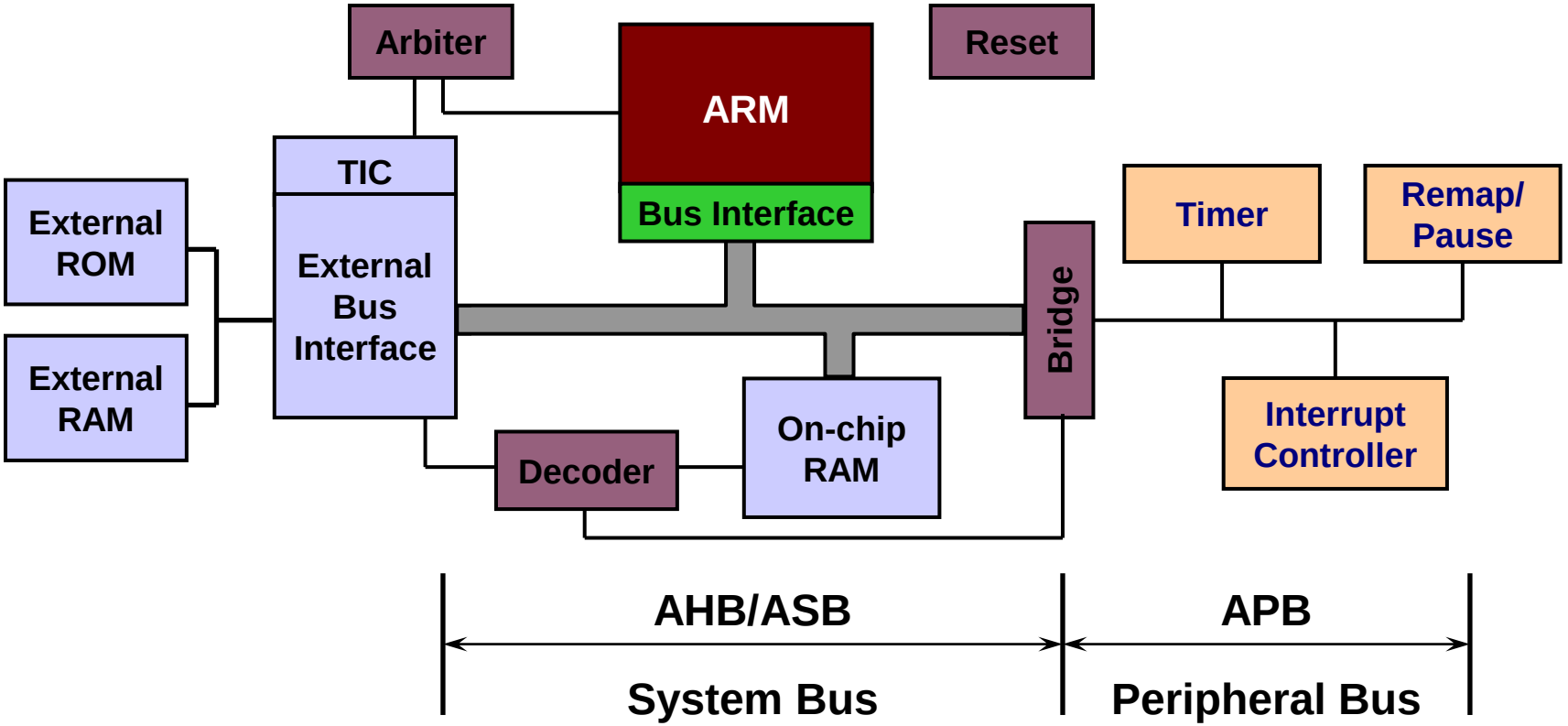
Typical ARM-based System



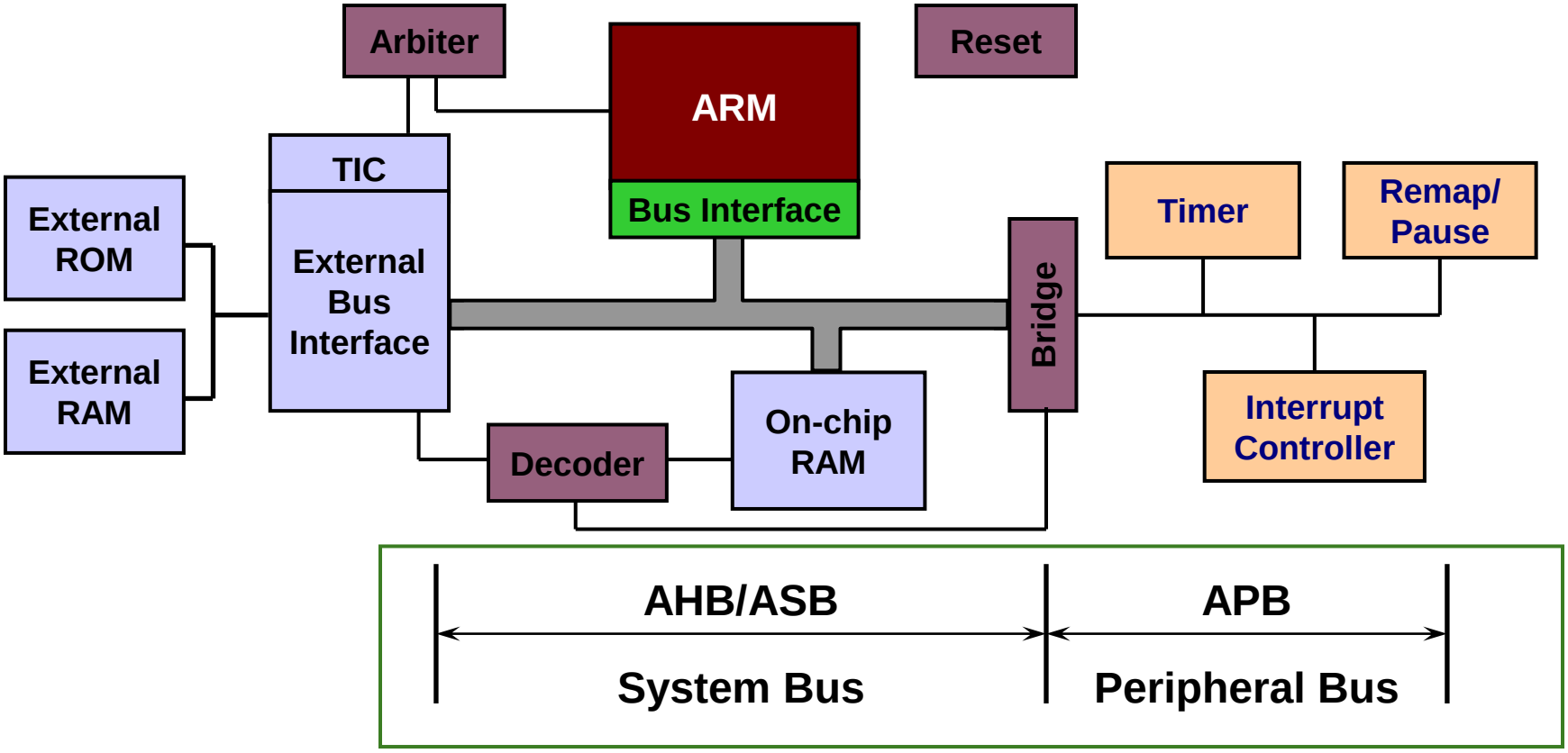
Typical ARM-based System



Typical ARM-based System



Typical ARM-based System



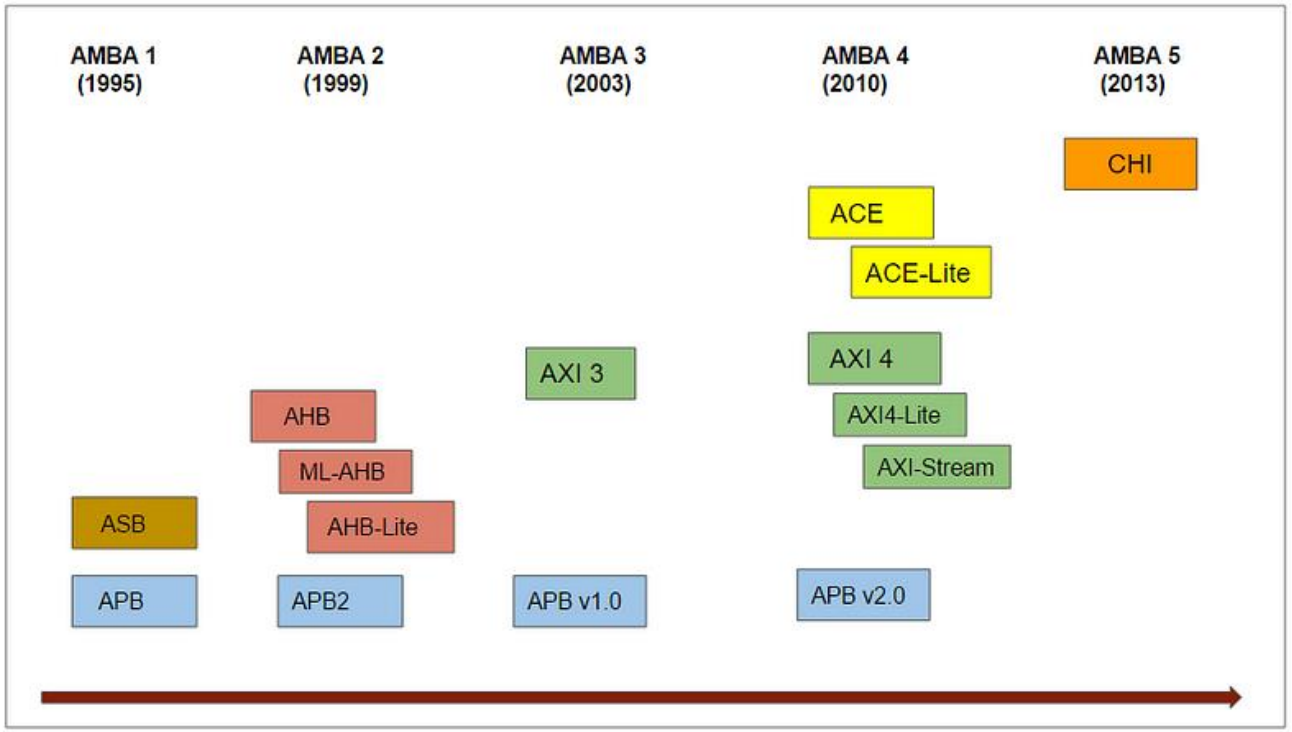
AMBA Protocol

A Parallel Protocol: AMBA Bus

- Advanced Microcontroller Bus Architecture (AMBA)
    - De facto standard in embedded processors
- AMBA is ARM’s on-chip bus specification.

A Parallel Protocol: AMBA Bus

- Advanced Microcontroller Bus Architecture (AMBA)
  - De facto standard in embedded processors
- AMBA is ARM's on-chip bus specification.



AMBA Sub-protocols

- **Advanced High-performance Bus (AHB)**
  - For high-performance, high clock frequency system modules.
  - All signal names start with H.

High Performance

AMBA Sub-protocols

- **Advanced High-performance Bus (AHB)**
  - For high-performance, high clock frequency system modules.
  - All signal names start with H.
- **Advanced System Bus (ASB)**
  - Simplified version of AHB.
  - Most signal names start with B.

High Performance

AMBA Sub-protocols

- **Advanced High-performance Bus (AHB)**
  - For high-performance, high clock frequency system modules.
  - All signal names start with H.
- **Advanced System Bus (ASB)**
  - Simplified version of AHB.
  - Most signal names start with B.
- **Advanced Peripheral Bus (APB)**
  - The AMBA APB is for low-power peripherals.
  - All signal names start with P.

High Performance

Minimal

AMBA Sub-protocols

- **Advanced High-performance Bus (AHB)**
  - For high-performance, high clock frequency system modules.
  - All signal names start with H.
- **Advanced System Bus (ASB)**
  - Simplified version of AHB.
  - Most signal names start with B.
- **Advanced Peripheral Bus (APB)**
  - The AMBA APB is for low-power peripherals.
  - All signal names start with P.

High Performance

Minimal

Advanced High-performance Bus (AHB)

- Recall I^2C and CAN
  - Address/data/control share one wire
  - Primary and secondary share one wire
  - Need to detect and avoid collision

	I2C	
Electrical Sharing	Yes	
Logical bus sharing		
Arbitration		

Advanced High-performance Bus (AHB)

- Recall I^2C and CAN
  - Address/data/control share one wire
  - Primary and secondary share one wire
  - Need to detect and avoid collision

	I2C	
Electrical Sharing	Yes	
Logical bus sharing	Yes	
Arbitration		

Advanced High-performance Bus (AHB)

- Recall I^2C and CAN
  - Address/data/control share one wire
  - Primary and secondary share one wire
  - Need to detect and avoid collision

	I2C	
Electrical Sharing	Yes	
Logical bus sharing	Yes	
Arbitration	Bitwise conflict detection (wired-AND)	

Advanced High-performance Bus (AHB)

- Recall I^2C and CAN
  - Address/data/control share one wire
  - Primary and secondary share one wire
  - Need to detect and avoid collision

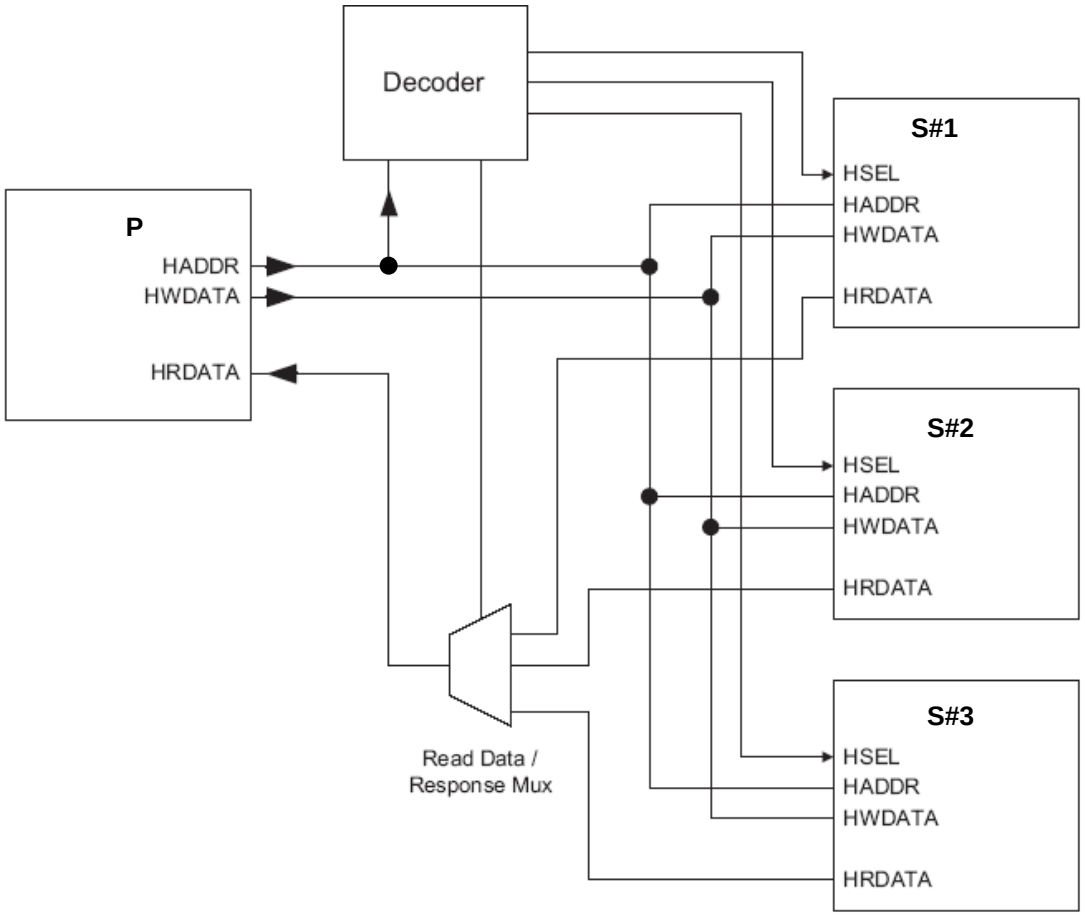
	I2C	AHB
Electrical Sharing	Yes	No (only one drives, unidirectional)
Logical bus sharing	Yes	Yes
Arbitration	Bitwise conflict detection (wired-AND)	Central arbiter selects

Advanced High-performance Bus (AHB)

- Recall I^2C and CAN
  - Address/data/control share one wire
  - Primary and secondary share one wire
  - Need to detect and avoid collision
- Wires not shared
  - ***All wires are unidirectional, point-to-point***
- Components
  - ***Primary*** — Initiates the transaction
  - ***Secondary*** — Responds to the transaction
  - ***Arbiter*** — Resolves priority
  - ***Decoder*** — Selects secondary and steers signal flow

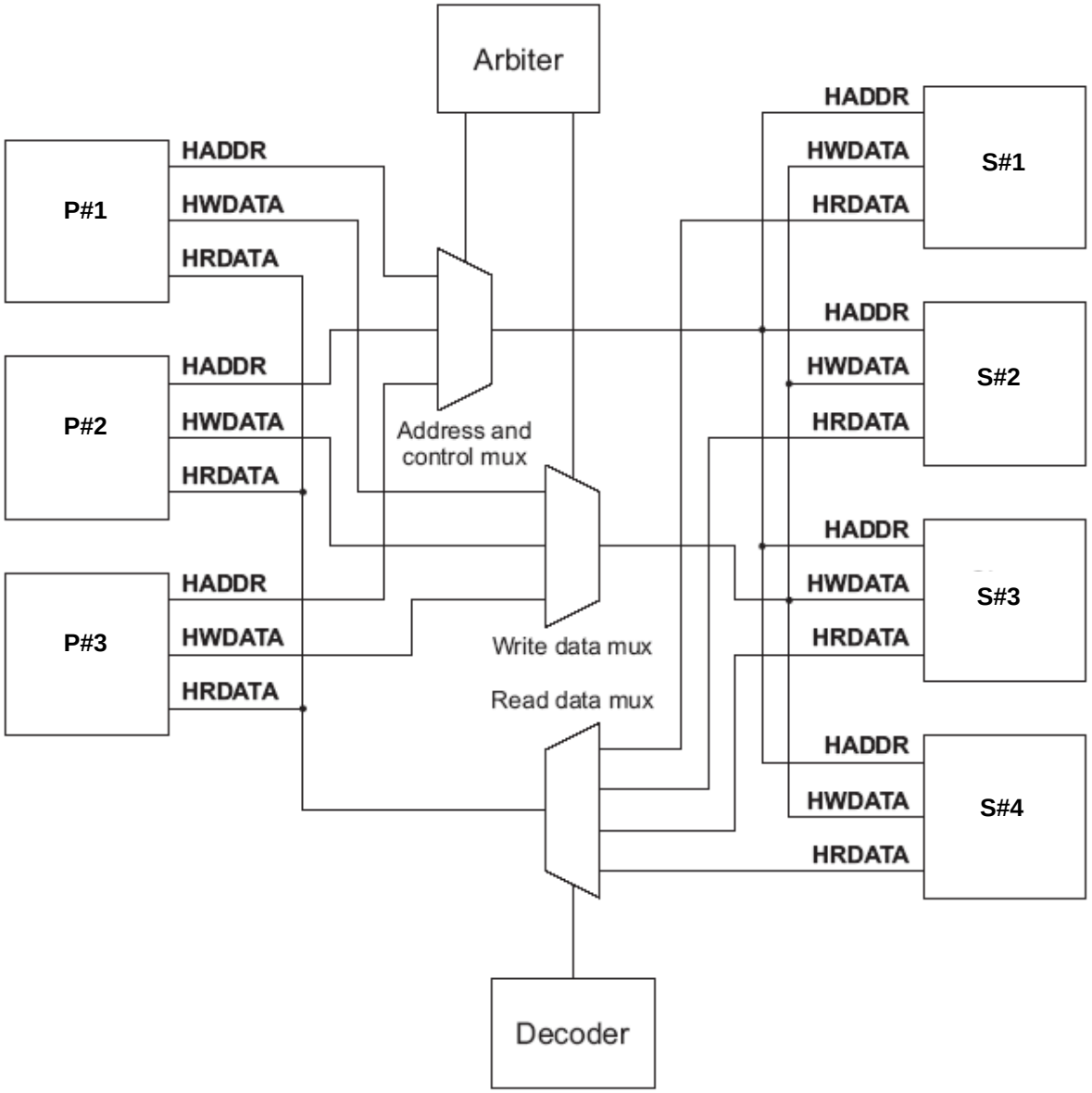
First, AHB-Lite

Single Primary: no arbiter needed



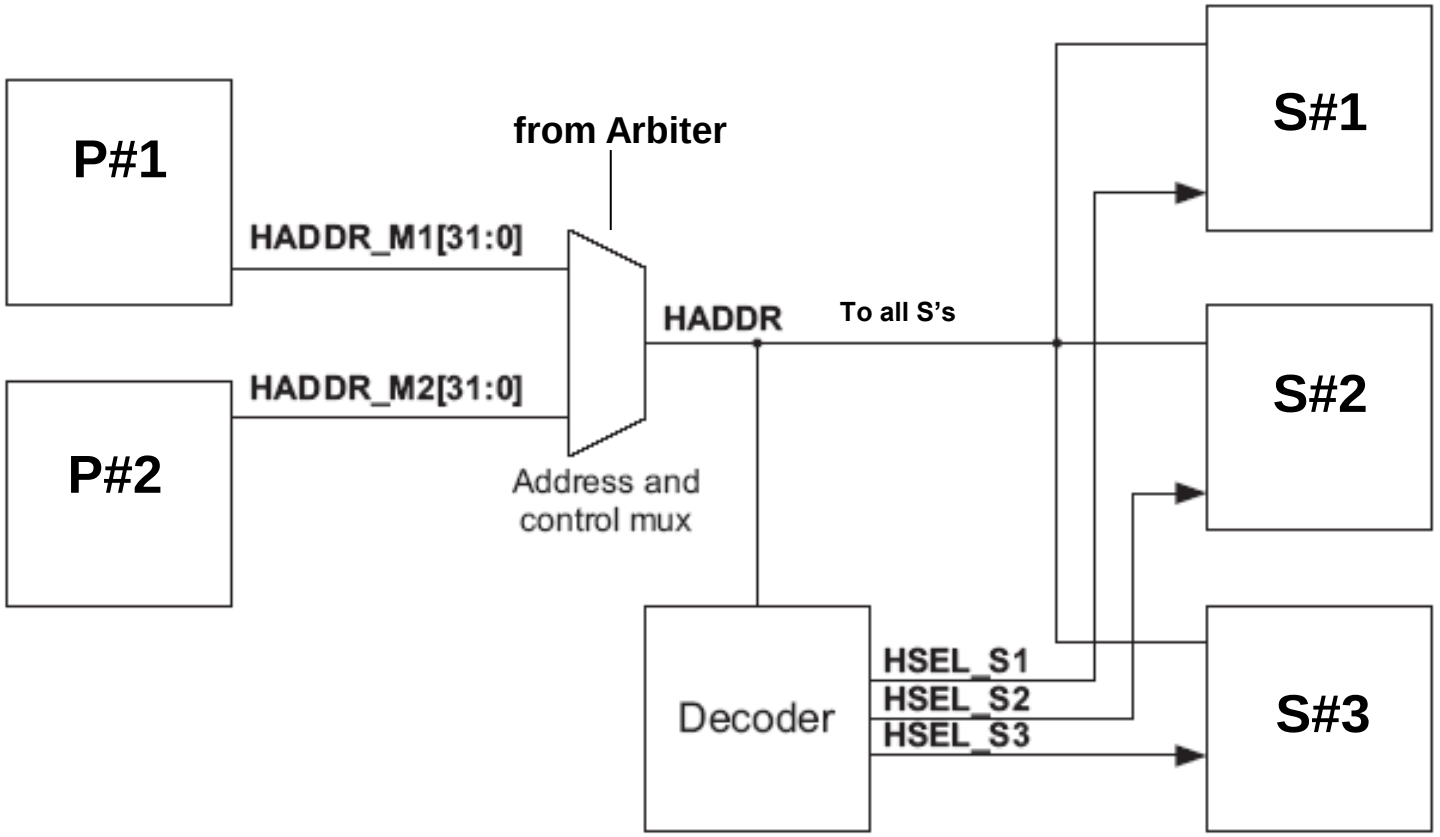
Full AHB

- Multiple primaries
- Arbiter controls address/data multiplexers



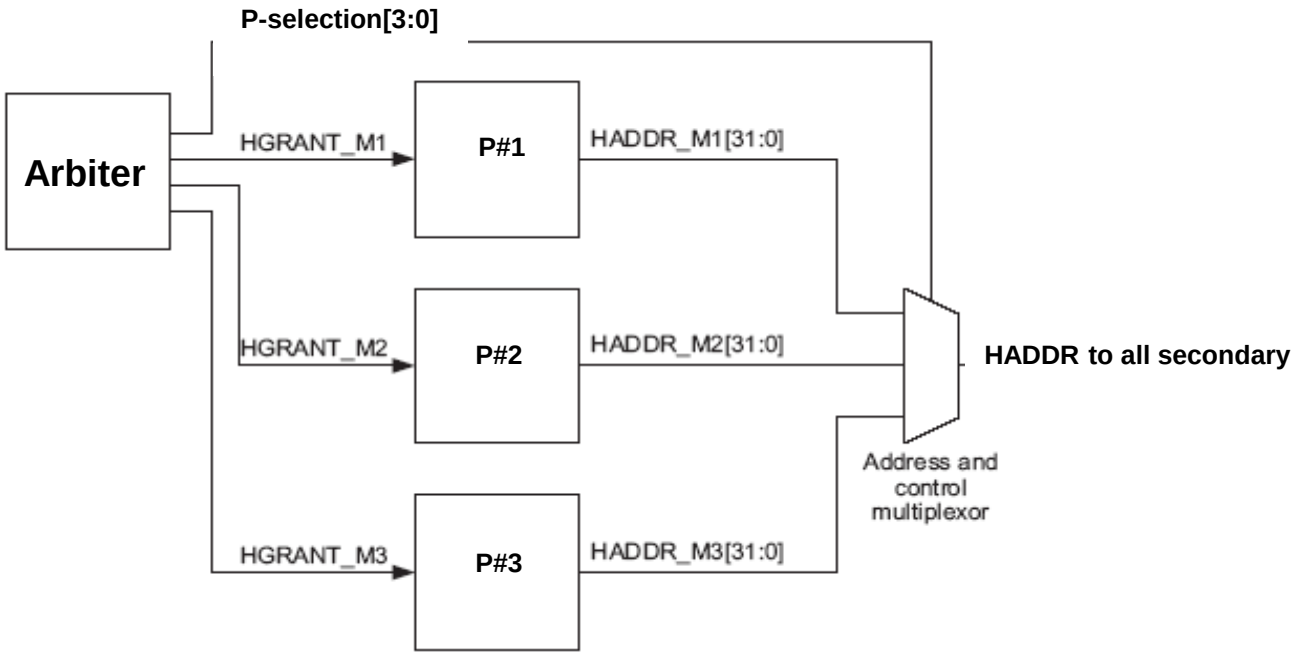
Address Decoding

- Decoder
  - Input: HADDR[31:0]
  - Output: HSELx



Arbitration

- Primary units send request to arbiter
- Arbiter grants access by asserting HGRANTx signals

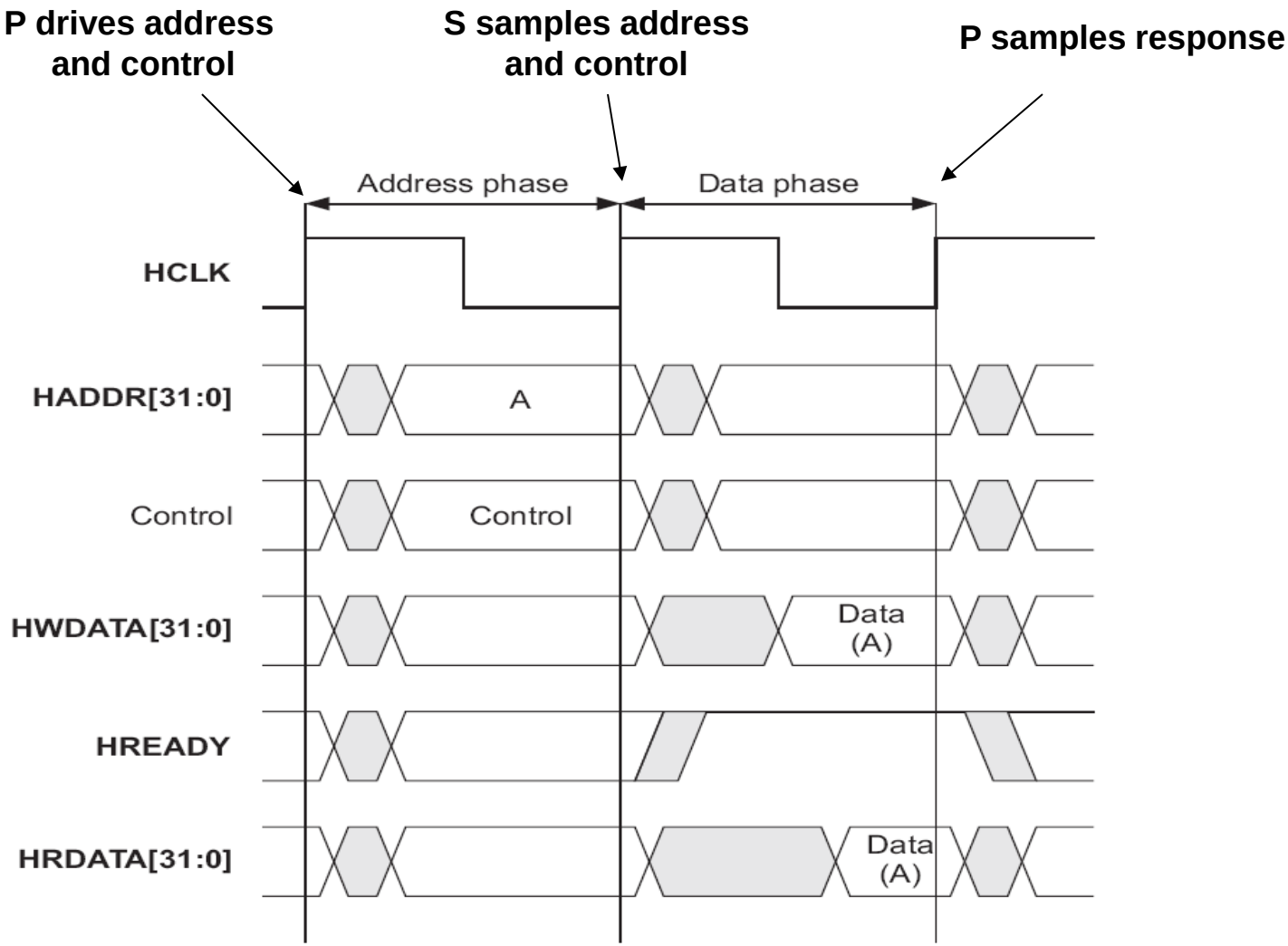


HBUSREQx from primary units not shown

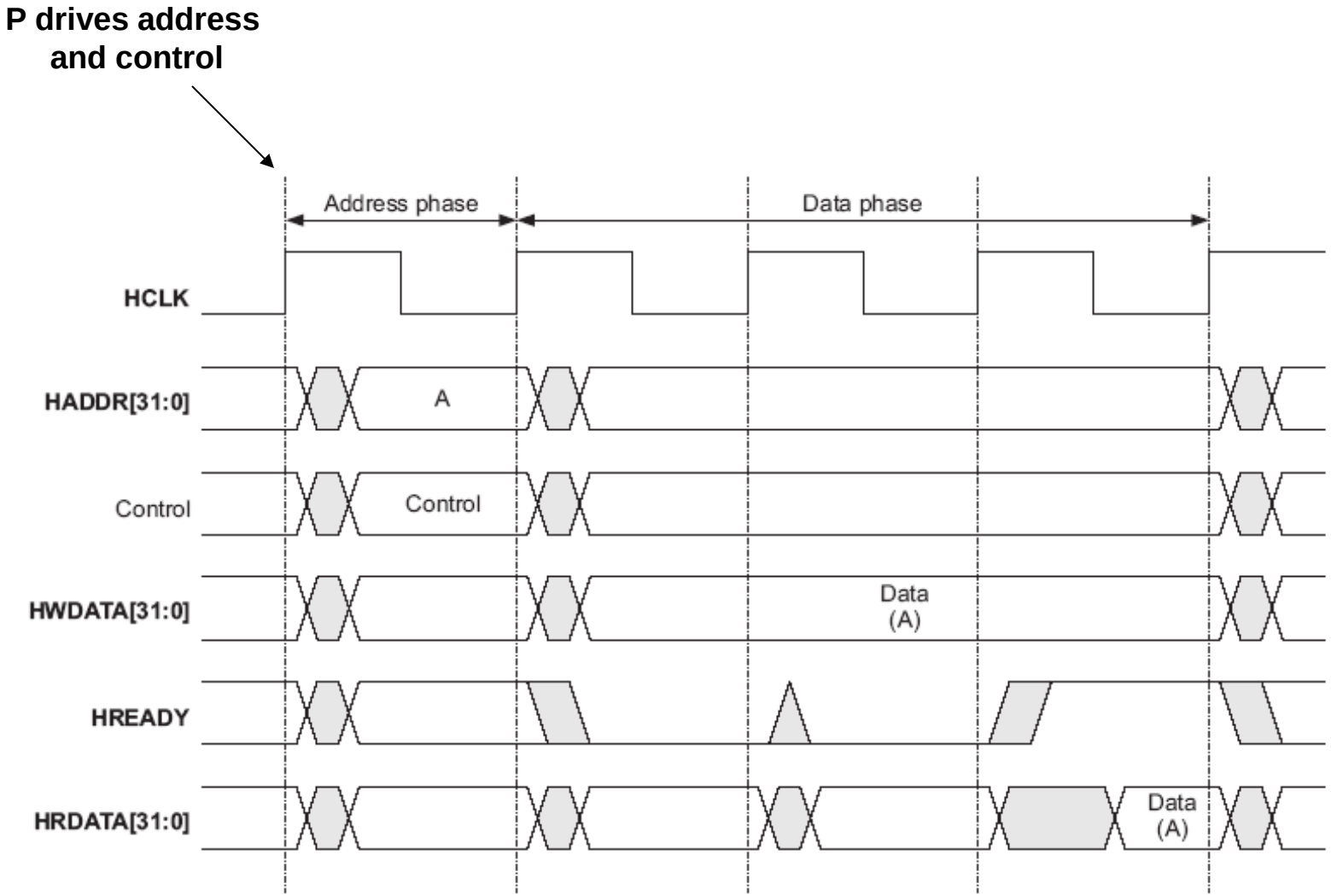
AHB Main Signals

name	source	
HCLK	Clock source	Global clock for all bus transactions
HADDR[31:0]	Primary	32bit address line
HWRITE	P	HI — write, LOW — read
HSIZE[2:0]	P	Transfer size (8bits – 1024bits)
HWDATA[31:0]	P	Data to write
HSELx	Decoder	Selects intended secondary
HREADY	Secondary	HI — a transfer finishes, LOW — wait
HRDATA[31:0]	S	Data to read by primary

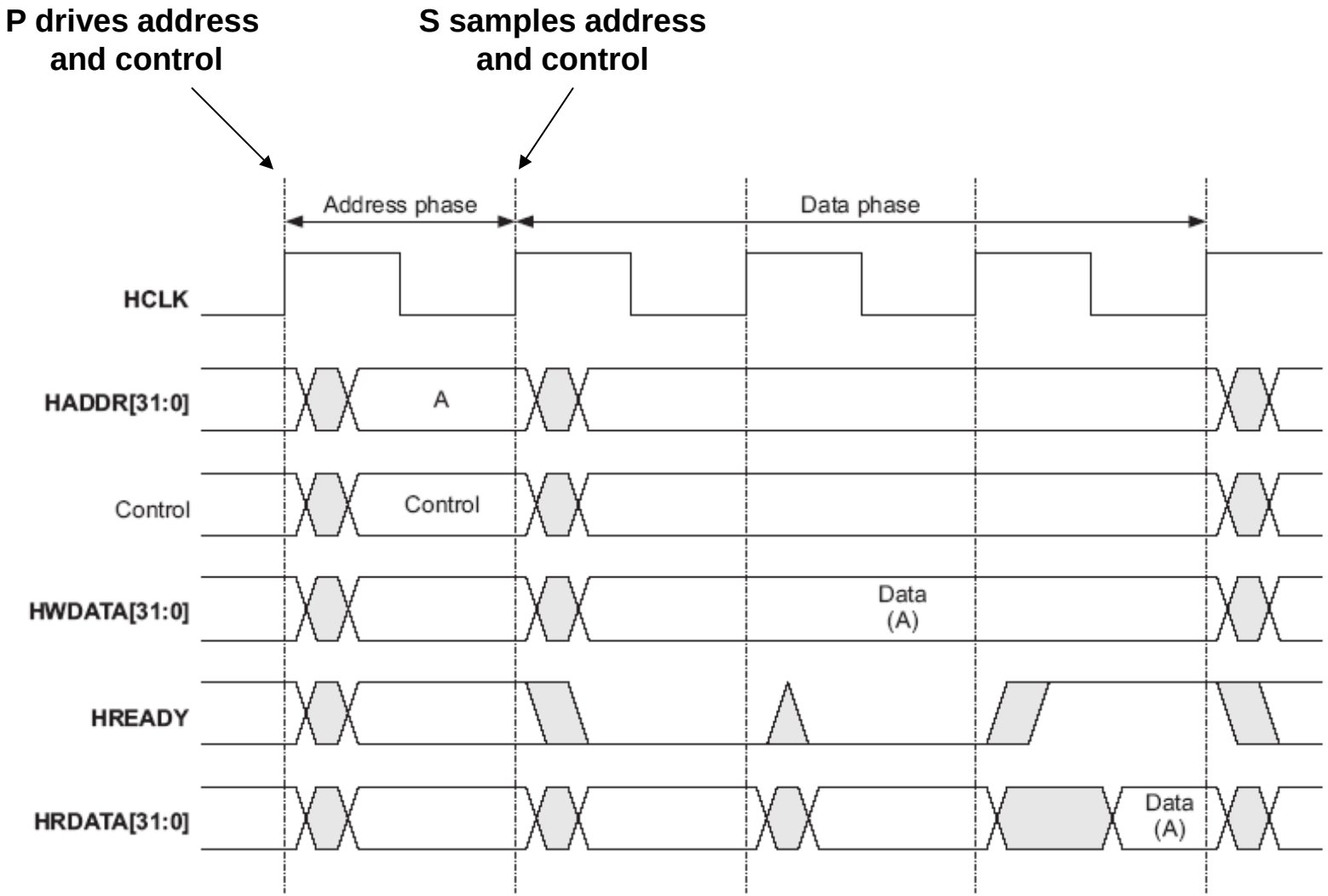
A Simple Transfer



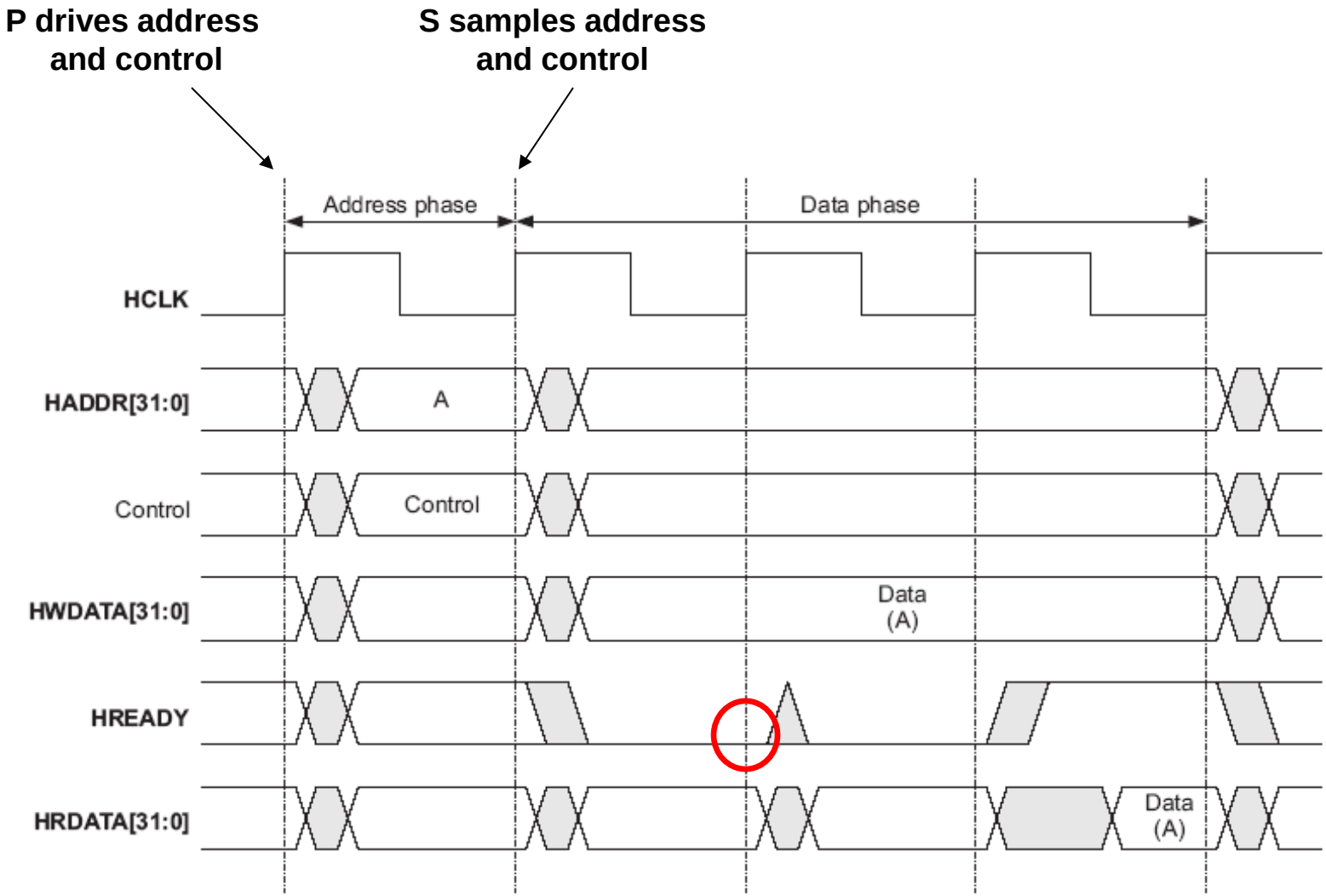
Transfer with Wait States



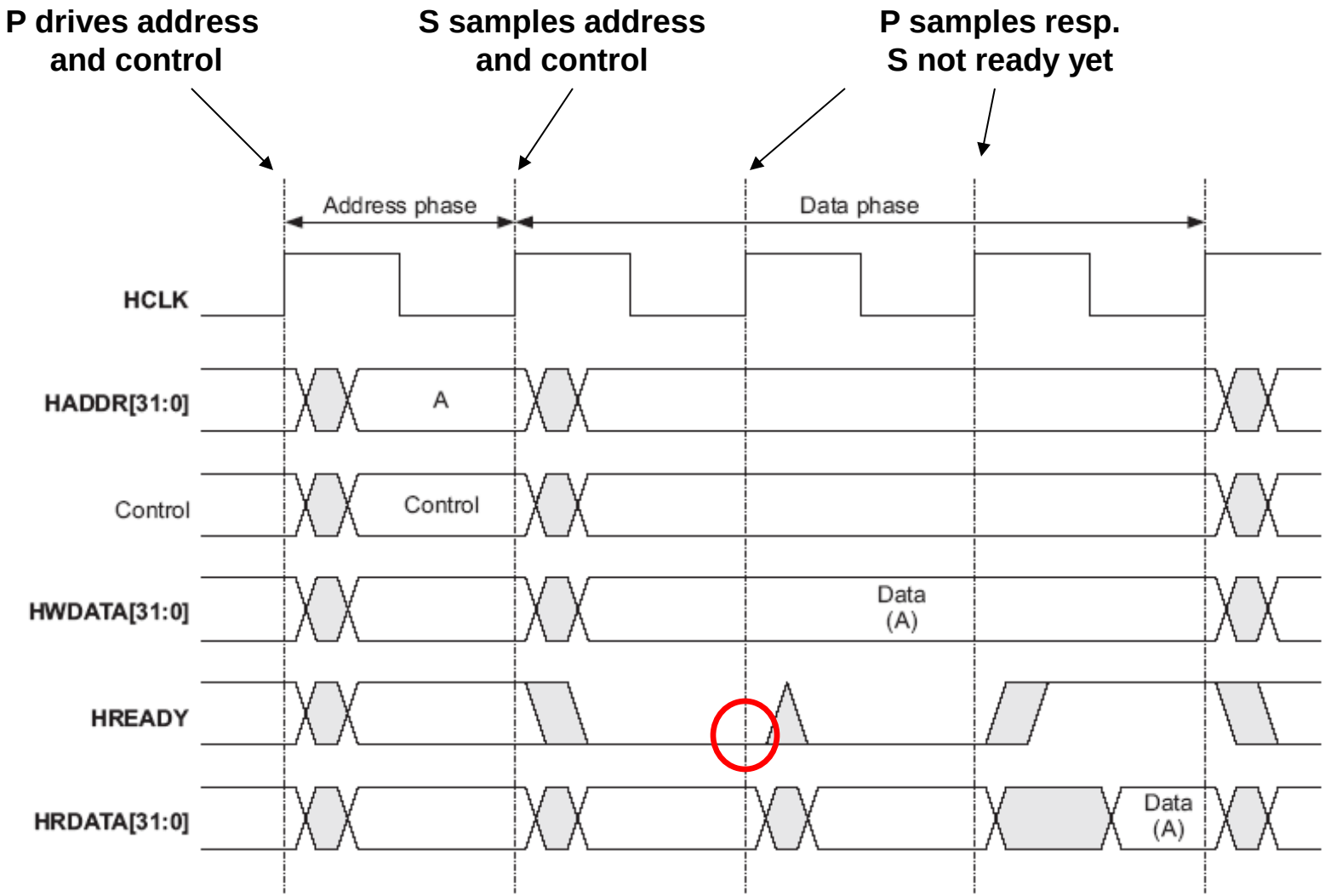
Transfer with Wait States



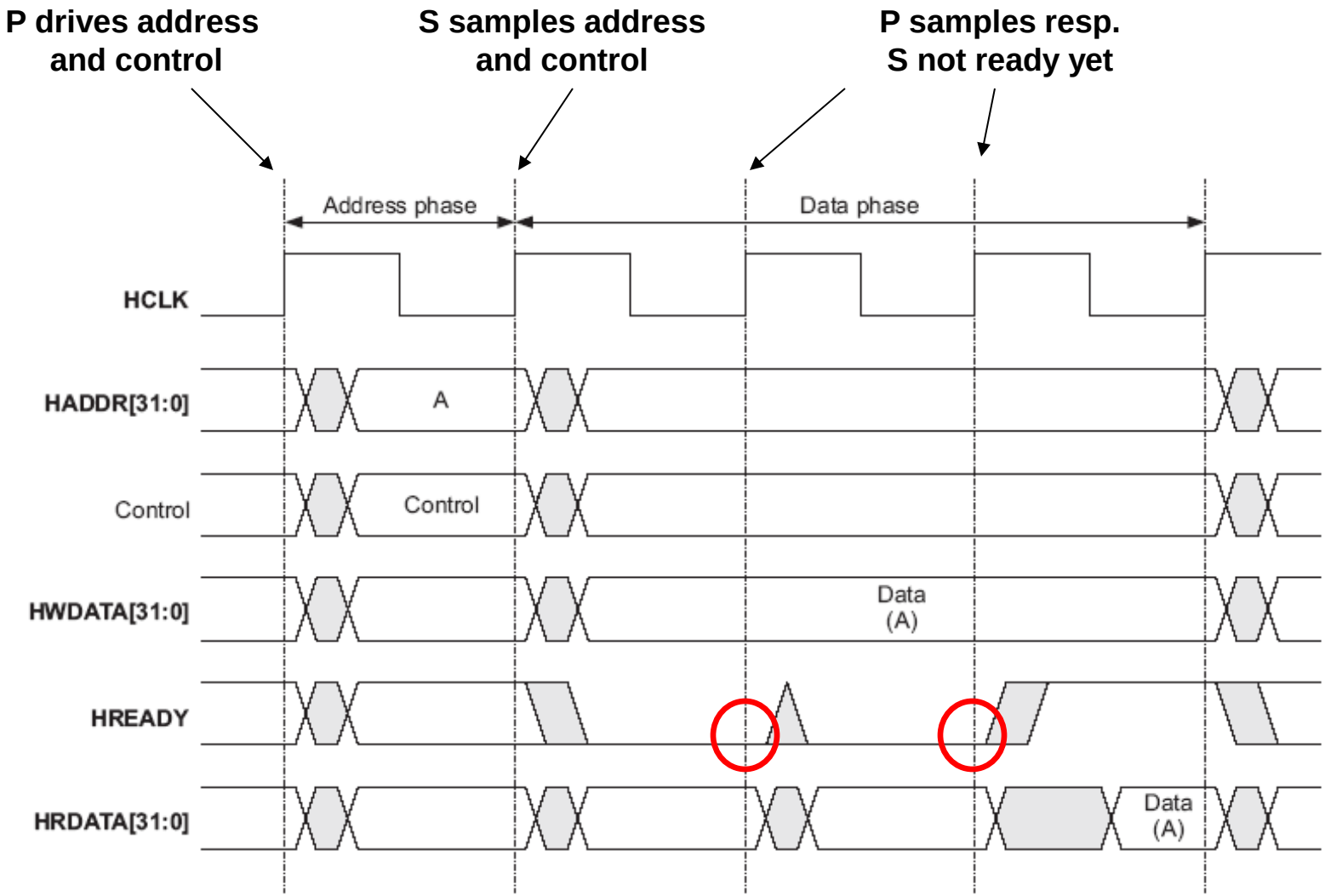
Transfer with Wait States



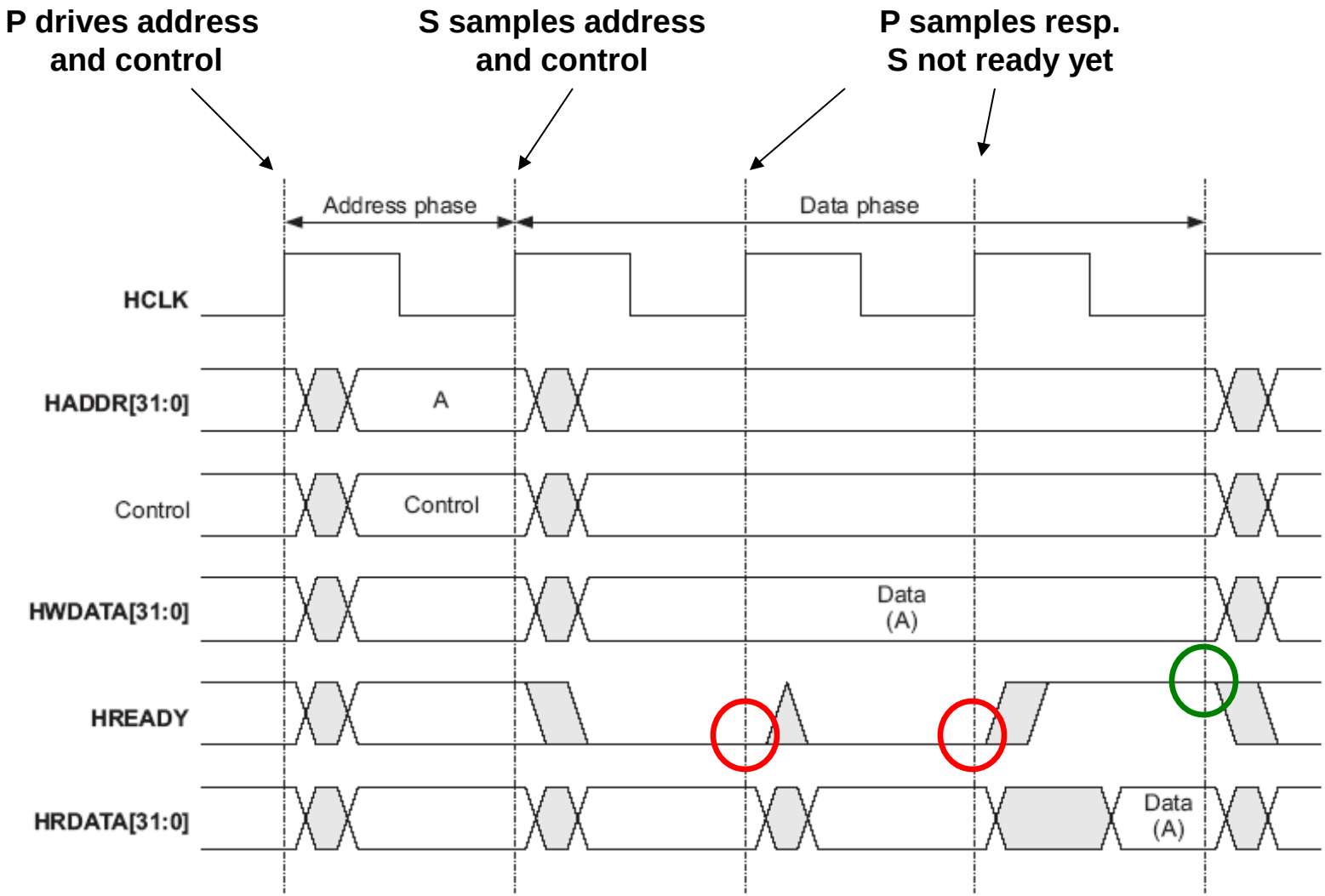
Transfer with Wait States



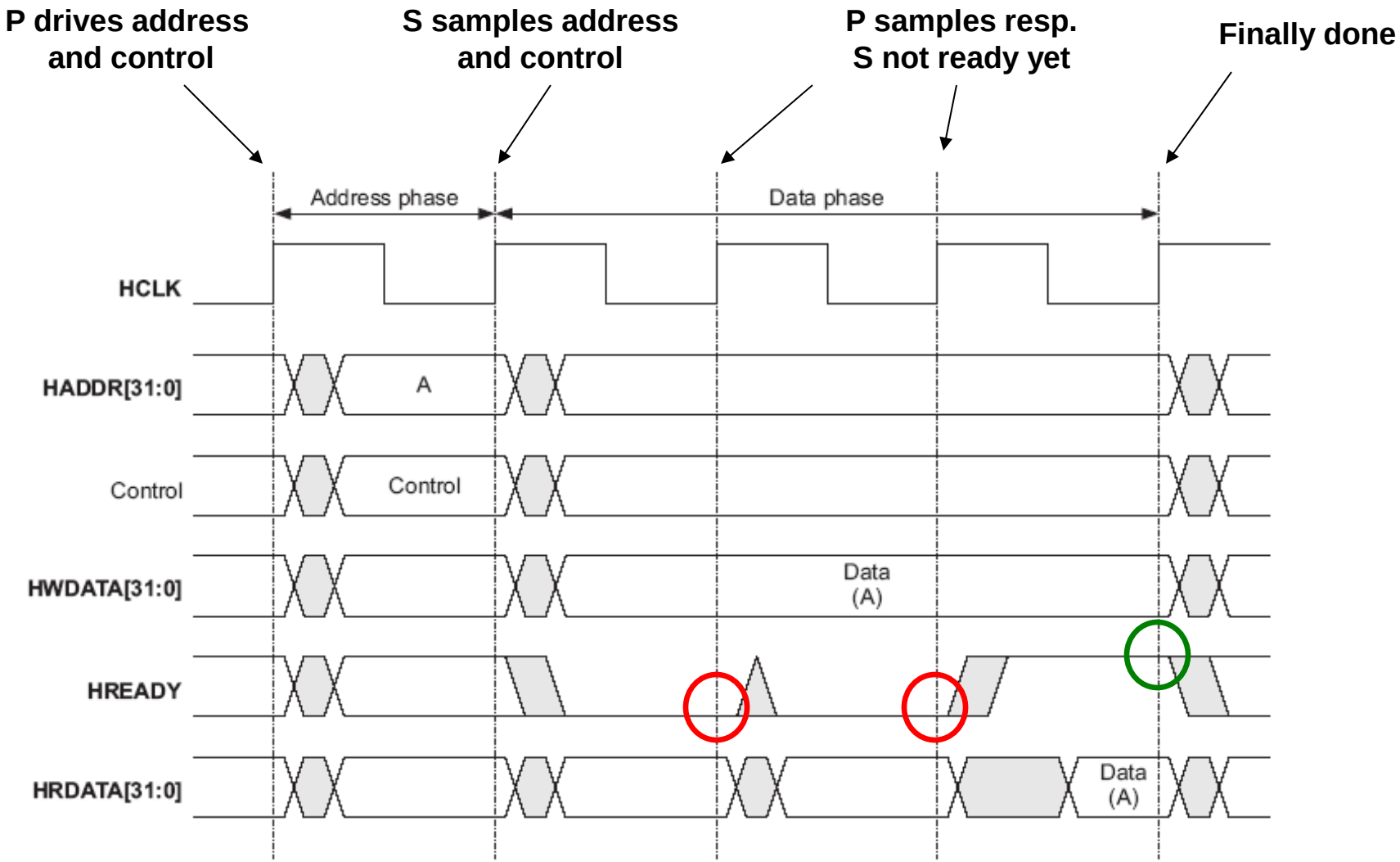
Transfer with Wait States



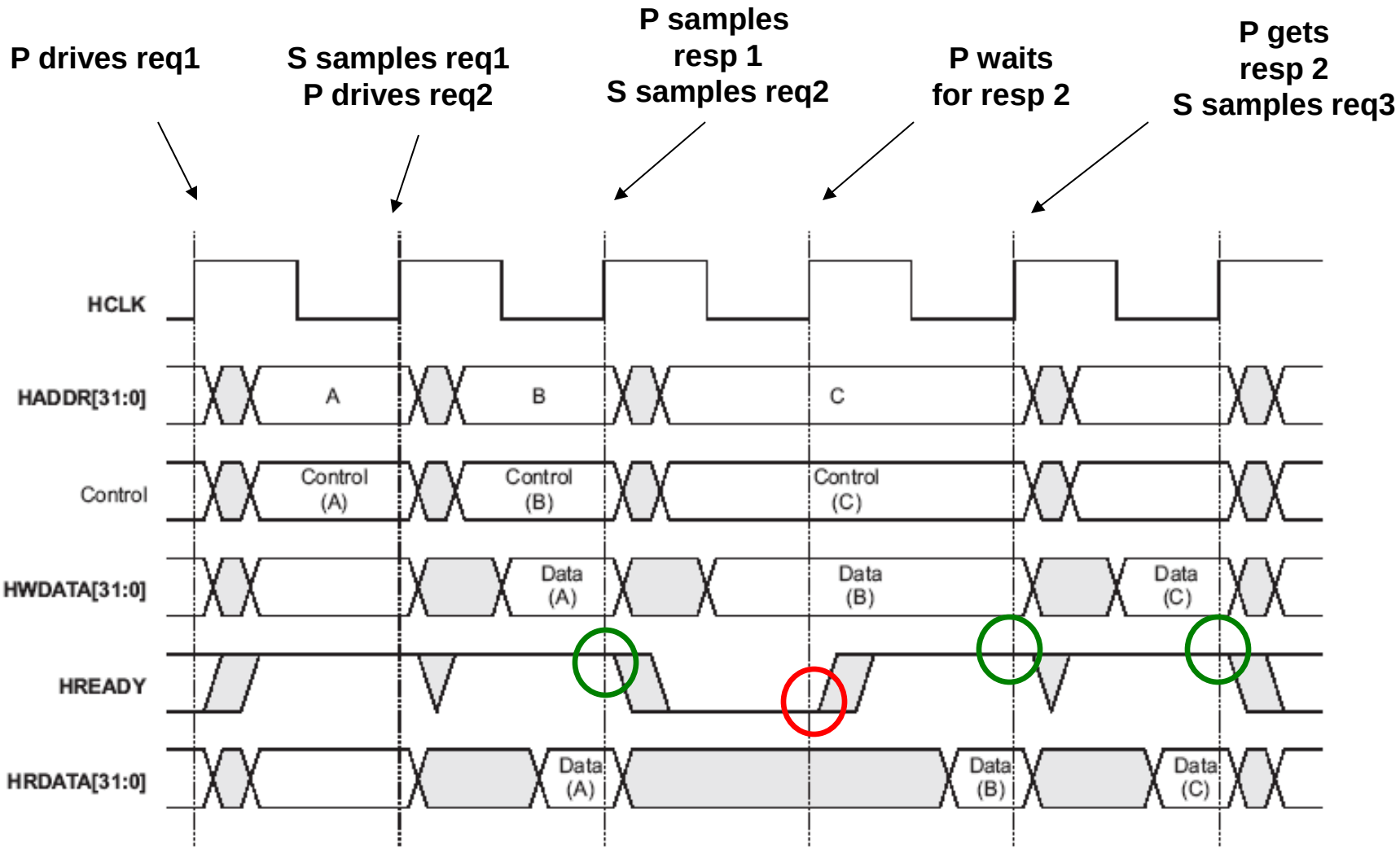
Transfer with Wait States



Transfer with Wait States



Pipelined Transfer



Transfer Types

- Primary drives HTRANS[1:0]
  - Idle — no data transfer
  - Busy — primary cannot prepare request in time, wait
  - NonSEQ — first transfer of a burst or single transfer
  - SEQ — remaining transfers of a burst

- Primary drives HBURST[2:0]
  - Single transfer
  - ***Burst*** transfer
    - Specified length: 4-beat, 8 beat, 16 beat
    - Unspecified length

Burst Mode

- A 4-beat increment transfer, word size