CPLDs/FPGAs

a) zur Ergänzung von Mikrocontrollern

- Funktionen, die es im Mikrocontroller oder Schaltkreissatz nicht gibt,
- zusätzliche Funktionen (z. B. Schnittstellen),
- kleiner Mikrocontroller + CPLD/FPGA als Alternative zum "dicken" Mikrocontroller (Kosten, Strombedarf).

Separate Hardware ist inhärent parallel.

Sie kann Funktionen sozusagen im Schlaf erledigen (minimaler Stromverbrauch), für die im Mikrocontroller ein Programm laufen müßte (deutlich höherer Stromverbrauch). Beispiel: Überwachung von Schnittstellen, um den Mikrocontroller aus dem Stromsparzustand aufzuwecken.

Pegelwandlung

Bedienung/Anzeige

Grundgedanke: eine Vielzahl von Neben- und Hilfsfunktionen im CPLD unterbringen.

Separate Hardware kann optimal an die jeweilige Aufgabe angepaßt werden. Kein unnötiger funktioneller Overkill (vgl. beispielsweise die neumodischen E-A-Ports mancher Mikrocontroller mit ihren vielen Funktionen (z. B. 16 I/O-Adressen je Port)).

b) komplette Alternative zum Mikrocontroller + zusätzliche Schaltkreise.
Alles (Prozessor, Peripherie usw.) in einem einzelnen Schaltkreis unterbringen (System on Silicon).

Was kennzeichnet einen Coprozessor?

Eine anwendungsspezifische Verarbeitungseinrichtung:

- Datenwege,
- Speicher / Register,
- Verarbeitungsschaltungen,
- Schnittstellen.

– Alles ist problemspezifisch. –

Es wird das implementiert, was der gewöhnliche (universelle) Prozessor leistungsmäßig nicht bewältigt.

Aber: Problem der Kommunikation und der Aufteilung der Verarbeitungsfunktionen auf zwei programmierbare Einrichtungen (funktionelle Partitionierung).

Heute – in den unteren Leistungsklassen, um die es hier geht – eher Stromsparproblem als Leistungsproblem. Wieviele FETs schalten in jeder Zeiteinheit? Problemspezifische Hardware kann weitgehend statisch ausgelegt werden. Nur dann schalten, wenn es sein muß, nicht mit hoher Geschwindigkeit zyklisch abfragen.

Coprozessor =

problemspezifsiche Datenwege und Verarbeitungsschaltungen = die zu steuernde Hardware

+ Steuerung + Kopplung mit dem Mikrocontroller.

Wie steuern?

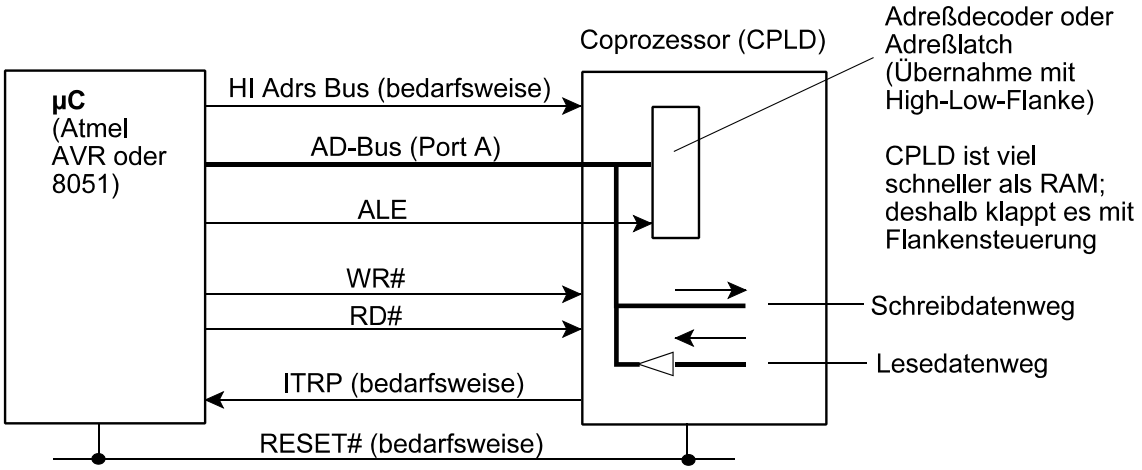
- systematisch entworfene Steuerautomaten (OHE),
- reguläre Sequencer
- Branch Sequencer
- Mikroprogrammsteuerung

Speicherprogrammerbare Steuerung (Branch Sequencer/Mikroprogramm) mit PLA statt RAM/ROM oder mit eingebautem Flash oder dergleichen (MAX II Flash 512 Worte zu 16 Bits (8 kBits)).

Interface zum Mikrocontroller – Wie?

- SPI
- I2C
- JTAG
- LPC
- Eigenbau:
  - bitseriell
  - herkömmlicher Bus nach Intel-Prinzip, also ähnlich RAM, ROM, ISA-Bus
  - herkömmlicher Bus nach Motorola-Prinzip, also ähnlich LCD-Anzeige
  - Ringbus (HyperTransport-ähnlich)
  - LPC-ähnlich
  - ähnlich SDRAM
- Welche fertigen Schnittstellen haben die Mikrocontroller?
- Welche Interfaces lassen sich im Mikrocontroller einfach implementieren (E-A-Ports + Software)?

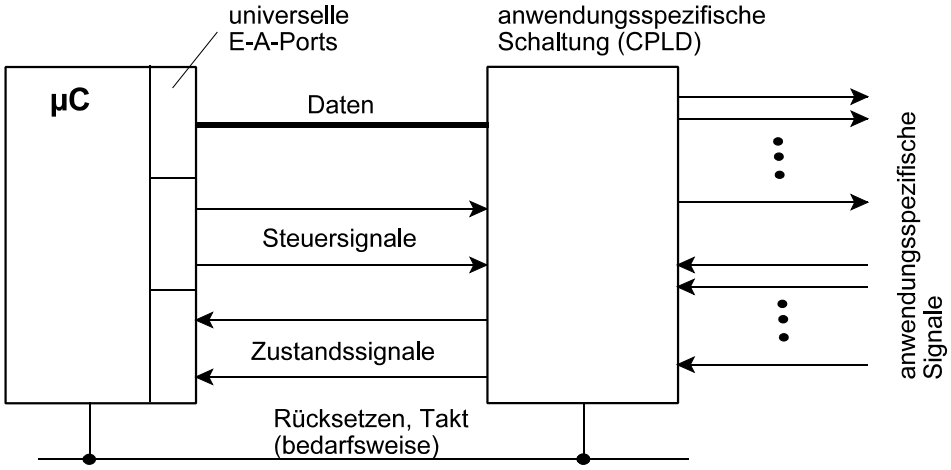
Speicherschnittstellen ausnutzen (herkömmliche und SDRAM).



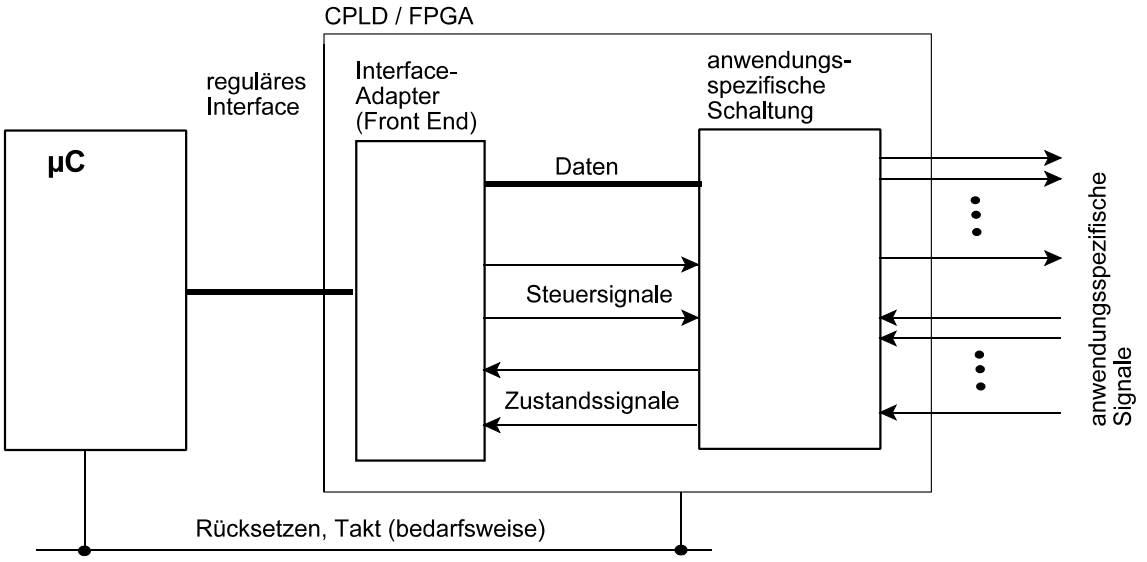
Problem: Auftragsübergabe und Rückmeldung.

1. Interne Übergabestellen definieren. Ergeben sich aus der jeweiligen Problemlösung (Datenregister, Steuerbits usw.)

2. Schnittstelle zum Mikrocontroller.
   - 1:1 an die E-A-Ports
   - Anpassung an reguläres oder standardisiertes Interface (Front End).



Aus der Anwendungslösung ergeben sich die Signale und die zeitlichen/funktionellen Anschlußbedingungen. Hier werden die Signale 1:1 mit universellen E-A-Ports des Mikrocontrollers verbunden. Alle Funktionen sind auszuprogrammieren, alle zeitlichen Anforderungen mit Software zu erfüllen.



Hier wird die anwendungsspezifische Schnittstelle über einen Interfaceadapter (Front End) auf ein reguläres (standardisiertes oder wenigstens vereinfachtes) Interface umgesetzt.

Probleme:
- Synchronisation
- sehr unterschiedliche Taktfrequenzen
- Auftragsübergabe,
- Ergebnisübernahme,
- Parameteränderung ohne Störung des aktuellen Ablaufs.

Synchronisationsproblem
- verschiedene Taktfrequenzen
- Zu-Ende-Kommen des aktuellen Ablaufs
- Auftragsübergabe
- Fertigmeldung.

Übergabeproblem:
- Nicht den aktuellen Ablauf stören: aktueller Ablauf muß korrekt zu Ede kommen. Pufferregister
- Werte müssen auf einmal wirksam werden, nicht stückweise. Assembly-Register.

- Strobes auf anwendungsspezifische Takte synchronisieren.
- Wartezustände
- Handshaking.

Ein gewisser Grad an asynchronen Schaltmitteln muß sein. Oder ein Takt, der schneller ist als jeder zu erwartende Immpuls oder Mindestimpulsdauer gemäß dem verfügbaren Takt vorgeben (vgl. LCD).

Wenn der Takt im CPLD schneller ist als der des Mikrocontrollers:
Dann müssen alle Steuerabläufe über eine Art Single-Shot State Machine geführt werden.

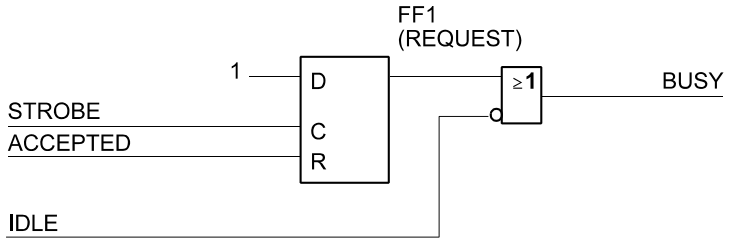
Übernahme von Daten mit schnellen Strobe- oder Taktimpulsen:
Pufferregister erforderlich, die ihren Takt direkt von der Quelle beziehen (Slave Ports).
Das betrifft sinngemäß Bustreiber für Lesezugriffe.

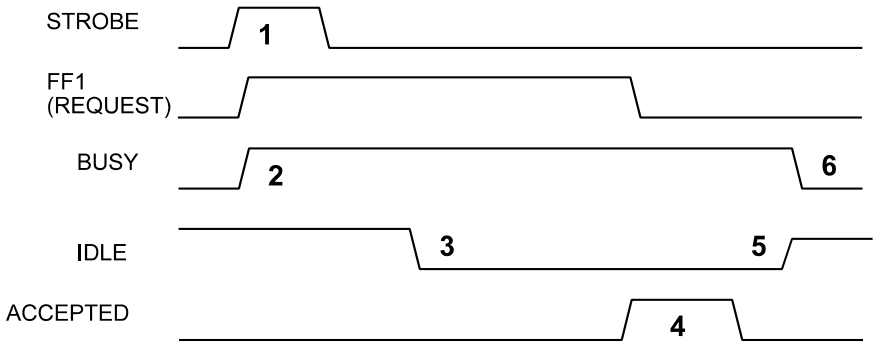
Wenn Datenbreite > Busbreite:
Abschnittsweise übertragen. Am Ziel zur gewünschten Breite zusammensetzen (Assembly). Es darf nicht sein, daß während des Zusammensetzens die Hardware mit Misch- und Zwischenwerten arbeitet. Ggf. Pufferregister erforderlich.Lesedaten abschnittsweise holen (Disassembly). Wenn sie sich während des Holens verändern können, sind Pufferregister erforderlich.

Programmseitige Einstellungen dürfen den aktuellen Ablauf nicht beeinträchtigen.

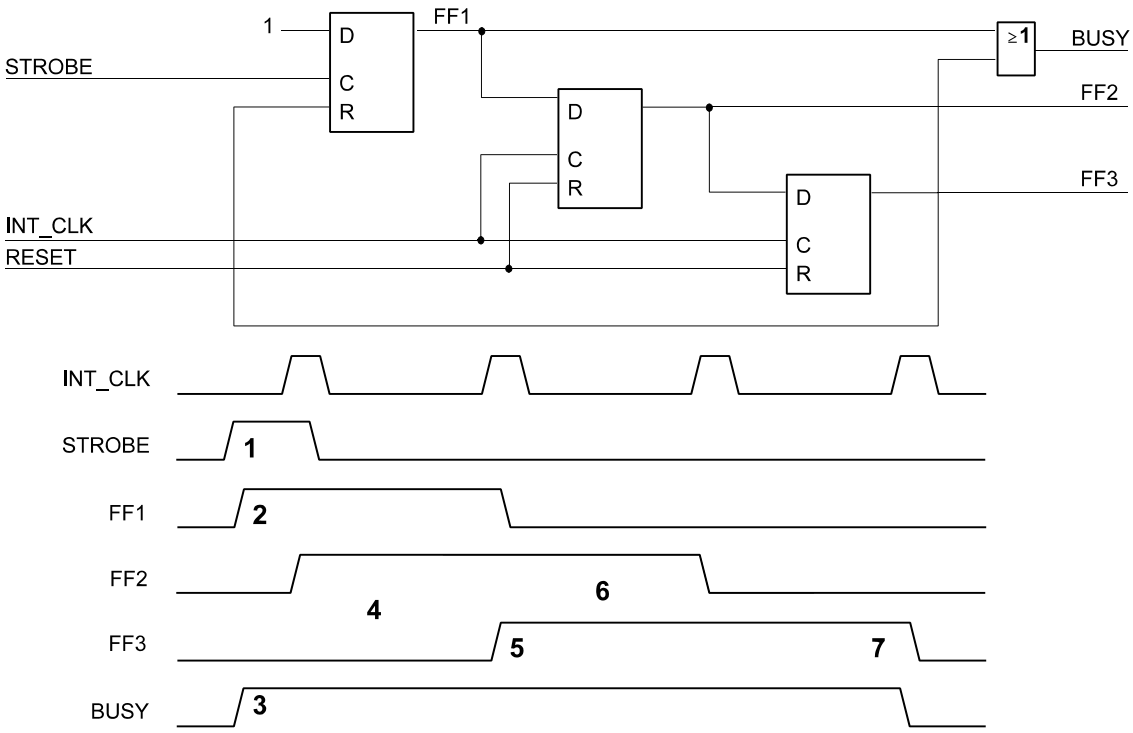
Der einfachste Fall: es ist möglich, den Ablauf anzuhalten und nach dem Übertragen der Einstellungen wieder zu starten (Start-Stop State Machine).

Wenn dies nicht möglich ist: es muß alles gepuffert werden. Die neuen Einstellungen dürfen erst dann übernommen werden, wenn dies ausdrücklich signalisiert wurde.

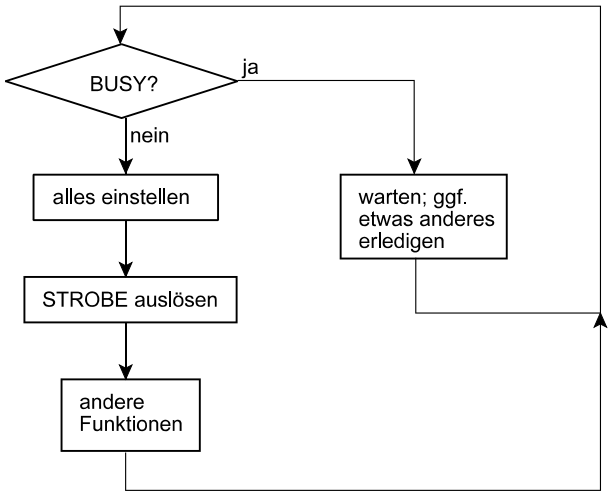




Synchronisation mit langsamerem Takt. Es wird eine interne State Machine angestoßen. STROBE bewirkt eine Art Tastenbetätigung. 1- STROBE-Impuls (kurz); 2 - Interface wird sofort besetzt; 3 - die interne State Machine verläßt ihren Ruhezustand; 4 - die interne State Machine signalisiert, daß die Anforderung erledigt ist (hierzu genügt irgend ein Zwischenzustand) – das Anforderungs-Flipflop FF1 wird zurückgesetzt; 5 - die interne Stae Machine erreicht wieder ihren Ruhezustand; 6 - damit wird auch der Besetztzustand am Interface beendet.



Erzeugung von Steuerimpulsen auf Grundlage eines langsameren Taktes. 1- STROBE-Impuls (kurz); 2 - FF1 schaltet ein; 3 - damit wird das Interface sofort besetzt; 4 - mit den nachfolgenden Taktimpulsen schalten nacheinander FF2 und FF3 ein; 5 - FF1 wird zurückgesetzt; 6 - mit den nachfolgenden Taktimpulsen schalten nacheinander FF2 und FF3 aus; 7 - mit dem Ausschalten von FF3 wird auch der Besetztzustand am Interface beendet.



Es darf nicht sein, daß sich an FF 1 Rücksetzen und Takt überschneiden – die Flipflops in FPGAs und CPLDs halten das womöglich nicht aus (Metastabilität).

Damit das funktioniert, muß eine gewisses Software-Protokoll eingehalten werden: STROBE nur senden, wenn das Interfacs frei ist (BUSY = 0).

Die andere Alternative: Fully Interlocked Handshaking. Dieses Signalprotokoll kommt mit beliebig langsamen Einrichtungen zurecht.