الجامعة الافتراضية السورية
SYRIAN VIRTUAL UNIVERSITY

أساسيات النظم المضمّنة

الدكتور عبد الناصر العاسمي

ISSN: 2617-989X

Books & Refrences

أساسيات النظم المضمّنة

الدكتور عبد الناصر العاسمي

من منشورات الجامعة الافتراضية السورية

الجمهورية العربية السورية 2020

Embedded System Fundamentals

Dr. Abdelnasser Assimi

Publications of the Syrian Virtual University (SVU)

Syrian Arab Republic, 2020

الفهرس

الفصل الأول: مدخل إلى النظم المضمنة 11

1. مقدمة 13

2. خصائص النظم المضمنّة 15

3. التحديات التصميمية وأمثلة مقاييس التصميم 11

4. تقنيات المعالجات المضمنة 20

4-1. معالجات ذات أغراض عامة - معالجات برمجية 20

4-2. معالجات ذات غرض وحيد - معالجات عتادية 21

4-3. معالجات مخصصة لتطبيق 21

5. تقنيات الدارات المتكاملة 23

5-1. دارات متكاملة مخصصة بشكل كامل 23

5-2. دارات متكاملة مخصصة جزئياً 23

5-3. دارات المنطق المبرمج 24

6. تقنيات التصميم 26

6-1. الترجمة والتركيب 27

6-2. استعمال المكتبات والملكية الفكرية 27

6-3. الاختبار والتحقق 28

7. خلاصة 28

8. أسئلة الفصل الأول 29

الفصل الثاني: أساسيات الدارات المنطقية 32

1. مقدمة 34

2. المنطق التراكبي 35

2-1. الترانزستورات والبوابات المنطقية.....35
2-2. تصميم دارات المنطق التراكبي.....35
2-3. العناصر التراكبية على سوية نقل السجلات.....36
3. المنطق التعاقبي.....40
3-1. القلابات.....40
3-2. العناصر التعاقبية على سوية نقل السجلات.....41
3-3. تصميم دارات المنطق المتتابع.....43
4. خلاصة.....45
5. أسئلة الفصل الثاني.....46
الفصل الثالث: المعالجات المخصصة لغرض وحيد.....50
1. مقدمة.....52
2. تصميم المعالج المخصص لغرض وحيد.....53
2-1. غرض المعالج: حساب القاسم المشترك الأكبر.....54
2-2. مخطط آلة الحالة المنتهية مع معطيات.....55
2-3. تصميم مسار المعطيات.....56
2-4. تصميم المتحكم.....57
3. أمثَلَة تصميم المعالج الخاص.....60
3-1. أمثَلَة البرنامج الأصلي.....60
3-2. أمثَلَة آلة الحالة المنتهية مع معطيات.....61
3-3. أمثَلَة مسار المعطيات.....62
3-4. أمثَلَة آلة الحالة المنتهية.....63
4. خلاصة.....63

5. أسئلة الفصل الثالث 64
الفصل الرابع: لغة التوصيف العتادي Verilog 67
1. مقدمة 69
2. لمحة تاريخية عن لغة Verilog 70
3. سويات التصميم 70
3-1. السوية السلوكية 70
3-2. سوية نقل السجلات 71
3-3. سوية البوابات 71
4. لغة التوصيف العتادي Verilog 71
4-1. العناصر المعجمية 72
4-2. أنماط المعطيات 73
4-3. تمثيل الأرقام 75
4-4. العمليات 76
5. هيكلية البرنامج 76
5-1. التصريح عن البوابات 76
5-2. جسم البرنامج 77
5-3. التصريح عن الإشارات 77
5-4. التوصيف البنيوي 78
5-5. مثال مقارن على بتين 78
6. الاختبار 81
7. خلاصة 83
8. أسئلة الفصل الرابع 84

الفصل الخامس: الدارات التراكبية في Verilog 88
1. مقدمة..... 90
2. العمليات الحسابية والمنطقية..... 91
2-1. ضبط عرض الإشارات..... 94
2-2. تركيب القيم ‘x’ و ‘z’ 95
3. كتلة "دائماً" للدارات التراكبية..... 97
3-1. الاسناد الاجرائي..... 97
3-2. عبارة "إذا- وإلا"..... 98
3-3. عبارة "في حال"..... 100
4. شبكة التوصيل..... 13
5. الثوابت والمعاملات 13
6. تطبيق: مفكك ترميز سباعي القطع 13
7. إرشادات عامة في التصميم 105
8. خلاصة 13
9. أسئلة الفصل الخامس..... 13
الفصل السادس: الدارات التعاقبية في Verilog 112
1. مقدمة..... 13
2. القلاب والسجل..... 13
3. النظام المتزامن..... 13
4. أنواع الدارات التعاقبية..... 117
4-1. الدارات التعاقبية النظامية 13
4-2. دارات آلة الحالة المنهية 13

3-4. دارات آلة الحالة المنتهية مع معطيات 13
5. خلاصة 126
6. أسئلة الفصل السادس.....13
الفصل السابع: المعالجات ذات الأغراض العامة.....13
1. مقدمة.....13
2. البنية الأساسية.....134
2-1. مسار المعطيات.....134
2-2. وحدة التحكم 134
2-3. الذاكرة.....135
3. عمل المعالج 137
3-1. تنفيذ التعليمة.....13
3-2. التواردية.....137
3-3. المعالجات المتعددة النوى 138
4. برمجة المعالج.....139
4-1. قائمة التعليمات 139
4-2. فضاء ذاكرة البرنامج وذاكرة المعطيات.....14
4-3. السجلات 14
4-4. الدخل والخرج.....14
4-5. المقاطعات.....141
5. المعالجات ذات التعليمات الموجهة لتطبيق محدد.....14
5-1. المتحكمات 142
5-2. معالجات الإشارة الرقمية.....143

6. خلاصة 14
7. أسئلة الفصل السابع.....14
الفصل الثامن: المعالجات الخاصة المعيارية: الطرفيات.....148
1. مقدمة.....150
2. المؤقّتات والعدّادات.....151
2-1. مؤقت بسيط.....151
2-2. مؤقت بنمطين 14
2-3. مؤقت مجال زمني.....152
2-4. مؤقتات متراصّة 14
2-5. مؤقت مع مقسّم.....154
2-6. مؤقت الملاحقة.....154
3. مرسل-مستقبل غير متزامن عام 156
4. معدِّل عرض النبضة 158
5. متحكم شاشات العرض LCD 160
6. متحكم لوحة المفاتيح.....163
7. ساعة الزمن الحقيقي (RTC) 164
8. خلاصة 164
9. أسئلة الفصل الثامن 165
الفصل التاسع: الذواكر.....14
1. مقدمة.....170
2. تصنيف الذواكر 172
2-1. قابلية الكتابة.....172

2-2. دوام التخزين 172
3. الأنواع الشائعة للذواكر 174
3-1. الذواكر القابلة للقراءة فقط..... 174
3-2. الذواكر القابلة للقراءة والكتابة..... 178
3-3. أمثلة على الذواكر 180
4. تركيب الذواكر 182
4-1. طريقة التركيب جنباً إلى جنب..... 182
4-2. طريقة التركيب من الأعلى إلى الأسفل 183
4-3. الحالة العامة 183
5. وحدة إدارة الذاكرة..... 184
6. خلاصة 15
7. أسئلة الفصل التاسع..... 15
الفصل العاشر: المواجهة باستعمال المساري 15
1. مقدمة..... 15
2. أساسيات الاتصال..... 191
3. مفاهيم أساسية في البرتوكولات 194
3-1. العقدة..... 194
3-2. اتجاه المعطيات..... 194
3-3. العناوين..... 194
3-4. التنضيد الزمني..... 194
3-5. طرق التحكم 195
3-6. مثال: بروتوكول المسرى ISA..... 15

4. مواجهة المعالجات.....15
4-1. الدخل والخرج باستعمال بوابات المعالج 199
4-2. الدخل والخرج باستعمال المسرى 199
4-3. مثال: الدخل والخرج في المسرى ISA 200
4-4. مثال: بروتوكول الذاكرة في المتحكم الصغري 8051.....201
5. التحكيم 202
5-1. تحكيم الأولوية.....202
5-2. تحكيم سلسلة ديزي 203
5-3. طرق تحكيم المسرى.....204
6. بنية المسرى متعدد السويات 205
7. أسئلة الفصل العاشر 207
الفصل الحادي عشر: بروتوكولات الاتصال 210
1. مقدمة.....212
2. أنواع الاتصالات.....213
2-1. الاتصال التفرعي.....213
2-2. الاتصال التسلسلي 213
2-3. الاتصال اللاسلكي 214
3. مفاهيم أساسية في الاتصالات.....215
3-1. الطبقية.....215
3-2. كشف الخطأ وتصحيحه.....215
4. البروتوكولات التسلسلية 217
4-1. بروتوكول I^2C 217

4-2. البروتوكول CAN.....219
4-3. السلك الناري 222
4-4. المسرى التسلسلي العام USB 222
5. البروتوكولات التفرعية.....225
5-1. المسرى PCI 225
5-2. المسرى ARM 225
6. البروتوكولات اللاسلكية.....226
6-1. بروتوكول الاتصال IrDA.....226
6-2. بروتوكول الاتصال Bluetooth.....226
6-3. بروتوكول الاتصال (WIFI) IEEE 802.11 226
7. خلاصة 227
8. أسئلة الفصل الحادي عشر 228
الفصل الثاني عشر: نظم التشغيل في الزمن الحقيقي.....232
1. مقدمة.....234
2. نبذة تاريخية عن نظم التشغيل.....235
3. تعريف نظام التشغيل في الزمن الحقيقي.....237
4. المُجدول 239
4-1. تعدد المهام.....239
4-2. تبديل السياق.....240
4-3. خوارزميات الجدولة 241
5. الأغراض 243
6. الخدمات 243

7. الخصائص العامة لنظم التشغيل في الزمن الحقيقي.....244
7-1. الموثوقية 244
7-2. السلوك القابل للتنبؤ به 244
7-3. الأداء.....244
7-4. الحجم المحدود 244
7-5. قابلية التقييس 244
8. بعض نظم التشغيل.....246
8-1. النظام VxWorks.....246
8-2. النظام QNS 246
8-3. النظام FreeRTOS.....246
9. خلاصة 247
10. أسئلة الفصل الثاني عشر 248

الفصل الأول:
مدخل إلى النظم المضمنة

الكلمات المفتاحية:

النظم المضمنة، مقاييس التصميم، المعالجات المضمنة، تقنيات الدارات المتكاملة.

Embedded systems, Design Metrics, Embedded Processors, Integrated Circuits technologies.

الملخص:

يهدف هذا الفصل إلى التعريف بمفهوم النظم المضمنة والخصائص التي تميزها عن النظم الحاسوبية التقليدية. نبين في هذا الفصل أهم التحديات التي تواجه تصميم النظم المضمنة المتعلقة بأمثلة مقاييس التصميم. كما نبين أنواع المعالجات المستعملة في النظم المضمنة وفوائد ومساوئ كل نوع من وجهة نظر مقاييس التصميم. كما نعرض تقنيات تصميم الدارات المتكاملة المستعملة في تنفيذ النظم المضمنة.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- مفهوم النظم المضمنة وخواصها
- مقاييس تصميم النظم المضمنة
- أنواع المعالجات المستعملة في النظم المضمنة
- تقنيات تصميم الدارات المتكاملة

المخطط:

يضم هذا الفصل 6 وحدات تعليمية هي:

1. لمحة عن النظم المضمنة.
2. خصائص النظم المضمنّة.
3. التحديات التصميمية وأمثلة مقاييس التصميم.
4. تقنيات المعالجات المضمنة.
5. تقنيات الدارات المتكاملة.
6. تقنيات التصميم.

يهدف هذا الفصل إلى التعريف بمفهوم النظم المضمنة والخصائص التي تميزها عن النظم الحاسوبية التقليدية. نبين في هذا الفصل أهم التحديات التي تواجه تصميم النظم المضمنة المتعلقة بأمثلة مقاييس التصميم. كما نبين أنواع المعالجات المستعملة في النظم المضمنة وفوائد ومساوئ كل نوع من وجهة نظر مقاييس التصميم. كما نعرض تقنيات تصميم الدارات المتكاملة المستعملة في تنفيذ النظم المضمنة. وفي النهاية نبين أهم منهجيات التصميم المساعدة في تحسين انتاجية التصميم.

1. مقدمة:

تنتشر النظم الحاسوبية في كل مكان من حياتنا اليومية. وربما ليس من المفاجئ معرفة أنه يتم تصنيع الملايين من النظم الحاسوبية كل عام من أجل صناعة الحواسيب الشخصية ومحطات العمل والمخدمَات. ولكن، قد يكون من المفاجئ فعلاً معرفة أن هناك أيضاً المليارات من النظم الحاسوبية التي تصنع لأغراض أخرى وتكون مضمنةً ضمن تجهيزات الكترونية واسعة النطاق للقيام بمهمة معينة. ولا يدرك المستخدم في أغلب الأحيان بوجود هذه النظم الحاسوبية داخل الجهاز الذي يستخدمه.

إن صياغة تعريف دقيق للنظم الحاسوبية المضمّنة أو ببساطة النظم المضمّنة ليست عملية سهلة. ولكن يمكن أن نعطي التعريف التالي:

النظام المضمّن: هو نظام حاسوبي يحتوي العديد من النظم الجزئية وهو مُصمّم لتنفيذ مهمة أو عدة مهام محددة سلفاً في الزمن الحقيقي.

بالطبع، إن هذا التعريف ليس دقيقاً أو كاملاً ولكنه يمكن أن يكون الأقرب للصحّة. يمكن أن نفهم بشكل أفضل تلك النظم من خلال إعطاء بعض الأمثلة الشائعة وخواصها العامة. سيكشف تفحص هذه الأمثلة عن قرب عن أهم التحديات التي تواجهنا عند تصميم هذه النظم.

تتواجد النظم المضمّنة في العديد من التجهيزات الالكترونية الشائعة ويمكن أن نذكر الأمثلة التالية:

- الأجهزة الالكترونية: هاتف خلوي, كاميرا رقمية، كاميرة فيديو، لعبة فيديو، حاسبة.
- التجهيزات الكهربائية المنزلية: فرن الميكروويف، الغسالة الآلية، المجيب الآلي.
- القطاع المصرفي: قارئ المنتجات، الصرافات الآلية، نظام الانذار.
- التجهيزات المكتبية: طابعة، ماسح ضوئي، جهاز فاكس.
- صناعة السيارات: نظام نقل الحركة الآلي، مثبت السرعة، ضخ الوقود في المحرك، نظام الكبح بدون اقفال ABS (Antilock Braking System).

ويمكننا القول أن معظم الأجهزة التي تعمل على الطاقة الكهربائية تحتوي في داخلها على نظام مضمّن أو سيتم تزويدها بهكذا نظام في المستقبل.

 هاتف محمول	 كاميرا رقمية
 فرن	 غسالة آلية
 جهاز فاكس	 طابعة
 صراف آلي	 قارئ بطاقة مصرفية
 قارئ باركود	

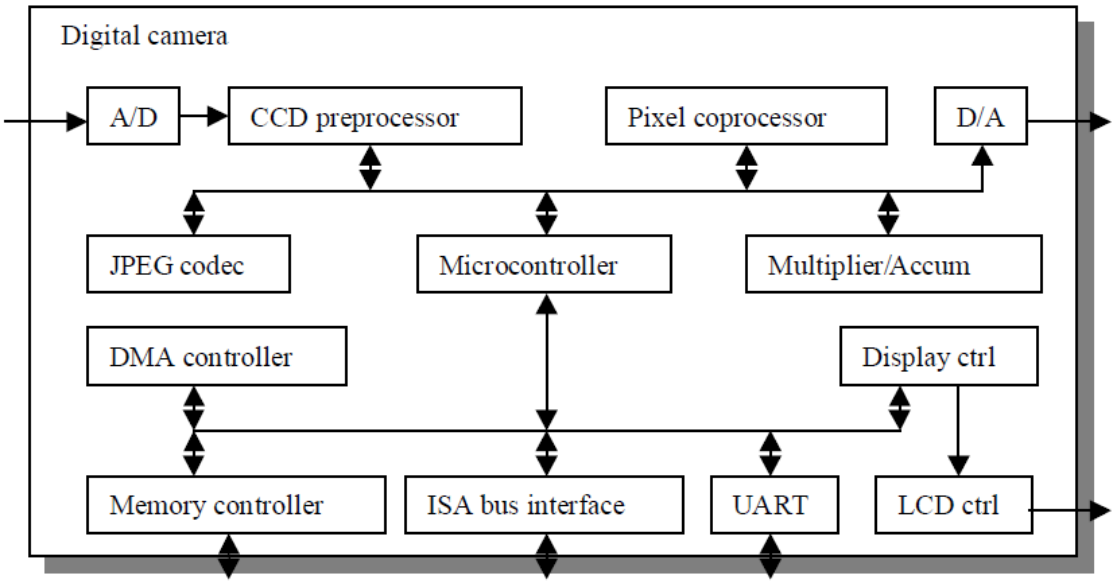
بعض الأمثلة عن النظم المضمنة

2. خصائص النظم المضمنّة:

للنظم المضمنّة العديد من الخصائص المشتركة، نذكر منها:

1. **هي ذات عمل محدد:** ينفذ النظام المضمّن عادةً برنامجاً وحيداً بشكل متكرر. على سبيل المثال فإن الطابعة هي دائماً تقوم بعمل الطباعة، بينما الحاسوب الشخصي يقوم بتنفيذ العديد من البرامج المختلفة مثل برامج تحرير النصوص، أو برامج تحرير الصور وغيرها الكثير.
2. **مقيَّدة بشكل كبير:** يتم تقييد مواصفات النظم الحاسوبية بشكل دائم ولكن يكون تقييد مواصفات النظم المضمنة أكبر. ويشمل ذلك في معظم الأحيان قيود على الكلفة والحجم واستهلاك الطاقة.
3. **تتجاوب بالزمن الحقيقي:** يجب أن يتفاعل النظام مع تغيرات الظروف المحيطة ويقوم بتنفيذ بعض الحسابات والحصول على النتائج من دون أي تأخير زمني يذكر. فعلى سبيل المثال يجب أن يقوم نظام مثبت السرعة في السيارة بقراءة قيم حساسات السرعة والمكابح بشكل مستمر ويقوم بحساب قيم التسارع أو التباطؤ بشكل مستمر ضمن وقت محدد وإلا سيفشل بالتحكم بسرعة السيارة. بالمقابل يركز الحاسب على الحسابات عند يستجيب لتغير في تجهيزات الدخل ويقدم النتائج للمستثمر ولو بعد فترة انتظار(وإن كان ذلك غير محبذ) دون أن يتسبب ذلك في فشل النظام.

لنأخذ مثال الكاميرا الرقمية ذات المخطط الصندوقي المبين في الشكل التالي:



المخطط الصندوقي لكاميرا رقمية.

تقوم دارات التبديل التماثلي الرقمي A/D و D/A بتحويل الصورة التماثلية إلى صورة رقمية وبالعكس. ويقوم المعالج CCD preprocessor بتحويل القيم التماثلية إلى القيم اللونية المكافئة. وبعدها يقوم المرمز JPEG codec بضغط الصورة باستعمال المعيار JPEG مما يمكن تخزين الصورة باستعمال حجم أقل لذاكرة الكاميرا. ويساعد المعالج pixel coprocessor على عرض الصورة على الشاشة بشكل سريع. يمكن متحكم

الذاكرة memory controller بالتحكم بالنفاذ إلى الذاكرة، بينما يمكن المتحكم DMA controller بالنفاذ المباشر للذاكرة بدون أن يتطلب ذلك اشغال المتحكم. تمكن وحدة UART بالتراسل التسلسلي مع الحاسوب لتحميل الصور. تتحكم الدارتان LCD ctrl وDisplay ctrl بعرض الصورة على الشاشة. تساعد الدارة Mutiplier/Accum ببعض معالجات الإشارة الرقمية. ويوجد في قلب النظام متحكم صغري يقوم بإدارة عمل النظام بالكامل. يمكن اعتبار أن كل دارة هي معالج مصمم لمهمة محددة، بينما يمكن اعتبار المتحكم الصغري بأنه معالج مصمم لتنفيذ مهام عامة.

يوضح هذا المثال خواص النظم المضمّنة الآنفة الذكر.

- يقوم هذا النظام بمهمة وحيدة بشكل متكرر وهي مهمة التقاط الصور حيث يقوم بالتقاط الصورة وضغطها وتخزينها في الذاكرة وعرضها على الشاشة مع إمكانية ارسالها إلى الحاسوب.
- يجب أن يكون سعر الجهاز رخيصاً ليكون متاحاً للشراء من قبل شريعة واسعة من الزبائن.
- يجب أن يكون صغير الحجم بالقياس الطبيعي لكاميرا رقمية.
- يجب أن يكون سريعاً للتمكن من التقاط الصور بسلاسة وبشكل متعاقب.
- يجب أن يستهلك القليل من الطاقة لكي تدوم بطارية الجهاز وقتاً أطول.

مع ذلك فإن هذا النظام لا يفرض درجة عالية من التقييد من حيث العمل في الزمن الحقيقي. يكفي أن يكون الجهاز قادراً على الاستجابة لكبسة المستخدم على زر أخذ الصورة وهو وقت طويل مقارنة بسرعة عمل المعالج.

3. التحديات التصميمية وأمثلة مقاييس التصميم:

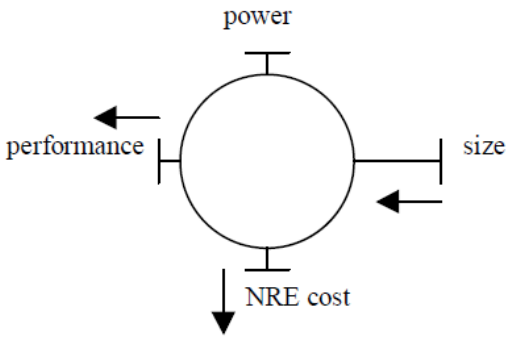
على مصمم النظام المضمّن أن يضع تصميماً يلبي بالطبع المتطلبات الوظيفية للنظام، ولكن يكمن التحدي الصعب في هذه المهمة هو أن يبني تصميماً يلبي مقاييس التصميم بشكل أمثلي.

مقياس التصميم: هو مقدار قابل للقياس يوصف ميزة معينة في تصميم النظام.

ومن المقاييس ذات الدلالة:

1. **الكلفة الإفرادية (unit cost):** كلفة تصنيع كل نسخة من النظام باستثناء الكلفة الهندسية غير متكررة.
2. **الكلفة الهندسية غير متكررة NRE (Non-Recurring Engineering):** كلفة تصميم النظام. حيث يمكن بعد تصميم النظام انتاج هذا النظام بأي عدد من النسخ دون كلفة تصميمة إضافية.
3. **الحجم (size):** الحجم الفيزيائي المطلوب لتنفيذ النظام وهو يقدر بالبايت بالنسبة للبنية البرمجية، وبعدد البوابات أو الترانزستورات بالنسبة للبنية العتادية.
4. **الأداء (performance):** سرعة عمل النظام أو معدل تدفق المعطيات في النظام.
5. **استهلاك الطاقة (power consumption):** كمية الطاقة المستهلكة من قبل النظام في واحدة الزمن والتي تحدد العمر التقديري للبطارية أو مقدار التبريد المطلوب للعناصر الالكترونية وذلك لأن استهلاك طاقة أكثر يعني تبديد حراري أكثر.
6. **المرونة (flexibility):** القدرة على تغيير عمل النظام من دون زيادة الكلفة التصميمة كثيراً لذلك يجب أن تكون البنية البرمجية للنظام ذات مرونة عالية.
7. **زمن الوصول للسوق (time to market):** وهو الزمن اللازم لتصميم وتصنيع النظام وصولاً لنقطة جاهزية بيع النظام للزبائن.
8. **زمن الوصول للنموذج (time to prototype):** وهو مقدار الزمن المطلوب لتنفيذ نسخة عملية للنظام والتي قد تكون أكبر أو ذات كلفة أعلى من النموذج النهائي ولكن تستخدم للتحقق من فائدة النظام وصحة عمله وتحسين عمله.
9. **صحة العمل (correctness):** وهي مقدار الثقة بصحة عمل النظام في مختلف الظروف الطبيعية الممكنة. يمكن التثبت من صحة عمل النظام من خلال اختباره بدارات اختبار مناسبة.
10. **السلامة (safety):** وهي احتمال أن لا يسبب النظام أي أذى.

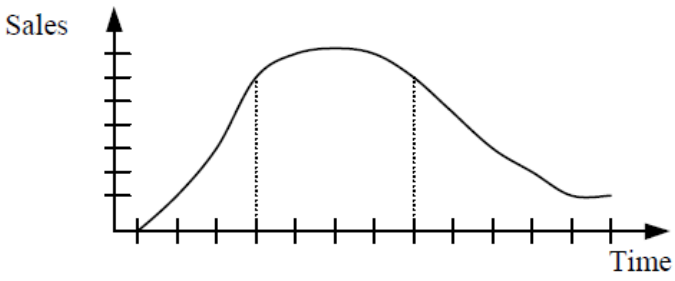
تتعارض عادةً هذه المقاييس فيما بينها، حيث يؤدي غالباً تحسين أحد هذه المقاييس إلى تدني مقياس آخر. قام البعض بتشبيه ذلك بظاهرة العجلة مع عدة مقابس على الأطراف. حيث يمثل كل مقبس مقياس تصميمي معين. فإذا قمت بدفع أحد المقابس نحو المركز (الحجم على سبيل المثال) ستخرج المقابس الأخرى باتجاه الخارج بسبب انتفاخ العجلة كما هو موضح في الشكل التالي:



شكل يمثل عجلة تعارض مقاييس التصميم.

لذلك يجب أن يكون المصمم متمكناً من العديد من التقنيات البرمجية والعتادية وقادراً على الانتقال من تقنية إلى أخرى من أجل الحصول على التصميم الأفضل للتطبيق المطلوب بقيود معينة على مقاييس التصميم. وبالتالي، لا يمكن للمصمم أن يكون فقط خبيراً في البرمجيات فقط أو خبيراً بالتصميم العتادي فقط وإنما عليه أن يكون خبير في كلا المجالين.

معظم المقاييس السابقة تفرض نفسها بشدة في تصميم النظم المضمّنة. وقد أصبح مقياس زمن الوصول للسوق الأكثر إلحاحاً في السنوات الأخيرة. يمكن أن يشكل دخول النظام المضمّن إلى السوق باكراً فرقاً كبيراً في الربح العائد وذلك لأن نافذة السوق الزمنية للمنتجات أصبحت ضيقة أكثر فأكثر. يوضح الشكل التالي مثالاً عن هذه النافذة والتي تعطي حجم المبيعات تبعاً للزمن.



منحني حياة المنتج المتعلق بتغير حجم المبيعات مع الزمن.

إن تفويت هذه النافذة، أي البدء ببيع المنتج بعد هذه النافذة يعني خسارةً كبيرةً في المبيعات. وفي بعض الأحيان قد يترجم تأخير كل يوم في إدخال المنتج للسوق بخسارة ملايين الدولارات.

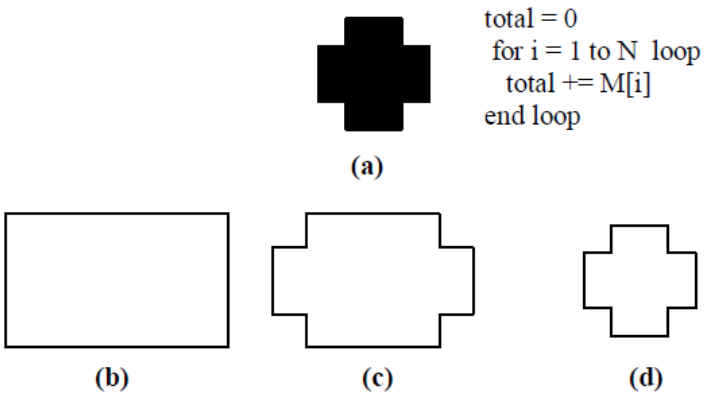
بالإضافة إلى صعوبة تلبية مقياس زمن الدخول للسوق نذكر حقيقة أن تعقيد النظم المضمنة يزداد بسبب الزيادة في سعة الدارات التكاملية IC. لقد نمت سعة الدارة التكاملية، والمقاسة بعدد الترانزستورات بالشريحة، بشكل أسّي خلال العقود الأخيرة. ولكن في المقابل لم تتطور وتيرة التصميم بنفس هذه السرعة. لذا يتوجب على مصمم النظم المضمّنة أن يكون على إطلاع مستمر بالحالة التي وصلت إليها التقانة بشكل دائم.

تُعرف **التقنية (technology)** بأنها الطريقة المتبعة لإنجاز مهمة معينة وخاصة باستعمال الآليات التقنية وطرق التنفيذ والمعارف ذات الصلة.

نركز في هذا المقرر على إعطاء لمحة شاملة عن ثلاث تقنيات مركزية لتصميم النظم المضمّنة وهي: تقنيات المعالجات وتقنيات الدارات التكاملية وتقنيات التصميم. سنوضح ذلك بشكل سريع هنا وسنعود إليها لاحقاً بشكل أكثر تفصيلاً في الفصول القادمة.

4. تقنيات المعالجات المضمنة:

تتضمّن تقنية المعالجات بنية وحدة الحساب المستعملة لتنفيذ وظيفة النظام المرغوب. ومن الشائع ربط مصطلح "المعالج" مع المعالجات القابلة للبرمجة ولكن يمكن اعتبار النظم الرقمية على أنها معالجات أيضاً ولكنها غير قابلة للبرمجة. ويختلف كل معالج عن البقية بتخصصه باتجاه بعض التطبيقات وبالتالي يُظهر مقاييس تصميم مختلفة. نوضح هذا المفهوم بالتمثيل البياني الموضح في الشكل التالي:



تمثيل رسومي يبين مدى ملائمة معالج ما لتطبيق معين حيث يمثل (a) التطبيق و(b) معالج ذو غرض عام و(c) معالج مخصص لمجموعة تطبيقات و(d) معالج ذو غرض وحيد.

فإذا كان التطبيق يتطلب وظيفة معينة نرمز لها هنا بشكل التقاطع، فيمكن للعديد من المعالجات تلبية متطلبات هذا التطبيق من حيث الوظيفة ولكن يكمن الاختلاف في أمثلة مقاييس التصميم فهناك معالجات قد تكون ذات امكانيات أكثر من المطلوب تؤدي إلى هدر غير مبرّر في الموارد وهذا حال استعمال معالج عام في هذا المثال. في التطبيقات العملية نستعمل غالباً مجموعة من المعالجات لكل منها وظيفة محددة وذلك لتلبية مقاييس التصميم بشكل أفضل كما في حالة الكاميرا الرقمية على سبيل المثال.

1.4. معالجات ذات أغراض عامة – معالجات برمجية:

تُصمّم المعالجات ذات الأهداف العامة (general–purpose processors) لتلبية العديد من التطبيقات وذلك لزيادة عدد القطع المباعة منها. لذلك يتم تزويد هذه المعالجات بإمكانيات متعددة مثل ذاكرة البرنامج والذاكرة الحية وسجلات العمل مع مسرى معطيات عريض بقدرٍ كافٍ مع وحدات حسابية ومنطقية تضم معظم العمليات الرياضية المستعملة مثل الضرب والجمع والعمليات المنطقية. يُمكن لمصمم النظام المضمّن أن يستخدم هكذا معالج في تنفيذ نظامه وتقتصر مهمة المصمم هنا على كتابة برنامج هذا المعالج لتنفيذ وظيفة النظام. ويتم الإشارة إلى هذا الجزء من التنفيذ ببساطة بالجزء البرمجي. نحصل من خلال استعمال معالج ذو غرض عام في تصميم النظام المضمّن على العديد من الفوائد. حيث يكون زمن التنفيذ والكلفة الغير متكررة

منخفضان بشكل عام لأن مهمة المصمم تقتصر على كتابة البرنامج فقط ولا يحتاج لتنفيذ أي تصميم رقمي. كما تكون الكلفة الإفرادية منخفضة من أجل عدد نسخ قليل بالإضافة إلى الأداء السريع عند استعمال معالجٍ متطور.

لكن بالرغم من ذلك، فهناك بعض السيئات في مقاييس التصميم. فيمكن أن تكون الكلفة الافرادية عالية من أجل تنفيذ عدد نسخ كبير ويكون الأداء بطيئاً من أجل بعض التطبيقات المتطلبة. كما يمكن أن ينتج ذلك عن حجم كبير للنظام بسبب استعمال معالج بإمكانيات كثيرة غير مستخدمة ولكنها تزيد من الحجم واستهلاك الطاقة أيضاً.

2.4. معالجات ذات غرض وحيد – معالجات عتادية:

المعالج ذو الغرض الوحيد (single-purpose processors) هو دارة رقمية مصممة لتنفيذ برنامج وحيد فقط. لنأخذ مثال الكاميرا. فكل المكونات باستثناء المتحكم الصغري هي معالجات ذات غرض وحيد. فعلى سبيل المثال يقوم مرمز JPEG بتنفيذ برنامج وحيد مهمته ضغط وفك ضغط الصور فقط. يمكن لمصمم النظام المضمن أن يبني معالجاً ذو غرض وحيد بتصميم دارات رقمية مخصصة أو يمكنه استعمال معالج جاهز. تتم الإشارة عادةً إلى هذا الجزء من التصميم بالتصميم العتادي. وعملياً قد نجد المعالجات ذات الغرض الوحيد تحت مسميات أخرى شائعة مثل: المعالج المساعد (coprocessor) والمُسرِّع (accelerator) والطرفية (peripheral).

ينتج عن استعمال المعالجات ذات الغرض الوحيد في النظم المضمنة العديد من المحاسن والمساوئ من وجهة نظر معايير التصميم والتي هي بشكل أساسي عكس تلك الناتجة عن استعمال المعالجات ذات الأغراض العامة. حيث يمكن الحصول على سرعة في الأداء وصغر في الحجم وخفض الطاقة المستهلكة وتقليل التكلفة الإفرادية من أجل كميات كبيرة. بينما في المقابل، يكون زمن التصميم طويلاً والكلفة الغير متكررة NRE عالية مع مرونة منخفضة وكلفة إفرادية عالية من أجل الكميات القليلة للمنتج.

3.4.معالجات مخصصة لتطبيق:

تمثل المعالجات ذات التعليمات المخصصة لتطبيق ASIP (Application-Specific Instruction-set Processors) حلاً وسطاً بين الخيارات الأخرى للمعالجات. فالمعالج من نوع ASIP هو معالجٌ تم تصميمه للاستعمال في صنف من التطبيقات ذات خواص مشتركة مثل تطبيقات التحكم المضمن أو معالجة الإشارة الرقمية أو الاتصالات. يمكن لمصمم هذا النوع من المعالجات أمثلة تصميم مسار المعطيات بما يخدم الصنف المعتبر من التطبيقات. بالإضافة إلى ذلك يمكن إضافة بعض الوحدات الحسابية الخاصة والشائعة الاستعمال في هذا الصنف من التطبيقات.

من فوائد استعمال معالج ASIP في نظام مضمن الحصول على المرونة مع الحفاظ على مستوى جيد من حيث الأداء وصرف الطاقة والحجم. ولكن يتطلب ذلك كلفة هندسية غير متكررة NRE عالية لبناء المعالج نفسة وبناء المترجم (compiler) الخاص به إذا لم يكن موجوداً مسبقاً.

من الأمثلة المعروفة عن المعالجات ASIP نذكر المتحكمات الصغرية (microcontroller) ومعالجات الإشارة الرقمية DSP (Digital Signal Processor).

المتحكم الصغري هو معالج مصمم لتطبيقات التحكم المضمن. حيث يتم عادةً في هذا النوع من التطبيقات عمليات مراقبة أو تغيير إشارات تحكم ممثلة على بت واحد بينما لا يوجد عمليات حسابية كبيرة. وبالتالي تميل المتحكمات لامتلاك مسار معطيات ذو بنية بسيطة وقادرة على تنفيذ العمليات على مستوى البت كعمليات القراءة والكتابة لإشارات ممثلة على بت واحد. بالإضافة إلى ذلك تحتوي المتحكمات ضمن نفس شريحة المعالج على العديد من الطرفيات الشائعة الاستعمال في تطبيقات التحكم مثل طرفيات التراسل التسلسلي والمؤقتات والعدادات ومُعدِّلات عرض النبضة (PWM) والمبدلات التماثلية الرقمية. يسمح وجود هذه الطرفيات على نفس الشريحة بتنفيذ نظم على شريحة واحدة وبالتالي الحصول على كلفة أقل وحجم أصغر.

تمثل معالجات الإشارات الرقمية DSP نوعاً آخر من معالجات ASIP. فمعالج DSP هو معالج مصمم لتنفيذ العمليات الشائعة في معالجة الإشارات الرقمية والتي هي الترميز الرقمي للإشارات التماثلية مثل إشارات الصوت والفيديو. وتشمل هذه العمليات على إجراء المهام الأساسية في معالجة الإشارة الرقمية مثل الترشيح والتحويلات المختلفة. تشمل هذه العمليات عادةً على حسابات رياضية كثيرة مثل الضرب والجمع أو الازاحة والجمع. ولتسريع هذا العمليات، يمتلك المعالج على مكونات ذات وظيفة محددة في مسار المعطيات مثل وحدة ضرب مع مراكمة (multiply–accumulate) التي تمكن من حساب عبارة من الشكل T=T+M[i]*k باستعمال تعليمة واحدة فقط. ولأن معالجات الإشارة الرقمية تتعامل مع مصفوفات معطيات كبيرة فقد تم تزويد المعالج بوحدة عتادية خاصة وهي وحدة النفاذ المباشر للذاكرة وذلك لتخزين (أو جلب) المعطيات في الذاكرة وذلك على التوازي مع العمليات الحسابية في المعالج لتسريع التنفيذ وعدم اشغال المعالج بعمليات ادخال واخراج الإشارات مثل تحصيل المعطيات من الميكرفون واخراج نتيجة المعالجة على مكبر الصوت.

5. تقنيات الدارات المتكاملة:

يجب تنجيز أي معالج على دارة متكاملة. تشمل تقنيات الدارات المتكاملة على الكيفية التي يتم فيها تنفيذ التصميم الرقمي على دارة متكاملة. تسمى الدارة المتكاملة عادةً بالشريحة (chip) والتي تحتوي على مجموعة من الترانزستورات المصنعة من أنصاف النواقل (semiconductors) بالإضافة إلى مكونات أخرى. ومن أكثر تقنيات أنصاف النواقل انتشاراً تقنية CMOS (complementary metal oxide semiconductor).

تختلف تقنيات الدارات المتكاملة بمقدار تخصيص الدارة المتكاملة لتصميم معين. ولنفهم هذا الفرق، نذكر أولاً بأن الدارة المتكاملة تتكون من عدة طبقات (layers). تشكل الطبقات الدنيا الترانزستورات. وتشكل الطبقات المتوسطة العناصر المنطقية. أما السويات العليا فتصل بين العناصر بواسطة أسلاك. ولتنفيذ الدارة المتكاملة، يجب تنفيذ جميع الطبقات. ويأتي السؤال هنا من يقوم بتصميم كل طبقة ومتى؟. لذلك تصنف الدارات المتكاملة إلى أنواع وفقاً لهذا السؤال. وسنذكر هذه الأنواع تباعاً.

1.5. دارات متكاملة مخصصة بشكل كامل:

في الدارات المتكاملة المخصصة كلياً (Full-custom/VLSI) المصنعة بتقنية VLSI (Very Large Scale Integration)، يتم تصميم وأمثلة جميع الطبقات في التنفيذ الرقمي لنظام مضمن معين. وتشمل هذه الأمثلة على وضع الترانزستورات على الشريحة في مواقع مناسبة لتقصير طول الوصلات بينها ويتم تحديد معاملات تصميم الترانزستورات لأمثلة نقل الإشارة ومن ثم وصل الأسلاك بين الترانزستورات. وعند الانتهاء من تصميم جميع الطبقات، يتم إرسالها إلى المصنع لتصنيع الدارة المتكاملة.

إن تصميم دارات متكاملة مخصصة كلياً، والذي يشار إليه عادةً بالمصطلح VLSI له كلفة غير متكررة NRE عالية ووقت تنفيذ طويل نسبياً قد يصل لعدة شهور قبل الحصول على الدارة المتكاملة. ولكنه بالمقابل يمكن من الحصول على أداء ممتاز وحجم صغير واستهلاك طاقة منخفض. تستخدم هذه التقنية فقط في حالات الانتاج الكمي الكبير أو للتطبيقات التي تتطلب درجة عالية من المواصفات.

2.5.دارات متكاملة مخصصة جزئياً:

في الدارات المتكاملة المخصصة لتطبيق ASIC (Application Specific IC)، يتم بناء الطبقات الدنيا بشكل جزئي أو كلي وترك عملية انهاء الطبقات العليا للمصمم وهذا ما نسميه دارات ASIC مخصصة جزئياً (semi-custom ASIC). ففي تقنية مصفوفة البوابات (gate-array ASIC) يتم بناء الدارة على سوية الترانزستورات والبوابات وتبقى مهمة المصمم في وصل هذه البوابات بشكل مناسب لتحقيق التصميم المرغوب. أما في تقنية الخلايا المعيارية (standard-cell ASIC) فيتم تصميم الخلايا في السوية المنطقية مسبقاً مثل بوابات AND أو تركيبة AND-OR-INVERT ويبقى للمصمم ترتيب هذه القطع على الشريحة ووصلها مع بعضها لإتمام التصميم. وتعتبر هذه التقنية من التقنيات الأكثر شيوعاً من تقنيات ASIC وذلك

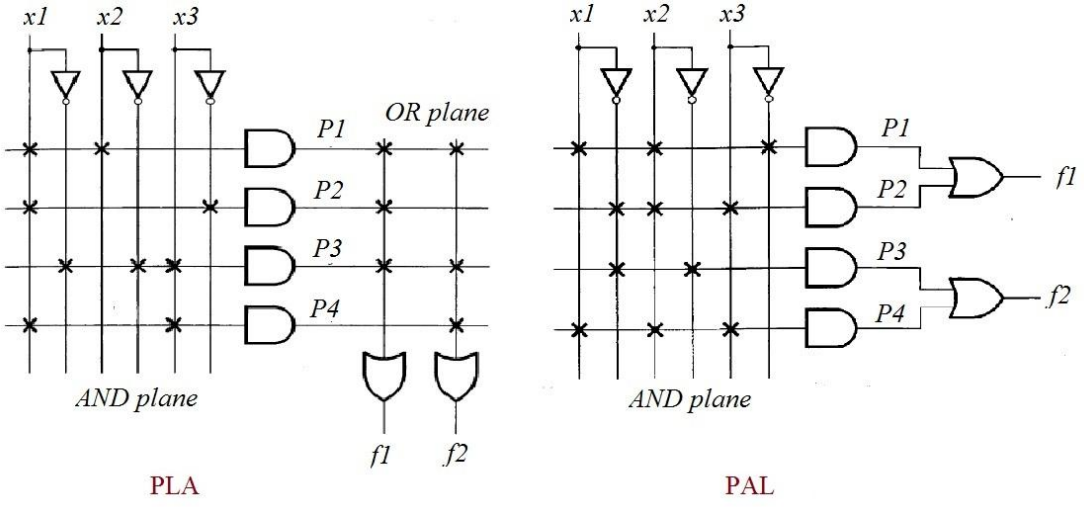
لأنها تعطي مواصفات جيدة من حيث الحجم والأداء وكلفة NRE أقل بكثير مقارنةً بتقنية الدارات المتكاملة المخصصة بالكامل. ولكن مع ذلك يتطلب تصنيع هذه الدارات عدة أسابيع أو حتى شهور.

3.5. دارات المنطق المبرمج:

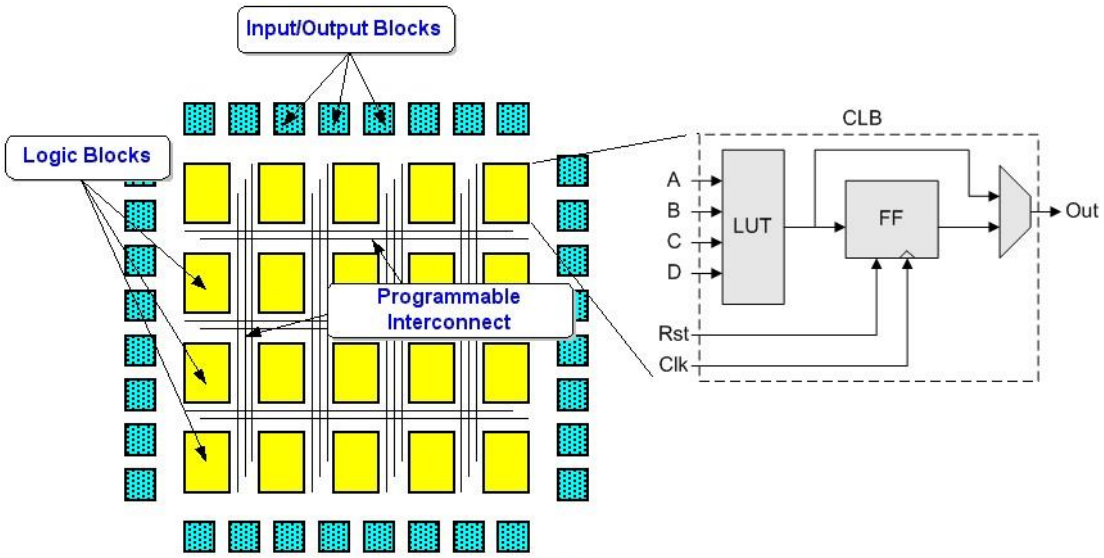
في دارات المنطق المبرمج PLD (Programmable Logic Device) تكون جميع الطبقات مصنعة مسبقاً في المعمل وبالتالي يمكن شراء الدارة المتكاملة وبرمجتها لتنفيذ تصميم معين. تشكل الطبقات الموجودة دارات قابلة للبرمجة إلا أن البرمجة هنا لها معنى ذو سوية أدنى من كتابة برنامج حاسوبي. حيث تشمل البرمجة التي تتم هنا على عملية انشاء أو قطع الوصلات بين الأسلاك التي تصل بين البوابات وذلك إما عن طريق تقنية قطع الصمامات (blowing fuse) أو وضع قيمة بت في قاطعة قابلة للبرمجة. تتم عادةً هذه البرمجة باستعمال تجهيزات صغيرة متصلة بالحاسب الشخصي.

يمكن تقسيم دارات المنطق المبرمج PLD إلى نوعين: دارات بسيطة ودارات معقدة.

تشكل دارات PLA (Programmable Logic Array) أحد الأنماط البسيطة لدارات PLD والمكونة من مصفوفة قابلة للبرمجة من بوابات AND ومصفوفة قابلة للبرمجة من بوابات OR. ونذكر أيضاً نمطاً بسيطاً آخر وهو دارات PAL (Programmable Array Logic) والتي تستعمل مصفوفة واحدة قابلة للبرمجة (بوابات AND فقط) والمصفوفة الأخرى ثابتة غير قابلة للبرمجة وذلك لخفض عدد المكونات القابلة للبرمجة ذات التكلفة المرتفعة.



وهناك نوع من دارات PLD المعقدة والتي تنمو في العقد الأخير بشكل متسارع جداً من حيث الانتشار وهي دارت FPGA (Field Programmable Gate Array). حيث تتميز دارات FPGA بقابلية وصل أكثر عمومية بين الكتل المنطقية بالمقارنة مع دارات PLA أو PAL وبالتالي فهي قادرة على تنفيذ تصاميم أكثر تعقيداً.



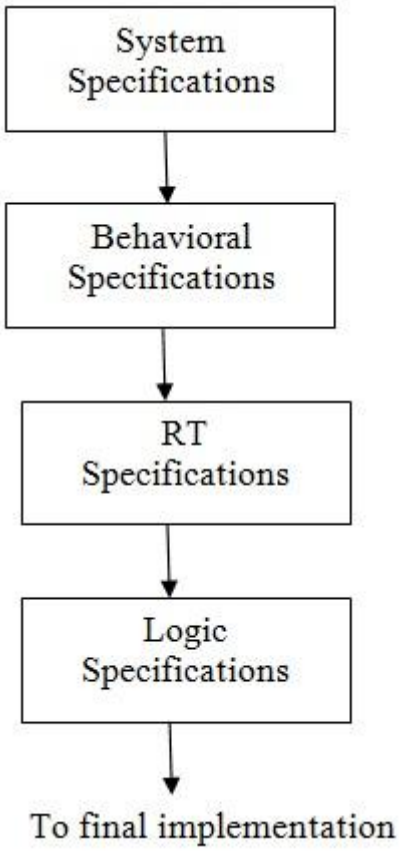
FPGA

توفر دارات PLD كلفة غير متكررة منخفضة وزمن تنفيذ سريع ولكن تكون عادةً ذات حجم أكبر من دارات ASIC وكلفة افرادية أعلى وتستهلك طاقة أكبر ومن الممكن أن تكون أبطئ (وخاصة دارات FPGA) ولكنها تقدم مواصفات معقولة ولذلك فهي مناسبة لتطوير النماذج الأولية بسرعة.

6. تقنيات التصميم:

تتضمن تقنيات التصميم الطريقة التي نحول فيها فكرة عمل النظام المرغوب إلى تنفيذ عملي. يجب علينا أن لا نكتفي فقط بتصميم النظام لأمثلة مقاييس التصميم وحسب ولكن يجب علينا فعل ذلك بسرعة لنواكب التطور في تقنيات الدارات المتكاملة. تقاس إنتاجية المصمم بعدد الترانزستورات التي يولدها المصمم (المستعملة في التصميم) في الشهر الواحد. لذلك كان تحسين الانتاجية للمصمم مركز اهتمام المطورين لعقود في المجالين البرمجي والبنية الصلبة.

لنفهم كيف نحسن عملية التصميم، يجب علينا أولاً فهم آلية التصميم نفسها. يبين الشكل واحدة من أهم منهجيات التصميم من الأعلى للأسفل (top–down) حيث يقوم المصمم بتفصيل التصميم حسب عدة مستويات تجريدية.



منهجية التصميم من الأعلى للأسفل ضمن عدة مستويات تجريدية.

في سوية النظام يوصف المصمم عمل النظام المرغوب بلغة ما (لغة طبيعية أو لغة برمجية مثل لغة C) ونسمي ذلك بمواصفات النظام (system specifications). ثم يفصل المصمم هذه المواصفات بتوزيع أقسام منها على العديد من المعالجات ذات الأغراض العامة و(أو) المعالجات ذات الغرض الوحيد وذلك ما يشكل التوصيف السلوكي (behavioral specifications) لكل معالج. ومن ثم يفصل المصمم هذه المواصفات إلى

مواصفات نقل السجل (register–transfer specifications) وذلك بتحويل التصميم السلوكي لمعالج ذو غرض عام إلى لغة التجميع أو بتحويل التصميم السلوكي لمعالج ذو غرض وحيد إلى عناصر نقل السجل (مثل سجلات، نواخب، مفككات ترميز، وحدات عاملة) وآلة الحالة (state machine). ثم يفصل المصمم مواصفات نقل السجل إلى مواصفات منطقية والتي تتألف من معادلات منطقية. ولا يشمل هذا التفصيل لغة التجميع للمعالجات ذات الأغراض العامة ولكن يشمل المعالجات ذات الهدف الوحيد فقط. وأخيراً، يفصل المصمم هذه المواصفات إلى لغة الآلة بالنسبة للمعالجات ذات الأغراض العامة وإلى سوية شبكة وصل البوابات (gate–level netlist) بالنسبة للمعالجات ذات الغرض الوحيد.

يوجد ثلاث منهجيات رئيسية في تحسين آلية التصميم لزيادة الإنتاجية والتي سنعرضها تباعاً فيمايلي:

1.6. الترجمة والتركيب:

تسمح عملية الترجمة والتركيب (compilation/synthesis) للمصمم من تحديد العمل المرغوب للنظام بطريقة تجريدية ويتم توليد السويات الدنيا من التنفيذ بشكل آلي.

تحول أدوات التركيب المعادلات المنطقية إلى شبكة متصلة من البوابات المنطقية تدعى شبكة وصل (netlist). كما تحول أدوات نقل السجل RT آلات الحالة المنتهية (finite–state machines) ونقل السجل إلى مسار للمعطيات ضمن عناصر RT ودارات تحكم بمعادلات منطقية. أما أدوات التركيب السلوكي فهي تحول البرنامج التسلسلي إلى آلات الحالة المنتهية وسجلات نقل. وبشكل مماثل يحول المترجم البرمجي البرنامج التسلسلي إلى مجموعة من تعليمات لغة التجميع. وأخيراً، تحول أدوات تركيب النظام مواصفات النظام إلى مجموعة من البرامج التسلسلية. ستتوضح معنا هذه المفاهيم بشكل أكبر في الفصول القادمة.

2.6.استعمال المكتبات والملكية الفكرية:

تسمح المكتبات (libraries) بإعادة استعمال تنفيذ معين موجود سلفاً. يسمح استعمال المكتبات بتحسين الانتاجية إذا كان زمن الحصول على هذه المكتبات واستعمالها أقل من زمن إعادة تنفيذها.

يمكن أن تتكون المكتبة في السوية المنطقية من تصاميم مكونة من بوابات وخلايا منطقية. ويمكن أن تتكون المكتبة في سوية نقل السجل RT من تصاميم لعناصر مثل السجلات والنواخب ومفككات الترميز وغيرها. كما يمكن أن تحتوي المكتبة في السوية السلوكية من مكونات شائعة الاستعمال مثل عناصر ضغط (compression) وواجهات مساري (bus interface) ومتحكمات عرض (controller display) أو حتى على معالجات ذات أغراض عامة.

لقد أدى التقدم الحاصل في تكامل النظم تغييراً في المكتبات المتوفرة. فبالإضافة إلى توفر المكتبات التي تتضمن دارات تكاملية جاهزة ومستقلة، أصبحت العديد من العناصر متوفرة كتصميم يمكن استعماله كجزء من نظام على جزء من دارة تكاملية جديدة يمكن تنفيذها بتقنية الدارات المبرمجة مثلاً. يدعى العنصر المتوفر

كتصميم بالنواة (core). دفع هذا التغيير في المكتبات على السوية السلوكية بظهور مصطلح الملكية الفكرية IP(Intellectual Property) للإشارة بأن هذه العناصر متوفرة بشكل تصميمي وليس مادي.
وأخيراً يمكن أن تتكون المكتبات على سوية النظام من نظم كاملة مصممة لحل مشكل معينة مثل معالجات مع نظم تشغيل مرافقة لتنفيذ واجهة ربط من خلال شبكة الانترنت مثلاً.

3.6. الاختبار والتحقق:

يشمل الاختبار والتحقق على التأكد من صحة عمل النظام المرغوب وذلك لتجنب عمليات التصحيح التصميمية على السويات التجريدية المنخفضة والتي تستهلك وقتاً كبيراً.
تشكل المحاكاة (simulation) الطريقة الأكثر شيوعاً لاختبار العمل الصحيح للتصميم. فمثلاً، تمكن المحاكيات على سوية البوابات المنطقية من عرض المخطط الزمني لإشارة الخرج من أجل إشارات دخل معطاة. وبالمثل تقوم محاكيات المعالجات ذات الأغراض العامة من تنفيذ تعليمات لغة التجميع.
وأيضاً، يقوم محاكي لغة التوصيف العتادية HDL (Hardware Description Language) بإجراء المحاكاة على سوية نقل السجل أو السوية السلوكية وعرض إشارات الخرج من أجل إشارات دخل معطاة.
على سوية النظام، يقوم محاكي النموذج (model simulator) بالتحقق من صحة المواصفات الابتدائية للنظام باستعمال نموذج حسابي مجرد ومستقل عن تقنية المعالج وذلك من أجل التحقق من صحة واكتمال المواصفات.

7. خلاصة:

يزداد عدد النظم المضمنة كل عام وذلك بسبب تزويد التجهيزات الكهربائية بالعناصر الحسابية. تشترك النظم المضمنة بالعديد من الخصائص والتي تميزها عن الحواسيب الشخصية وتضع مجموعة من التحديات الهامة للتصميم. ومن أبرز هذه التحديات هو أمثلة مقاييس التصميم والذي هو من الأمور الصعبة لتعارض المقاييس فيما بينها. ومن أبرز هذه المقاييس هو زمن الوصول للسوق وذلك بسبب التطور المتسارع للنظم المضمنة. الأمر الذي أدى إلى ظهور تقنيات جديدة في تصنيع الدارات المتكاملة مثل دارات المنطق المبرمج والأدوات البرمجية المصاحبة لتنفيذ التصميم باستعمال هذه التقنيات الجديدة.

8. أسئلة الفصل الأول:

أسئلة عامة:

1. عرف النظام المضمن.

هو نظام حاسوبي يحتوي العديد من النظم الجزئية وهو مُصمّم لتنفيذ مهمة أو عدة مهام محددة سلفاً ضمن الزمن الحقيقي.

2. اذكر ثلاثة من خصائص النظم المضمنة التي تميزها عن النظم الحاسوبية التقليدية.

- ذات عمل محدد: ينفذ النظام المضمّن عادةً برنامجاً وحيداً بشكل متكرر.
- مقيدة بشكل كبير: ويشمل ذلك في معظم الاحيان قيود على الكلفة والحجم واستهلاك الطاقة.
- تتجاوب بالزمن الحقيقي: يجب أن يتفاعل النظام مع تغيرات الظروف المحيطة ويقوم بتنفيذ بعض الحسابات والحصول على النتائج من دون أي تأخير زمني يذكر.

3. ما هو المقياس التصميمي؟

مقياس التصميم هو مقدار قابل للقياس يوصف ميزة معينة في تصميم النظام.

4. اذكر اثنين من المقاييس التصميمية التي تتعارض فيما بينها مع تعليل ذلك بالشرح.

(ملاحظة: الإجابات متعددة)

مقياس الحجم ومقياس الأداء. فكلما صغرنا حجم النظام يعاني الأداء من انخفاض في السرعة.

5. ماهي الكلفة الهندسية غير المتكررة NRE؟

هي كلفة تصميم النظام التي تدفع لمرة واحدة فقط.

6. عدد تقنيات المعالجات المستعملة في النظم المضمنة؟

- المعالجات ذات الأغراض العامة
- المعالجات ذات الغرض الوحيد
- المعالجات المخصصة لتطبيق

7. عدد تقنيات الدارات المتكاملة.

- دارات متكاملة مخصصة بشكل كامل
- دارات متكاملة مخصصة جزئياً
- دارات المنطق المبرمج

8. ما هو الفرق بين دارات PLA ودارات PAL القابلة للبرمجة؟

تحتوي دارات PLA على صفين من البوابات المنطقية القابلة للبرمجة AND وOR. أما في دارات PAL فصف البوابات OR يكون موصول بشكل ثابت وغير قابل للبرمجة.

أسئلة خيارات متعددة:

1. المتحكم الصغري هو نوع من:

A. المعالجات ذات الأغراض العامة

B. المعالجات المخصصة لتطبيق

C. المعالجات ذات غرض وحيد

D. معالج عتادي

2. من مزايا دارات المنطق المبرمج مقارنة بدارات المنطق المبرمج الأخرى

A. ذات كلفة قليلة

B. تحتوي على صفين من بوابات AND وOR قابلين للبرمجة.

C. بقابلية وصل أكثر عمومية بين الكتل المنطقية.

D. سرعة أكبر في الأداء.

3. الدارات المتكاملة VLSI هي دارات:

A. مخصصة بشكل كامل.

B. مخصصة بشكل جزئي.

C. قابلة للبرمجة.

D. معقدة كبيرة الحجم.

4. النظام المضمن بالزمن الحقيقي يعني أنه:

A. يحتوي على ساعة بالزمن الحقيقي للدلالة على الوقت.

B. يتفاعل النظام مع تغيرات الظروف المحيطة دون تأخير زمني يذكر.

C. نظام مضمن ذو مواصفات مقيدة من حيث الكلفة واستهلاك الطاقة.

D. يواكب التطورات الحالية ويكون أسرع في الظهور إلى السوق.

5. تتميز النظم المضمنة عن النظم الحاسوبية التقليدية بما يلي:

A. هي ذات عمل محدد

B. مقيدة بشكل كبير من حيث الكلفة والحجم واستهلاك الطاقة.

C. تتجاوب بالزمن الحقيقي.

D. كل ما سبق.

6. ماذا يعني مفهوم برمجة دارات المنطق المبرمج؟

A. كتابة برنامج التطبيق لمعالج مخصص لتطبيق.

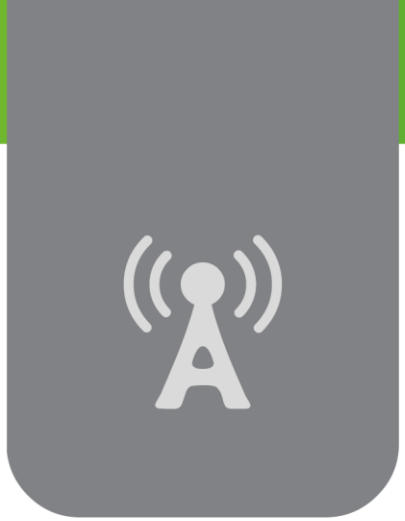
B. تحميل البرنامج على المعالج العام باستخدام تجهيز موصول مع الحاسب.

C. عملية انشاء أو قطع الوصلات بين الأسلاك التي تصل بين البوابات.

D. كتابة برنامج يتحكم بدارات منطقية.

الإجابات الصحيحة:

رقم التمرين	الإجابة الصحيحة
1	A
2	C
3	A
4	B
5	D
6	C



الفصل الثاني:
أساسيات الدارات المنطقية

الكلمات المفتاحية:

دارات المنطق التراكبي، دارات المنطق التعاقبي، جدول الحقيقة، مخططات كارنو، آلة مور، آلة ميلي.
Combinational Logic, Sequential Logic, Truth Table, Karnaugh diagrams, Moore machine, Mealy Machine.

الملخص:

يهدف هذا الفصل إلى تقديم مراجعة سريعة لطرق تصميم الدارات المنطقية والمفاهيم الأساسية ذات الصلة وذلك تمهيداً للفصول القادمة التي ترتكز على المفاهيم المقدمة في هذا الفصل.
نشرح في البداية طرق تصميم الدارات التراكبية والعناصر المنطقية الأساسية المستعملة في التصميم وهي البوابات المنطقية الأساسية مع تذكير بجدول الحقيقة ومخططات كارنو. ثم ننتقل إلى طرق تصميم الدارات التعاقبية والعناصر المنطقية الأساسية المستعملة مثل السجل والناخب ومفكك الترميز. نبين بشكل خاص تصميم آلة الحالة بنوعيها آلة الحالة مور (Moore) وآلة الحالة ميلي (Mealy).

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- البوابات المنطقية الأساسية وجدول الحقيقة.
- العناصر الأساسية في المنطق التراكبي وتشمل الناخب والجامع والمقارن ومفكك الترميز.
- طرق تصميم الدارات التراكبية باستخدام مخططات كارنو.
- العناصر الأساسية في المنطق التعاقبي مثل القلابات والسجل والعداد.
- تقنيات تصميم الدارات التعاقبية وتشمل على تصميم آلة الحالة المنتهية.

المخطط:

يضم هذا الفصل 3 وحدات تعليمية هي:

1. مقدمة.
2. تصميم دارات المنطق التراكبي وعناصره الأساسية.
3. تصميم دارات المنطق التعاقبي وعناصره الأساسية.

يهدف هذا الفصل إلى تقديم مراجعة سريعة لطرق تصميم الدارات المنطقية والمفاهيم الأساسية ذات الصلة وذلك تمهيداً للفصول القادمة التي ترتكز على المفاهيم المقدمة في هذا الفصل.
نشرح في البداية طرق تصميم الدارات التراكبية والعناصر المنطقية الأساسية المستعملة في التصميم وهي البوابات المنطقية الأساسية مع تذكير بجدول الحقيقة ومخططات كارنو. ثم ننتقل إلى طرق تصميم الدارات التعاقبية والعناصر المنطقية الأساسية المستعملة مثل السجل والناخب ومفكك الترميز. نبين بشكل خاص تصميم آلة الحالة بنوعيها آلة الحالة مور (Moore) وآلة الحالة ميلي (Mealy).

1. مقدمة:

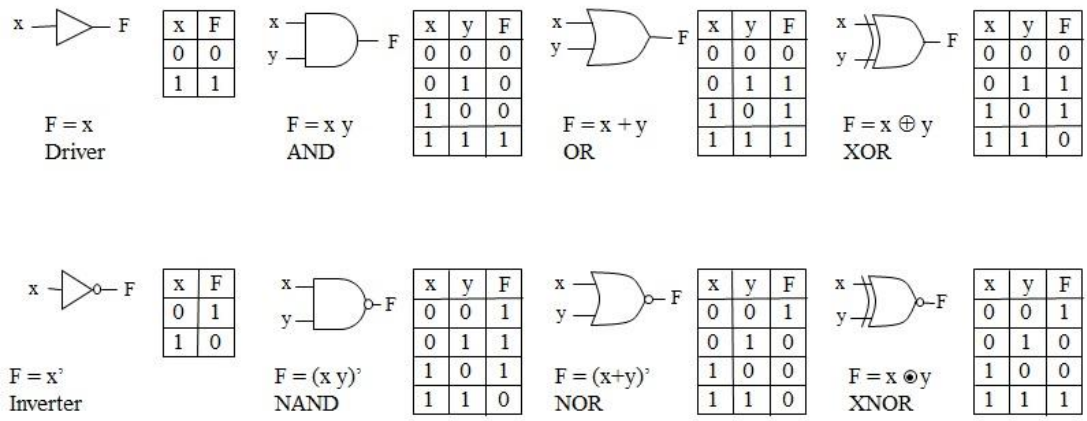
تتكون الدارات الرقمية من مجموعة من الترانزستورات المجمعة وفق تراكيب معينة، حيث يعمل الترانزستور كقاطعة فصل أو وصل بناءً على إشارة تحكم. تتكون الدارات الرقمية الحديثة من عدد كبير من الترانزستورات والذي قد يصل إلى المليارات. وبسبب هذا التعقيد المتزايد للدارات الرقمية الحديثة، لم يعد المصمم يعمل على سوية الترانزستورات بل يعمل باستخدام عناصر بنوية بسويات تجريدية أعلى. في السوية الثانية للتصميم، يستعمل المصمم البوابات المنطقية مثل AND وOR وNOT، حيث تتكون كل بوابة من تركيبة معينة من عناصر السوية الأدنى أي الترانزستورات. وفي السوية الثالثة يستعمل المصمم عناصر أكثر تعقيداً من البوابات مثل النواخب والجوامع والسجلات وغيرها والتي تتكون من تراكيب معينة من البوابات المنطقية. وتسمى هذه السوية بسوية نقل السجلات (register transfer level). وفي السوية الرابعة، يستعمل المصمم الوصف السلوكي للدارة الرقمية من حيث تحديد قيمة المخارج المرغوبة بدلالة قيم إشارات الدخل. وتسمى هذه السوية بالسوية السلوكية (behavioral level) والتي تراكيب معينة تستخدم عناصر من سوية نقل السجلات لتكوين عناصر مثل آلة حالة منتهية أو وحدات توصيفية أخرى سنتعرف عليها لاحقاً.
سوف نستعرض في هذا الفصل طرق تصميم الدارات الرقمية على سوية البوابات. كما سنتعرف على العناصر الأساسية المستعملة في سوية نقل السجلات. يمكن تصنيف الدارات الرقمية إلى صنفين وهما: دارات المنطق التراكبي (combinational logic circuits) ودارات المنطق التعاقبي (sequential logic circuits).

2. المنطق التراكبي:

دارة المنطق التراكبي (combinational circuit) هي دارة يكون خرجها في لحظة ما تابعاً فقط لقيم الدخل في هذه اللحظة، أي إن هذه الدارة لا تحتوي على ذاكرة لقيم الدخل السابقة.
ويعتمد تصميم هذه الدارات على المعادلات المنطقية والتي يمكن تنفيذها باستعمال البوابات المنطقية.

1.2. الترانزستورات والبوابات المنطقية:

يشكل الترانزستور العنصر الأساسي في الدارات الرقمية، ويشكل تركيب مجموعة من الترانزستورات عناصر من سوية تجريدية أعلى تسمى البوابات المنطقية (logic gates) التي يستخدمها المصممون في بناء النظم الرقمية.

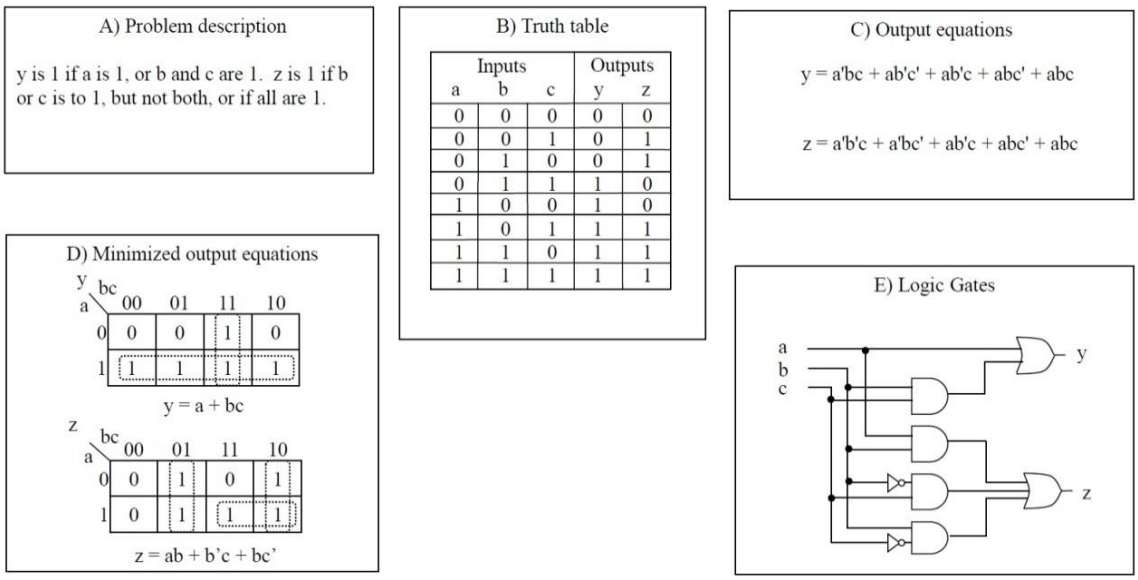


البوابات المنطقية الشهيرة وجدول الحقيقة الموافق

يبين الشكل ثماني بوابات منطقية أساسية. تُمثّل كل بوابة منطقية من خلال الرمز البياني والمعادلة المنطقية وجدول الحقيقة. في جدول الحقيقة نضع مداخل البوابة على اليسار ومخرجها الوحيد على اليمين، فعلى سبيل المثال، يكون خرج بوابة AND مساوياً 1 عندما يكون كلا المدخلين قيمتهما 1. ويكون خرج بوابة OR مساوياً 1 عندما يكون أحد المدخلين على الأقل قيمته 1.
مع أن بوابات AND وOR هي بوابات أسهل للفهم منطقياً إلا أن البوابات NAND وNOR هي أكثر استخداماً وذلك بسبب سهولة تنفيذها باستخدام ترانزستورات من الناحية العملية.

2.2. تصميم دارات المنطق التراكبي:

يمكن استخدام تقنيات بسيطة لتصميم دارة تراكبية باستعمال البوابات المنطقية الأساسية كما هو مبين في الشكل التالي.



مثال عن تصميم دارة تراكبية بسيطة

في البداية نبدأ بتوصيف المسألة من خلال توصيف المخارج بدلالة المداخل، ومن ثم نقوم بترجمة هذا التوصيف إلى جدول الحقيقة حيث نحدد قيم الخرج من أجل جميع القيم الممكنة للدخل. وانطلاقاً من جدول الحقيقة نستطيع تحديد المعادلة المنطقية لكل مخرج من المخارج بدلالة المداخل وذلك على شكل مجموعة من الحدود بحيث يوافق كل حد سطراً من جدول الحقيقة يكون الخرج فيه مساوياً 1. بعد ذلك نترجم هذه المعادلات إلى مخطط دارة رقمية.

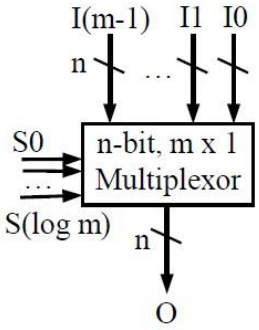
باستخدام هذه الطريقة المباشرة نحصل غالباً على دارات ذات عدد كبير من البوابات، إلا أنه يمكن تبسيط المعادلات المنطقية باستخدام العمليات الجبرية على المعادلات بهدف تقليل عدد البوابات المنطقية اللازمة لتحقيق كل معادلة. كما يمكن استخدام مخططات كارنو (Karnaugh) لهذا الغرض، وعند الحصول على المعادلة النهائية يمكن تحويلها إلى مخطط دارة منطقية.

3.2. العناصر التراكبية على سوية نقل السجلات:

مع أننا نستطيع تصميم جميع الدارات التراكبية بالطريقة السابقة إلا أن هذه الطريقة تصبح عالية التعقيد من أجل الدارات ذات العدد الكبير من المداخل، فعلى سبيل المثال، من أجل دارة ذات 16 مدخل نحصل على جدول حقيقة يضم 216 سطراً. وتتمثل إحدى الطرق المستخدمة لتخفيض هذا التعقيد في استخدام عناصر تراكبية ذات سوية أعلى من البوابات المنطقية تُدعى عناصر على سوية نقل السجلات (Register–Transfer (RT). سنذكر فيما يلي بعضاً منها بشكل مختصر.

1.3.2. الناخب:

يسمح الناخب(Multiplexer) بتمرير مدخل واحد فقط Im من المداخل إلى الخرج O، حيث يتم تحديد رقم المدخل المطلوب m على خطوط الانتخاب S. فإذا كان للناخب m مدخلاً كان عدد خطوط الانتخاب اللازمة لاختيار المدخل المطلوب هو $\log_2(m)$. نسمي هذا الناخب ناخب m×1 لأن له m مدخلاً ومخرجاً واحداً. على سبيل المثال يكون للناخب 8×1 ثمانية مداخل ومخرج وحيد، ويكون عدد خطوط الانتخاب مساوياً $\log_2(8)=3$.



O =
I0 if S=0..00
I1 if S=0..01
…
I(m-1) if S=1..11

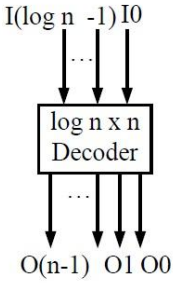
ناخب m×1

في الحياة العملية، نستخدم غالباً نواخب أكثر تعقيداً من ذلك وتسمى نواخب على N بت، حيث تكون كل المداخل والمخارج مكونة من عدد من البتات بدلاً من بت واحد، ويبقى عدد خطوط الانتخاب نفسه وهو يتعلق بعدد المداخل m وليس N. على سبيل المثال، يقوم الناخب 8×1 على 4 بت بتوصيل أحد المداخل الثمانية (المكون من 4 بتات) إلى الخرج المكون من 4 بتات أيضاً باستخدام 3 خطوط انتخاب.

2.3.2.مفكك الترميز:

يمتلك مفكك الترميز (Decoder) m مخرجاً وعدداً من خطوط الانتخاب قدره $\log_2(m)$ وليس له مداخل معطيات. فإذا كانت القيمة على خطوط الانتخاب هي m يكون المخرج ذو الرقم m مساوياً الواحد، أما بقية المخارج فتكون جميعها صفراً. نسمي هذا العنصر مفكك ترميز m×$\log_2(m)$.

على سبيل المثال يمتلك مفكك الترميز 3×8 ثمانية مخارج و ثلاثة خطوط انتخاب. كما ويحتوي مفكك الترميز على خط دخل إضافي يدعى خط التأهيل (enable). عندما يكون خط التأهيل صفراً تكون جميع المخارج صفراً، وعندما يكون خط التأهيل على القيمة 1 يعمل مفكك الترميز كما ذكرنا سابقاً.

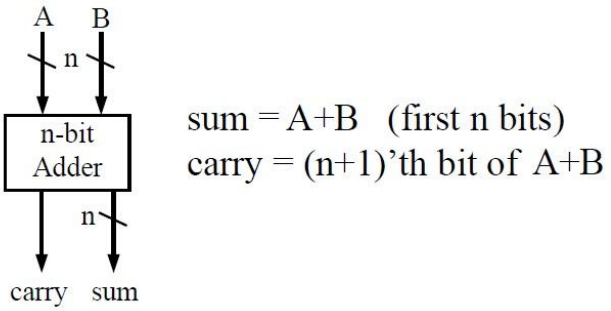


O0 =1 if I=0..00
O1 =1 if I=0..01
…
O(n-1) =1 if I=1..11

مفكك ترميز

3.3.2. الجامع:

الجامع (Adder) هو دارة رقمية تجمع قيمتين A و B كل منهما على N بت ويعطي ناتج الجمع sum على N بت مع بت إضافي يعبر عن قيمة الحمل (carry). على سبيل المثال، يمتلك الجامع على 4 بت مدخلين A و B على 4 بت وخرج sum على 4 بت ومخرجاً للحمل على بت واحد، فإذا كان الدخل A=1010 والدخل B=1001 يكون الخرج sum=0011 وبت الحمل carry=1.

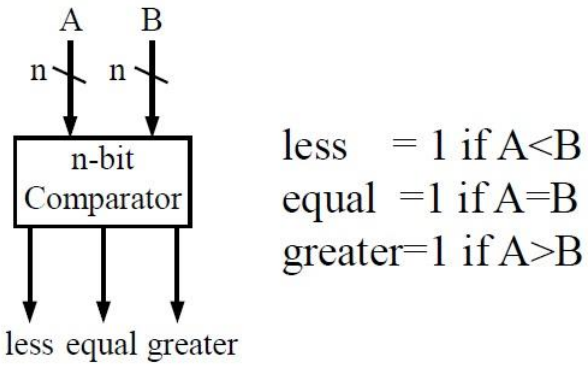


جامع على n بت

غالباً ما يتم تزويد الجامع بمدخل حمل أيضاً وذلك من أجل ربط الجوامع بشكل متعاقب لتشكيل جامع على عدد بتات أكبر.

4.3.2. المُقارِن:

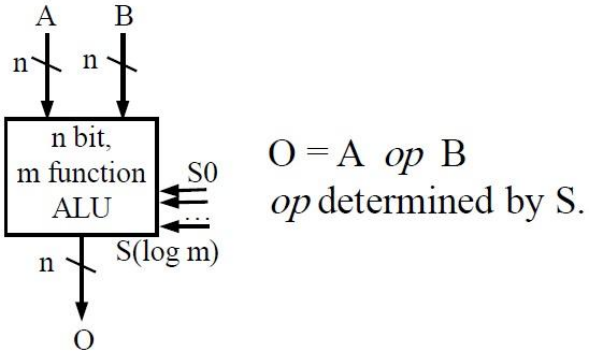
يمتلك المقارن (Comparator) مدخلين A و B كل منهما على n بت وله ثلاثة مخارج هي less و equal و greater. تشير المخارج إلى نتيجة المقارنة بين A و B، فإذا كان A=1010 و B=1001 يكون less=0 و equal=0 و greater=1.



مقارن على n بت

5.3.2. وحدة الحساب والمنطق:

تنفذ وحدة الحساب والمنطق ALU (Arithmetic and Logic Unit) العديد من العمليات الحسابية والمنطقية بين مدخلين A وB كل منهما على n بت، ويتم اختيار العملية المطلوبة من خلال خطوط الانتخاب S. من العمليات الشائعة التي تنفذها هذه الوحدة هي عمليات الجمع والطرح وAND و OR.



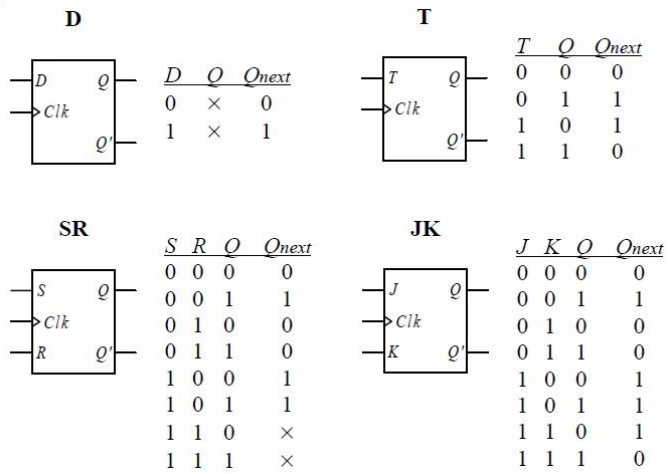
وحدة حساب ومنطق على n بت

3. المنطق التعاقبي:

دارة المنطق التعاقبي (Sequential Logic) هي دارة رقمية يكون خرجها في لحظة ما تابعاً لقيم الدخل في اللحظة الحالية واللحظات السابقة، أي إن دارات المنطق المتتابع تحتوي على ذاكرة. ومن أشهر دارات المنطق المتتابع الأساسية دارة القلاب (flip–flop).

1.3. القلابات:

يخزن القلاب (flip–flop) بتاً واحداً، وهناك عدة أنواع للقلابات وهي كالتالي:



انواع القلابات

- **القلاب من نوع T**
- **القلاب من نوع D (D flip–flop):** وهو أبسط أنواع القلابات. يمتلك هذا القلاب مدخلين هما D وclock. عندما يكون clock=1 تخزن قيمة الدخل D في القلاب وتظهر هذه القيمة على الخرج Q. وعندما يكون clock=0 يتم تجاهل قيمة الدخل D ويبقى الخرج Q على القيمة المخزنة سابقاً.
- **القلاب من النوع SR (SR flip–flop):** لهذا القلاب ثلاثة مداخل وهي S وR وclock. عندما يكون clock=1 يتم اختبار قيم الدخل S وR، فإذا كانت S=1 يتم تخزين قيمة 1 في القلاب، وإذا كانت قيمة R=1 يتم تخزين القيمة 0 في القلاب، وإذا كان كلاهما صفراً لا يقوم القلاب بأي تغيير، أما إذا كان كلاهما 1 تكون حالة القلاب غير محددة. وبالتالي يعبر المدخل S عن عملية وضع على الواحد (set)، ويعبر المدخل R عن عملية تصفير (reset). عندما تكون clock=0 يكون تصرف القلاب من هذا النوع مشابهاً لتصرف القلاب من النوع D أي يتم تجاهل قيم الدخل ويبقى الخرج على القيمة المخزنة سابقاً.

- **القلاب من النوع JK (JK flip–flop):** يشبه هذا القلاب النوع SR باستثناء أنه عندما يكون كلا المدخلين J وK على الواحد فإن القلاب يعكس القيمة المخزنة سابقاً فإذا كانت 0 تصبح 1 وإذا كانت 1 تصبح 0.

غالباً ما يتم تصميم القلابات لتعمل على الجبهة (edge–triggered) لإشارة الساعة clock أي فقط عند انتقال هذه الإشارة من الصفر إلى الواحد (جبهة صاعدة) أو الانتقال من الواحد إلى الصفر (جبهة هابطة)، وذلك لتجنب العمل العشوائي للقلاب عند وجود ضجيج على إشارات الدخل.

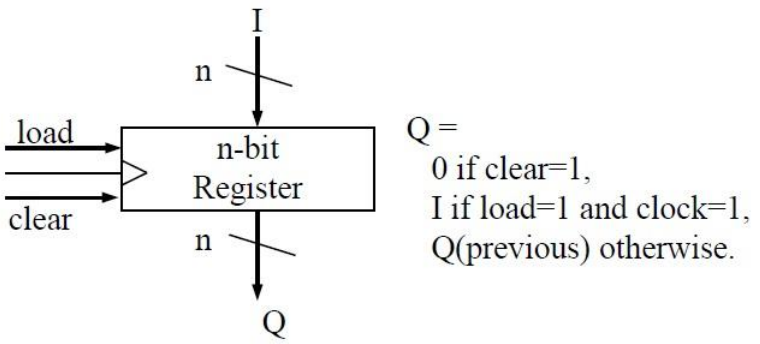
2.3. العناصر التعاقبية على سوية نقل السجلات:

كما أن هناك عناصر تراكبية على سوية نقل السجلات، هناك أيضاً عناصر تتابعية أكثر تعقيداً من القلابات على سوية نقل السجلات. نذكر فيما يلي أهم هذه العناصر:

1.2.3. السجل:

يخزن السجل (register) N بت من إشارات الدخل I المكونة من N بت، وتظهر هذه القيمة المخزنة على المخرج Q المكون من N بت أيضاً.

يحتوي السجل عادة على مدخلي تحكم على الأقل وهما clock وload، حيث لا يقوم السجل بتخزين قيم الدخل I عند حصول جبهة على إشارة الساعة clock إلا إذا كانت قيمة load=1، كما أن للسجل مدخل تحكمي آخر شائع الاستخدام وهو مدخل reset يؤدي إلى تصفير محتوى السجل بغض النظر عن قيم الدخل I.

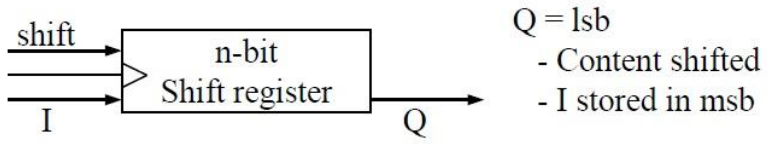


سجل على n بت

بما أن تخزين المعطيات في السجل يتم دفعة واحدة من قيم الدخل I على N بت فإننا نسمي هذا السجل بالسجل المتوازي (parallel register) وذلك لتمييزه عن نوع آخر من السجلات وهو سجل الإزاحة (shift register) الذي سنشرح عمله في الفقرة التالية.

2.2.3. سجل الإزاحة:

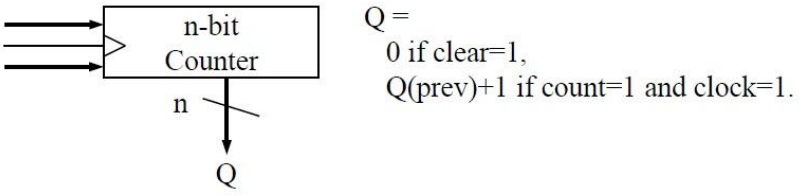
يخزن سجل الإزاحة (shift register) N بت ولكن لا يتم ذلك على التوازي، حيث يمتلك هذا السجل مدخلاً للمعطيات على بت واحد فقط بالإضافة إلى مدخلي تحكم على الأقل وهما clock و shift. عند حصول جبهة على إشارة الساعة clock وقيمة الدخل shift=1 يتم تخزين قيمة الدخل في البت رقم N ويتم نقل قيمة البت N السابقة إلى البت رقم N−1، وقيمة البت رقم N−1 إلى البت رقم N−2 وهكذا نزولاً حيث يتم تخزين البت الثاني في البت الأول. يكون لهذا السجل عادة مخرج وحيد يظهر عليه قيمة البت الأول.



سجل إزاحة على n بت

3.2.3. العداد:

العداد (counter) هو سجل يزيد محتواه بمقدار واحد عند حصول جبهة على إشارة الساعة ومدخل تأهيل العد count=1. يمتلك العداد في أبسط أشكاله مدخل clear يؤدي إلى تصفير العداد، وهناك أشكال متقدمة للعداد قادرة على الزيادة أو الإنقاص بمقدار واحد باستخدام إشارة تحكم إضافية Up و DN حيث يدل المدخل UP=1 على أننا نريد عملية عد تصاعدي، ويدل المدخل DN=1 على أننا نريد عملية عد تنازلي. كما يتم تزويد بعض العدادات بميزة شحن القيمة الابتدائية لكي يقوم العداد بالعد انطلاقاً من هذه القيمة الابتدائية وليس من الصفر. في هذه الحالة يجب أن يكون للعداد مدخلاً I على N بت ومدخلاً تحكمياً load لإعطاء أمر الشحن بالقيمة الابتدائية.



عداد على n بت

يمكن أن تعمل جميع إشارات التحكم بشكل متزامن (synchronous) أو غير متزامن (asynchronous). يأخذ الخط التحكمي المتزامن مفعوله فقط عند حصول جبهة على خط الساعة، أما الخط التحكمي غير

المتزامن فيأخذ مفعوله فوراً بغض النظر عن إشارة الساعة. على سبيل المثال، يكون المدخل التحكمي clear غير متزامن، أي إنه يتم تصفير السجل أو العداد فوراً عند تفعيل هذا المدخل.

3.3. تصميم دارات المنطق المتتابع:

يمكن تصميم دارات المنطق المتتابع بشكل ممنهج وفق عدة خطوات، ويعتمد مفهوم هذه الدارات على مفهوم حالة الآلة المنتهية (Finite-State Machine) أو FSM اختصاراً. حيث يتم نمذجة عمل دارة المنطق المتتابع على شكل تتابع عدد منتهٍ من الحالات وتنتقل الدارة من حالة إلى أخرى حسب الحالة الحالية للدارة وقيم الدخل.

هناك نوعين لحالة الآلة المنتهية وهما:

- آلة مور (Moore machine): تعتمد قيم الخرج لحالة الآلة المنتهية من هذا النوع في لحظة ما على الحالة في هذه اللحظة.
- آلة ميلي (Mealy machine): تعتمد قيم الخرج لحالة الآلة المنتهية من هذا النوع في لحظة ما على الحالة وقيم الدخل في هذه اللحظة.

نقتصر في دراستنا هنا على حالة الآلة المنتهية المتزامنة، أي إنه لا يتم الانتقال من حالة إلى أخرى إلا عند حدوث جبهة لإشارة الساعة.

يتم ترقيم جميع حالات الدارة بأرقام مختلفة (مرتبة أو غير مرتبة) ويتم تخزين قيمة الحالة في سجل الحالة ضمن سجل على N بت ونختار قيمة N بحيث تكون كافية لتخزين أرقام جميع الحالات، فعلى سبيل المثال إذا كان للدارة أربع حالات مختلفة وتم ترقيمها من 0 إلى 3 فيكفي اختيار N=2 لتمثيل أرقام جميع الحالات.

مثال:

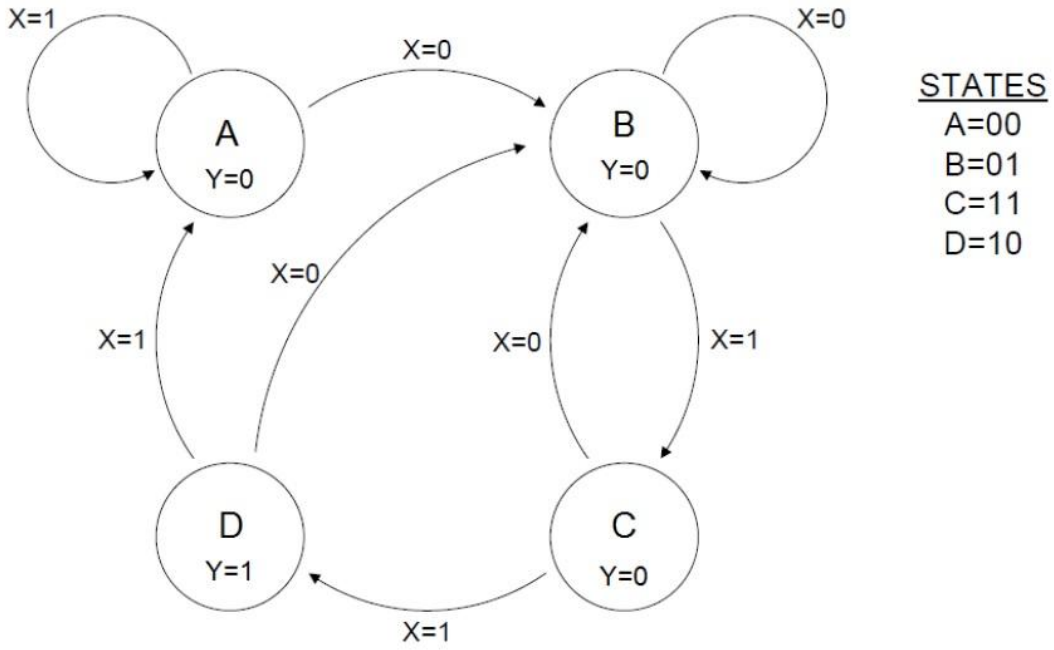
لنفرض أننا نريد تصميم دارة تتابعية لها مدخلان x وclock ومخرج وحيد هو y. يمثل الدخل x سلسلة من البتات وclock هي إشارة الساعة الموافقة. نريد أن يعطي الخرج y نبضة بعد الكشف عن تعاقب القيم 010 في إشارة الدخل x كما هو موضح في المثال التالي:

x= 0 1 1 0 1 0 0 1 0 1 0 1 1

y= 0 0 0 0 0 1 0 0 1 0 1 0 0

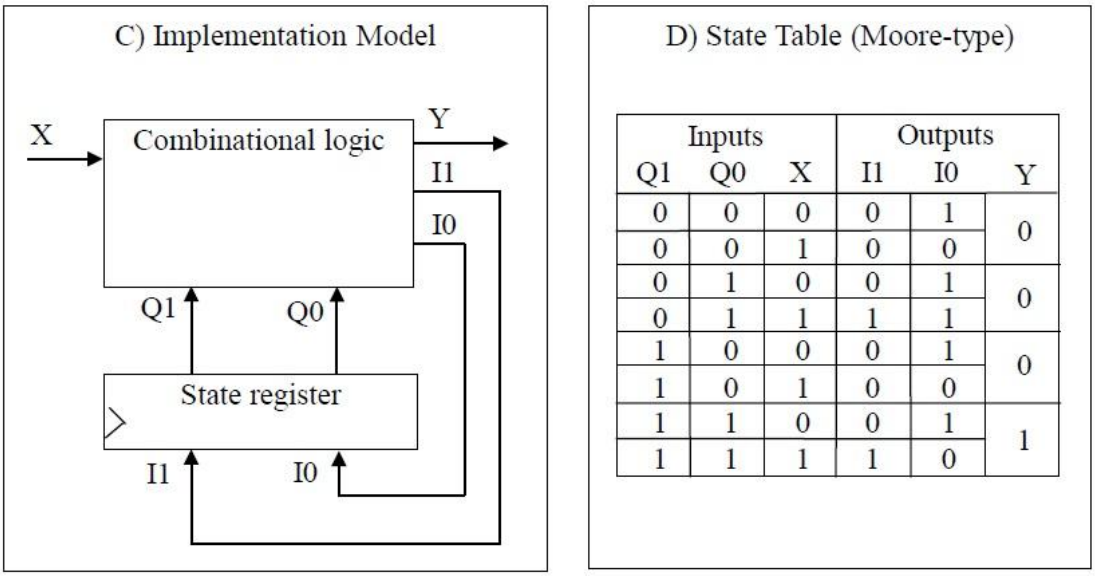
لتصميم هذه الدارة نقوم بتحويل المسألة إلى مخطط حالة آلة منتهية FSM. في الحالة الأولى تكون الدارة في حالة انتظار كشف 0 في إشارة الدخل x وبعد الكشف عن 0 تنتقل الدارة إلى حالة الكشف عن 1 يلي الصفر الأول، وبعد ذلك تنتقل الدارة إلى حالة الكشف عن 0 يلي الواحد، وفي هذه الحالة يتم وضع قيمة الخرج على y=1 أما في بقية الحالات فيكون الخرج y=0.

لدينا هنا أربع حالات للدارة (A, B, C, D) سنقوم بترقيمها بالأرقام (00, 01, 11, 10). يتم تمثيل كل حالة بدائرة وعليها اسم الحالة كما هو موضح في الشكل التالي:



مخطط الحالة لكاشف سلسلة

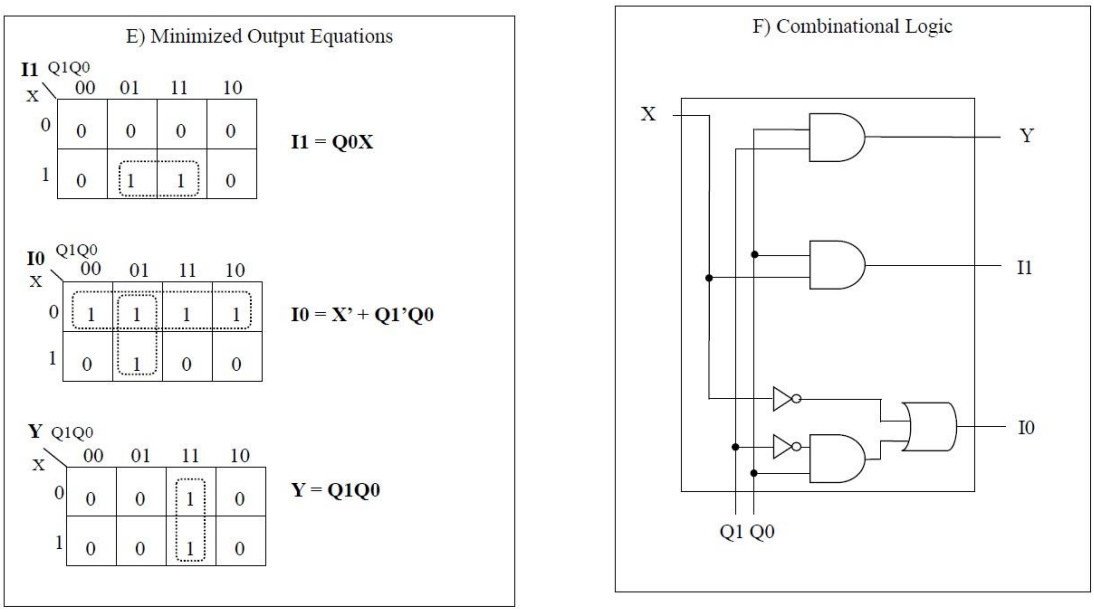
بما أن الخرج يتعلق بالحالة فقط فلدينا هنا حالة آلة منتهية من نمط مور، وعلينا تصميم دارة تراكبية لتحديد قيمة الحالة التالية انطلاقاً من قيمة الحالة الحالية وقيمة الدخل x.



D) State Table (Moore-type)

Inputs			Outputs		
Q1	Q0	X	I1	I0	Y
0	0	0	0	1	0
0	0	1	0	0	
0	1	0	0	1	0
0	1	1	1	1	
1	0	0	0	1	0
1	0	1	0	0	
1	1	0	0	1	1
1	1	1	1	0	

المخطط الصندوقي وجدول الحقيقة لكاشف السلسلة



اختصار المعادلات المنطقية باستعمال مخططات كارنو والدارة النهائية

4.خلاصة:

تتكون الدارات الرقمية من عناصر تحتوي على ترانزستورات مجمعة وفق تركيبة معينة. تقسم هذه العناصر إلى عدة سويات تجريدية وفقاً لتعقيد العنصر من البوابات المنطقية البسيطة إلى العناصر على سوية نقل السجلات مثل الناخب والجامع والسجل والعداد. تصمم الدارات الرقمية التراكبية على سوية البوابات باستعمال منهجية بسيطة تعتمد على جدول الحقيقة والمعادلات المنطقية ومخططات كارنو. ويمكن تصميم الدارات التعاقبية باستعمال آلة الحالة المنتهية والتي تحتوي بدورها على سجلات ودارات تراكبية أو يمكن استعمال طرق خاصة أخرى ولكنها غير منهجية وتتعلق بالتصميم المطلوب.

5. أسئلة الفصل الثاني:

أسئلة عامة:

1. ما هي دارة المنطق التراكبي؟

هي دارة يكون خرجها في لحظة ما تابعاً فقط لقيم الدخل في هذه اللحظة، أي إن هذه الدارة لا تحتوي على ذاكرة لقيم الدخل السابقة.

2. ارسم جدول الحقيقة للبوابة المنطقية NAND مداخلها x و y ومخرجها F.

يبين الجدول التالي جدول الحقيقة لبوابة NAND.

x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

3. ما هي وظيفة الناخب m×1؟

يسمح الناخب بتمرير مدخل واحد فقط من المداخل إلى الخرج، حيث يتم تحديد رقم المدخل المطلوب m على خطوط الانتخاب S.

4. ما هي وظيفة مفكك الترميز n×m؟

لمفكك الترميز m مخرج و n خط انتخاب.

إذا كانت القيمة على خطوط الانتخاب هي m يكون المخرج ذو الرقم m مساوياً الواحد، أما بقية المخارج فتكون جميعها صفراً.

5. ما هي مداخل ومخارج الجامع على n بت؟

يمتلك الجامع مدخلين A و B كل منهما على n بت وله مخرجين وهما sum على n بت يعبر عن نتيجة الجمع ومخرج على بت واحد هو carry ويعبر عن بت الحمل الناتج.

6. ما هي مداخل ومخارج المقارن على n بت؟

يمتلك المقارن (Comparator) مدخلين A و B كل منهما على n بت وله ثلاثة مخارج هي less و equal و greater.

7. ما هي مداخل ومخارج القلاب من النوع JK وما هو عمله؟

لهذا القلاب ثلاثة مداخل وهي J و K و clock ومخرجين Q و Q'.

- عندما J=0 و K=0 يكون Q(next)=Q.
- عندما J=1 و K=0 يكون Q(next)=1.
- عندما J=0 و K=1 يكون Q(next)=1.
- عندما J=1 و K=1 يكون Q(next)=Q'.

8. ما هي دارة المنطق التعاقبي؟

دارة المنطق التعاقبي (Sequential Logic) هي دارة رقمية يكون خرجها في لحظة ما تابعاً لقيم الدخل في اللحظة الحالية واللحظات السابقة، أي إن دارات المنطق المتتابع تحتوي على ذاكرة.

9. ما الفرق بين آلة مور وآلة ميلي في دارات آلة الحالة المنتهية؟

- **آلة مور (Moore machine):** تعتمد قيم الخرج لحالة الآلة المنتهية من هذا النوع في لحظة ما على الحالة في هذه اللحظة.
- **آلة ميلي (Mealy machine):** تعتمد قيم الخرج لحالة الآلة المنتهية من هذا النوع في لحظة ما على الحالة وقيم الدخل في هذه اللحظة.

10. سمِّ عنصرين من العناصر التعاقبية على سوية نقل السجلات.

السجل والعداد.

أسئلة خيارات متعددة:

1. من الناحية العملية، البوابات NAND وNOR هي أكثر استخداماً من بوابات AND وOR وذلك بسبب:

A. لسهولة تنفيذها باستخدام ترانزستورات

B. لأنها أسرع في الأداء

C. لأنها تحتوي على بوابة NOT إضافية

D. لأنها أقل استهلاكاً للطاقة.

2. يعني مفكك الترميز 3×8 مايلي:

A. له 3 خطوط انتخاب و8 مداخل.

B. له 8 خطوط انتخاب و3 مخارج.

C. له 3 خطوط انتخاب و8 مخارج.

D. له 24 مدخلاً ومخرجاً.

3. ما هو خرج الجامع على 4 بت من أجل الدخل A=1010 و B=1001؟

A. Sum=0011 و carry=0.

B. Sum=0011 و carry=1.

C. Sum=1011 و carry=0.

D. Sum=1011 و carry=1.

4. ما هو خرج المقارن على 4 بت من أجل الدخل A=1010 و B=1001؟

A. يكون less=0 و equal=0 و greater=0.

B. يكون less=1 و equal=1 و greater=0.

C. يكون less=1 و equal=0 و greater=0.

D. يكون less=0 و equal=0 و greater=1.

5. ما هو العنصر الذي ينتمي لعناصر المنطق التعاقبي من هذه العناصر ؟

A. المقارن

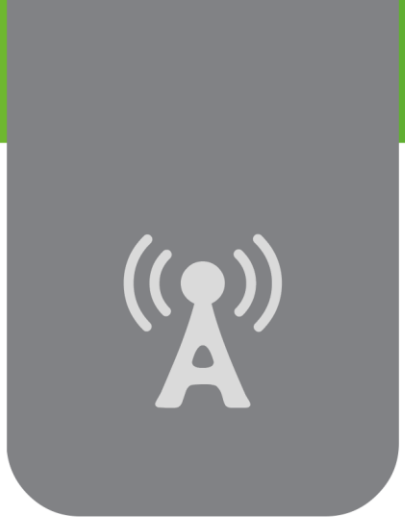
B. العداد

C. الجامع

D. المقارن

الإجابات الصحيحة:

رقم التمرين	الإجابة الصحيحة
1	A
2	C
3	B
4	D
5	B



الفصل الثالث:
المعالجات المخصصة لغرض وحيد

الكلمات المفتاحية:

تصميم المعالج المخصص لغرض وحيد، آلة الحالة المنتهية مع معطيات، مسار المعطيات، المتحكم.
Custom Single-Purpose Processor Design, Finite-State Machine with Data, Data-Path, Controller.

الملخص:

يهدف هذا الفصل إلى شرح آليات تصميم المعالجات المخصصة لغرض وحيد على سوية البوابات المنطقية. نشرح في هذا الفصل مراحل التصميم باستعمال طريقة منهجية من خلال مثال بسيط عن حساب قيمة القاسم المشترك الأكبر لعددين. حيث نتعرف على مفاهيم جديدة مثل آلة الحالة المنتهية مع معطيات (FSMD) ومسار المعطيات (datapath) والمتحكم (controller). نناقش بعد ذلك التحسينات الممكنة على التصميم ومفاهيم الجدولة (scheduling) والتخصيص (Allocation) والتكليف (binding) وترميز الحالات (state encoding). يتم ذلك على سوية البوابات ليتلمس الطالب الحجة الماسة لأدوات التصميم بمساعدة الحاسوب. كل هذا بهدف فهم أفضل للغة التوصيف العتادي الذي سنعرضه في الفصول القادمة.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- طريقة منهجية في تصميم المعالجات المخصصة لغرض وحيد
- بنية آلة الحالة المنتهية وتحويلها إلى مسار معطيات ومتحكم
- بعض طرق أمثلة التصميم والمفاهيم المتعلقة
- تلمس الحاجة لأدوات التصميم بمساعدة الحاسوب على سوية تجريدية أعلى من البوابات المنطقية

يهدف هذا الفصل إلى شرح آليات تصميم المعالجات المخصصة لغرض وحيد على سوية البوابات المنطقية. نشرح في هذا الفصل مراحل التصميم باستعمال طريقة منهجية من خلال مثال بسيط عن حساب قيمة القاسم المشترك الأكبر لعددين. حيث نتعرف على مفاهيم جديدة مثل آلة الحالة المنتهية مع معطيات (FSMD) ومسار المعطيات (datapath) والمتحكم (controller). نناقش بعد ذلك التحسينات الممكنة على التصميم ومفاهيم الجدولة (scheduling) والتخصيص (Allocation) والتكليف (binding) وترميز الحالات (state encoding). يتم ذلك على سوية البوابات ليتلمس الطالب الحجة الماسة لأدوات التصميم بمساعدة الحاسوب. كل هذا بهدف فهم أفضل للغة التوصيف العتادي الذي سنعرضه في الفصول القادمة.

1. مقدمة:

المعالج هو دارة رقمية مصممة لتنفيذ مهمات حسابية. يتكون المعالج من مسار معطيات (datapath) قادر على تخزين ومعالجة المعطيات، ومن متحكم (controller) قادر على تحريك المعطيات ضمن مسار المعطيات.

يُصمم المعالج العام (general-purpose processor) لكي يكون قادراً على تنفيذ طيف واسع من المهمات الحسابية التي تم توصيفها بواسطة مجموعة من التعليمات المتاحة للمبرمج. في المقابل، يُصمم المعالج الخاص (single-purpose processor) لتنفيذ مهمة حسابية محددة. هنالك العديد من المهمات الحسابية الشائعة الاستخدام والتي قد نجد لها معالجات خاصة لتنفيذها، وبالتالي يمكن استعمالها كدارات جاهزة في تنجيز النظام المضمَّن. من جهة ثانية، توجد مهمات حسابية فريدة وخاصة بالنظام الضمن الذي نرغب به. في هذه الحال قد يكون من الأفضل أن يقوم مصمم النظام المضمَّن بتصميم معالج خاص لتنفيذ هذه المهمة.

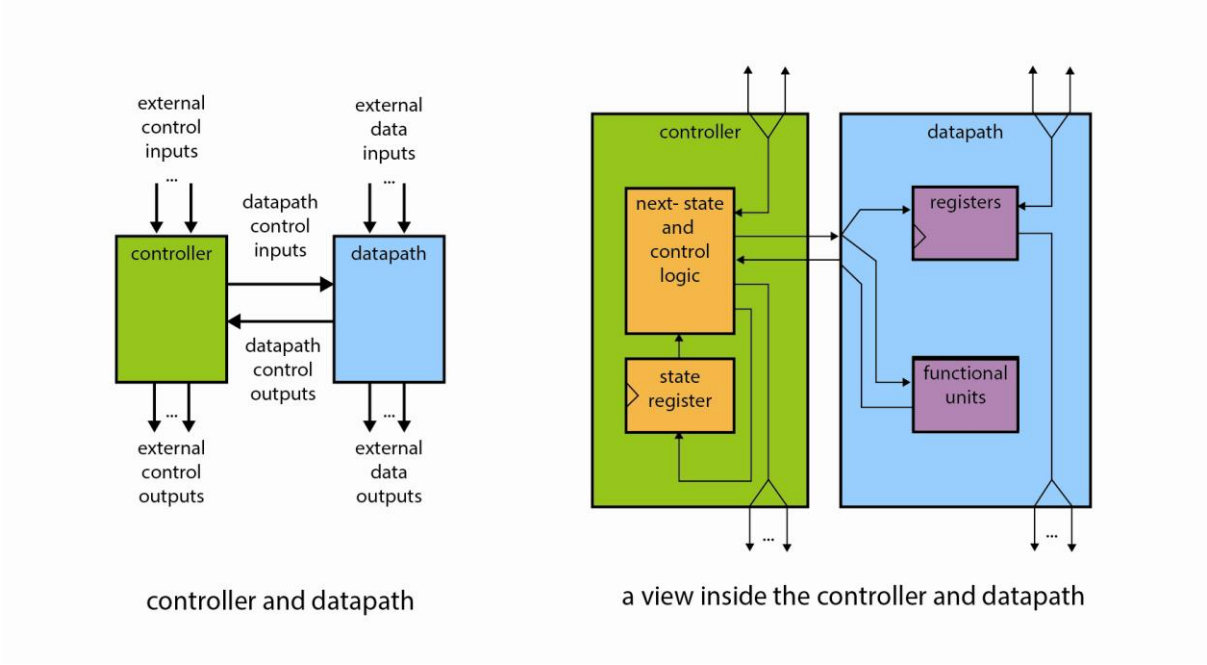
نشرح في هذا الفصل التقنيات الأساسية في تصميم المعالجات الخاصة. ثم نشرح الطرائق المستعملة في تحويل الخوارزميات والبرامج إلى معالجات خاصة.

2. تصميم المعالج المخصص لغرض وحيد:

أصبح لدينا الآن المعارف الضرورية لبناء معالج مخصص لغرض وحيد (Custom Single-Purpose Processor) أو اختصاراً المعالج الخاص أو المعالج العتادي لتمييزه عن المعالجات ذات الأغراض العامة والتي يتم تصميمها لتنفيذ برنامج يكتب من قبل المطور.

يتكون المعالج الخاص من متحكم (controller) ومسار معطيات (datapath).

يقوم مسرى المعطيات بتخزين ومعالجة معطيات النظام، ومن الأمثلة على المعطيات التي يتعامل معها النظام المضمَّن نذكر الأرقام الثنائية التي تمثل المقادير الفيزيائية مثل الحرارة والسرعة والحروف التي تُعرض على شاشة الإظهار والصور الرقمية التي يتم ضغطها وتخزينها.



بنية المعالج الخاص: المتحكم ومسار المعطيات

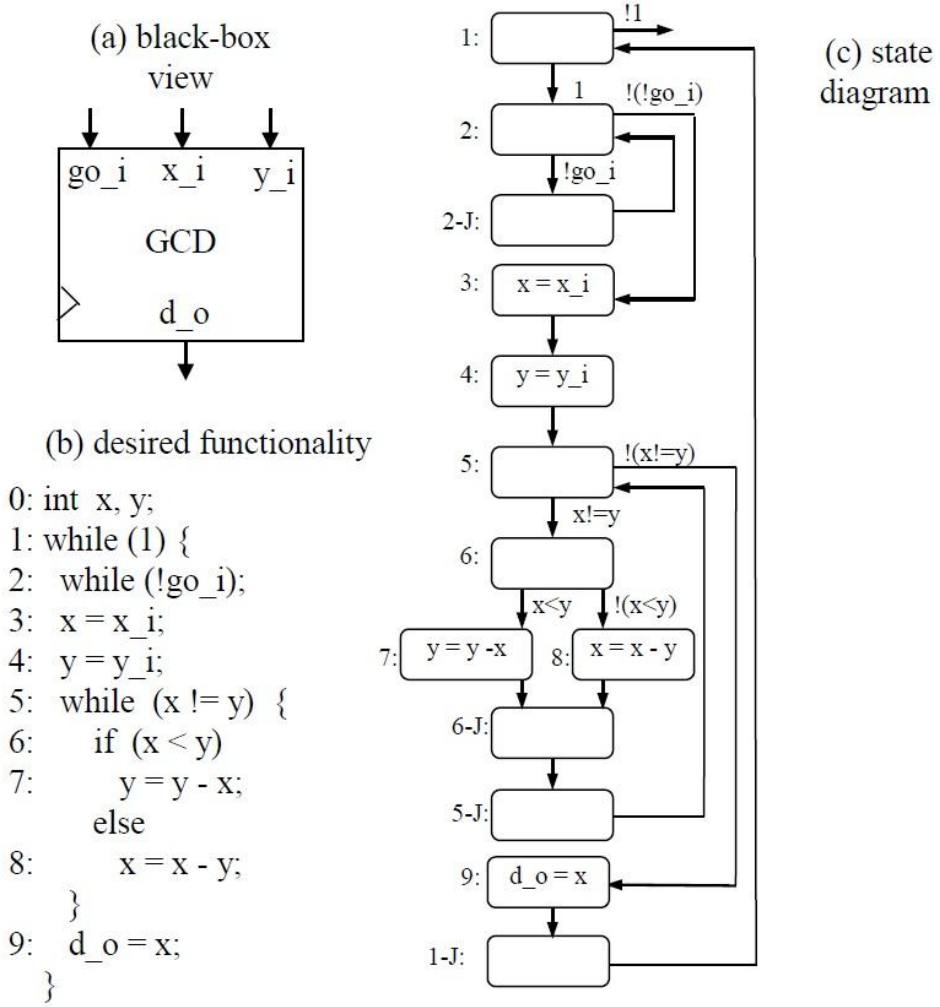
يحتوي مسار المعطيات على وحدات السجلات ووحدات عاملة (functional units) ووحدات وصل مثل الأسلاك والنواخب. يمكن تنظيم مسار المعطيات ليقرأ المعطيات من سجلات معينة وتوجيه هذه المعطيات إلى الوحدات العاملة المُعدّة لتنفيذ عمليات حسابية معينة مثل الجمع والإزاحة ومن ثم تخزين النتائج في سجلات معينة.

يتولى المتحكم عملية تنظيم عمل مسار المعطيات من خلال توليد إشارات التحكم بمسار المعطيات مثل إشارات شحن السجلات (load) وخطوط الانتخاب (select) للنواخب وذلك لضمان العمل الصحيح لمسار المعطيات في كل لحظة. يولد المتحكم أيضاً إشارات التحكم الخاصة بمعطيات الدخل الخارجية ومخارج مسار المعطيات الخارجية.

1.2. غرض المعالج: حساب القاسم المشترك الأكبر:

سوف نستخدم تقنيات تصميم الدارات التراكبية والدارات التعاقبية التي عرضناها سابقاً في تصميم المتحكم ومسار المعطيات للمعالج الخاص. سنقوم الآن بعرض طريقة لترجمة تنفيذ مهمة حسابية من خلال بناء المعالج الخاص بها. سنبدأ بعرض برنامج تتابعي يصف المهمة الحسابية التي نرغب بتنفيذها.

يبين الشكل أدناه مثالاً عن مهمة حساب القاسم المشترك الأكبر GCD (Greatest Common Divisor). يمثل الشكل مخطط الصندوق الأسود للنظام المطلوب والذي يمتلك المداخل x_i و y_i والمخرج d_o بالإضافة إلى المدخل go_i الذي يعطي أمر البدء بتنفيذ المهمة الحسابية. هذا بالإضافة إلى مدخل إضافي غير مُبيّن في الشكل وهو إشارة الساعة clock التي سيعمل عليها النظام.



نموذج الصندوق الأسود وخوارزمية العمل ومخطط الحالة الموافق

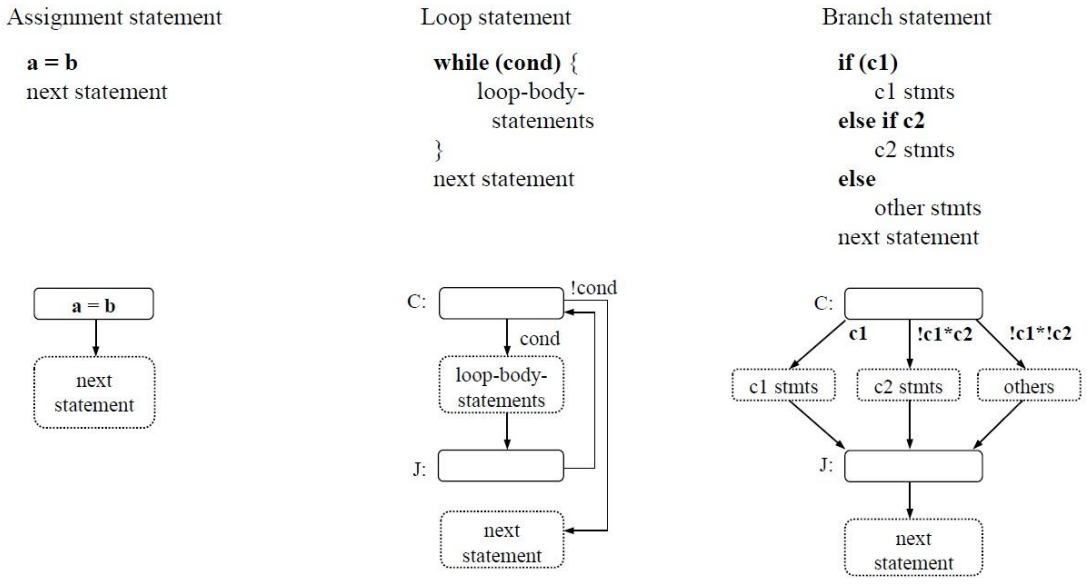
يُبيّن الشكل أيضاً برنامجاً بسيطاً يوضح الخوارزمية المستعملة في حساب القاسم المشترك الأكبر بشكل تكراري حيث يمكن تتبّع هذا البرنامج من أجل قيمة معينة لرقمي الدخل والتأكد من أن الناتج هو فعلاً القاسم المشترك لهذين العددين.

2.2. مخطط آلة الحالة المنتهية مع معطيات:

للبدء ببناء المعالج الخاص الذي سينفذ مهمة إيجاد القاسم المشترك الأكبر لعددين سنقوم أولاً بتحويل البرنامج إلى مخطط حالة مركب حيث يمكن أن تحتوي الحالات والأسهم فيه على عبارات حسابية. وقد تتضمن هذه العبارات الحسابية مداخل أو مخارج خارجية أو متحولات داخلية، وذلك بخلاف مخطط حالة الآلة المنتهية التي عرضناها سابقاً والتي تحتوي فقط على عبارات منطقية قد تتضمن مداخل ومخارج خارجية ولكن ليس متحولات داخلية.

يعتبر هذا المخطط المتطور للحالة بشكل أساسي برنامجاً تعاقبياً تمت جدولة العبارات البرمجية فيه ضمن حالات. نسمي هذا المخطط المتطور للحالة "آلة الحالة المنتهية مع معطيات" FSMD (Finite State Machine with Data).

يمكننا استعمال نماذج جاهزة لتحويل البرنامج إلى FSMD كما هو موضح في الشكل التالي، حيث نقوم في البداية بتصنيف عبارات البرنامج إلى عبارة إسناد، أو عبارة حلقة، أو عبارة تفريع من الشكل (if–then–or case else).



نماذج ترجمة البرنامج إلى مخطط حالة

عبارة الإسناد: فمن أجل عبارة الإسناد ننشئ حالة يكون الفعل (action) المنفّذ فيها هو عبارة التعويض المطلوبة. كما نضيف سهماً من هذه الحالة إلى الحالة التالية بدون شرط (وهي الحالة الأولى في العبارة التالية).

عبارة الحلقة: من أجل عبارة الحلقة، ننشئ حالة شرطية C وحالة مرتبطة J وكلاهما بدون أي فعل. نضيف سهماً مرتبطاً بتحقق العبارة الشرطية من الحالة C إلى الحالة الأولى في جسم الحلقة (loop-body) ونضيف سهماً آخر مرتبطاً بعدم تحقق العبارة الشرطية إلى أول حالة في العبارة التي تلي الحلقة.

عبارة التفريع: من أجل عبارة التفريع، نقوم بإنشاء حالة شرطية C وحالة مرتبطة J وكلاهما بدون أي فعل. نضيف سهماً مرتبطاً بشرط التفريع الأول من الحالة C إلى أول حالة في عبارة تحقق هذا الشرط ونكرر ذلك لبقية الشروط، ثم نضيف أسهماً من جميع العبارات الشرطية إلى الحالة المرتبطة J، كما نضيف سهماً من هذه الحالة المرتبطة إلى حالة العبارة التالية.

باستخدام هذه النماذج الجاهزة نحول برنامج حساب القاسم المشترك الأكبر إلى المخطط المبيّن في الشكل الذي يحتوي على البرنامج. لاحظ أن المتحولات التي تم إسنادها في بعض الحالات كفعل لهذه الحالة تحتوي أيضاً على عمليات حسابية.

تشمل الخطوة التالية في تصميم المعالج الخاص تقسيم عمل النظام إلى قسم مسار المعطيات وقسم التحكم.

3.2. تصميم مسار المعطيات:

يجب أن يحتوي مسار المعطيات على عناصر تراكيبية وعناصر تتابعية مرتبطة فيما بينها، بينما يجب أن يحتوي المتحكم على آلة حالة منتهية FSM صِرْفة أي تحتوي على أفعال منطقية وشروط من دون عمليات حسابية على معطيات.

نبني مسار المعطيات وفق أربع خطوات:

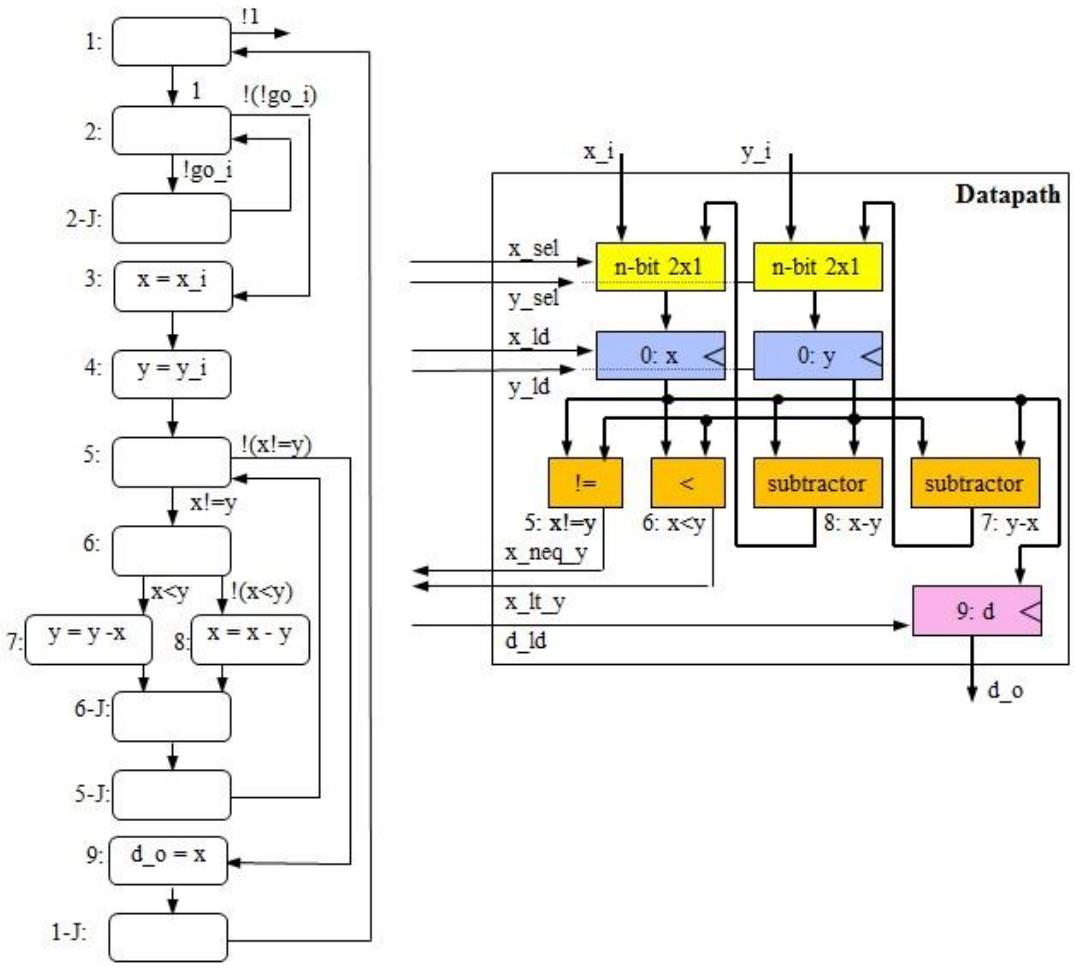
A. بدايةً، ننشئ سجلاً لكل المتحولات الداخلية وهي في مثالنا الحالي x و y. نعامل بوابة الخرج كمتحول ضمني لذلك ننشئ سجلاً d ونربطه ببوابة الخرج.

B. ثانياً، ننشئ وحدة عاملة من أجل كل عملية حسابية في مخطط الحالة. في مثالنا توجد عمليتا طرح وعملية مقارنة (<) وعملية مقارنة (عدم مساواة) مما يعطي عمليتي طرح وعمليتي مقارنة.

C. ثالثاً، نصل البوابات والسجلات والوحدات العاملة. فمن أجل كل عملية كتابة في متحول في مخطط الحالة، نرسم وصلة من مصدر الكتابة إلى سجل المتحول، حيث يمكن أن يكون مصدر الكتابة بوابة أو وحدة عاملة أو سجل آخر. ومن أجل كل عملية حسابية أو منطقية نصل المصادر إلى المداخل المناسبة للوحدة العاملة. وعندما يكون هناك أكثر من مصدر موصول إلى سجل فإننا نضيف ناخباً بالقياس المناسب.

D. أخيراً، ننشئ مُعرِّفاً (Identifier) وحيداً لكل إشارة تحكم دخل أو خرج من عناصر مسار المعطيات مثل المعرّفات x_scl و x_neq_j.

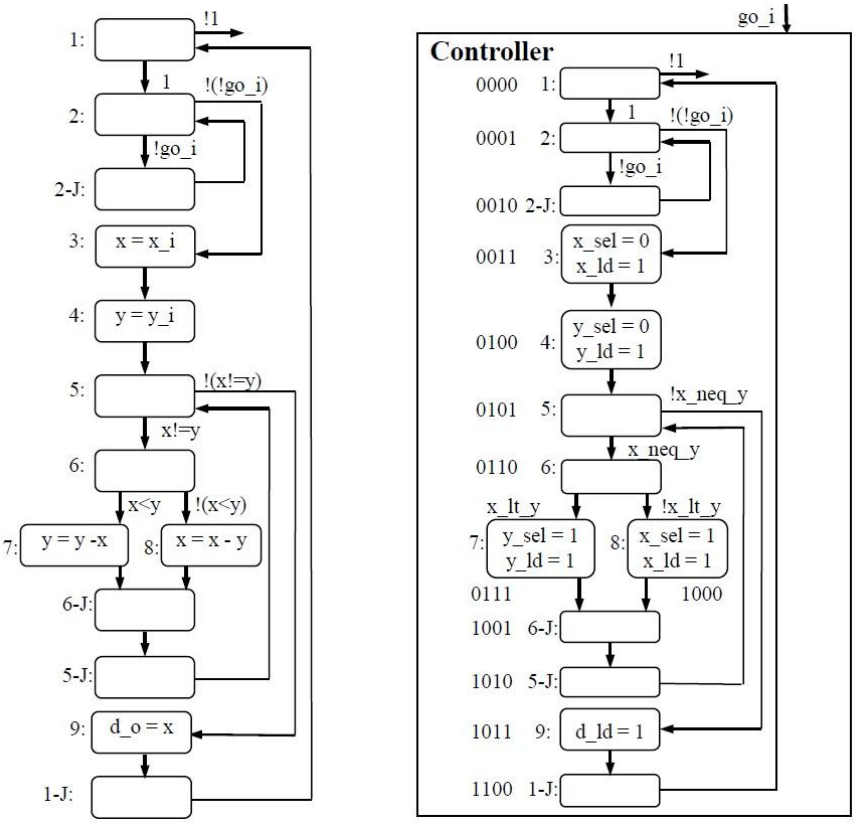
نحصل بنتيجة تطبيق هذه الخطوات على الشكل التالي لمسار المعطيات.



بنية مسار المعطيات

4.2. تصميم المتحكم:

الآن وبعد اكتمال مسار المعطيات نستطيع تحويل آلة الحالة المنتهية مع معطيات FSMD إلى آلة حالة منتهية FSM والتي تمثل المتحكم المبين في الشكل التالي. ولآلة الحالة المنتهية هذه نفس حالات وانتقالات آلة الحالة المنتهية مع معطيات ولكن نبدل الأفعال المركبة والشروط بأخرى منطقية باستخدام إشارات مسار المعطيات، حيث نبدل عمليات الكتابة في المتحولات بوضع إشارات الانتخاب للنواخب المرتبطة بسجل المتحول بحيث يمرر الناخب المصدر الصحيح وكذلك نولد إشارة الشحن في السجل load. كما نبدل كل عملية منطقية شرطية بخرج التحكم من الوحدة العاملة الموافقة التي تنفذ عملية المقارنة.



آلة الحالة المنتهية للمتحكم

نفترض في مخطط آلة الحالة أن كل إشارة لم يتم إسنادها بشكل صريح في أية حالة فإنه يتم إسناد قيمة 0 لهذه الإشارة ضمنياً. على سبيل المثال، يتم إسناد 0 إلى الإشارة x_ld في كل الحالات ما عدا في الحالات 3 و8 حيث يتم إسناد 1 لهذه الإشارة.

نستطيع بعد ذلك إتمام تصميم المتحكم بتنجيز آلة الحالة FSM باستخدام تقنيات المنطق المتعاقب كما رأينا سابقاً. نقوم ببناء جدول الحقيقة لآلة الحالة حيث لدينا هنا 7 إشارات دخل و9 إشارات خرج، وذلك يعني أن جدول الحقيقة يجب أن يحتوي على 128 سطراً ولكننا اختصرنا بعض التراكيب الممكنة للإشارات باستخدام * لبعض المداخل وذلك لأن قيمتها الفعلية لا تهم في بناء إشارات الخرج كما هو موضح في الجدول التالي. ولكن مع ذلك نرى أن عملية إنشاء المعادلات المنطقية للخرج تبقى عملية طويلة ومعقدة، ولهذا السبب يتم استخدام أدوات التصميم بمساعدة الحاسوب CAD (Computer Aided Design) التي تقوم بأتمتة عملية تصميم دارات المنطق المتراكب والمتعاقب والتي تولد دارات البوابات المنطقية انطلاقاً من البرامج التعاقبية. تسمى هذه الأدوات أدوات التركيب (Synthesis Tools).

Inputs							Outputs								
Q3	Q2	Q1	Q0	x_neq_y	x_lt_y	go_i	I3	I2	I1	I0	x_sel	y_sel	x_ld	y_ld	d_ld
0	0	0	0	*	*	*	0	0	0	1	X	X	0	0	0
0	0	0	1	*	*	0	0	0	1	0	X	X	0	0	0
0	0	0	1	*	*	1	0	0	1	1	X	X	0	0	0
0	0	1	0	*	*	*	0	0	0	1	X	X	0	0	0
0	0	1	1	*	*	*	0	1	0	0	0	X	1	0	0
0	1	0	0	*	*	*	0	1	0	1	X	0	0	1	0
0	1	0	1	0	*	*	1	0	1	1	X	X	0	0	0
0	1	0	1	1	*	*	0	1	1	0	X	X	0	0	0
0	1	1	0	*	0	*	1	0	0	0	X	X	0	0	0
0	1	1	0	*	1	*	0	1	1	1	X	X	0	0	0
0	1	1	1	*	*	*	1	0	0	1	X	1	0	1	0
1	0	0	0	*	*	*	1	0	0	1	1	X	1	0	0
1	0	0	1	*	*	*	1	0	1	0	X	X	0	0	0
1	0	1	0	*	*	*	0	1	0	1	X	X	0	0	0
1	0	1	1	*	*	*	1	1	0	0	X	X	0	0	1
1	1	0	0	*	*	*	0	0	0	0	X	X	0	0	0
1	1	0	1	*	*	*	0	0	0	0	X	X	0	0	0
1	1	1	0	*	*	*	0	0	0	0	X	X	0	0	0
1	1	1	1	*	*	*	0	0	0	0	X	X	0	0	0

جدول الحقيقة لآلة الحالة المنتهية للمتحكم

من الجدير بالذكر هنا أننا نستطيع أمثَلة التصميم سواء في بنية مسار المعطيات أو في بنية المتحكم، فعلى سبيل المثال، يمكننا دمج الوحدات العاملة في مسار المعطيات فبدلاً من استخدام وحدتي طرح ووحدتي مقارنة نستطيع استخدام وحدة طرح واحدة ووحدة مقارنة واحدة، ولكن يجب اختيار معاملات الدخل المناسبة عند إجراء كل عملية. نستطيع أيضاً حذف بعض الحالات غير الضرورية في المتحكم، الأمر الذي يؤدي إلى تصغير حجمه وتسريع التنفيذ نظراً لتوفير الزمن اللازم لهذه الحالات غير الضرورية.

وأخيراً فمن الضروري التنويه إلى مسألة هامة تتعلق بتزامن الأفعال ضمن آلة الحالة المنتهية مع معطيات FSMD، حيث تُعتبر جميع الأفعال ضمن نفس الحالة أفعالاً متزامنة، أي إنها تحدث في نفس الوقت وليس بصورة متعاقبة كما في البرامج التعاقبية. على سبيل المثال، إذا كانت قيمة المتحول x=0 قبل الدخول إلى الحالة A، وكانت الأفعال التي يتم تنفيذها ضمن هذه الحالة هي x=x+1 و y=x فإن قيمة y هي 0 بعد تنفيذ هذه الحالة وليس 1 وذلك لأن العبارتين يتم تنفيذهما في نفس الوقت حيث يكون x=0، وبالتالي لا يؤثر ترتيب الأفعال ضمن الحالة نفسها. إضافة إلى ذلك، فإن المتحولات التي يتم تغييرها ضمن الحالة لا تأخذ قيمها الجديدة إلا عند نبضة الساعة التي تلي الدخول في الحالة، وذلك لأن المتحولات عبارة عن سجلات تتم الكتابة فيها عند جبهات الساعة. وكذلك فإن الأسهم الخارجة من الحالة والتي تستخدم المتحول x كشرط للانتقال فإن هذا الشرط سيستخدم القيمة القديمة للمتحول أي 0 في مثالنا السابق وليس القيمة الجديدة، وهذا من الأخطاء الشائعة في تصميم آلات الحالة، فإذا أردنا أن يكون التفريع باستخدام القيمة الجديدة للمتحول علينا إضافة حالة إضافية ومنها نفرع باستخدام القيمة الجديدة.

3. أمثَلَة تصميم المعالج الخاص:

عندما قمنا بتصميم مثال المعالج الخاص لحساب القاسم المشترك الأكبر تجاهلنا الكثير من عمليات الأَمثَلة لتحسين مقاييس التصميم، وكان ذلك مقصوداً للحفاظ على المنهجية العامة في تصميم المعالج. لا نريد هنا الخوض كثيراً في تقنيات أمثلة التصميم بل نريد إعطاء فكرة عامة عن بعض عمليات الأَمثَلة البسيطة التي يمكن أن نجريها على التصميم الأولي بهدف تحسين مقاييس التصميم من حيث الحجم والسرعة على عدة مستويات.

1.3. أمثَلَة البرنامج الأصلي:

سنقوم أولاً بتحسين البرنامج الأصلي الذي استندنا إليه في تصميمنا وذلك من خلال تحليل عدد العمليات الحسابية وحجم المتحولات المطلوب في الخوارزمية، وبعبارة أخرى تحليل التعقيد الزمني والتعقيد التخزيني للخوارزمية. يمكننا البحث عن خوارزمية بديلة لحساب القاسم المشترك الأكبر. فإذا فرضنا أننا نستطيع استخدام عملية باقي القسمة بين عددين % (modulo) فإننا نستطيع أن نعيد كتابة الخوارزمية بالشكل التالي:

original program

```
0: int  x, y;
1: while (1) {
2:   while (!go_i);
3:   x = x_i;
4:   y = y_i;
5:   while  (x != y)  {
6:     if  (x < y)
7:       y = y - x;
       else
8:       x = x - y;
     }
9:   d_o = x;
   }
```

replace the subtraction operation(s) with modulo operation in order to speed up program

optimized program

```
0: int  x, y, r;
1: while (1) {
2:   while (!go_i);
     // x must be the larger number
3:   if (x_i >= y_i)  {
4:     x=x_i;
5:     y=y_i;
     }
6:   else {
7:     x=y_i;
8:     y=x_i;
     }
9:   while  (y != 0)  {
10:    r = x % y;
11:    x = y;
12:    y = r;
     }
13:  d_o = x;
   }
```

GCD(42, 8) - 9 iterations to complete the loop

x and y values evaluated as follows : (42, 8), (43, 8), (26,8), (18,8), (10, 8), (2,8), (2,6), (2,4), (2,2).

GCD(42,8) - 3 iterations to complete the loop

x and y values evaluated as follows: (42, 8), (8,2), (2,0)

أمثلة البرنامج الأصلي للخوارزمية

لنقارن بين أداء هذه الخوارزمية الجديدة وتلك القديمة من حيث عدد الخطوات اللازمة لحساب القاسم المشترك الأكبر من أجل قيم الدخل 42 و 8.

سيتم تكرار العمليات الحسابية في الحلقة الداخلية للخوارزمية القديمة من أجل القيم التالية للمتحولات (x,y): (42,8), (34,8), (26,8), (18,8), (10,8), (2,8), (2,6), (2,4), (2,2) ويكون الناتج هو 2.

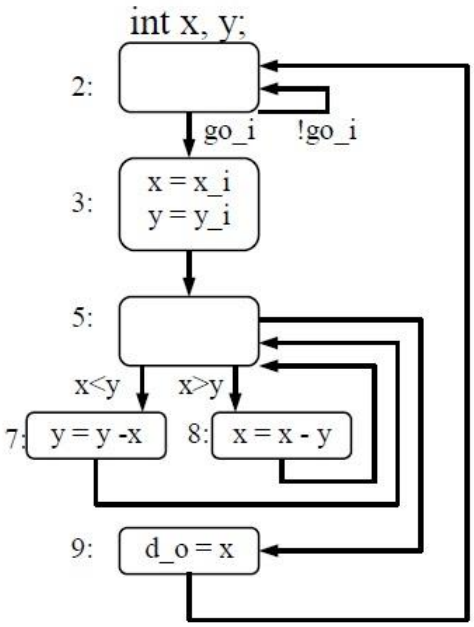
أما في الخوارزمية الجديدة فسيتم تكرار العمليات الحسابية في الحلقة الداخلية من أجل القيم التالية للمتحولات (x,y): (42,8), (8,2), (2,0) ويكون الناتج هو 2 أيضاً.
وبالتالي فإن الخوارزمية الجديدة أكثر كفاءة من حيث زمن التنفيذ، لهذا فإن للاختيار الصحيح لخوارزمية البرنامج الأثر الأكبر على فعالية المعالج المصمم.

2.3. أمثَلَة آلة الحالة المنتهية مع معطيات:

بعد الاستقرار على خوارزمية معينة، نستطيع تحويل البرنامج الذي يوصّف تلك الخوارزمية إلى FSMD. تعطي الطريقة التي استخدمناها في تحويل البرنامج بالاعتماد على النماذج الجاهزة لتحويل العبارات البرمجية الكثير من الحالات غير الضرورية التي يمكن اختصارها أو دمجها مع حالات أخرى.
الجدولة (scheduling) هي عملية إسناد العمليات في البرنامج إلى حالات في آلة الحالة FSMD. يمكننا تحسين الجدولة التي قمنا بها سابقاً بالاستناد إلى الملاحظات التالية:

- يمكننا حذف الحالة رقم 1 لأن شرط التفريع محقق دائماً ويؤدي إلى الحالة رقم 2 دائماً.
- يمكن دمج الحالة J–2 مع الحالة 2 لأن الحالة J–2 لا تقوم بأي عمل وتعود دائماً إلى الحالة 2.
- يمكن دمج الحالتين 3 و4 في حالة واحدة لأن العمليات الحسابية التي نجريها في كل حالة مستقلة عن بعضها البعض ويمكن إجراؤها معاً في حالة واحدة.
- يمكن حذف الحالة 6 والتفريع مباشرة من الحالة 5 إلى الحالتين 7 و8.
- يمكن حذف الحالتين j–6 وj–5 لأنهما تقومان بالوصل المباشر إلى الحالة 5 انطلاقاً من الحالتين 7 و8، وبالمثل يمكن حذف الحالة j–1.

بتطبيق هذه الملاحظات نحصل على مخطط آلة الحالة FSMD المبين في الشكل التالي الذي يحتوي على 6 حالات فقط بدلاً من 13 حالة في مخطط الحالة الأصلي.



آلة الحالة FSMD المحسنة

تم تحسين الجدولة في هذا المثال بهدف تقليص عدد الحالات وبالتالي تقليص الزمن اللازم للحصول على النتيجة، إلا أنه في بعض التطبيقات قد يكون زمن ظهور النتيجة محدد سلفاً وبالتالي قد نضطر أحياناً لإضافة حالات بدلاً من إنقاصها لضمان ظهور النتائج في الوقت المحدد.

في مثالنا السابق عن كاشف السلسلة "010" كان تصرّف آلة الحالة محكوماً زمنياً بشكل دقيق وبالتالي لا يمكن إضافة أو حذف حالات في مخطط آلة الحالة، وبالتالي فعلى التصميم أن يأخذ بعين الاعتبار مسألة تزامن إشارات الخرج.

3.3. أمثلة مسار المعطيات:

في طريقتنا التي عرضناها سابقاً في تصميم مسار المعطيات من خلال أربع خطوات، تم إنشاء وحدة عاملة لكل عملية حسابية في مخطط FSMD، ولكن غالباً ما تكون المقابلة واحد لواحد غير ضرورية ونرغب بتقليص عدد الوحدات العاملة لتخفيض حجم النظام. يمكن أن تتشارك عدة عمليات حسابية في نفس الوحدة العاملة إذا كانت هذه العمليات الحسابية ضمن حالتين مختلفتين.

في مثال حساب القاسم المشترك الأكبر تُنفذ كلا الحالتين 7 و8 عملية طرح، وبالتالي يمكننا هنا استخدام طارح وحيد مع إضافة ناخب على دخل الطارح لاختيار ما إذا كنا نريد تنفيذ x–y أو y–x.

إضافة إلى ذلك لدينا العديد من العناصر على مستوى السجلات RT التي يمكن استعمالها في بناء مسار المعطيات مثل وحدات الحساب والمنطق ALU القادرة على تنفيذ عمليات مختلفة.

نعرّف مفهوم التخصيص (Allocation) بأنه عملية اختيار العناصر على مستوى السجلات RT في بناء مسار المعطيات. كما نعرّف مفهوم التكليف (Binding) بأنه عملية ربط العمليات الحسابية في آلة الحالة FSMD مع العناصر المخصصة.

إن عمليات الجدولة والتخصيص والتكليف مرتبطة فيما بينها حيث إن طريقة جدولة ما تُحدد خيارات إمكانيات التخصيص الممكنة وبالعكس. لذلك نقوم أحياناً بدراسة هذه المهام معاً لاختيار الأنسب.

4.3. أمثَلَة آلة الحالة المنتهية:

عند تصميم دارة تتابعية لتنجيز آلة الحالة FSM يمكن إجراء بعض عمليات الأمثَلَة التي تتعلق بعملية ترميز الحالات (أي ترقيمها) وتقليص عدد الحالات.

ترميز الحالات (state encoding) هي عملية إسناد رقم معين (بالصيغة الثنائية مثلاً) ووحيد لكل حالة في آلة الحالة FSM.

سيعمل أي ترقيم وحيد بشكل صحيح ولكن طول سجل الحالة وحجم دارات المنطق المتراكب لحساب الحالة التالية سيختلف باختلاف طريقة الترميز. على سبيل المثال يمكن ترقيم أربع حالات A وB وC وD بالأرقام 00، 01، 10، 11 على الترتيب. كما يمكن ترقيمها بالشكل 11، 10، 00، 01 على الترتيب. في الواقع إذا كان عدد الحالات في آلة حالة منتهية هو n (حيث n من قوى العدد 2 أي $n=2^q$) فهناك $n!$ طريقة مختلفة لعملية ترقيم هذه الحالات.

عندما تكون قيمة n كبيرة، تُعتبر القيمة $n!$ كبيرة جداً لكي يتم تفحّص جميع احتمالات الترقيم لاختيار الأفضل، وعند اختيار عدد من البتات أكثر من $\log_2(n)$ لتمثيل الحالات سيزداد عدد إمكانيات الترقيم الممكنة. لذا فإن استخدام تقنيات التصميم بمساعدة الحاسوب CAD ضرورية لإجراء هذا النوع من الأمثَلَة.

نُعرّف مفهوم تقليص الحالات (state minimization) بأنه عملية دمج الحالات المتكافئة في حالة وحيدة. نعتبر أن حالتين متكافئتين إذا كانتا تعطيان نفس المخارج ونفس الانتقال إلى الحالة التالية من أجل جميع قيم الدخل الممكنة. إن مفهوم عملية دمج الحالات التي قمنا بها عند أمثَلَة آلة الحالة FSMD سابقاً يختلف عن مفهوم تقليص الحالات المُعرّف هنا، وذلك لأن الدمج الذي قمنا به يغير من تزامن الخرج، أي إن مفهوم تقليص الحالات المقصود هنا هو عملية الدمج التي لا تؤدي إلى أي تغيير في قيم الخرج أو تسلسلها الزمني.

4. خلاصة:

قدمنا في هذا الفصل طريقة منهجية في تصميم المعالجات المخصصة لغرض وحيد. حيث يشمل هذا الغرض تنفيذ خوارزمية معينة. نبدأ عادةً هذا التصميم باستعمال آلة الحالة مع معطيات. ثم نفصل مسار المعطيات عن المتحكم. نحتاج في الغالب على أدوات التصميم بمساعدة الحاسب لإكمال التصميم وأمثلته. الأمر الذي أدى إلى ظهور لغات التوصيف العتادي التي تعمل على سوية تجريدية أعلى من سوية البوابات وتقوم بعمليات التصميم الروتينية بشكل آلي كما سنرى في الفصول القادمة.

5. أسئلة الفصل الثالث:

أسئلة عامة:

1. ما هي وظيفة مسار المعطيات في المعالج الخاص؟

يقوم مسرى المعطيات بتخزين ومعالجة معطيات النظام

2. ما هي وظيفة المتحكم في المعالج الخاص؟

يتولى المتحكم عملية تنظيم عمل مسار المعطيات من خلال توليد إشارات التحكم بمسار المعطيات وذلك لضمان العمل الصحيح لمسار المعطيات في كل لحظة.

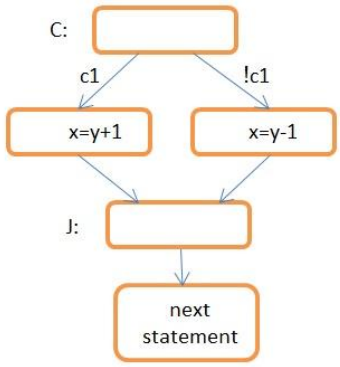
3. ما هو مخطط آلة الحالة المنتهية مع معطيات FSMD؟

مخطط حالة مركب حيث يمكن أن تحتوي الحالات والأسهم فيه على عبارات حسابية. وقد تتضمن هذه العبارات الحسابية مداخل أو مخارج خارجية أو متحولات داخلية

4. ارسم مخطط الحالة الموافق لعبارة التفريع التالية:

```
if(c1)
    x=y+1
else
    x=y-1
```

نقوم بإنشاء حالة شرطية C وحالة مرتبطة J وكلاهما بدون أي فعل. نضيف سهماً مرتبطاً بشرط التفريع c1 من الحالة C إلى الحالة x=y+1 وسهماً مرتبطاً بشرط التفريع !c1 من الحالة C إلى الحالة x=y−1، ثم نضيف أسهماً من جميع العبارات الشرطية إلى الحالة المرتبطة J، كما نضيف سهماً من هذه الحالة المرتبطة J إلى حالة العبارة التالية.



5. كيف يتم وصل مداخل الوحدات العاملة مع المصادر عندما يكون هناك أكثر من مصدر لنفس المعامل؟
عندما يكون هناك أكثر من مصدر موصول إلى سجل فإننا نضيف ناخباً بالقياس المناسب.

6. كيف يتم تصميم المتحكم انطلاقاً من آلة الحالة المنتهية؟
يتم ذلك على خطوتين وهما:

- انشاء آلة حالة منتهية بنفس حالات وانتقالات آلة الحالة المنتهية مع معطيات.
- نبدل الأفعال المركبة والشروط بأخرى منطقية باستخدام إشارات مسار المعطيات.

7. لماذا نلجأ إلى استخدام أدوات التصميم بمساعدة الحاسوب CAD في تصميم المعالجات الخاصة؟
لأن عملية إنشاء المعادلات المنطقية لمخارج آلات الحالة المنتهية هي عملية طويلة ومعقدة عندما يكون عدد المداخل كبيراً.

أسئلة خيارات متعددة:

1. عند تصميم مسار المعطيات. ننشئ سجلاً من أجل
A. بوابة الدخل.
B. تخزين اشارات التحكم.
C. كل حالة في مخطط الحالة.
D. المتحولات الداخلية.

2. عند تصميم مسار المعطيات. ننشئ وحدة عاملة من أجل
A. كل عملية حسابية
B. لمعالجة كل إشارة دخل
C. لتوليد كل إشارة خرج
D. عمليات الجمع والطرح فقط.

3. ما هو الحل الأنسب لتقليص حجم الدارة عند تصميم مسار المعطيات

A. استخدام وحدات عاملة مختصة لكل عملية.

B. دمج كل وحدتين عاملتين في وحدة واحدة مع نواخب

C. تنفيذ العمليات الحسابية على التوازي.

D. استخدام وحدة حساب ومنطق واحدة لتنفيذ العمليات الحسابية.

4. ماذا يعني مفهوم الجدولة في عملية انشاء آلة حالة منتهية مع معطيات؟

A. هي عملية ترتيب تنفيذ العمليات الحسابية عبر الزمن.

B. هي عملية انشاء جداول الحقيقة لآلة الحالة المنتهية مع معطيات.

C. هي عملية إسناد العمليات في البرنامج إلى حالات في آلة الحالة FSMD.

D. هي عملية ترقيم الحالات بأرقام فريدة تسهل تصميم دارات إشارات الخرج.

5. ماذا يعني مفهوم التخصيص في عملية انشاء آلة حالة منتهية مع معطيات؟

A. عملية اختيار العناصر على مستوى السجلات RT في بناء مسار المعطيات.

B. هي عملية إسناد العمليات في البرنامج إلى حالات في آلة الحالة FSMD.

C. هي تخصيص سجلات لكل متحولات البرنامج.

D. هي عملية تخصيص وحدة حساب ومنطق ALU لإجراء العمليات الحسابية.

الإجابات الصحيحة:

رقم التمرين	الإجابة الصحيحة
1	D
2	A
3	D
4	C
5	A

الفصل الرابع

لغة التوصيف العتادي Verilog

الكلمات المفتاحية:

لغة التوصيف العتادي ، لغة Verilog، أنماط المعطيات، الوحدة، الوحدات الأولية، وحدة الاختبار.

Hardware Description Language, Verilog Language, Data types, Module, Primitive Module, Testbench.

الملخص:

يهدف هذا الفصل إلى شرح أساسيات لغة التوصيف العتادي Verilog حيث نقتصر في هذا الفصل على توصيف الدارات التراكبية على سوية البوابات فقط. سنعتمد في هذا الشرح على مثال بسيط لتوضيح المفاهيم المختلفة لهذه اللغة بشكل تدريجي. سيتعرف الطالب إلى مصطلحات هذه اللغة وأنماط المعطيات المستعملة. ثم سنشرح بنية الوحدة التي تشكل اللبنة الأساسية في بناء الدارات الرقمية في هذه اللغة. ثم سنعرض طرق استعمال الوحدات بهدف التوصيف البنوي للدارات. ونختم هذا الفصل بشرح بسيط لطرق اختبار ومحاكاة الوحدات باستعمال برامج الاختبار.

الأهداف التعليمية:

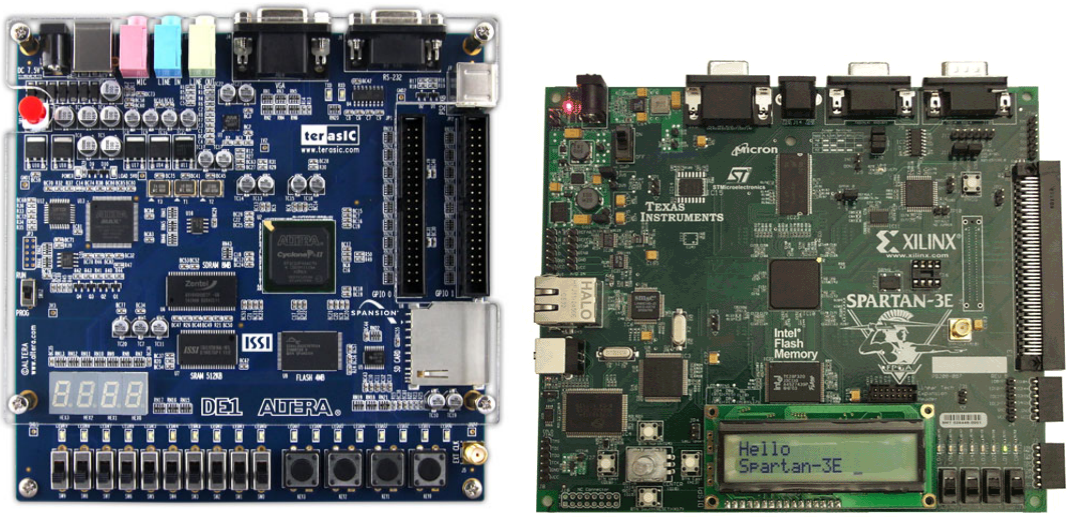
يتعرف الطالب في هذا الفصل على:

- السويات التجريدية للتوصيف العتادي.
- المصطلحات المعجمية في لغة Verilog وأنماط المعطيات.
- مكونات الوحدة الأساسية في هذه اللغة وطرقة استدعائها.
- بناء الوحدة الاختبارية في هذه اللغة.

يهدف هذا الفصل إلى شرح أساسيات لغة التوصيف العتادي Verilog حيث نقتصر في هذا الفصل على توصيف الدارات التراكبية على سوية البوابات فقط. سنعتمد في هذا الشرح على مثال بسيط لتوضيح المفاهيم المختلفة لهذه اللغة بشكل تدريجي. سيتعرف الطالب إلى مصطلحات هذه اللغة وأنماط المعطيات المستعملة. ثم سنشرح بنية الوحدة التي تشكل اللبنة الأساسية في بناء الدارات الرقمية في هذه اللغة. ثم سنعرض طرق استعمال الوحدات بهدف التوصيف البنوي للدارات. ونختم هذا الفصل بشرح بسيط لطرق اختبار ومحاكاة الوحدات باستعمال برامج الاختبار.

١. مقدمة

تسمح لغة التوصيف العتادي HDL (Hardware Description Langauges) مع تقنيات الدارات القابلة للبرمجة FPGA (Field Programmable Gates Array) بالتطوير السريع للدارات الرقمية ومحاكاتها وتنفيذها واختبارها بشكل فيزيائي. حيث أصبح بالإمكان تصميم وتنفيذ النماذج الأولية لنظم رقمية معقدة باستعمال حاسب شخصي وبطاقة تطوير FPGA غير مكلفة.



بطاقات تطوير FPGA

يوجد لغتين شهيرتين للتوصيف العتادي وهما لغة VHDL و لغة Verilog. حيث تشبه لغة VHDL في صياغتها لغة باسكال PASCAL، أما لغة Verilog فهي شبيهة للغة C. ولكن يجب الإشارة هنا أن لغات التوصيف العتادي ليست لغات برمجة، وإنما طريقة عالية المستوى لتوصيف بنية العتاد الصلب في دارة رقمية. وهي مقابلة لطريقة توصيف الدارات الرقمية باستعمال المخططات المنطقية بشكل رسومي.

في لغات البرمجة للمعالجات مثل C أو باسكال، تنفذ التعليمات بشكل متعاقب، أي تعليمة تلو الأخرى. ولذلك يكون لترتيب التعليمات أهمية كبيرة على عمل البرنامج والنتائج التي نحصل عليها. بينما في لغات التوصيف العتادي مثل VHDL و Verilog فإن التعليمات تعبر عن دارات تعمل على التوازي. وبالتالي يمكن أن نغير ترتيب العبارات دون أن يغير ذلك من عمل الدارة. ولذلك نطلق على عبارات التوصيف في هذه اللغات بالعبارات المتنافسة أو المتزامنة (concurrent statements).

تُعد لغة Verilog أبسط وأسهل للتعلم من لغة VHDL. ولكن تعتبر لغة VHDL متماسكة أكثر وقادرة على كشف أخطاء التصميم بشكل أفضل من لغة Verilog.

٢. لمحة تاريخية عن لغة Verilog

بدأت لغة Verilog في عام 1984 تقريباً كوسيلة لنمذجة الدارات الرقمية وهي خاصة بشركة Gateway Design Automation Inc. قامت هذه الشركة بتسويق هذه اللغة مع برامج المحاكاة المرتبطة بها حتى عام 1990 حيث قامت شركة Cadence Design System بالاستحواذ على هذه الشركة واستمرت بتسويق منتجاتها. وهنا فكرت شركة Cadence في جعل هذه اللغة عامة الاستعمال من قبل الجميع وعدم بقائها مغلقة ضمن منتجاتها وذلك لتجنب توجه الآخرين إلى لغة VHDL التي كانت معيارية منذ عام 1987 من قبل منظمة IEEE وموصفة بالمعيار ذو الرقم IEEE 1076. أدى فتح لغة Verilog للاستعمال العام إلى جعل شركات عديدة تسوق منتجات التصميم والمحاكاة باستعمال Verilog مع إدخال تغييرات على هذه اللغة وفقاً لمصالح كل شركة. لذلك تم توجيه الجهود لتوحيد هذه اللغة ضمن معيار واحد في عام1993 من قبل منظمة IEEE واصدار المعيار IEEE 1364. وبعد عدة سنوات تم تطوير هذا المعيار وحل معظم مشكلاته واصدار نسخة معدلة عنه في عام 2001 تحت اسم IEEE 1364-2000. أما المعيار الأصلي فيشار إليه بالاسم IEEE 1364-1993.

٣. سويات التصميم

يمكن تصميم الدارات الرقمية باستعمال منهجية التصميم من الأعلى للأسفل (top-down) أو بالعكس من الأسفل للأعلى (bottom-up).

تسمح لغات التوصيف العتادي بتصميم الدارات الرقمية على عدة سويات من التجريد. ومن أهم هذه السويات والأكثرها انتشاراً:

- السوية السلوكية (Behavioral Level)
- سوية نقل السجلات RTL (Register Transfer Level)
- سوية البوابات (Gate Level)

سوف نقتصر في هذا الفصل على تصميم الدارات التراكبية على سوية البوابات ومن ثم نتعرف على تصميم الدارات التعاقبية وباقي السويات في الفصول القادمة.

١.٣. السوية السلوكية

تصف هذه السوية النظام باستعمال خوارزميات متزامنة. وقد تتكون الخوارزمية من مجموعة من العمليات المتعاقبة مثل العدادات أو آلة الحالة. ومن الكتل الأساسية المستعملة في هذه السوية هي كتل التوابع (Functions) والمهام (Tasks) وتعليمة "دائماً" (Always).

٢.٣. سوية نقل السجلات

في هذه السوية يتم توصيف النظام من خلال تحديد خواص عمل النظام من حيث نقل المعطيات بين السجلات ولذلك يتم استعمال ساعة عمل بشكل صريح. يمكن في هذه السوية تحديد التزامن الدقيق للإشارات بالإضافة إلى تحديد لحظة تنفيذ أي عملية.

٣.٣. سوية البوابات

في هذه السوية يتم توصيف عمل الدارة باستعمال روابط منطقية مع تحديد خواصها الزمنية. أي يتم توصيف الدارة الرقمية باستعمال بوابات منطقية أساسية مثل AND و OR و NOT وغيرها. تأخذ الإشارات في هذه السوية قيماً منطقية محددة وهي (‘0’, ‘1’, ‘X’, ‘Z’) حيث يستعمل الرمز ‘Z’ للدلالة على حالة إشارة عائمة (tristate) أي مفصولة عن أي مصدر، ويستعمل الرمز ‘X’ للدلالة على عدم تعيين قيمة الإشارة (أي قيمة ‘0’ أو ‘1’ أو ‘Z’). تستعمل هذه السوية لتوصيف الدارات البسيطة فقط. إذا أن العمل ضمن هذه السوية لتوصيف النظم المعقدة سيكون مضنياً جداً. لذلك تستعمل أدوات التصميم بمساعدة الحاسوب لتوليد الدارات آلياً ضمن هذه السوية انطلاقاً من السويات التجريدية الأعلى.

٤. لغة التوصيف العتادي Verilog

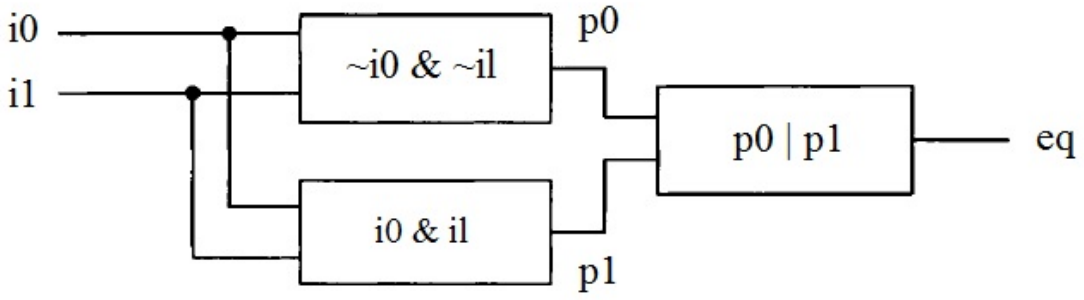
سوف نأخذ مثالاً بسيطاً لشرح مفردات لغة التوصيف العتادي Verilog. لنعتبر مقارن مساواة على بت واحد. لهذا المقارن مدخلين i0 و i1 ومخرج واحد eq. يأخذ المخرج قيمة ‘1’ عندما يكون هناك مساواة بين i0 و i1. يبين الجدول التالي جدول الحقيقة لهذا المقارن:

inputs		output
i_0	i_1	eq
0	0	1
0	1	0
1	0	0
1	1	1

يمكن كتابة المعادلة المنطقية لهذا المقارن بالشكل:

$$eq = i_0.i_1 + i_0'.i_1'$$

وانطلاقاً من هذه المعادلة، يمكن التعبير عن المقارن بشكل بياني كالتالي:



يتم توصيف الدارات الرقمية بلغة التوصيف Verilog ضمن وحدات تسمى module. يبين الترميز التالي وصفاً للمقارن باستعمال وحدة تسمى eq1.

```
module eql
// I/O ports
(
input wire i0, il,
output wire eq
) ;
// signal declaration
wire p0 pl;

// body
// sum of  two product terms
assign eq = p0 | pl;
// product  terms
assign p0 = ~i0 & ~il;
assign pl = i0 & il;
endmodule
```

تتكون الوحدة من ثلاث أقسام أساسية وهي:

- قسم الدخل والخرج: والذي يعبر عن مداخل الوحدة (i0,i1) ومخارجها (eq).
- قسم التصريح عن الإشارات: وهي الإشارات الداخلية التي تربط بين مكونات الوحدة (p0,p1).
- قسم جسم الوحدة: وهو مكون من بنية الدارة وهي تحتوي على ثلاث عمليات اسناد دائمة. حيث تتكون كل عملية اسناد من عمليات منطقية أساسية.

١.٤ العناصر المعجمية

نبين فيما يلي أهم العناصر المعجمية (lexical elements) للغة Verilog وهي المصطلحات المستخدمة في هذه اللغة لتسمية مكوناتها.

المعرف (Identifier)

يعطي المعرف اسماً وحيداً لكل غرض من الدارة مثل eq1 أو i0 أو p0. وهو مكون من حروف أبجدية و المحرف '_' و '$'. يتم التمييز بين الحروف الكبيرة والصغيرة في Verilog كما هي الحال في لغة C. ومن الأفضل انتقاء اسماء معبرة عند تسمية الأغراض.

الكلمات المفتاحية (Keywords)

وهي الكلمات المحجوزة لتعريف تعليمات لغة Verilog. ونشير إليها هنا باستعمال الخط الغامق مثل **module** و **wire**.

الفراغ الأبيض (White space)

يمكن استعمال أحرف الفراغ (space) وفراغ الجدولة (tab) وبداية سطر جديد (new line) بشكل حر في Verilog من أجل جعل العبارات أكثر وضوحاً للقراءة.

التعليقات (Comments)

يمكن اضافة التعليقات للرماز باستعمال '//' لإضافة تعليق ضمن سطر واحد أو الرمزين '/*' و '*/' لإضافة تعليق على عدة أسطر بشكل مشابه للتعليقات في لغة C.

٢.٤. أنماط المعطيات

هناك العديد من أنماط المعطيات (data types) المستخدمة في لغة Verilog. وتعتمد هذه الأنماط على النظام المنطقي بأربعة قيم حيث تستعمل أربع قيم أساسية لأي إشارة منطقية وهي:

- '0': من أجل قيمة الصفر المنطقي.
- '1': من أجل قيمة الواحد المنطقي.
- 'z': من أجل القيمة العائمة (tristate) أو ممانعة عالية (high impedance).
- 'x': من أجل قيمة غير محددة أو غير معلومة. ويستخدم خلال النمذجة والمحاكاة فقط.

يوجد في لغة Verilog مجموعتين من الأنماط وهما مجموعة الشبكة (net group) مجموعة المتحولات (variable group).

١.٢.٤. مجموعة الشبكة

تعبر أنماط المتحولات في هذه المجموعة عن الوصلات الفيزيائية بين العناصر العتادية. وتستعمل كخرج لعبارات الاسناد أو لإشارات الوصل بين الوحدات. النمط الأكثر استعمالاً في هذه المجموعة هو النمط **wire** والذي يدل على وصلة سلكية.

مثال: من أجل سلك وحيد

```
wire p0,p1;
```

مثال: من أجل مجموعة من الأسلاك أو مسرى يمكن استعمال شعاع من الأسلاك:

```
wire [7:0] data1, data2;  //8 bit data
wire [31:0] addr;         //32-bit address
wire [0:7] revers_data;   // ascending index should be avoided
```

يمكن جعل دليل الشعاع في الترتيب التصاعدي أو التنازلي، ولكن من المفضل استعمال الترتيب التنازلي لتتوافق مع صياغة الأرقام بالنظام الثنائي حيث يكون البت ذو الدلالة العظمى إلى اليسار.

مثال: أحيانا نحتاج لأشعة ببعدين (مصفوفات) لتمثيل عناصر الذاكرة مثل:

```
wire [3:0]  mem1 [31:0] ; // 32-by-4 memory
```

ويوجد أيضاً ضمن هذه المجموعة أنماط مثل **wand** الذي يعبر عن وصلة عن طريقة بوابة AND وأيضاً **supply0** للتعبير عن خط الأرضي في الدارة. كما يدعم verlog-2001 نمط **signed** للدلالة على مجموعة بتات بصيغة المتمم الثنائي.

٢.٢.٤. مجموعة المتحولات

تعبر أنماط المعطيات في مجموعة المتحولات عن عنصر تخزين وهو الذي يشكل خرجاً لإسناد إجرائي في السوية السلوكية. يوجد خمسة أنماط في هذه المجموعة وهي:

reg, integer, real, time, realtime

ومن أكثرها استعمالاً هو النمط **reg** وهو مستخدم لتركيب الدارة، أما الأنماط الثلاثة الأخيرة فهي مستخدمة فقط لأغراض المحاكاة ولا تستخدم في تركيب الدارة لذلك نصفها بأنها أنماط غير قابلة للتركيب. أما النمط **integer** فسنأتي على تفصيله في الفقرة التالية.

ملاحظة هامة جداً

تحتوي لغة Verilog على مجموعة من الكلمات المفتاحية وأنماط للمتحولات وعمليات حسابية مختلفة. إلا أنه يستخدم جزء منها فقط لتوصيف بنية الدارة العتادية. أما بقية الكلمات المفتاحية فتستخدم لأغراض المحاكاة والتحقق فقط. وسنشير إلى أي عملية لا يمكن استعمالها بتوصيف الدارة بوصفها بالمصطلح "غير قابلة للتركيب" (non synthesizable).

٣.٤. تمثيل الأرقام

يمكن تمثيل الثوابت الصحيحة في لغة Verilog بالصيغة العامة التالية:

[sign] [size]'[base] [value]

حيث:

sign: يعبر عن اشارة الرقم.

size: عدد البتات المستخدمة لتمثيل الرقم.

base: الأساس المستخدم في كتابة الرقم ويأخذ القيم التالية:

- B: Binary
- O: Octal
- H: Hexadecimal
- D: Decimal

الحقل size هو حقل اختياري، وعند تحديده نقول أن الرقم محدد القياس (sized number)، و إلا فإننا نقول أن الرقم غير محدد القياس (unsized number). عند عدم تحديد القياس فإن قياس الرقم الفعلي يكون هو القياس الافتراضي للحاسوب وهو 32 بت في أغلب الأحيان.

يبين الشكل التالي بعض الأمثلة عن تمثيل الأرقام:

Number	Stored Value	Comment
5’b11010	11010	
5’b11_010	11010	‘_’ ignored
5’o32	11010	
5’h1a	11010	
5’d26	11010	
5’b0	00000	0 extended
5’b1	00001	0 extended
5’bz	zzzzz	Z extended
5b’x	xxxxx	X extended
5’bx01	xxx0l	X extended
-5’b00001	11111	2’s complement
'b11010	00000000000000000000000000011010	Extended to 32 bits
'hee	00000000000000000000000011101110	Extended to 32 bits
1	00000000000000000000000000000001	Extended to 32 bits
-1	11111111111111111111111111111111	Extended to 32 bits

٤.٤. العمليات

هناك الكثير من العمليات (operations) المضمنة في لغة Verilog. ومن العمليات التي تهمنا على سوية البوابات المنطقية نذكر: &(AND), |(OR), ~(NOT), ^(XOR).

٥. هيكلية البرنامج

عند تحليل برنامج التوصيف العتادي، يجب التفكير في البرنامج بأنه وسيلة لتوصيف التنظيم العتادي وليس تعليمات برمجية متسلسلة. وترتكز لغة Verilog على مفهوم "الوحدة" (module) في بناء الدارات. حيث يقابل ذلك مفهوم التابع في لغات البرمجة. ولهذه الوحدة بوابات دخل وبوابات خرج. ويحتوي جسم الوحدة على كتل لتوصيف بنيتها العتادية الداخلية.

١.٥. التصريح عن البوابات

يتم التصريح عن بوابات الدخل والخرج (port declaration) للوحدة الوصفية بالصيغة العامة التالية:

```
module [module-name]
(
[mode] [data- type] [port-names] ,
[mode] [data- type] [port-names] ,
. . .
[mode] [data- type ] [port-names]
);
```

حيث يأخذ النمط [mode] القيم التالية: **input, output, inout**.

يأخذ نمط المعطيات [data- type] أي نمط معطيات تم ذكره سابقاً ويمكن حذفه إذا كان **wire**.

في المعيار Verilog-1995 يتم ذكر النمط بشكل مستقل كالتالي:

```
module eql (i0, il, eq); // only port names in brackets
// declare mode
input i0, il;
output eq;
// declare data type
wire i0, il;
wire eq;
```

٢.٥. جسم البرنامج

يحتوي جسم برنامج التوصيف على وصف أجزاء من الدارات التي تعمل على التوازي وبشكل متزامن. وتتكون هذه الأجزاء من:

- عبارة اسناد دائم (continuous assignment): وهي مفيدة من أجل الدارات التراكبية حيث تأخذ الصيغة التالية:

assign [signal-name] = [expression] ;

حيث يعبر [signal-name] عن خرج الدارة التراكبية الموصفة بالعبارة [expression].

- كتلة "دائماً" (always): وهي تعبر عن عمليات اسناد إجرائية وسنأتي على شرحها لاحقاً.
- استدعاء وحدة أخرى (module instantiation): وهي عملية إضافة وحدة عتادية موصفة مسبقاً ضمن وحدة مستقلة ويتم تضمين نسخة منها في الوحدة الحالية.

٣.٥. التصريح عن الإشارات

يحتوي قسم التصريح عن الإشارات (signal declaration) على أسماء الإشارات التي تربط بين أجزاء الوحدة. مثل:

wire p0, p1;

ليس من الضروري التصريح المسبق عن الإشارات في Verilog. فإن تم استعمال إشارة لم يتم التصريح عنها مسبقاً بشكل صريح، فإن نمط المعطيات لهذه الإشارة سيكون من مجموعة نمط الشبكة ضمناً (implicit net) وسيكون النمط الافتراضي هو **wire**. يوضح المثال التالي كيفية كتابة وحدة المقارن باستعمال إشارات ضمنية.

```
module eql
// I/O ports
(
input wire i0, il,
output wire eq
) ;
// no internal signal declaration

// product terms must be  placed in front
assign p0 = ~i0 & ~il;   // implicit declaration
assign pl = i0 & il;      // implicit declaration
// sum of two product terms
assign eq = p0 | pl;

endmodule
```

٤.٥ التوصيف البنيوي

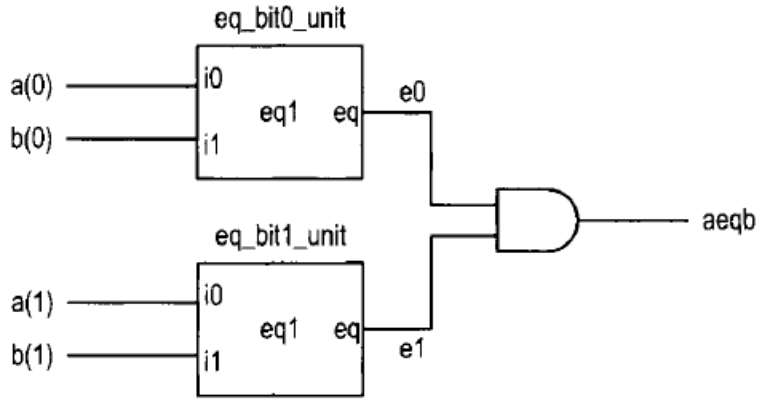
يتكون النظام الرقمي عادةً من العديد من النظم الجزئية. وذلك يسمح لنا ببناء النظام بشكل أسهل حيث نستعمل وحدات جاهزة أو مصممة مسبقاً لبناء النظام المطلوب. تسمح لغة Verilog بعملية استدعاء الوحدات. تسمى عملية التوصيف هذه بالتوصيف البنيوي (structural description) للوحدة المطلوبة. لتوضيح كيفية استعمال هذه الميزة، سوف نأخذ المثال التالي.

٥.٥ مثال مقارن على بتين

سنقوم الآن بتطوير وحدة المقارن السابق ليقوم بمقارنة مقدارين كل منهما على بتين اثنين كما هو مبين في التوصيف التالي:

```
module eq2_sop
// I/O ports
(
input wire [1:0]  a, b,
output wire  aeqb
) ;
// internal signal declaration
wire p0, p1, p2, p3;

// sum o f two product terms
assign aeqb = p0 | p1 | p2 | p3;
// product terms
assign p0 = (~a[1] & ~b[1]) & (~a[0] & ~b[0]);
assign p1 = (~a[1] & ~b[1]) & ( a[0] &  b[0]);
assign p2 = ( a[1] &  b[1]) & (~a[0] & ~b[0]);
assign p3 = ( a[1] &  b[1]) & ( a[0] &  b[0]);

endmodule
```

يمكن أيضاً تصميم هذا المقارن على بتين باستعمال المقارن على بت واحد الذي قمنا بتصميمه سابقاً كما هو مبين في الشكل التالي:



مقارن على بتين باستعمال مقارنين على بت واحد

حيث تكون المساواة بين a و b إذا كان كلا البتين متساويين بينهما. لذلك نستخدم مقارنين على بت واحد يعملان على التوازي. ومن ثم نجري عملية AND منطقية بين الخرجين (e0, e1) للحصول على الخرج النهائي (aeqb).

```
module eq2
 (
input wire [1:0]  a, b,
output wire  aeqb
) ;
wire e0, e1;

// instantiate two 1-bit comparators
eq1  eq_bit1_unit(.i0(a[0]), .i1(b[0]), .eq(e0) );
eq1  eq_bit2_unit(.i0(a[1]), .i1(b[1]), .eq(e1) );
// a and b are equal if individual bits are equal
assign aeqb = e0 & e1;

endmodule
```

يتم استدعاء وحدة بالشكل التالي: نضع أولاً اسم الوحدة التصميمة، أي eq1 هنا. ونعطي اسماً لهذه الكتلة (eq_bit1_unit) لتمييزها عن بقية الكتل. ثم نربط مداخل ومخارج هذه الكتلة بالإشارات المناسبة. يتم الربط بوضع الإشارة المناسبة ضمن قوسين بعد اسم المدخل أو المخرج الموافق للوحدة eq1. في هذا المثال، استعملنا نسختين من وحدة المقارنة على بت واحد، وأعطينا كل نسخة اسماً مختلفاً.

يبين هذا المثال التقابل الواضح بين رماز الوحدة والمخطط الصندوقي لها. أي أن رماز الوحدة ما هو إلا طريقة للتعبير عن المخطط البنيوي للدارة. لذلك يجب على مصمم الوحدة بلغة

التوصيف العتادي أن يضع المخطط في ذهنه عند كتابة رماز الوحدة. كما أنه ليس لترتيب الكتل المختلفة أي أثر في بنية الوحدة المصممة.

ملاحظة:

يمكن استدعاء الوحدة بشكل مختصر كالتالي:

```
eq1  eq_bit1_unit(a[0], b[0], e0);
eq1  eq_bit2_unit(a[1]), b[1], e1);
```

ولكن يجب المحافظة على الترتيب الصحيح للإشارات بحسب ترتيب المداخل والمخارج المتبع في كتابة الوحدة التي نستدعيها. نسمي هذه الطريقة في وصل المداخل والمخارج بطريقة الوصل بترتيب الأسماء. صحيحٌ أن هذا يعطي كتابة مختصرة، إلا أن ذلك غير منصوح به بشكل عام وذلك تجنباً للأخطاء في ترتيب الإشارات أو تغيير هذا الترتيب في تعريف الوحدة نفسها في وقت لاحق. لذلك سوف نقوم دائماً باستعمال طريقة الوصل بالإسم.

هناك العديد من الوحدات الأولية (Verilog primitive) الجاهزة في Verilog والتي يمكن استدعاؤها مباشرة ضمن الوحدة. ويشمل ذلك على البوابات المنطقية الأساسية. وكمثال على ذلك يمكن إعادة كتابة رماز وحدة المقارنة على بت واحد بالشكل التالي:

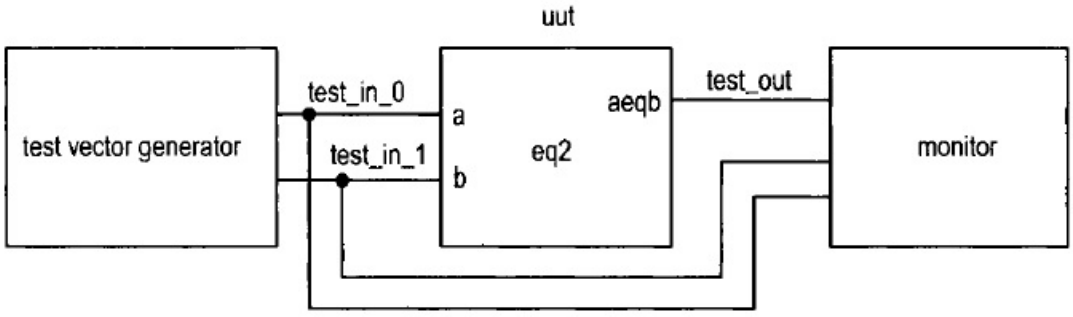
```
module eql_primitive(input wire i0, i1,output wire eq);
//  internal signal declaration
wire i0_n, il_n, p0, p1;
// primitive gate instantiations
not unit1 (i0_n, i0);          // i0_n = ~i0;
not unit2 (i1_n, i0);          // i1_n = ~il;
and unit3 (p0, i0_n, il_n); // p0 = i0_n & i1_n ;
and unit4 (p1, i0, i1);        // p1 = i0 & il;
or   unit5 (eq, p0, pl);       // eq = p0 | p1;
endmodule
```

تعتبر الكتابة بهذه الطريقة لكامل الوحدة متعبة جداً ويمكن الاستغناء عنها بكتابة عبارات اسناد منطقية. كما يمكن للمستخدم من تعريف وحدات أولية خاصة به من خلال جدول الحقيقة وتسمى وحدات أولية معرفة من قبل المستخدم UDP (User Defined Primitive). وكمثال على ذلك نكتب وحدة المقارنة على بت واحد كوحدة أولية بالشكل التالي:

```
primitive eql_udp(eq, i0, i1);
  output  eq;
  input   i0, i1;

table
// i0  i1 : eq
  0   0 : 1
  0   1 : 0
  1   0 : 0
  1   1 : 1
endtable
endprimitive
```

٦. الاختبار

بعد كتابة رماز الدارة الرقمية، يمكن محاكاتها على الحاسوب للتحقق من عملها الصحيح ومن ثم تشكيلها على دارة فيزيائية قابلة للبرمجة. يمكن اجراء المحاكاة ضمن نفس إطار لغة التوصيف HDL. وذلك بكتابة برنامج خاص يعرف باسم طاولة الاختبار (testbench) وذلك تقليداً لاسم طاولة المختبر التي نستعملها لاختبار الدارات الكهربائية. يبين الشكل التالي مخطط الاختبار لوحدة المقارن على بتين eq2 الذي كتبناه سابقاً.



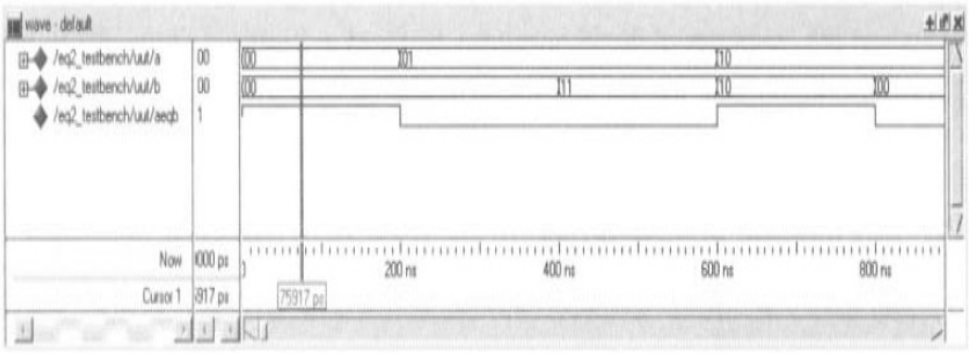
في هذا الشكل:

- تمثل الكتلة المسماة uut (unit under test) الوحدة التي نريد اختبارها.
- تمثل كتلة مولد أشعة الاختبار (testvector generator) الكتلة التي تولد قيم الدخل.
- تمثل كتلة الاظهار (monitor) الكتلة التي تظهر قيم الخرج وتتأكد من صحتها.

يبين الرماز التالي برنامج الاختبار لوحدة المقارن على بتين:

```
// The 'time scale directive specifies that the simulation time unit is 1 ns
// and the simulation time step is 10 ps
'timescale 1 ns/10 ps

module eq2-testbench;
        // signal declaration
        reg [1:0] test_in0, test_in1 ;
        wire test_out ;
        // instantiate the circuit under test
        eq2 uut( .a(test_in0), .b(test_inl), .aeqb(test_out) );
        // testvector generator
        initial
        begin
                // testvector 1
                test_in0 = 2'b00;
                test_in1 = 2'b00;
                # 200;
                // testvector 2
                test_in0 = 2'b01;
                test_in1 = 2'b00;
                # 200;
                // testvector 3
                test_in0 = 2'b01;
                test_in1 = 2'bll;
                # 200;
                // testvector 4
                test_in0 = 2'b10;
                test_in1 = 2'bl0;
                # 200;
                // testvector 5
                test_in0 = 2'bl0;
                test_in1 = 2'b00;
                # 200;
                // testvector 6
                test_in0 = 2'bll;
                test_in1 = 2'bll;
                # 200;
                // testvector 7
                test_in0 = 2'bll;
                test_in1 = 2'b01;
                # 200;
                // stop simulation
                $stop;
        end
endmodule
```

في هذا البرنامج، فإنه ليس لبرنامج الاختبار والذي تتم كتابته بنفس طريقة كتابة الوحدات العتادية من مداخل أو مخارج. يتكون جسم البرنامج من عملية استدعاء للوحدة التي نريد اختبارها ونطبق عليها إشارات الاختبار. تقوم كتلة مولد أشعة الاختبار بتوليد عدة قيم اختبارية ليتم تطبيقها بشكل متسلسل حيث تكون فترة تطبيق كل منها 200 وحدة زمنية. تقابل الوحدة الزمنية هنا 1 ns وهو ما تم تحديده في بداية البرنامج باستعمال الموجهة **'timescale**. عند الوصول إلى التعليمة **$stop** يتم ايقاف المحاكاة والعودة إلى برنامج المحاكاة. لم نقم هنا بتنفيذ وحدة الإظهار والمراقبة. حيث يمكن القيام بذلك برسم اشارات الدخل والخرج والتحقق من صحة النتائج من قبل المستخدم. كما يمكن إضافة التعليمات الخاصة للتحقق الآلي من صحة العمل ولكن ذلك يتطلب معلومات متقدمة في بناء برامج الاختبار لا داعي للخوض فيها الآن. يبين الشكل التالي رسماً لإشارات الدخل والخرج التي يمكن الحصول عليها باستعمال إحدى برامج المحاكاة.



إشارات الدخل والخرج لوحدة المقارن على بتين

٧. خلاصة

لغة التوصيف العتادي Verilog هي لغة تستعمل لتوصيف الدارات الرقمية بأي سوية من سويات التجريد من سوية البوابات إلى سوية نقل السجلات إلى السوية السلوكية. ترتكز هذه اللغة على قواعد شبيهة بلغة C في البرمجة. حيث تعرفنا على أنماط المعطيات وطرق تمثيل الأرقام. يتم توصيف الدارات بشكل بنوي باستعمال الوحدات. وتتألف الوحدة من مداخل ومخارج وإشارات داخلية وكتل بناء أساسية. يتم التحقق من عمل الوحدات عن طريق المحاكاة باستعمال برامج خاصة تسمى برامج الاختبار.

8. أسئلة الفصل الرابع:

أسئلة عامة

١. ما هي القيم الأربعة المستعملة للإشارات المنطقية مع تحديد معنى كل قيمة؟

القيم المنطقية الأربعة هي:

- '0': من أجل قيمة الصفر المنطقي.
- '1': من أجل قيمة الواحد المنطقي.
- 'z': من أجل القيمة العائمة (tristate) أو ممانعة عالية (high impedance).
- 'x': من أجل قيمة غير محددة أو غير معلومة. ويستخدم خلال النمذجة والمحاكاة فقط.

٢. ماذا نستعمل لتوصيف الدارات الرقمية على سوية البوابات؟

نستعمل البوابات المنطقية الأساسية مثل AND و OR و NOT وغيرها.

٣. اكتب المعادلة المنطقية لمقارن على بت بمدخلين i0 و i1 ومخرج واحد eq

يمكن كتابة المعادلة المنطقية لهذا المقارن بالشكل:

$$eq = i_0.i_1 + i_0'.i_1'$$

٤. ما هي الأقسام الأساسية الثلاثة للوحدة في لغة Verilog؟

الأقسام الثلاثة للوحدة هي:

- قسم الدخل والخرج: والذي يعبر عن مداخل الوحدة ومخارجها.
- قسم التصريح عن الإشارات: وهي الإشارات الداخلية التي تربط بين مكونات الوحدة.
- قسم جسم الوحدة: وهو مكون من بنية الدارة.

٥. عرف إشارة باسم x من نمط wire تحتوي على ٤ بتات.

```
wire [٣:0] x;  //4 bit signal
```

٦. اكتب رماز الوحدة الأولية تسمى or_udp بمدخلين x,y ومخرج o تقوم بعملية OR منطقية بين الدخلين وذلك باستعمال جدول الحقيقة.

```
primitive or_udp(o, x, y);
  output  o;
  input    x, y;

table
// x  y : o
  0    0 : 0
  0    1 : 1
  1    0 : 1
  1    1 : 1
endtable
endprimitive
```

٧. ما هي طرق وصل المداخل والمخارج عند استدعاء وحدة بلغة Verilog ؟

يوجد طريقتين وهما: طريقة الوصل بالاسم و طريقة الوصل بترتيب الأسماء.

٨. ماذا تعني التعليمة التالية (**timescale** 1 ns/10 ps‘) المستخدمة في أول ملف الاختبار في لغة verilog؟

تعني أن الوحدة الزمنية في المحاكاة تساوي 1ns وخطوة المحاكاة هي 10ps.

أسئلة خيارات متعددة

١. لغة verilog هي لغة تستعمل بهدف

A. التوصيف العتادي للدارات الرقمية

B. برمجة المعالجات العامة ذات سوية منخفضة

C. توصيف خوارزميات النظم المضمنة

D. برمجة منصات اختبار للمعالجات العامة والخاصة.

٢. تم اصدار المعيار الذي ينظم لغة Verilog لأول مرة في عام

A. 1984

B. 1987

C. 1993

D. 2001

٣. ما هو نمط المعطيات الذي لا ينتمي لمجموعة الشبكة (net group) في لغة Verilog؟

A. wire

B. reg

C. wand

D. supply0

٤. عندما نقول عن كلمة مفتاحية في لغة Verilog بأنها غير قابلة للتركيب (non synthesizable) فإن ذلك يعني؟

A. بأنها غير مستعملة في توصيف الدارات.

B. بأنها تستعمل لأغراض المحاكاة فقط.

C. تستعمل في وحدات الاختبار.

D. كل ما سبق.

٥. ما هي القيمة المخزنة من أجل القيمة 5'b00001- في لغة Verilog؟

A. ٠٠١٠١

B. ١٠١٠١

C. ١١١١١

D. ٠٠٠٠١

٦. تسمى طريقة وصل المداخل والمخارج المستعملة في الاستدعاء التالي لوحدة بلغة Verilog

```
eq1  eq_bit1_unit(.i0(a[0]), .i1(b[0]), .eq(e0) );
```

A. طريقة الوصل بالاسم.

B. طريقة الوصل بترتيب الأسماء.

C. طريقة الوصل المباشر.

D. طريقة الوصل بالدليل.

٧. ما هي مداخل ومخارج برنامج وحدة الاختبار لوحدة في Verilog؟

A. مداخل ومخارج الوحدة تحت الاختبار

B. مولد شعاع الاختبار ووحدة الإظهار.

C. جميع ما سبق.

D. لا يوجد له مداخل أو مخارج.

٨. ما هو النمط القابل للتركيب من بين الأنماط التالية في لغة Verilog؟

A. reg
B. real
C. time
D. realtime

الإجابات الصحيحة

رقم التمرين	الإجابة الصحيحة
1	A
2	C
3	B
4	D
5	C
6	A
7	D
8	A

الفصل الخامس:
الدارات التراكبية في Verilog

الكلمات المفتاحية:

العمليات الحسابية والمنطقية في Verilog، ضبط عرض الإشارات، كتلة "دائماً"، عبارة الإسناد الإجرائي، عبارة "إذا – وإلا"، عبارة "في حال"، شبكة التوصيل.

Operations in Verilog, bit-length adjustment, “always” block, Procedural assignment, “If-else” statement, “case” statement, Routing network.

الملخص:

يهدف هذا الفصل إلى التعريف بكيفية تصميم الدارات التراكبية على سوية نقل السجلات باستخدام عناصر متقدمة أكثر مثل الجوامع والمقارنات والنواخب وذلك بفضل العمليات الحسابية الجاهزة التي توفرها لغة Verilog. كما سنتعرف على طريقة التوصيف السلوكي من خلال استعمال عبارات متقدمة ضمن كتلة "دائماً" والتي قد تشمل على عبارات اسناد إجرائية أو عبارات شرطية مثل "إذا–وإلا" و"في حال". نختم هذا الفصل ببعض الإرشادات العامة لتجنب الوقوع في الأخطاء التصميمة الشائعة في لغة Verilog.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- العمليات الحسابية والمنطقية في لغة Verilog
- بنية كتلة "دائما" وقائمة التحسس في الدارات التراكبية
- أنواع العبارات الإجرائية وتشمل على الإسناد الإجرائي والعبارات الشرطية "إذا–وإلا" و"في حال"
- استعمال الثوابت والمعاملات لتكييف تصميم الوحدات مع إشارات ذات أطول متغيرة
- اثر اختيار العبارات على التصميم الفعلي للدارة.
- إرشادات عامة في التصميم

يهدف هذا الفصل إلى التعريف بكيفية تصميم الدارات التراكبية على سوية نقل السجلات باستخدام عناصر متقدمة أكثر مثل الجوامع والمقارنات والنواخب وذلك بفضل العمليات الحسابية الجاهزة التي توفرها لغة Verilog. كما سنتعرف على طريقة التوصيف السلوكي من خلال استعمال عبارات متقدمة ضمن كتلة "دائماً" والتي قد تشمل على عبارات اسناد إجرائية أو عبارات شرطية مثل "إذا–وإلا" و"في حال". نختم هذا الفصل ببعض الإرشادات العامة لتجنب الوقوع في الأخطاء التصميمية الشائعة في لغة Verilog.

1. مقدمة:

استعملنا في الفصل السابق البوابات المنطقية البسيطة لتوصيف الدارات على سوية البوابات المنطقية. ولكن تصميم الدارات الرقمية المعقدة على هذه السوية قد يكون مجهداً جداً. لذلك تم تطوير العديد من الإمكانيات في لغة Verilog لتدعم التصميم في السويات العليا مثل سوية نقل السجلات من خلال توفير استعمال العديد من العمليات الحسابية التي تعتمد في تصميمها على العناصر التراكبية على سوية نقل السجلات مثل الجوامع والنواخب. يمكن هذا الأمر بالانتقال إلى سوية أعلى في تصميم الدارات مما يسرع عملية التصميم ويوفر الوقت والجهد على المصمم في بناء جداول الحقيقة واختزال العبارات المنطقية. إذ يتم ذلك آلياً ضمن مترجم لغة Verilog لإعطاء التصميم الأمثلي.

لا تقتصر إمكانيات لغة Verilog على سوية نقل السجلات بل تتعدى ذلك لتوفر أدوات التصميم على السوية السلوكية للدارة من خلال استعمال عبارات متقدمة مثل العبارات الشرطية "إذا–وإلا" و "في حال" بشكل مشابه للغات البرمجة عالية المستوى. صحيح أننا هنا نستخدم عبارات شبيهة بلغات البرمجة التتابعية على المعالجات ولكن يجب دائماً الأخذ بعين الاعتبار أن هذه العبارات ما هي إلا وسيلة لتوصيف دارة رقمية تستعمل عناصر عتادية على سوية نقل السجلات. ولذلك يجب فهم كيف يتم ترجمة هذه التعليمات التوصيفية إلى العتاد الموافق.

سنتعلم في هذا الفصل عملية استخدام امكانيات لغة Verilog على الدارات التراكبية. بينما سنعالج الدارات التعاقبية في الفصل القادم.

2. العمليات الحسابية والمنطقية:

تدعم لغة Verilog العديد من العمليات الحسابية والمنطقية بشكل مشابه للعمليات في لغة C.

نجد في الصفحات التالية الجداول التي تضم قائمة بالعمليات التي تدعمها لغة Verilog وقواعد الأولوية المتبعة وبعض الأمثلة على هذه العمليات.

من الجدير بالذكر أن هناك بعض العمليات غير قابلة للتركيب مثل العمليات: '/' و'%'. وإذا ما أردنا تنفيذ هذه العمليات فعلينا تصميم الدارات الموافقة بأنفسنا ضمن وحدات مستقلة، أو الحصول على تصميمها من مكتبات جاهزة.

Table 1 Verilog operators

Type of operation	Operator symbol	Description	Number of operands
Arithmetic	+	addition	2
	–	subtraction	2
	*	multiplication	2
	/	division	2
	%	modulus	2
	**	exponentiation	2
Shift	>>	logical right shift	2
	<<	logical left shift	2
	>>>	arithmetic right shift	2
	<<<	logical left shift	2
Relational	>	greater	2
	<	lees than	2
	>=	greater than or egual to	2
	<=	less than or egual to	2
Equality	==	equality	2
	!=	inequality	2
	===	case equality	2
	!==	case inequality	2
Bitwise	~	bitwise negation	1
	&	bitwise and	2

	|	bitwise or	2
	^	bitwise xor	2
Reduction	&	reduction and	1
	|	reduction or	1
	^	reduction xor	1
Logical	!	logical negation	1
	&&	logical and	2
	||	logical or	2
Concatenation	{ }	concatenation	any
	{ { } }	replication	any
Conditional	? :	conditional	3

Table 2 Operator precedence

Operator	precedence
!~ + − (unary)	highest
** */ % + − (binary)	
>> << >>> <<<	
< <= > >= == ! = === !==	
& ^ |	
&& ||	
? :	lowest

Table 3 Shift operation examples

a	a >> 2	a >>> 2	a << 2	a <<< 2
0100_1111	0001_0011	0001_0011	0011_1100	0011_1100
1100_1111	0011_0011	1111_0011	0011_1100	0011_1100

Table 4 Logical and bitwise operation examples

a	b	a&b	a | b	a&&b	a || b
0	1	0	1	0 (false)	1 (true)
000	000	000	000	0 (false)	0 (false)
000	001	000	001	0 (false)	1 (true)
011	001	001	011	1 (true)	1 (true)

لإدراك فائدة هذه العمليات في تبسيط العمل على سوية نقل السجلات، نعيد كتابة برنامج المقارن على بتين الذي عرضناه في الفصل السابق باستخدام العمليات المتاحة بالشكل التالي:

```
module eq2_rt (input wire [1:0]  a, b, output wire aeqb) ;

assign aeqb = (a == b);

endmodule
```

من المعروف أن العمليات المنطقية التالية '&' و'|' و'^' تتم بين معاملين. تتيح لغة Verilog امكانية تطبيقها على معامل وحيد مؤلفاً من مجموعة البتات كالتالي:

```
wire [3:0] a;
wire y;
assign y=& a;  //y=a[3] & a[2] & a[1] & a[0];
```

وهذا مكافئ للعبارة:

```
assign y=a[3] & a[2] & a[1] & a[0];
```

1.2. ضبط عرض الإشارات:

بخلاف تمثيل الأرقام على الحاسب الذي يتم على عدد معين من البتات بحسب نمط المتحول المستعمل، فإن تمثيل الإشارات والمتحولات في البنية العتادية للنظام الرقمي يتم على عدد من البتات يختلف بين إشارة وأخرى بحسب المجال الديناميكي لها. ولذلك عند اجراء العمليات الحسابية والمنطقية بين إشارات مختلفة في عدد البتات، علينا أولاً توحيد عدد البتات للمعاملات في الطرف الأيمن من العبارة. ويتم ذلك بتمديد طول الإشارات إلى أطول معامل. وتدعى هذه العملية بعملية ضبط عرض الإشارة أو الطول بعدد البتات (bit-adjustment length). تجري عملية التمديد وفق القواعد التي عرضناها في الفصل السابق وذلك بحسب تمثيل الإشارة بالصيغة الثنائية أو بصيغة المتمم الثنائي.

بعد ذلك تتم العملية الحسابية وتكون النتيجة ممثلة على نفس الطول الأعظمي للمعاملات. ثم يتم اسناد نتيجة العملية في الإشارة على الطرف الأيسر من عبارة الاسناد حيث يتم قص (truncate) البتات الزائدة من جهة البتات ذات الدلالة العظمى (MSB) إذا كانت النتيجة أطول من طول الإشارة التي يتم إليها الاسناد. أو يتم تمديد النتيجة لتصبح بطول الإشارة التي يتم إليها الاسناد.

مثال:

لنأخذ الأمثلة التالية على بعض العمليات الحسابية:

```
wire [7:0] a , b;
wire [7:0] sum8;
wire [8:0] sum9;

assign a=8'b1111_1111;
assign b=1;
assign sum8 = a + b;
assign sum9 = a + b;
assign sum8 = (a+b+0)>>1;
```

تم تعريف الإشارات a و b و sum8 على 8 بتات والإشارة sum9 على 9 بتات.

- في عبارة الاسناد الأولى تم اسناد ثابت بطول 8 بتات إلى a وذلك واضح وصريح.
- في العبارة الثانية، يتم اسناد قيمة 1 إلى الإشارة b. بما أن قيمة 1 غير محددة الطول فإنه يتم تمديدها إلى 32 بت كما رأينا في الفصل السابق. وعند الاسناد في الإشارة b، فإنه يتم الحفاظ على البتات الثمانية الدنيا فقط. فيكون محتوى b هو "00000001".
- في عملية الجمع الأولى، المعاملان a و b بنفس الطول وهو 8 بتات. تتم عملية الجمع ويكون الناتج على 8 بتات بالإضافة إلى بت الحمل. بما أن طول الإشارة المسند إليها sum8 هو 8، فيتم تجاهل بت الحمل. أي أن sum8=00000000.
- في عملية الجمع الثانية، يتم الحفاظ على بت الحمل ويخزن في البت التاسع من الإشارة، أي أن sum9=100000000.

- في عملية الجمع مع الإزاحة الأخيرة, نجد أن الطول الأعظمي للمعاملات هو 32 وذلك بسبب تمديد قيمة 0 لأنها غير محدد الطول. تجري عملية الجمع على 32 بت. ويكون الناتج هو: 00000000 00000000 00000001 00000000. وبعد عملية الإزاحة لليمين بمقدار بت واحد، تكون النتيجة النهائية للعبارة هي:

00000000 00000000 00000000 10000000

عند تخزين هذه النتيجة في الإشارة sum8 يتم الحفاظ على البتات الثمانية الدنيا فقط، أي:

sum8=100000000

من هذا المثال نجد أنه علينا الانتباه عند إجراء العمليات الحسابية إلى مسألة طول المعاملات وقواعد التمديد للحصول على النتائج المطلوبة. تتم عملية التمديد والقص بشكل آلي في Verilog عند إجراء العمليات الحسابية. ولكن يمكننا اجراء هذه العمليات بشكل يدوي إذا ما رغبنا بذلك وذلك كالتالي:

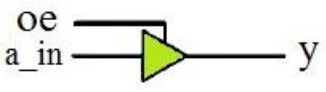
- عملية تمديد a بثلاث بتات صفري على اليسار: {3'b000,a}=c.
- عملية تمديد a بتكرار البت الأخير ثلاث مرات: {3{a[7]}, a}=c.
- عملية دوران a نحو اليمين بمقدار بتين: {a[1],a[0],a[7:3]}=c.

أي باستخدام رمز المجموعة {.} يتم تجميع مجموعة قطع لتكوين عنصر أطول. كما يتم القص بتحديد المجال المرغوب من دليل الشعاع.

2.2. تركيب القيم 'x' و 'z':

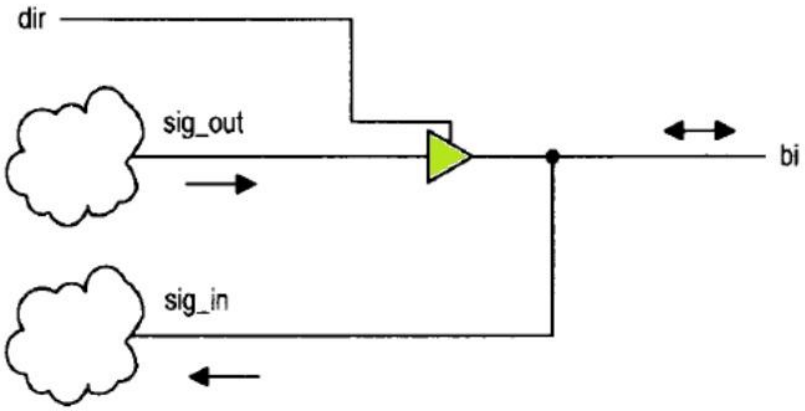
كما سبق وذكرنا بأن قيمة الإشارة 'x' هي قيمة ترمز لتجاهل قيمة الإشارة. فالقيم المنطقية الفعلية في الدارات الرقمية هي 0 أو 1. وبالتالي لا يوجد مقابل فيزيائي للقيمة x. عند تنفيذ الدارة، يستبدل المترجم القيمة 'x' بإحدى القيمتين 0 أو 1 وذلك حسب ما يسهل عملية تنفيذ الدارة.

وكذلك فإن القيمة 'z' التي تعبر عن حالة إشارة عائمة أو ممانعة عالية لا يتم تنفيذها بشكل فعلي داخل الدارات الرقمية، إلا من أجل المرابط الخارجية فقط للدارة المبرمجة والتي تكون مجهزة بعوازل يمكن وضعها في الحالة العائمة. وعادةً ما يتم استخدام الحالة العائمة للمرابط ثنائية الاتجاه **inout**. فعندما نريد أن يكون مربط ما بحالة دخل فإنه يجب وضعه بالحالة العائمة لتتمكن الدارة الخارجية من وضع إشارتها على هذا الخط.



oe	y
0	z
1	a_in

رمز وجدول عمل عازل ثلاثي الحالة (tri-state buffer).



عازل دخل / خرج ثنائي الاتجاه.

3. كتلة "دائماً" للدارات التراكبية:

من أجل تسهيل نمذجة الدارات، تمتلك لغة Verilog العديد من العبارات الإجرائية والتي يتم تنفيذها بشكل متعاقب. وبسبب الطبيعة المختلفة لهذه العبارات عن الكتل المتزامنة، يتم تجميعها في كتلة "دائماً" (**always**) أو كتلة "استهلال" (**initial**). تستخدم الكتلة "استهلال" لأغراض المحاكاة فقط. أما الكتلة "دائماً" فتستخدم في تركيب الدارات. وبما أن العبارات الإجرائية ذات طابع مجرد أكثر، لذلك تدعى هذه الطريقة في كتابة الرماز بالتوصيف السلوكي (behavioral description).

تحتوي كتلة "دائماً" على العديد من العبارات الإجرائية، وسنهتم في هذا الفصل بالعبارات التراكبية فقط. وتحديداً سوف نقتصر على ثلاث عبارات وهي:

- الإسناد الاجرائي الكابح (blocking assignment).
- عبارة "إذا–وإلا" (if–else statement).
- عبارة "في حال" (case statement).

سنشرح هذه العبارات بعد أن نشرح أولاً بنية كتلة "دائماً" وسلوكها.

يتم كتابة كتلة "دائماً" بالشكل العام التالي:

```
always @([sensitivity-list])
begin [optional name]
        [optional local variable declaration];

        [procedural statement];
        [procedural statement] ;
        ...
end
```

حيث تعبر [sensitivity–list] عن قائمة الشروط التي يجب أن تتحقق حتى يتم تنفيذ العبارات الإجرائية داخل الكتلة. ومن أجل الدارات التراكبية فإن هذه القائمة تحتوي على جميع الإشارات المستخدمة داخل الكتلة. وذلك يعني أنه يتم تنفيذ الكتلة عند حدوث أي تغير في إحدى هذه الإشارات. يمكن في بداية الكتلة التصريح عن المتحولات المستعملة ومن ثم يتم استعمال العبارات الإجرائية لتصميم الكتلة.

1.3.الإسناد الاجرائي:

يمكن استخدام الاسناد الاجرائي (procedural assignment) في كتلة **always** أو كتلة **initial** فقط وله نوعان: الاسناد الإجرائي الكابح (blocking assignment) والاسناد الإجرائي غير الكابح (nonblocking assignment). ويتم التعبير عنهما في لغة Verilog بالشكل التالي:

```
[[variable-name] = [expression] ;    // blocking assignment
[variable-name] <= [expression] ;  // nonblocking assignment
```

في الاسناد الكابح، يتم اسناد قيمة العبارة فوراً للمتحول قبل الانتقال للعبارة التالية. لذلك سوف نسميه بالإسناد الفوري للوضوح. أي أن المتحول سيحتوي على القيمة الجديدة إذا ما أُستعمل في العبارات التالية. أما في الإسناد غير الكابح فإنه لا يتم اسناد قيمة العبارة للمتحول إلا في نهاية تنفيذ كتلة "دائماً"، لذلك سوف نسميه بالإسناد المؤجل. أي أن المتحول سيحافظ على قيمته القديمة إذا ما أُستعمل في العبارات التالية.

عادةً ما يتم الخلط بين هذين النوعين من الاسناد. ولتجنب الأخطاء هناك قاعدة عامة لطريقة الاستعمال وهي:

يستعمل الإسناد الفوري في الدارات التراكبية، ويستعمل الإسناد المؤجل في الدارات التعاقبية.

مثال: لتوضيح عمل كتلة "دائماً"، سنعيد كتابة مثال المقارن على بت واحد بالشكل التالي:

```
module eql-always
(
input wire i0, i1,
output reg eq  //eq declared as reg
) ;
reg p0, p1;  //p0, p1 declared as reg

        always @(i0, i1) // i0 and i1 must be in sensitivity list
        begin
                // the order of statements is important
                p0 = ~i0 & ~ i1;
                p1 =  i0 & i1;
                eq =  p0 | p1;
        end
endmodule
```

بما أنه تم اسناد المتحولات p0 و p1 و eq داخل كتلة "دائماً" لذلك تم التصريح عنها باستخدام النمط **reg.** وهذا ضروري جداً لصحة توصيف الدارة. تحتوي قائمة التحسس على i0, i1. يتم تفعيل الكتلة عند تغير أحدهما لحساب قيمة المخارج الجديدة. في الدارات التراكبية، يجب اضافة كافة الإشارات المستخدمة إلى قائمة التحسس. لذلك يمكن اختصار ذلك باستعمال الصيغة **always@*.**

2.3. عبارة "إذا– وإلا":

تشبه طريقة صياغة عبارة "إذا–وإلا" (if–else statement) في لغة Verilog لطريقة صياغتها في لغة C. حيث تأخذ الشكل التالي:

```
if (boolean_expr)
  begin
        [procedural statement];
        …
  end
else
```

```
begin
    [procedural statement];
    …
end;
```

ولتوضيح طريقة استخدام هذه العبارة في Verilog، سنأخذ المثال التالي:

نريد تصميم دارة مفكك ترميز من خطين إلى 4 خطوط مع خط تفعيل en. يبين الشكل التالي جدول الحقيقة لهذه الدارة. حيث يتم تفعيل المخرج ذو الرقم الموافق لقيمة الدخل إذا كان خط التفعيل en=1 وإلا فإن جميع المخارج تكون غير مفعلة.

en	Input a(1)	a(0)	Output y
0	–	–	0000
1	0	0	0001
1	0	1	0010
1	1	0	0100
1	1	1	

جدول الحقيقة لمفكك ترميز مع إشارة تأهيل

يبين الرماز التالي طريقة توصيف مفكك الترميز باستخدام عبارة "إذا–وإلا".

```
module decoder_2_4_if
        (
        input wire [1:0] a,
        input wire en,
        output reg [3:0] y
        );

        always @*
                if (en==1'b0)  // can be written as (~en)
                   y = 4'b0000;
                else  if (a=2'b00)
                   y = 4'b0001;
                else  if (a=2'b01)
                   y = 4'b0010;
                else  if (a=2'b10)
                   y = 4'b0100;
                else
                   y = 4'b1000;

endmodule
```

لتجنب تكرار تعليمة if عندما نريد اختبار عدة حالات فإنه من المناسب أكثر استعمال عبارة "في حال".

3.3.عبارة "في حال":

تشبه طريقة صياغة عبارة "في حال" (case statement) في لغة Verilog لطريقة صياغتها في لغة C. حيث تأخذ الشكل التالي:

```
case [case_expr]
[item]:
        begin
          [procedural statements];
        end
[item]:
        begin
          [procedural statements];
        end
…
default:
        begin
          [procedural statements];
        end
endcase
```

لتوضيح عمل هذه العبارة، سوف نعيد كتابة المثال السابق باستخدام عبارة "في حال". يبين الرماز التالي طريقة توصيف مفكك الترميز السابق.

```
module decoder_2_4_case
        (
        input wire [1:0] a,
        input wire en,
        output reg [3:0] y
        );

        always @*
                case({en, a})
                  3'b000, 3'b010, 3'b100, 3b110:    y = 4'b0000;
                  3'b100:  y = 4'b0001;
                  3'b101:  y = 4'b0010;
                  3'b110:  y = 4'b0100;
                  3'b111:  y = 4'b1000;
                endcase
endmodule
```

وهنا تم اختبار الحالات على المتحول الجديد {en, a} الناتج عن دمج en مع a.

من أجل تجنب تكرار جميع القيم الممكنة عندما يكون en=0، يمكن استخدام عبارة **casez** أو عبارة **casex** بدلاً من **case** مع استبدال الحالة الأولى بالقيمة 3'b0??. حيث يدل الرمز '?' إلى تجاهل قيمة البت الموافق. وبالتالي، نتجاهل قيمة بتات a إذا كان البت en=0.

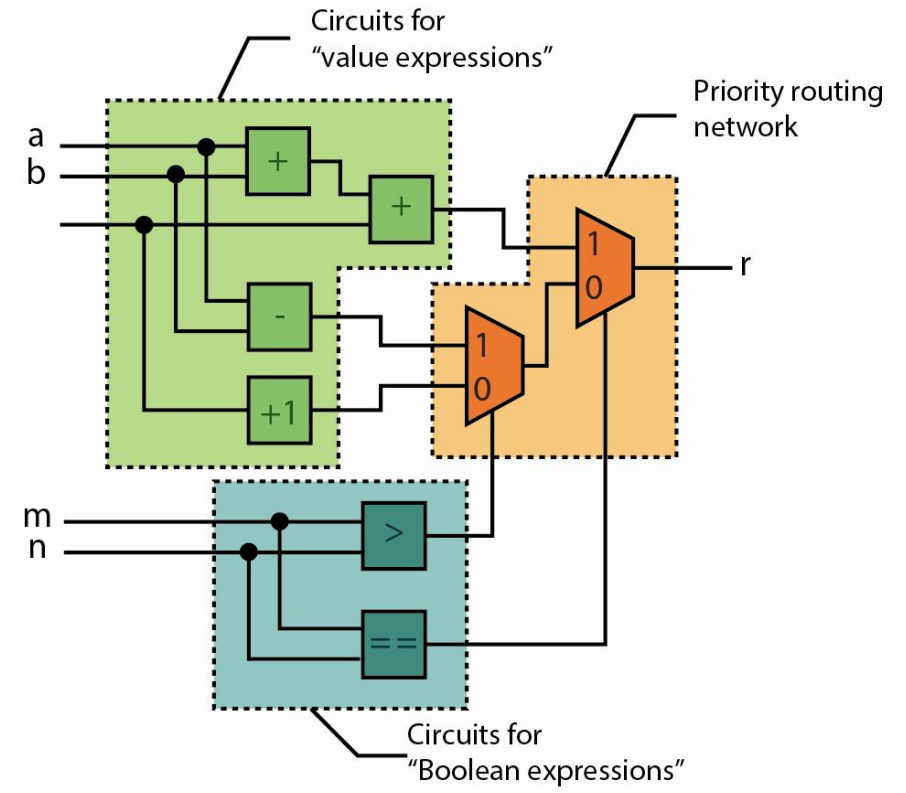
4. شبكة التوصيل:

شبكة التوصيل (routing network) هي عملية انتخاب قيمة إشارة من بين مجموعة من القيم التي يتم حسابها في دارت مختلفة على التوازي. لتوضيح ذلك سنأخذ المثال التالي:

لنفرض أننا نريد تنفيذ عبارة "إذا–وإلا" التالية:

```
if (m==n)
        r = a+b+c;
else if (m > n)
        r = a-b;
else
        r = c+1;
```

حيث r هو قيمة الخرج المطلوبة التابعة لقيم الدخل a, b, c بحسب نتيجة مقارنة قيم الدخل m و n. يعطي برنامج الترجمة للغة Verilog لهذه العبارة الدارة التالية:



دارة تركيب عبارة "إذا–وإلا"

تتألف هذه الدارة من ثلاثة أقسام:

- القسم الأول يقوم بحساب جميع العبارات الحسابية الموجودة داخل عبارة "إذا– وإلا".
- القسم الثاني يقوم بإجراء عمليات المقارنة بين m و n.
- القسم الثالث يقوم بانتخاب نتيجة العملية الحسابية المناسبة حسب نتائج المقارنة باستعمال ناخب بمدخلين ومخرج وحيد.

أي أن المترجم يستخدم عدة نواخب 2x1 ويكون كل ناخب محكوم بإشارة منطقية تعبر عن قيمة الشرط لتنفيذ العبارة الشرطية "إذا–وإلا".

أما تنفيذ عبارة "في حال" فيتم باستخدام ناخب وحيد بعدة مداخل بعدد الحالات الواردة. وبالتالي ستنعكس طريقة كتابتنا للرماز على البنية العتادية للدارة الناتجة. لذلك توفر معظم برامج التصميم باستعمال لغات التوصيف العتادية إمكانية عرض نتيجة الترجمة كمخطط دارة رقمية. لذلك يجب أن يكون المصمم مدركاً لطريقة الترجمة. إذ أن التوصيف الغير مناسب قد يعطي دارات عالية التعقيد بشكل لا ضرورة له.

5. الثوابت والمعاملات:

نستخدم في لغات التوصيف العتادي عادةً الثوابت (constants) في العبارات وأبعاد الإشارات. وهي ثوابت لا يمكن تغييرها. ولكن من المفضل إعطاء هذه الثوابت أسماء بدل من كتابتها كأرقام وهذا بهدف توضيح كتابة الرماز مما يسهل عملية المراجعة والتطوير للبنية العتادية فيما بعد. يتم التصريح عن هذه الثوابت باستخدام الكلمة المفتاحية (**localparam**).

بالإضافة إلى ذلك هناك إمكانية لتعريف معاملات يمكن تغييرها من قبل المستخدم عند استدعاء الوحدة، ويتم التصريح عنها في بداية الوحدة باستعمال الكلمة المفتاحية (**parameter**). لتوضيح دور الثوابت والمعاملات في لغة Verilog، سوف نأخذ مثال الجامع مع حمل مستقل على عدد من البتات N. حيث نعطي للمعامل N قيمة افتراضية N=4، ولكن نستطيع تغييرها عند أي استدعاء للوحدة إذا ما أردنا استعمال هذه الوحدة لتنفيذ الجمع على أي عدد آخر من البتات. لذلك فإنه عند كتابة وحدات بمعاملات متغيرة، يجب أن يكون التصميم عاماً بحيث يصلح لأي قيمة لمعاملات الدخل الممكنة.

يبين الرماز التالي طريقة كتابة وحدة الجامع:

```
module adder_carry
#(parameter N=4)
(
input wire [N-1:0] a , b,
output wire [N-1:0] sum,
output wire cout    // carry-out
);
// constant declaration
localparam N1 = N-1;
// signal declaration
wire [N:0] sum_ext ;
//body
assign sum-ext = {l'b0, a} + {1'b0, b};
assign sum = sum_ext[N1: 0] ;
assign cout= sum_ext[N];
endmodule
```

يقوم هذ الجامع بجمع رقمين a و b على N بت، ويعطي الناتج في sum على N بت أيضاً الحمل الناتج يخرج عبر الخط cout.

يمكن استدعاء هذه الوحدة بالشكل الاعتيادي من أجل إشارات بالعرض الافتراضي (4 بت)، أي:

```
// instantiate 4-bit adder
adder_carry unit1(.a(a4), .b(b4), .sum(sum4), . cout(c4));
```

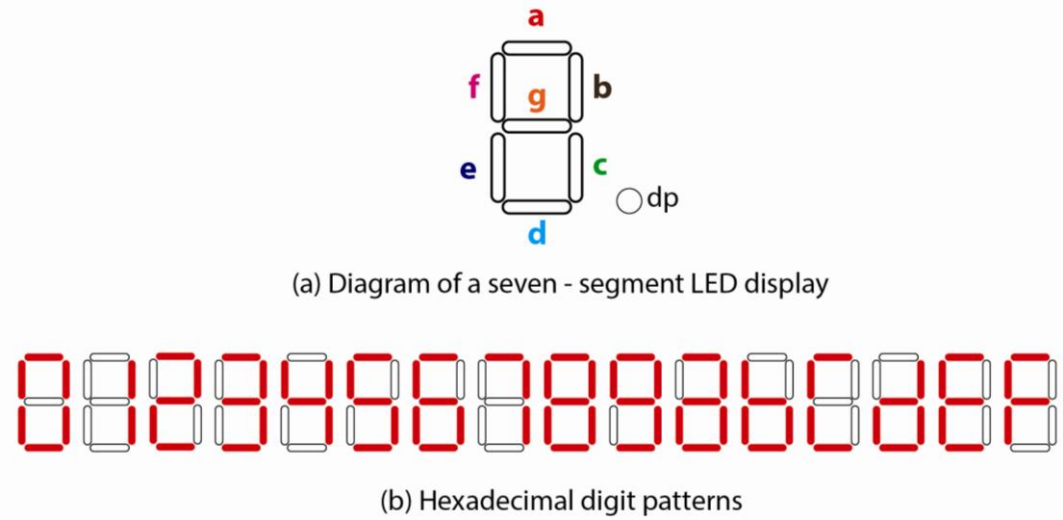
كما يمكن استدعاء وحدة الجامع من أجل إشارات بعرض 8 بت مثلاً بالشكل:

```
// instantiate 8-bit adder
adder_carry  #(. N(8))  unit2(. a(a8), . b(b8), . sum(sum8), . cout (c8)) ;
```

وهذا يعني أننا نستطيع توصيف وحدات بالصيغة العامة باستخدام المعاملات والثوابت، ومن ثم استدعاؤها بقيم مختلفة للمعاملات حسب الحاجة.

6. تطبيق: مفكك ترميز سباعي القطع:

تتألف شاشة الإظهار الرقمية من سبعة قطع لكل خانة. كل قطعة عبارة عن ديود ضوئي. لإظهار رقم ما على الشاشة، يتم اختيار القطع المناسبة التي يجب تشغيلها. لهذه الشاشة 7 خطوط للتحكم بإضاءة القطع السبع، بالإضافة إلى خط ثامن لإضاءة النقطة المجاورة للرقم. تسمى هذه القطع عادة بالرموز {a, b, c, d, e, f, g, h, dp} كما يبين الشكل التالي:



(a) Diagram of a seven - segment LED display

(b) Hexadecimal digit patterns

شاشة الإظهار الرقمية سباعية المقاطع

نريد إظهار رقم ست عشري ممثل على 4 بتات على هذه الشاشة. لذلك يجب بناء جدول لإضاءة القطع المناسبة من أجل كل رقم. نفرض أن القطعة تضيء عند وضع 0 على الخط الموافق لها. بينما تكون مطفأة عند وضع 1 على هذا الخط (أي أن القطعة فعالة على السوية المنخفضة). نريد بناء دارة تحول قيمة الدخل hex المكون من 4 بتات الذي يمثل الرقم الذي نريد اظهاره بالإضافة إلى بت الدخل dp يمثل النقطة العشرية إلى الخرج sseg المكون من 8 بتات والذي يتحكم بإضاءة جميع القطع على الشاشة.

تشكل عبارة "في حال" وسيلة مناسبة لتنفيذ هذه الدارة. يبين الرماز التالي وصف الدارة الموافقة.

```
module hex-to-sseg
(
input wire [3:01 hex,
input wire dp,
output reg [7:0] sseg // output active low
)
    always @*
    begin
        case (hex)
        4'h0: sseg[6:0] = 7'b0000001;
        4'hl: sseg[6:0]  = 7'b1001111;
        4'h2: sseg[6:0] = 7'b0010010;
        4'h3: sseg[6:0] = 7'b0000110;
        4'h4: sseg[6:0] = 7'b1001100;
        4'h5: sseg[6:0] = 7'b0100100;
        4'h6: sseg[6:0] = 7'b0100000;
        4'h7: sseg[6:0] = 7'b0001111;
        4'h8 : sseg[6:0] = 7'b0000000 ;
        4'h9 : sseg[6:0] = 7'b0000100 ;
        4'ha: sseg[6:0]  = 7'b0001000;
        4'hb: sseg[6:0]  = 7'b1100000;
        4'hc: sseg[6:0]  = 7'b0110001;
        4'hd: sseg[6:0]  = 7'b1000010;
        4'he: sseg[6:0]  = 7'b0110000;
        default : sseg[6:0] = 7'b0111000;  //4'hf
     endcase
        sseg[7] =dp;
    end
endmodule
```

7. إرشادات عامة في التصميم:

نورد فيما يلي بعض القواعد العامة لتجنب الأخطاء الشائعة في كتابة التوصيف العتادي للدارات التراكبية.

- أسند قيم كل متحول في كتلة واحدة فقط من كتل "دائماً".
- استعمل الاسناد الفوري للدارات التراكبية.
- استعمل *@ لتضمين جميع المداخل في قائمة التحسس.
- تأكد من تغطية جميع الحالات في عبارات "إذا–وإلا".
- أضف دائماً حالة "**default**" في كتابة عبارة "في حال".
- تأكد من اسناد قيمة للخرج في جميع الحالات الشرطية. ومن إحدى الطرق لتلبية ذلك القيام بإسناد القيم الافتراضية للخرج في بداية كتلة "دائماً".
- فكر بالبنية العتادية عند كتابة الرماز ولا تفكر بطرقة كتابة البرامج بلغة C.

8. خلاصة:

توفر لغة Verilog العديد من العمليات الحسابية والمنطقية المصممة مسبقاً لتسهيل توصيف الدارات التراكبية على سوية نقل السجلات. لا يمكن تركيب القيمة 'x' ولكن تستعمل لأغراض المحاكاة أو برامج الاختبار فقط. أما القيمة 'z' فلا يمكن تركيبها إلا في المرابط الخارجية للدارة المبرمجة. نستعمل كتلة "دائماً" لضم مجموعة من العبارات الإجرائية مثل الاسناد الإجرائي. كما تسمح عبارة "إذا–وإلا" وعبارة "في حال" من تصميم الدارات على السوية السلوكية للدارة مما يسهل ويسرع زمن تطوير تصميم الدارة. يمكن استعمال الثوابت والمعاملات لإعطاء تصميم الوحدات الصبغة العمومية مما يساعد في استدعاء الوحدة من أجل إشارات متعددة ومختلفة في عدد البتات. تؤثر طريقة كتابة الرماز على الدارة الرقمية الناتجة لذلك يجب ادراك طريقة ترجمة كل عبارة وانتقاء العبارة المناسبة التي تعطي التصميم الأصغر.

9. أسئلة الفصل الخامس:

أسئلة عامة:

1. هنالك نوعين للإسناد الإجرائي في لغة Verilog. ما هما هذان النوعان وما هو الفرق بينهما؟

- **الاسناد الكابح (أو الفوري)**، يتم اسناد قيمة العبارة فوراً للمتحول قبل الانتقال للعبارة التالية. أي أن المتحول سيحتوي على القيمة الجديدة إذا ما أُستعمل في العبارات التالية.
- **الاسناد غير الكابح (أو المؤجل)** فإنه لا يتم اسناد قيمة العبارة للمتحول إلا في نهاية تنفيذ كتلة "دائماً"، أي أن المتحول سيحافظ على قيمته القديمة إذا ما أُستعمل في العبارات التالية.

2. بفرض أن a=8'b1111_1111 و b=8'b0000_0001. ماهي القيمة المخزنة في المتحول sum9 المعرف على 9 بتات بعد تنفيذ العملية (**assign** sum9 = a + b;) مع التعليل؟

يتم جمع a و b على 8 بت لأن المعاملين بنفس الطول وهو 8 فنحصل على ناتج الجمع على 8 بت أي 0000_0000 بالإضافة إلى بت الحمل وهو 1 فيتم الحفاظ على بت الحمل ويخزن في البت التاسع من الإشارة، أي أن sum9 = 100000000.

3. بفرض أن a=5'b11001 و b=3'b101. ماهي القيمة المخزنة في المتحول sum4 المعرف على 4 بتات بعد تنفيذ العملية (**assign** sum4 = a + b;) مع التعليل؟

يتم جمع a و b على 5 بت لأنه الطول الأعظمي للمعاملين a و b حيث يتم تمديد قيمة b على 5 بتات لنحصل على 00101 فنحصل على ناتج الجمع على 5 بت أي 11110 بالإضافة إلى بت الحمل وهو 0. بما أن طول النتيجة sum4 على 4 بت فيتم قصر النتيجة بالاحتفاظ بالبتات ذات الدلالة الدنيا أي أن sum4 = 1110.

4. كيف نمدد قيمة عدد صحيح في لغة Verilog لم يتم تحديد طوله بشكل صريح مثل قيمة العدد 3 في العبارة التالية: **assign** sum = a+3;؟

5. مما تتكون قائمة التحسس لكتلة "دائماً" في لغة Verilog من أجل الدارات التراكبية؟

في الدارات التراكبية، يجب إضافة كافة الإشارات المستخدمة إلى قائمة التحسس.

6. مما تتكون شبكة التوصيل (rounting network) في لغة Verilog من أجل عبارة "إذا–وإلا"؟

تتكون الشبكة من عدة نواخب 2x1 ويكون كل ناخب محكوم بإشارة منطقية تعبر عن قيمة الشرط لتنفيذ العبارة الشرطية "إذا–وإلا".

7. مما تتكون شبكة التوصيل (rounting network) في لغة Verilog من أجل عبارة "في حال"؟

تتكون الشبكة من ناخب وحيد بعدة مداخل بعدد الحالات الواردة.

8. ماهي الفائدة من استعمال الثوابت والمعاملات في تصميم الوحدات في لغة Verilog؟

الفائدة هي أننا نستطيع توصيف وحدات بالصيغة العامة باستخدام المعاملات والثوابت، ومن ثم استدعاؤها بقيم مختلفة للمعاملات حسب الحاجة.

9. اكتب رماز وحدة تقوم بجمع عددين على N بت ممثلين بصيغة الإشارة والمطال (القيمة المطلقة). أي أن الرقم القيمة 3 و3– تمثلان بالصيغة 0011 و1011 بهذه الصيغة على 4 بتات. تتم العملية بالشكل التالي:

إذا كان لكلا الرقمين نفس الإشارة فتكون إشارة المجموع هي نفسها ونجمع المطالين.

إذا كانت الإشارات مختلفة نطرح الأصغر من الأكبر وتكون إشارة المجموع هي إشارة الأكبر.

الحل:

يمكن تنفيذ هذه الدارة إلى قسمين:

القسم الأول يقوم بترتيب الدخلين وفق قيمة المطال للحصول على قيمتين min وmax.

القسم الثاني يفحص الإشارات ويقوم بعملية الجمع أو الطرح تبعاً لنتيجة المقارنة.

يبين الرماز التالي بنية الوحدة المطلوبة:

```
module sign_mag_add
# (parameter N=4)
(input wire [N-1:0] a, b,
output reg [N-1:0]sum
);
 // signal declaration
reg [N-2:0] mag_a, mag_b , mag_sum, max, min;
reg sign_a, sign_b, sign_sum;
        / / body
        always @*
        begin
                // separate magnitude and sign
                mag_a = a[N-2:0];
                mag_b = b[N-2:0];
                sign_a = a[N-1];
                sign_b = b [N-1];
                // sort according to magnitude
                if (mag_a > mag_b)
                begin
                        max = mag_a;
                        min = mag_b;
                        sign_sum = sign_a;
                end
                else
                begin
                        max = mag_b;
                        min = mag_a;
                        sign_sum = sign_b;
                end
                // add/sub magnitude
                if (sign_a==sign_b)
                        mag_sum = max + min;
                else
                        mag_sum = max - min;
                 // form output
                sum = {sign_sum, mag_sum};
        end
endmodul
```

أسئلة خيارات متعددة:

1. ما هي العملية غير قابلة للتركيب في لغة Verilog من بين العمليات التالية

A. الجمع (+)

B. الطرح (−)

C. الضرب (*)

D. التقسيم (/)

2. بفرض أن a=8'b1111_1111 و b=8'b0000_0001. ماهي القيمة المخزنة في المتحول sum8 المعرف على 8 بتات بعد تنفيذ العملية (**assign** sum8 = (a+b+1)>>1;) ؟

A. 0000_0000

B. 1000_0000

C. 0000_0001

D. 1000_0001

3. ماذا يعني مفهوم قائمة التحسس في كتلة "دائماً" في لغة Verilog؟

A. يعني أنه يتم تفعيل الكتلة عند تغير أي إشارة من إشارات قائمة التحسس.

B. يعني أنه يتم تفعيل الكتلة عند تفعيل أي إشارة من إشارات قائمة التحسس.

C. يعني قائمة المتحولات المستخدمة في الكتلة.

D. يعني مجموعة إشارات الخرج للكتلة.

4. على ماذا يدل استعمال **always** @* في تصميم الدارات التراكبية في لغة Verilog؟

A. يدل على تضمين جميع المداخل في قائمة التحسس.

B. يدل على عدم وجود قائمة تحسس.

C. يدل على استعمال نفس قائمة التحسس لكتلة "دائماً" السابقة.

D. تدل على استعمال مداخل الوحدة التي تحتوي على الكتلة كقائمة تحسس.

5. ماذا تعني العبارة التالية (**assign** y=&a;) في لغة Verilog؟

A. تعني عملية NOT منطقية لإشارة a.

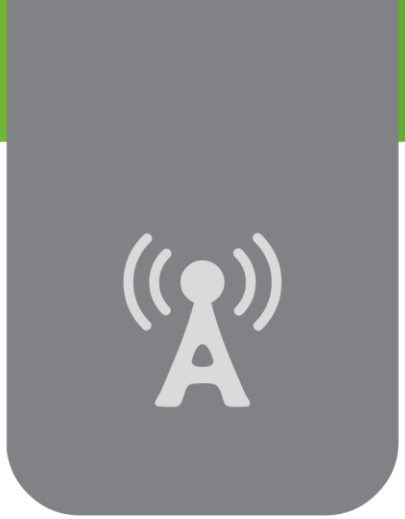
B. تعني عملية AND منطقية بين a و y ثم وضع الناتج في y.

C. تعني عملية AND منطقية بين جميع بتات الإشارة a وتخزين الناتج في y.

D. تعني عملية AND منطقية بين النصف الأدنى والنصف الأعلى من بتات الإشارة a.

الإجابات الصحيحة:

رقم التمرين	الإجابة الصحيحة
1	D
2	B
3	A
4	A
5	C



الفصل السادس:
الدارات التعاقبية في Verilog

الكلمات المفتاحية:

السجل، الدارات المتعاقبة المتزامنة, الدارات التعاقبية النظامية، آلة الحالة المنتهية.
Register, synchronous sequential circuit, regular sequential circuits, Finite-State Machine.

الملخص:

يهدف هذا الفصل إلى التعريف بكيفية تصميم الدارات التعاقبية في لغة Verilog. يشكل السجل المفصل الأساسي الذي يميز الدارات التعاقبية عن الدارات التراكبية. بعد التعريف بالقلابات والسجلات، نشرح البنية الشائعة للدارات التعاقبية المتزامنة وهي بنية آلة الحالة المنتهية. ونميز بين ثلاثة أنواع لهذه الدارات وهي الدارات التعاقبية النظامية وآلة الحالة المنتهية وآلة الحالة المنتهية مع معطيات. يتعرف الطالب أيضاً على الفرق بين آلة مور وآلة ميلي من خلال مثال تطبيقي.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- عمل القلابات المتزامنة والسجلات المتزامنة
- مبدأ تقدير التردد الأعظمي لعمل الدارة الرقمية
- أنواع الدارات التعاقبية وطريقة تنفيذه في لغة Verilog
- الفرق بين آلة مور وآلة ميلي في تنجيز دارات آلة الحالة المنتهية

المخطط:

يحتوي هذا الفصل على 4 وحدات تعليمية وهي:

1. مقدمة.
2. القلاب والسجل.
3. النظام المتزامن.
4. أنواع الدارات التعاقبية.
   - 1.4. الدارات التعاقبية النظامية.
   - 2.4. دارات آلة الحالة المنهية.
   - 3.4. دارات آلة الحالة المنتهية مع معطيات.

يهدف هذا الفصل إلى التعريف بكيفية تصميم الدارات التعاقبية في لغة Verilog. يشكل السجل المفصل الأساسي الذي يميز الدارات التعاقبية عن الدارات التراكبية. بعد التعريف بالقلابات والسجلات، نشرح البنية الشائعة للدارات التعاقبية المتزامنة وهي بنية آلة الحالة المنتهية. ونميز بين ثلاثة أنواع لهذه الدارات وهي الدارات التعاقبية النظامية وآلة الحالة المنتهية وآلة الحالة المنتهية مع معطيات. يتعرف الطالب أيضاً على الفرق بين آلة مور وآلة ميلي من خلال مثال تطبيقي.

1. مقدمة:

الدارة التعاقبية هي دارة منطقية تحتوي على ذاكرة، حيث يشكل محتوى هذه الذاكرة حالة الدارة الداخلية (internal state). وبخلاف الدارات التراكبية التي يكون فيها الخرج تابعاً للدخل فقط، يكون الخرج في الدارات التعاقبية تابعاً للدخل والحالة الداخلية للدارة أيضاً.

ومن المنهجيات الواسعة الاستعمال في الدارات التعاقبية هو منهجية التصميم المتزامن (synchronous design). في هذه المنهجية، يتم التحكم بجميع عناصر التخزين بواسطة ساعة مركزية. حيث يتم تخزين المعطيات على الجبهة الصاعدة أو الجبهة الهابطة للساعة. وهذا ما يسهل بشكل كبير تصميم النظم الكبيرة.

2. القلاب والسجل:

إن العنصر الأساسي المستعمل في التخزين في الدارات التعاقبية هو القلاب من نوع D (D flip–flop) أو DFF اختصاراً. يبين الشكل التالي رمز هذا العنصر وجدول الحقيقة الموافق لثلاثة أنواع من هذا القلاب.

clk	q*
0	q
1	q
↓	d

(a) DFF

reset	clk	q*
1	-	0
0	0	q
0	1	q
0	↓	d

(b) DFF with asynchronous reset

reset	clk	en	q*
1	-	-	0
0	0	-	q
0	1	-	q
0	↓	0	q
0	↓	1	d

(c) DFF with synchronous enable

في الشكل (a) نجد القلاب الأساسي الذي يخزن على الجبهة الصاعدة لإشارة الساعة clk.

وفي الشكل (b) نجد القلاب مع خط تصفير غير متزامن مع الساعة. أي أن قيمة الخرج تصبح صفراً مباشرة بعد تفعيل خط التصفير reset بغض النظر عن إشارة الساعة.

أما في الشكل (c) فنجد القلاب مع خط تفعيل en متزامن مع إشارة الساعة. أي لا يتغير الخرج إلا عند ورود الجبهة الصاعدة للساعة.

يوجد ثلاث معاملات زمنية أساسية لاستجابة القلاب وهي:

المعامل الأول هو زمن التأخير بين جبهة الساعة والخرج Tcq (clock–to–q). وهو الزمن الفاصل بين الجبهة الصاعدة للساعة وتغير الخرج للقيمة الجديدة. ولكي يعمل القلاب بشكل صحيح، يجب أن يكون الدخل d مستقراً عند حصول الجبهة الصاعدة للساعة.

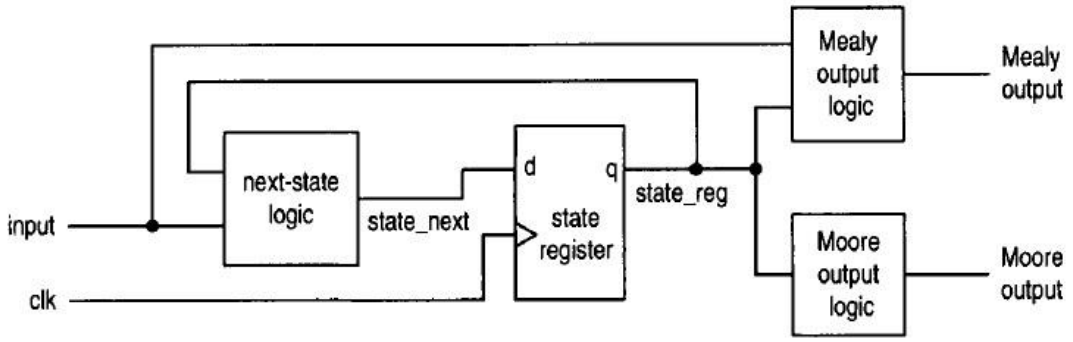
المعامل الثاني هو الزمن الأصغري لاستقرار قيمة الدخل قبل حصول الجبهة ونرمز له بالرمز Tsetup.

المعامل الثالث هو الزمن الأصغري لاستقرار قيمة الدخل بعد حصول الجبهة ونرمز له بالرمز Thold.

بتجميع عدد N من القلابات نحصل على سجل بطول N بت. وهو عنصر ضروري جداً في بناء الدارات التعاقبية.

3. النظام المتزامن:

يبين الشكل التالي المخطط الصندوقي لدارة تعاقبية متزامنة:



المخطط الصندوقي لدارة تعاقبية متزامنة

تتكون هذه الدارة من الأجزاء التالية:

- **سجل الحالة (state register):** وهو مجموعة من القلابات المحكومة بنفس الساعة.
- **دارات تحديد الحالة التالية (next state logic):** وهي دارات تراكبية تستعمل المداخل والحالة الداخلية للدارة (أي خرج سجل الحالة) لتحديد القيمة التالية لسجل الحالة.
- **دارات الخرج (output logic):** وهي دارات تراكبية لتوليد قيم الخرج للدارة. وهناك نوعان من دارات الخرج هما دارات مور (Moore) ودارات ميلي (Mealy) سنأتي على ذكرهما لاحقاً.

من الأمور الهامة في تصميم الدارات التعاقبية هو احترام أزمنة الاستقرار للدخل (Tsetup وThold). ولذلك يجب أن يكون زمن دور الساعة طويلاً بشكلٍ كافٍ لاحترام هذه الأزمنة وزمن الانتشار (Tcq). أي أن دور الساعة يجب أن يكون على الأقل مساوي للمقدار:

$$Tclock = Tcq + Tsetup + Thold$$

وهذا ما يقابل تردد أعظمي للساعة $f_{max}=1 / Tclock$ وذلك لكي يعمل السجل بشكل صحيح. وبالاعتماد على هذه المعاملات، نستطيع أيضاً تحديد أزمنة الانتشار في الدارة المصممة لتحديد التردد الأعظمي الذي تستطيع الدارة العمل عليه. عملياً، يتم حساب هذا التردد بشكل آلي في معظم برامج التصميم بمساعدة الحاسوب بعد تصميم الدارة.

4. أنواع الدارات التعاقبية:

بالنظر إلى المخطط الصندوقي للدارة التعاقبية المتزامنة والذي تم فيه فصل الذاكرة عن باقي أجزاء النظام نجد أنه فيما عدا السجل، تتكون الدارة التعاقبية من دارات تراكبية يمكن تصميمها كما رأينا في الفصل السابق. وتعد هذه المنهجية في التصميم (أي فصل الذاكرة عن بقية الدارات التراكبية) من الطرق التي توضح بنية الدارة الداخلية وتسهل عملية التصميم.

يمكن تقسيم الدارات التعاقبية إلى ثلاث فئات وفقاً لبنية دارة الحالة التالية وهي:

الدارات التعاقبية النظامية: وهي الدارات التي تتعاقب فيها الحالات بشكل منتظم مثل الزيادة بمقدار واحد دوماً. وفي هذه الحالات نستخدم عناصر أساسية لتنفيذ هذه الدارة مثل عداد أو سجل إزاحة.

دارات آلة الحالة المنتهية FSM: وهي الدارات التي لا تتعاقب فيها الحالات وفق نمط معين، أي بشكل شبه عشوائي. لذلك نستخدم آلة الحالة المنتهية لتحديد الحالة التالية.

دارات آلة الحالة المنتهية مع معطيات FSMD: وهي الدارات التي تشمل على عمليات حسابية متعاقبة بشكل محدد وفيها يتم استعمال مسار معطيات مع مسار تحكم يشمل على آلة حالة منتهية للتحكم بمسار المعطيات.

سنناقش هذه الحالات الثلاث في هذا الفصل من خلال أمثلة توضيحية.

1.4. الدارات التعاقبية النظامية:

نريد تصميم دارة عداد يعد من 0 إلى M−1 عند كل نبضة ساعة ومن ثم يعود إلى 0 ويكرر عملية العد باستمرار. نفرض أن M=10 في هذا المثال. يمتلك العداد مدخل تصفير reset غير متزامن. للعداد مخرج q مكون من أربعة بتات (N=4) تظهر عليه قيمة العداد، بالإضافة إلى مخرج نبضة max_tick بحيث يكون مساوياً الواحد عندما تكون قيمة العداد q= M−1 وصفر بخلاف ذلك. يسمى هذا العداد بالعداد الثنائي على M قيمة (mod−M binary counter).

يبين الرماز التالي توصيف وحدة العداد بلغة Verilog.

```
module mod_m_counter
# (parameter N=4,   // number of bits in counter
            M=10 // mod-M
  )
  (
      input wire clk, reset,
      output wire max_tick,
      output wire[N-1:0] q
  ) ;
      // signal declaration
      reg [N-1:0] r_reg;
      wire [N-1:0] r_next ;
// body
      // register
      always @ ( posedge clk , posedge reset )
            if (reset)  //equivalent to reset==1
                  r_reg <= 0 ;
            else
                  r_reg <= r-next;
      //next-state logic
      assign r_next = (r_reg==(M-1)) ? 0 : r_reg + 1;
      // output logic
      assign q = r_reg;
      assign  max_tick = (r_reg==(M-1)) ? l'b l : l'b0;
endmodule
```

عرَّفنا في هذه الوحدة معاملين للتصميم وهما M الذي يعبر عن القيمة العظمى للعداد وN الذي يعبر عن عدد بتات العداد والذي يجب أن يكون أكبر أو يساوي log2 (M). أعطينا القيم الافتراضية لهذين المعاملين M = 10 و N = 4.

أيضاً، عرّفنا متحولين: الأول هو r_reg من نمط **reg** لتخزين قيمة العداد، والثاني هو r_next الذي يعبر عن رقم الحالة التالية، والتي تعبر في نفس الوقت عن قيمة العداد التالية. حيث تأخذ الحالات هنا قيماً متعاقبة.

تشمل كتلة "دائماً" على عملية تخزين القيمة التالية للحالة عند الجبهة الصاعدة للساعة بشرط أن يكون خط التصفير غير مفعل. لذلك أدرجنا في قائمة التحسس لكتلة "دائماً" حدثين وهما الجبهة الصاعدة للساعة باستعمال الكلمة المفتاحية **posedge**، وكذلك الجبهة الصاعدة لخط التصفير reset. استعملنا داخل كتلة "دائماً" عبارة "إذا–وإلا" التي تختبر خط التصفير، فإذا كان مفعلاً نضع قيمة 0 في السجل، وإلا يأخذ السجل الحالة التالية ليخزنها.

لاحظ أننا استعملنا هنا إشارة الاسناد الإجرائي المؤجل (<=) الذي يستعمل في الدارات التعاقبية. وذلك يعني أن المتحول المسند إليه (r_reg) سيأخذ قيمته الجديدة في نهاية كتلة "دائماً".

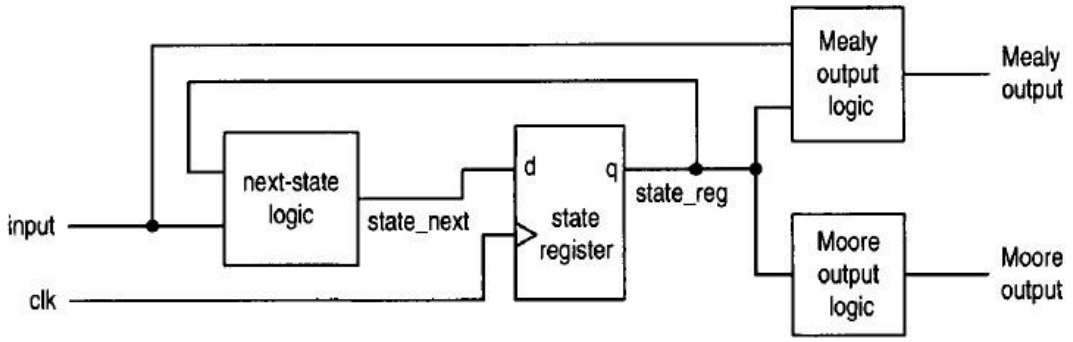
تحتوي دارة الحالة التالية (وهي دارة تراكبية) على عبارة شرطية (شرط؟ عبارة:عبارة) وهي مماثلة لعبارة "إذا–وإلا"، ولكنها مختصرة أكثر. نحصل على الحالة التالية بزيادة قيمة سجل الحالة بمقدار واحد إلا عند الحالة M−1 حيث يجب العودة للصفر.

وأخيراً، تتكون دارة الخرج من عملية إسناد قيمة سجل الحالة إلى الخرج q، ونضع الخرج max_tick=1 إذا كانت قيمة السجل مساوية M-1 وإلا يكون max_tick=0.

2.4. دارات آلة الحالة المنهية:

تُستعمل آلة الحالة المنتهية FSM لنمذجة نظام يتنقل بين عدد منتهي من الحالات الداخلية. ويعتمد الانتقال (أي رقم الحالة الجديدة) على الحالة الراهنة والمداخل الخارجية. وبخلاف الدارات التعاقبية النظامية، لا يتبع الانتقال بين الحالات نمطاً منتظماً.

يبين الشكل التالي المخطط الصندوقي لآلة حالة منتهية. وهي تتكون من سجل حالة ودارة الحالة التالية ودارة الخرج.



المخطط الصندوقي لآلة حالة منتهية متزامنة

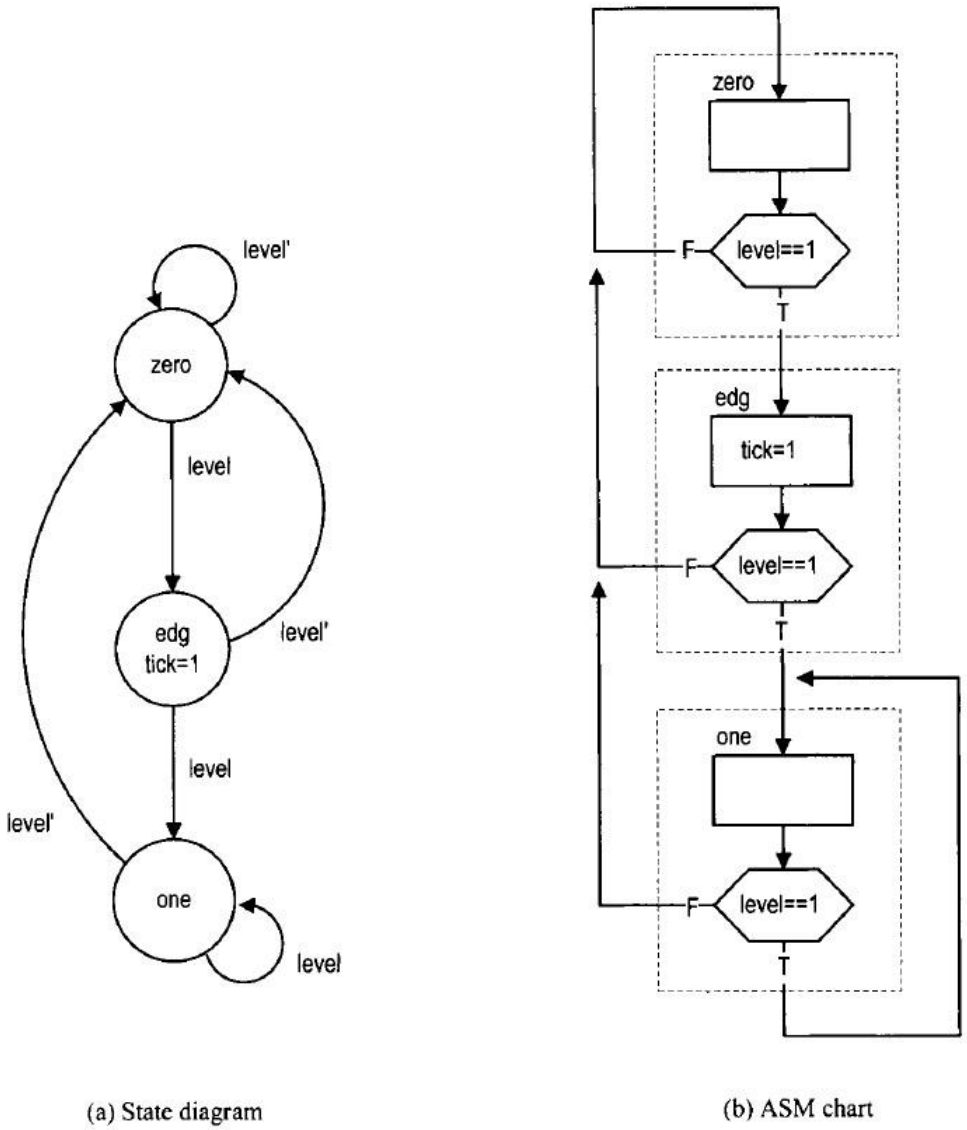
نقول عن آلة الحالة المنتهية بأنها آلة "مور" (Moore) إذا كان الخرج تابعاً للحالة الراهنة فقط. ونقول عنها بأنها آلة "ميلي" (Mealy) إذا كان الخرج تابعاً للحالة الراهنة وإشارات الدخل الخارجية أيضاً. يمكن أن نجد كلا النوعين من المخارج في آلة حالة مركبة. هناك تشابه بين مخارج آلة ميلي وآلة مور، ولكنهما غير متماثلين تماماً. سنوضح هذا الاختلاف الدقيق من خلال المثال التالي.

مثال: كاشف الجبهة الصاعدة:

دارة كاشف الجبهة الصاعدة هي دارة تولد نبضة واحدة بطول دور ساعة واحد عندما تتغير إشارة الدخل من 0 إلى 1. وتستعمل عادة للكشف عن بداية تفعيل إشارة تتغير ببطء. فيما يلي، سنعرض تصميم هذه الدارة باستعمال كل من آلة مور وآلة ميلي وسنقارن بينهما.

التصميم باستعمال آلة مور:

يبين الشكل التالي مخطط الحالة، والمخطط التدفقي للتصميم باستعمال آلة مور.



(a) State diagram (b) ASM chart

كاشف الجبهة الصاعدة باستعمال آلة مور

في مخطط الحالة، تشير الحالتين zero و one إلى أن إشارة الدخل level مستقرة على قيمة 0 أو 1 على الترتيب. عند انتقال الدخل من 0 إلى 1، تنتقل الآلة إلى حالة edg وفيها يتم تفعيل نبضة الخرج tick. يعبر

```
module edge_detect_moore (
        input wire clk, reset,
        input wire level,
        output reg tick
);
// symbolic state declaration
localparam [1:0]
        zero = 2'b00,
        edg  = 2'b01,
        one  = 2'bl0;
// signal declaration
reg [1:0] state_reg , state_next ;
        // state register
        always @ ( posedge clk , posedge reset)
          if (reset)
                state_reg <= zero;
          else
                state_reg <= state_next;
        //next state logic and output logic
        always @*
        begin
                state_next = state_reg; // default state: the same
                tick = 1'b0; // default output: 0
                case (state_reg)
                  zero:
                        if ( level)
                          state_next = edg ;
                  edg :
                        begin
                          t ick = l'bl ;
                          if (level)
                            state_next = one ;
                          else
                            state_next = zero;
                        end
                  one :
                        if (~level)
                          state_next = zero ;
                  default : state_next = zero;
                endcase
        end
endmodule
```

المخطط التدفقي عن عمل الدارة بشكل آخر ولكن بشكل مكافئ تماماً لمخطط الحالة. يسمى هذا المخطط التدفقي بمخطط حالة الآلة الخوارزمية ASM (Algorithmic State Machine). يبين الرماز التالي طريقة توصيف هذه الدارة.

في هذه الوحدة، عرفنا ثلاث حالات وهي zeros, edg, one وأعطيناها الأرقام 00, 01, 10 بالصيغة الثنائية باستعمال الكلمة المفتاحية localparam.

استعملنا كتلتي "دائماً": الأولى لتنفيذ دارة سجل الحالة والثانية لتنفيذ دارة الحالة التالية ودارة الخرج. تسهل عملية الدمج هذه تصميم دارة الحالة التالية ودارة الخرج ضمن تنفيذ آلة الحالة المنتهية FSM. استعملنا عبارة "في حال" لتنفيذ آلة الحالة المنتهية وذلك لتحديد قيمة الحالة التالية وقيمة الخرج في كل حالة. في البداية تم وضع القيم الافتراضية لهذه القيم لاستعمالها في حال لم يتم اسنادهما بشكل صريح في أي حالة من الحالات.

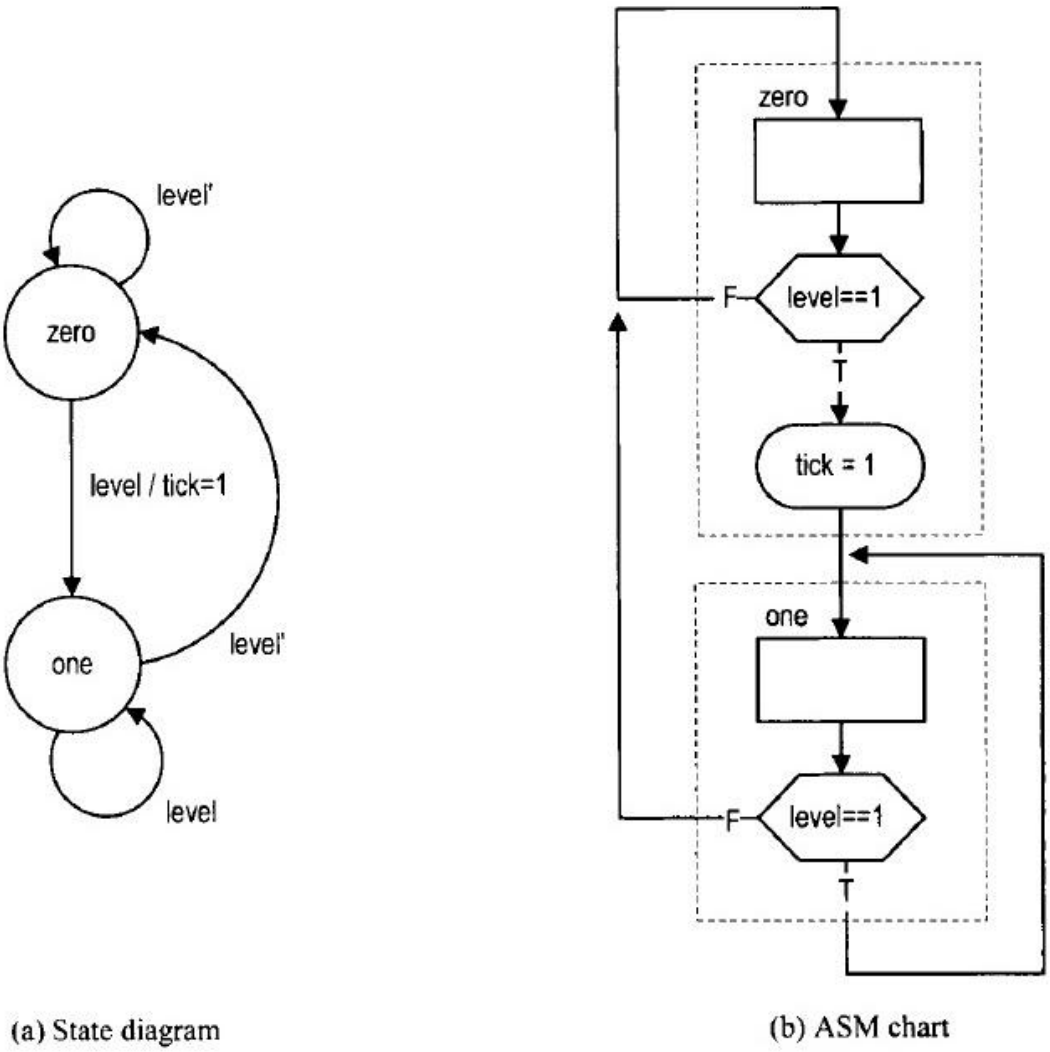
في الحالة zero وحسب مخطط الحالة، تكون الحالة التالية هي edg إذا أصبحت قيمة الإشارة level=1. وإلا فإن الحالة تبقى على الحالة الراهنة كما هو محدد ضمن قيمة الاسناد الافتراضية للمتحول state_next. لم يتم تحديد قيمة الخرج في هذه الحالة، لذلك تكون قيمة الخرج هي القيمة الافتراضية أي tick=0.

بنفس الطريقة تم توصيف الحالة edg والحالة one. بالإضافة إلى ذلك، تم إضافة الحالة default وهي تشمل قيم الحالة state_reg = 2'b11 وهي لا يمكن أن تحدث بشكل طبيعي ولكن إذا حدث وأصبحت قيمة سجل الحالة 11 بالخطأ فإننا نعود إلى الحالة الأولى zero وذلك منعاً لبقاء آلة الحالة في حالة غير متوقعة وعدم القدرة على الخروج منها.

تعتبر هذه الطريقة من الأمور الأساسية في تصميم البنية العتادية بلغة التوصيف لتجنب الوقوع في الأخطاء. حيث يجب دائماً تغطية جميع الحالات الممكنة ولو كانت غير نظامية. فمن أسباب حدوث خطأ في قيمة سجل الحالة وجود ضجيج كهربائي على مداخل سجل الحالة أو بسبب تأخير الإشارات بسبب الطول المختلف لأسلاك البتات وخصوصاً عند العمل بسرعات كبيرة وغيرها من الأسباب التي لا داعي للخوض فيها هنا.

التصميم باستعمال آلة ميلي:

يبين الشكل التالي مخطط الحالة، والمخطط التدفقي للتصميم باستعمال آلة ميلي.



كاشف الجبهة الصاعدة باستعمال آلة ميلي

استعملنا هنا حالتين فقط هما one و zero، وذلك لأن مخارج آلة ميلي يتم وضعها مباشرة عند الانتقالات (من الحالة zero إلى الحالة one وبالعكس) وليس ضمن الحالات وبالتالي ليس هناك من ضرورة لإنشاء حالة لوضع قيمة نبضة الخرج كما في حالة آلة مور. أي أن الخرج سيتغير فوراً عند تغير قيمة الدخل level وليس عندما تصبح الدارة في الحالة الجديدة عن نبضة الساعة التالية.

تجدر الإشارة هنا إلى أن قيمة إشارة الخرج الافتراضية هي tick = 0 إذا لم يتم ذكرها على وصلات الانتقال.

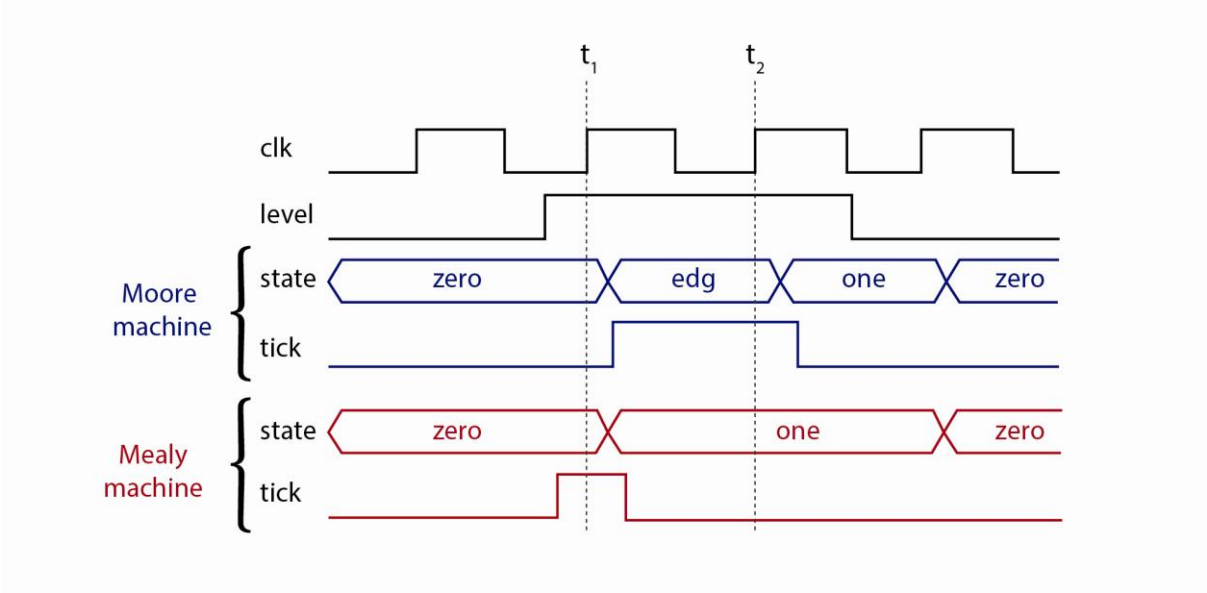
عند انتقال الإشارة من قيمة 0 إلى 1، يتم مباشرة تفعيل النبضة tick. وعند الجبهة الصاعدة التالية للساعة يتم إزالة تفعيل النبضة سواء بقيت الدارة في نفس الحالة أو تغيرت.

يبين الرماز التالي طريقة توصيف هذه الدارة.

```
module edge_detect_mealy
(
        input wire clk, reset,
        input wire level,
        output reg tick
);

// symbolic state declaration
localparam
        zero = 1'b0,
        one  = 1'bl;

// signal declaration
reg state_reg , state_next ;
        // state register
        always @ ( posedge clk , posedge reset)
          if (reset)
                state_reg <= zero;
          else
                state_reg <= state_next;
        //next state logic and output logic
        always @*
        begin
                state_next = state_reg; // default state: the same
                tick = 1'b0; // default output: 0
                case (state_reg)
                  zero:
                        if ( level)
                        begin
                          tick =1'b1;
                          state_next = one ;
                        end

                  one :
                        if (~level)
                          state_next = zero ;
                  default : state_next = zero;
                endcase
        end
endmodule
```

مقارنة:

يبين المخطط الزمني التالي، تغيرات الحالة وإشارة الخرج تبعاً لتغيرات إشارة الدخل وإشارة الساعة في كل من التصميمين السابقين لكاشف الجبهة.



المخطط الزمني لعمل كاشفي الجبهة الصاعدة

من الشكل نلاحظ بأن كل من التصميمين يعطي نبضة عن الجبهة الصاعدة لإشارة الدخل ولكن يكمن الاختلاف في النقاط التالي:

- تصميم دارة ميلي يحتوي على حالات أقل وبالتالي فهو أبسط من ناحية التعقيد.
- دارة ميلي أسرع في اخراج النبضة ولكن عرض النبضة متغير. حيث يظهر الخرج في آلة مور بعد دورة ساعة واحدة ولكن بعرض ثابت بمقدار دور ساعة.
- تمرر دارة ميلي النبضات الضيقة في الدخل (glitches) إلى الخرج. ففي حال انتقل الدخل من 0 إلى 1 ثم عاد سريعاً إلى 0، فإن آلة ميلي تخرج نبضة بينما لا يحدث ذلك في آلة مور إلا حصل هذا الانتقال في لحظة الجبهة الصاعدة للساعة.

وبالتالي، يعتمد الاختيار بين التصميمين على الهدف من إشارة الخرج والدارة التالية التي ستذهب إليها هذه الإشارة.

3.4. دارات آلة الحالة المنتهية مع معطيات:

تستعمل آلة الحالية المنتهية في التطبيقات العملية لتصميم المتحكم في نظام رقمي كبير. حيث يتم مراقبة إشارات الأوامر الخارجية والحالة لتوليد الإشارات التحكمية المناسبة للتحكم بمسار المعطيات المكون عادةً من دارات تعاقبية نظامية. تعرف هذه الدارات بدارات آلة الحالة المنتهية مع معطيات FSMD.

يشبه توصيف آلة الحالة المنتهية مع معطيات بلغة Verilog طريقة توصيف آلة الحالة المنتهية ولكن يتم إضافة العبارات الحسابية على المعطيات عند تنفيذ آلة الحالة.

يمكن للمهتمين العودة لمراجع هذه المادة من أجل دراسة بعض الأمثلة على ذلك ومزيد من التفاصيل حول التقنيات المستخدمة والتي هي خارج إطار هذه المادة.

5. خلاصة:

الدارة التعاقبية هي دارة منطقية تحتوي على ذاكرة، حيث يشكل محتوى هذه الذاكرة حالة الدارة الداخلية. نستخدم القلابات المتزامنة لتحقيق الذواكر حيث يتكون السجل من مجموعة من القلابات المحكومة بنفس إشارة الساعة. تعتمد سرعة الدارة المصممة على أزمنة الانتشار داخل سجلات الدارة. يؤول تصميم الدارات التعاقبية إلى تصميم دارات تراكبية بالإضافة إلى دارة السجلات لتكوين ما يسمى بآلة الحالة المنتهية مع أو بدون معطيات. حيث يمكن استعمال العبارات الشرطية في لغة Verilog لتحديد الحالة الجديدة للدارة. كما نستعمل دارات تراكبية لتحديد قيمة إشارات الخرج انطلاقاً من الحالة الراهنة للدارة وقيم إشارات الدخل. تتميز آلة ميلي عن آلة مور بأنها اقل تعقيداً وذات استجابة أسرع إلا أنها تعاني من عدم انتظام استجابتها وإمكانية تمريرها للإشارات النبضية القصيرة في الدخل إلى الخرج. ويعتمد الاختيار بينهما على التطبيق.

6. أسئلة الفصل السادس:

أسئلة عامة:

1. ما هي الدارة المنطقية التعاقبية؟

الدارة التعاقبية هي دارة منطقية تحتوي على ذاكرة. ويكون الخرج فيها تابعاً للدخل والحالة الداخلية للدارة أيضاً.

2. ما هي العناصر المنطقية التي تكون السجل على N بت في بنيته الداخلية؟

يتكون السجل من مجموعة مكونة من N قلاب وتكون محكومة بنفس إشارة الساعة.

3. ماهي الأقسام الرئيسية الثلاثة للدارة التعاقبية المتزامنة؟

الأقسام هي:

- سجل الحالة
- دارات تحديد الحالة التالية
- دارات الخرج

4. ما هو التردد الأعظمي لعمل سجل ذو المواصفات الزمنية التالية: Tcq=2 ns و Tsetup=2ns و Thold=1 ns؟

دور الساعة الأصغري يعطي بالعلاقة

$$Tclock = Tcq + Tsetup + Thold$$

أي:

$$2 + 2 + 1 = 5ns = Tclock$$

وبالتالي يكون التردد الأعظمي هو:

$$f_{max} = 1/Tclock = 1/5ns = 200\ MHz$$

5. ماذا تعني العبارة التالية في لغة verilog؟

```
assign r_next = (r_reg==(M-1)) ? 0: r_reg + 1;
```

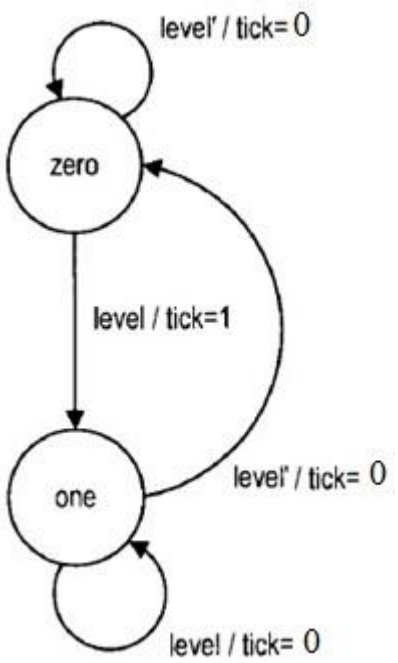
تعني اسناد قيمة 0 إلى المتحول r_next إذا كان الشرط (M-1)==r_reg محققاً وإلا يتم اسناد قيمة r_reg + 1 إلى المتحول r_next.

6. اشرح معنى التوصيف التالي في لغة Verilog.

```
always @ (posedge clk, posedge reset )
if (reset) r_reg <= 0 ;
else r_reg <= r–next;
```

نستعمل هنا كتلة "دائماً" بقائمة تحسس مكونة من حدثين وهما الجبهة الصاعدة للساعة باستعمال الكلمة المفتاحية **posedge**، وكذلك الجبهة الصاعدة لخط التصفير reset.
استعملنا داخل كتلة "دائماً" عبارة "إذا–وإلا" التي تختبر خط التصفير، فإذا كان مفعلاً نضع قيمة 0 في السجل، وإلا يأخذ السجل الحالة التالية r-next ليخزنها.

7. ارسم مخطط الحالة لدارة كشف الجبهة الصاعدة للإشارة level وذلك فق آلة ميلي والخرج هو الإشارة tick.



أسئلة خيارات متعددة:

1. في المواصفات الزمنية للسجل، يرمز الزمن Tcq إلى:

A. زمن تردد الساعة الأصغري لكي يعمل السجل بشكل صحيح.

B. الزمن الفاصل بين الجبهة الصاعدة للساعة وتغير الخرج للقيمة الجديدة.

C. الزمن الأصغري لاستقرار قيمة الدخل قبل حصول الجبهة.

D. الزمن الأصغري لاستقرار قيمة الدخل بعد حصول الجبهة.

2. في لغة Verilog، لماذا تستعمل الكلمة المفتاحية **posedge**؟

A. تستعمل لإضافة الجبهة الصاعدة للإشارة في قائمة التحسس لكتلة "دائماً".

B. تستعمل في العبارة الشرطية "إذا–وإلا".

C. تستعمل في العبارة الشرطية "في حال".

D. تستعمل لتحديد نوع السجل المستعمل في الدارة التعاقبية.

3. ما ذا يعني خط تصفير reset غير متزامن للسجل؟

A. يعني أنه صادر من إشارة تراكبية لتصفير السجل على الجبهة الصاعدة للساعة.

B. يعني أنه يأخذ مفعوله عندما تكون الساعة في السوية المنخفضة.

C. يعني أن تردد إشارة هذا الخط تختلف عن تردد إشارة الساعة.

D. يعني أنه يقوم بتصفير السجل فوراً بغض النظر عن الساعة.

4. في حال لم يتم تحديد قيمة متحول بشكل صريح في عبارة "في حال". ماذا تكون قيمة المتحول؟

A. تكون قيمته صفراً.

B. يحافظ على قيمته السابقة.

C. القيمة الافتراضية المحددة ضمن الكتلة.

D. قيمة عشوائية غير محددة.

5. تتميز آلة ميلي عن آلة مور بالتالي:

A. تصميم دارة ميلي يحتوي على حالات أقل.

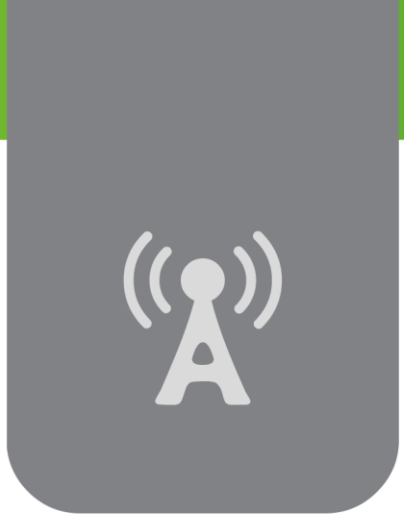
B. تصميم دارة ميلي أبسط من ناحية التعقيد.

C. دارة ميلي أسرع في اخراج قيمة الخرج.

D. كل ما سبق

الإجابات الصحيحة:

رقم التمرين	الإجابة الصحيحة
1	B
2	A
3	D
4	C
5	D



الفصل السابع:
المعالجات ذات الأغراض العامة

الكلمات المفتاحية:

المعالجات ذات الأغراض العامة، المتحكمات الصغرية، معالجات الإشارة الرقمية، التواردية، فضاء الذاكرة.
General Purpose Processors, Microcontrollers, Digital Signal Processors, Pipelining, Memory space.

الملخص:

يهدف هذا الفصل إلى التعرف على بنية المعالجات ذات الأغراض العامة والفائدة من استعمالها في النظم المضمنة. وهو يعتبر مراجعة سريعة لمادة المعالجات والمتحكمات الصغرية مع توضيح خصائص استخدام هذه المعالجات في النظم المضمنة والأمور التي يجب على المصمم التركيز عليها بشكل أكبر والقيود المفروضة عند تطوير برنامج لنظام مضمن. نتعرف في هذا الفصل على البنية العامة للمعالجات العامة ومكوناتها الداخلية. وأخيراً نتعرف على مفهوم المعالجات الموجهة لتطبيقات محددة والتي تعتبر الأكثر شيوعاً في تصميم النظم المضمنة مثل المتحكمات الصغرية ومعالجات الإشارة الرقمية.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- البنية الأساسية للمعالجات ذات الأغراض العامة
- المعارف التي يجب الالمام بها لبرمجة المعالجات في النظم المضمنة
- خصائص المعالجات الموجهة لتطبيقات محددة

يهدف هذا الفصل إلى التعرف على بنية المعالجات ذات الأغراض العامة والفائدة من استعمالها في النظم المضمنة. وهو يعتبر مراجعة سريعة لمادة المعالجات والمتحكمات الصغرية مع توضيح خصائص استخدام هذه المعالجات في النظم المضمنة والأمور التي يجب على المصمم التركيز عليها بشكل أكبر والقيود المفروضة عند تطوير برنامج لنظام مضمن. نتعرف في هذا الفصل على البنية العامة للمعالجات العامة ومكوناتها الداخلية. وأخيراً نتعرف على مفهوم المعالجات الموجهة لتطبيقات محددة والتي تعتبر الأكثر شيوعاً في تصميم النظم المضمنة مثل المتحكمات الصغرية ومعالجات الإشارة الرقمية.

1. مقدمة:

المعالج ذو الأغراض العامة (General-Purpose Processor) أو اختصاراً المعالج العام هو نظام رقمي قابل للبرمجة ومُعد لتنفيذ المسائل الحسابية في الكثير من التطبيقات وفي العديد من المجالات مثل الاتصالات والأتمتة والنظم المضمّنة الصناعية. لاستعمال المعالجات العامة في النظم المضمّنة الصناعية العديد من الفوائد مثل الكلفة الهندسية المنخفضة بسبب الإنتاج الكمي للمعالجات العامة الذي ينعكس على السعر الإفرادي للقطعة. تقتصر الكلفة الهندسية في تصميم النظام على كتابة البرنامج دون الحاجة إلى تصميم معالجات خاصة وتصنيعها، وبالتالي يكون زمن دخول النظام إلى السوق أقصر. إضافة إلى ذلك، تتمتع النظم التي تعتمد على معالجات عامة بالمرونة في تغيير البرنامج وإجراء التحديثات اللاحقة.

سنتعرف في هذا الفصل على المكونات الشائعة للمعالجات العامة وأنواعها وكيفية عملها.

2. البنية الأساسية:

يتكون المعالج العام، والذي نطلق عليه أحياناً اسم وحدة المعالجة المركزية CPU (Central Processing Unit)، من مسار المعطيات (Data Path) ووحدة التحكم (Control Unit) إضافة إلى ذاكرة.
سنناقش فيما يلي هذه المكونات باختصار.

1.2. مسار المعطيات:

يتكون مسار المعطيات (Data Path) من الدارات التي تعالج المعطيات وتخزنها. حيث يحتوي مسار المعطيات على وحدة الحساب والمنطق ALU (Arithmetic –Logic Unit) القادرة على معالجة المعطيات عن طريق تنفيذ عمليات متعددة مثل الجمع والطرح والإزاحة والعمليات المنطقية. تولد هذه الوحدة أيضاً العديد من إشارات الحالة التي تُخزن عادة في سجل يسمى سجل الحالة (Status Register) وتسمى كل إشارة مخزنة راية (flag) وهي تدل على تحقق شروط معينة في المعطيات التي تعالجها. مثلاً، هناك راية للدلالة على ما إذا كان ناتج العملية مساوياً الصفر، وهناك راية تدل على وجود حمل ناتج عن جمع رقمين.
يحتوي مسار المعطيات أيضاً على سجلات لتخزين المعطيات المؤقتة مثل المعطيات المقروءة من الذاكرة والتي لم تذهب بعد إلى وحدة الحساب والمنطق، أو المعطيات الناتجة عن وحدة الحساب والمنطق والتي نحتاجها في عمليات حسابية لاحقة أو تخزينها في الذاكرة.
يتم نقل المعطيات ضمن مسار المعطيات باستعمال مساري معطيات (Data Bus) داخلية، بينما يتم نقل المعطيات من وإلى الذاكرة باستعمال مساري معطيات خارجية.
غالباً ما يتم تمييز المعالجات من خلال عرض عناصر مسار المعطيات بوحدة البت، فمثلاً نقول عن معالج أنه على 8 بت إذا كانت وحدة الحساب والمنطق تتعامل مع معطيات على 8 بت، وكذلك يكون عرض المسرى الداخلي وسجلات العمل الداخلية على 8 بت أيضاً.
من القياسات الشائعة لعرض مسرى المعطيات نجد معالجات على 8 أو 16 أو 32 أو 64 بتاً، ومع ذلك فقد نجد معالجات تحتوي على سجلات بقياسات مختلفة، أو يكون عرض مسرى المعطيات الخارجي مختلفاً عن عرض مسرى المعطيات الداخلي، فمثلاً يمكن أن نجد معالجاً على 16 بتاً يحتوي على مسار معطيات خارجي على 8 بت فقط. لذلك لا يُعد هذا التعريف دقيقاً لقياس المعالج.

2.2. وحدة التحكم:

تتكون وحدة التحكم (Control Unit) من الدارات التي تجلب تعليمات البرنامج من الذاكرة، وتسير المعطيات ضمن مسار المعطيات وفقاً لهذه التعليمات.
تحتوي وحدة التحكم على سجل يسمى عداد البرنامج PC (Program Counter) يخزن عنوان الذاكرة للتعليمة التالية. كذلك تحتوي على سجل آخر يسمى سجل التعليمة IR (Instruction Register) يخزن التعليمة الحالية المجلوبة من ذاكرة البرنامج. إضافة إلى ذلك، تحتوي وحدة التحكم على الدارات المنطقية التي

تولد إشارات التحكم بمسار المعطيات بشكل مشابه لما سبق ذكره في تصميم المعالجات الخاصة في الفصل الثاني.

يعبّر عرض السجل PC عن عرض مسرى العناوين الذي يحدد عدد مواقع الذاكرة التي يمكن النفاذ إليها بشكل مباشر. نسمي هذا العدد فضاء العناوين (Address Space) أو فضاء الذاكرة (Memory Space)، فمثلاً إذا كان عرض السجل PC يساوي 16 يكون فضاء الذاكرة مساوياً 2^{16}= 65.536 أي 64K (1K=1024).

تُعتبر وحدة التحكم مسؤولة عن إدارة عملية تنفيذ البرنامج وتجهيز السجل PC بعنوان التعليمة التالية وتنفيذ التعليمات. تدير وحدة التحكم تنفيذ كل تعليمة من خلال عدة مراحل متسلسلة تشمل جلب التعليمة من الذاكرة وتفسيرها (Decode) وجلب معاملات التعليمة (Fetch Operands) إن وجدت وتنفيذ التعليمة في مسار المعطيات وتخزين النتائج. تشمل كل مرحلة من المراحل على نبضة أو أكثر من نبضات ساعة المعالج، وبالتالي تحتاج التعليمة لعدة نبضات ساعة كي تنفّذ، وتعتمد سرعة المعالج على تردد ساعة العمل وعدد النبضات اللازمة لتنفيذ التعليمات والذي يختلف من معالج لآخر. لذلك فإن المقارنة بين سرعة المعالجات بالاعتماد على تردد ساعة العمل فقط هو أمر غير دقيق.

3.2. الذاكرة:

في الوقت الذي نستعمل فيه السجلات لتخزين المعطيات على الأمد القصير نستعمل الذاكرة لتخزين المعطيات على الأمد المتوسط والطويل.

يمكن تصنيف المعطيات المخزنة في الذاكرة إلى صنفين: يتكون الصنف الأول من تعليمات البرنامج (Program) ويتكون القسم الثاني من المعطيات التي يقوم البرنامج بمعالجتها (Data). يمكن تخزين هذين الصنفين (البرنامج والمعطيات) معاً ضمن الذاكرة نفسها أو في ذاكرتين منفصلتين.

في البنية المسماة Princeton Architecture (أو أيضاً Von Neumann) يخزن البرنامج والمعطيات في فضاء الذاكرة نفسه، بينما في البنية المسماة Harvard Architecture يخزن البرنامج في فضاء مستقل عن فضاء ذاكرة المعطيات كما هو مبين في الشكل X.

تتميز البنية الأولى ببساطتها لأنها تتطلب مسرى واحداً مع الذاكرة، بينما نحتاج في البنية الثانية إلى مسريين، ولكنها تكون أفضل من حيث السرعة بسبب إمكانية إجراء عمليات القراءة والتخزين في ذاكرة المعطيات في الوقت نفسه الذي يتم فيه جلب التعليمات من ذاكرة البرنامج.

يمكن أن تكون الذاكرة من نوع ذاكرة قابلة للقراءة فقط ROM (Read Only Memory) وهي تستخدم غالباً لتخزين البرنامج لأن البرنامج نفسه لا يتغير في النظم المضمّنة مقارنة مع النظم الحاسوبية المكتبية، كما يمكن أن تكون الذاكرة من نوع الذاكرة الحية RAM (Random Access Memory) الذي يستعمل لتخزين المعطيات. يمكن أن تكون الذاكرة مضمّنة مع المعالج على الشريحة نفسها (On Chip) ولكنها تكون بحجم محدود أو يمكن أن تكون منفصلة على شريحة مستقلة (Off Chip) وعندها يمكن أن نختار حجمها كما نريد.

غالباً ما تكون الذاكرة الداخلية من النوع السريع SRAM (Static RAM) وهي غالية الثمن، بينما تكون الذاكرة الخارجية من النوع البطيء DRAM (Dynamic RAM) وهي منخفضة الكلفة، لذلك يكون زمن نفاذ المعالج إلى الذاكرة الداخلية المضمّنة في الشريحة نفسها أقصر من زمن النفاذ إلى الذاكرة الخارجية. لتقليص زمن النفاذ إلى الذاكرة الخارجية يتم تخصيص جزء من الذاكرة الداخلية كذاكرة مخبأة (Cache Memory).

يعتمد عمل الذاكرة المخبأة على المبدأ التالي: عندما يقوم المعالج في لحظة ما بالنفاذ إلى موقع معين في الذاكرة فإنه من المرجح أن يقوم المعالج بالنفاذ إلى المواقع المجاورة لهذا الموقع في اللحظات التالية، لذلك يتم نسخ كتلة كاملة من الذاكرة الخارجية حول الموقع الحالي إلى الذاكرة المخبأة، وبالتالي يكون زمن النفاذ إلى المواقع المجاورة قصيراً في التعليمات التالية.

عندما يحتاج المعالج للنفاذ إلى الذاكرة فإنه ينظر أولاً إذا ما كان هناك نسخة من هذا الموقع موجودة في الذاكرة المخبأة من خلال البحث في جدول الذاكرة المخبأة. في هذه الحالة نقول أن لدينا إصابة ذاكرة (Cache Hit) وإلا فإننا نحتاج إلى جلب كتلة جديدة حول الموقع المرغوب من الذاكرة الخارجية إلى الذاكرة المخبأة وهذا يأخذ وقتاً كبيراً بالطبع. في هذه الحالة نقول إنه لدينا فقد ذاكرة (Cache Miss).

لذلك تكون فعالية الذاكرة المخبأة أكبر في تسريع عملية النفاذ كلما زادت نسبة عدد مرات الإصابة إلى عدد مرات الفقد، وهذا يتطلب وضع خوارزميات إدارة ذكية للذاكرة المخبأة. من الجدير بالذكر أنه يمكن استعمال ذاكرة مخبأة للبرنامج وذاكرة مخبأة للمعطيات.

3. عمل المعالج:

1.3. تنفيذ التعليمة:

يتم تنفيذ كل تعليمة في المعالج على عدة مراحل:

1. جلب التعليمة: وهي مهمة قراءة التعليمة التالية من ذاكرة البرنامج إلى سجل التعليمة IP.
2. تفسير التعليمة: وهي مهمة تحديد العملية التي تمثلها التعليمة الموجودة في سجل التعليمة.
3. جلب المعاملات: وهي مهمة نقل معاملات التعليمة إن وجدت إلى السجلات المناسبة.
4. تنفيذ التعليمة: وهي مهمة إجراء العملية على المعاملات ضمن وحدة الحساب والمنطق ALU وتخزين النتيجة في السجل المناسب.
5. تخزين النتائج: وهي مهمة كتابة سجل النتيجة في الذاكرة.

وهكذا نجد أن تنفيذ كل تعليمة قد يأخذ عدة دورات للساعة، فمثلاً إذا كانت كل مرحلة تتطلب دورة ساعة واحدة فإنه يلزمنا 5 نبضات ساعة لتنفيذ التعليمة بشكل كامل.

2.3. التواردية:

التواردية (Pipelining) هي طريقة لزيادة معدل تنفيذ التعليمات في المعالج، فعوضاً عن الانتظار حتى إنهاء المراحل الخمسة لتنفيذ التعليمة ثم البدء بتنفيذ التعليمة التالية، يمكن تخصيص وحدة عتادية لتنفيذ كل مرحلة من المراحل، حيث تقوم الوحدة الأولى بجلب التعليمة وتسليمها لوحدة التفسير. في الوقت الذي تقوم فيه الوحدة الثانية بتفسير التعليمة الأولى تقوم وحدة الجلب بجلب التعليمة الثانية وهكذا. تسمى طريقة تنظيم العمل هذه لتنفيذ التعليمات بالتواردية. فإذا كان تنفيذ كل مرحلة يتطلب زمناً دوره ساعة واحدة فإن زمن تنفيذ كل تعليمة لا يزال مساوياً 5 نبضات ساعة، ولكن هنا نأخذ تعليمة جديدة عند كل نبضة ساعة، وبالتالي يكون معدل تنفيذ التعليمات هو تعليمة واحدة في كل نبضة ساعة بدلاً من تعليمة كل 5 نبضات ساعة.

لكي تعمل طريقة التواردية بشكل صحيح، لا بد أن يكون زمن تنفيذ المراحل المختلفة متساوياً. إضافة إلى ذلك، تشكل عمليات القفز ضمن البرنامج مشكلة بالنسبة للتواردية، إذ أننا لا نعلم بعملية القفز إلا عند الوصول إلى مرحلة التنفيذ. في هذا الوقت تكون التعليمات التالية لتعليمة القفز قد دخلت في خط التواردية، وبالتالي يجب تجاهل هذه التعليمات والبدء من جديد بمرحلة جلب التعليمة الأولى من المكان الجديد الذي سيقفز إليه البرنامج. من الجدير بالذكر أن هناك طرقاً متقدمة تقوم على مبدأ التنبؤ بالمكان الذي سيتم القفز إليه وذلك بهدف استثمار الوقت بشكل أفضل عند وجود عمليات قفز ضمن البرنامج.

3.3. المعالجات المتعددة النوى:

لتسريع عمل المعالج يمكن استعمال أكثر من وحدة حساب ومنطق في المعالج الواحد، بحيث تعمل هذه الوحدات معاً على التوازي لتنفيذ عدة تعليمات في آن واحد.

المعالج المتعدد النوى (Superscalar Microprocessor) هو معالج يحتوي على عدة وحدات حساب ومنطق، وفي هذا المعالج يتم في مرحلة الجلب قراءة عدة تعليمات معاً، وتكون مجهزة سلفاً عند ترجمة البرنامج. تشكل مجموعة التعليمات هذه ما يسمى كلمة تعليمة طويلة جداً VLIW (Very Long Instruction Word).

4. برمجة المعالج:

يكتب المبرمج تعليمات البرنامج الذي يحقق العمل المطلوب المعالج العام. في الحقيقة، لا يحتاج المبرمج أن يكون لديه معرفة تفصيلية ببنية المعالج، ولكن يتعامل مع المعالج كبنية مجردة تخفي الكثير من التفاصيل، وتعتمد سوية التجريد على سوية لغة البرمجة. يمكن التمييز بين سويّتين من البرمجة: الأولى هي البرمجة بلغة التجميع (Assembly) والثانية هي البرمجة بلغة عالية المستوى كلغة C مثلاً.

يستعمل المبرمج عند كتابة البرامج بلغة التجميع تعليمات خاصة بالمعالج، بينما يستعمل عند كتابة البرامج بلغة عالية المستوى تعليمات لا تعتمد على نوع المعالج المستخدم، وفي هذه الحالة يُستعمل المترجم (Compiler) لترجمة تعليمات اللغة العالية المستوى إلى تعليمات التجميع الخاصة بالمعالج. بشكل عام، لا يحتاج المبرمج بلغة عالية المستوى إلى معلومات عن المعالج ولكن في حالة النظم المضمّنة لا بد من أن يحيط المبرمج علماً ببعض المعلومات عن المعالج المستخدم.

سنذكر فيما يلي أهم هذه المعلومات التي يجب على المبرمج الإلمام بها:

1.4.قائمة التعليمات:

عند البرمجة بلغة التجميع يحتاج المبرمج إلى معرفة قائمة تعليمات (Instruction Set) المعالج.

تتألف التعليمة من قسمين: حقل رمز التعليمة (opcode) وحقل المعاملات (operands).

يحدد حقل التعليمة عمل التعليمة، ويمكن تصنيف التعليمات إلى ثلاث فئات:

- تعليمات نقل المعطيات (Data transfer): وهي التعليمات التي تنقل المعطيات بين الذواكر والسجلات أو بين المداخل/المخارج والسجلات أو بين السجلات نفسها.
- التعليمات الحسابية والمنطقية (Arithmetic/Logical): وهي التعليمات التي تقوم بتنفيذها وحدة الحساب والمنطق ALU.
- تعليمات القفز (Branch): وهي التعليمات التي تغير مكان التنفيذ في البرنامج وتوجه المعالج للتنفيذ من مكان آخر، ولهذه التعليمات عدة أنواع، فمنها تعليمات القفز اللاشرطية (Unconditional Jumps) وتعليمات القفز الشرطية (Conditional Jumps) وتعليمات استدعاء الإجرائيات الجزئية (Procedures) وتعليمات العودة من هذه الإجرائيات. تحدد هذه التعليمات عنوان التعليمة التالية التي سيقفز إليها المعالج، وفي حال تعليمات القفز الشرطية يقوم المعامل بالقفز إلى المكان الجديد فقط عند تحقق شرط معين بالاعتماد على رايات سجل الحالة.

يحدد حقل المعاملات مكان وجود المعطيات المرتبطة بالتعليمة. وللمعاملات نوعان إما معاملات مصدر (Source) وهي معاملات الدخل للتعليمة، أو معاملات وجهة (Destination) وهي مواقع الخرج التي سيتم فيها تخزين نتيجة التعليمة. يتغير عدد المعاملات حسب التعليمة، حتى أن هناك تعليمات بدون معاملات.

يتم تحديد عناوين مواقع المعاملات في حقل المعاملات باستعمال أنماط عنونة (Addressing modes) مختلفة:

1. العنونة الفورية (Immediate): يضم حقل المعاملات المعطيات نفسها التي ستعالجها التعليمة.
2. العنونة بالسجل (Register): يحتوي حقل المعاملات على رقم (عنوان) السجل الذي يحتوي على معطيات التعليمة.
3. العنونة المباشرة (Direct): يحتوي حقل المعاملات على عنوان الذاكرة التي تحتوي على معطيات التعليمة.
4. العنونة غير المباشرة بالسجل (Register Indirect): يحتوي حقل المعاملات على رقم السجل الذي يحتوي على عنوان الذاكرة التي تحتوي على معطيات التعليمة.
5. العنونة غير المباشرة (Indirect): يحتوي حقل المعاملات على عنوان الذاكرة التي تحتوي على عنوان ذاكرة أخرى تحتوي على معطيات التعليمة.
6. العنونة بالدليل (Indexed): هنا يمكن استعمال جميع أنواع العنونة السابقة ولكن ليس للدلالة على عنوان وجود معطيات التعليمة، بل على قيمة انزياح معين تُجمع مع محتوى سجل محدد ضمنياً (implicit) أي ثابت ومعروف سلفاً للحصول على عنوان وجود معطيات التعليمة، وهذا النوع من العنونة شائع الاستعمال عند التعامل مع الأشعة، حيث يتم تخزين عنوان بداية شعاع المعطيات ضمن سجل مخصص لهذه الغاية ويسمى سجل القاعدة BP (Base Pointer) ويتم تحديد دليل العنصر المرغوب من الشعاع ضمن حقل معاملات التعليمة.

إضافة إلى أنواع العنونة السابقة، هناك تعليمات لا تحتوي على حقل معاملات، وذلك لأن التعليمة لا تحتاج إلى معاملات. كما أن حقل المعاملات قد لا يحتوي على جميع معاملات التعليمة بل على بعضها فقط، وتكون المعاملات الأخرى محددة ضمنياً لهذه التعليمة. كمثال على ذلك نأخذ التعليمة التالية لأحد المعالجات ADD A, R5. تقوم هذه التعليمة بجمع معامل الدخل R5 المحدد في حقل معاملات التعليمة مع سجل المراكم A وتخزن النتيجة ضمناً في المراكم. في هذه التعليمة يمكن فقط تغيير السجل المستعمل R5 (أي R1 أو R2) ولكن ليس المراكم A لأنه سجل ضمني لهذه التعليمة.

وكما ذكرنا سابقاً، قد لا يحتاج المبرمج بلغة عالية المستوى إلى معرفة قائمة تعليمات التجميع للمعالج، ولكن في النظم المضمّنة غالباً ما يحتاج المبرمج لكتابة أجزاء معينة من البرنامج بلغة التجميع وذلك لأسباب عديدة: إما لأسباب تتعلق بسرعة التنفيذ حيث نريد تنفيذ جزء من البرنامج بسرعة أكبر لتحقيق متطلبات العمل ضمن الزمن الحقيقي، أو لأسباب تتعلق بدقة تزامن إشارات الدخل والخرج عند كتابة الإجرائيات التي تتعامل مع الطرفيات مثل بطاقة الصوت أو بطاقة شاشة العرض في النظام المضمّن. تسمى هذه الإجرائيات التي تتعامل مع الطرفيات سواقات برمجية (Software Drivers) وهي غالباً ما تُكتب بلغة التجميع.

2.4. فضاء ذاكرة البرنامج وذاكرة المعطيات:

في النظم المضمّنة ثمة دائماً حجم محدود لكل من ذاكرة البرنامج وذاكرة المعطيات، لذا يجب على المبرمج أن يعرف حدود الذاكرة المتاحة له كي لا يتجاوزها، وهو ما سيحدد طريقة كتابته للبرنامج. تُعتبر هذه من الأمور التي لا يدركها مبرمج النظم الحاسوبية المكتبية بسبب كبر حجم الذاكرة المتاح، ولكن ذلك يمثل قيداً كبيراً لمبرمج النظم المضمّنة في بعض الأحيان.

3.4. السجلات:

عند كتابة البرامج بلغة التجميع لا بد من معرفة السجلات الداخلية المتاحة لأغراض التخزين العام والتي نسميها سجلات العمل (Working Registers) إضافة إلى سجلات الوظائف الخاصة (Special Function Registers) مثل سجل القاعدة مثلاً. عند كتابة البرنامج بلغة عالية المستوى فهناك سجلات وظائف خاصة على المبرمج أن يُلم بها وهي تتعلق عادة بسجلات التحكم في الطرفيات مثل المؤقتات والعدادات وطرفيات التراسل التسلسلي وغيرها.

4.4.الدخل والخرج:

على مبرمج النظم المضمّنة معرفة إمكانيات الدخل والخرج (Inputs/Outputs) للتعامل مع الطرفيات المحيطية، ويشمل ذلك كيفية التعامل مع مرابط المعالج لإجراء عمليات القراءة والكتابة عن طريق بوابات المعالج أو عن طريق مسرى المعطيات الخارجي المكون من مسرى عناوين ومسرى معطيات.

5.4. المقاطعات:

تسبب المقاطعة (Interrupt) تعليق عمل المعالج في البرنامج الأساسي بشكل مؤقت والقفز إلى إجرائية تخديم المقاطعة ISR (Interrupt Service Routine) لمعالجة المقاطعة، وذلك بعد حفظ موقع التنفيذ الحالي في البرنامج، أي محتوى عداد البرنامج PC في الذاكرة، وبعد إنهاء إجرائية المقاطعة يعود المعالج ليتابع تنفيذ البرنامج الأساسي من المكان الذي توقف عنده من خلال إعادة شحن السجل PC بالقيمة المحفوظة في الذاكرة مسبقاً. على المبرمج الإلمام بأنواع المقاطعات المدعومة من قبل المعالج والقدرة على كتابة إجرائيات تخديم المقاطعات عند اللزوم. فعلى سبيل المثال، تفرض معظم المعالجات وجود برنامج تخديم كل مقاطعة في مكان محدد من ذاكرة البرنامج، ولذلك يجب معرفة الطريقة التي يوفرها المترجم للقيام بهذا الأمر عند كتابة إجرائيات تخديم المقاطعات.

5. المعالجات ذات التعليمات الموجهة لتطبيق محدد:

تتطلب التطبيقات المضمّنة المعاصرة مثل التلفزة العالية الدقة HDTV قدرة حسابية عالية ووظائف خاصة جداً. لا يمكن دائماً تلبية متطلبات هذه التطبيقات بفعالية باستعمال معالجات عامة، حيث تتعلق هذه المتطلبات بالسرعة والطاقة المستهلكة والحجم والكلفة. حتى أن المعالجات ذات الغرض الوحيد قد لا تناسب هذه التطبيقات بسبب عدم المرونة في القدرة على التطوير، وهنا تشكل المعالجات ذات التعليمات الموجهة لتطبيق محدد (ASIP) أو لمجال من التطبيقات حلاً وسطاً بين الحالتين السابقتين فهي ذات مواصفات أعلى من المعالجات العامة مع الحفاظ على مرونة البرمجة وقصر زمن تطوير النظام وصولاً للمنتج النهائي.
مع ذلك تبقى هذه المعالجات ذات كلفة أكبر من المعالجات العامة بسبب محدودية الإنتاج الكمي لها وغلاء أدوات التطوير من مترجمات وبطاقات التطوير. مع ذلك ساهم انتشار استعمال بعض أنواع المعالجات الموجهة إلى خفض كلفة هذه المعالجات مع الزمن. بالتالي يبقى الأمر متروكاً للمقايضة في استعمال المعالج المناسب حسب الظروف الراهنة عند تصميم النظام المضمّن.

يمكن تمييز نوعين من المعالجات الموجهة. النوع الأول هو المتحكمات الصغرية (Microcontrollers) الموجهة لتطبيقات التحكم والأتمتة. والنوع الثاني هو معالجات الإشارة الرقمية DSP (Digital Signal Processors) الموجهة لتطبيقات معالجة الإشارة الرقمية التي تتعامل مع معدل تدفق معطيات كبير.

1.5. المتحكمات:

يُنتج الكثير من المُصنّعين متحكمات (Microcontrollers) للنظم المضمّنة الموجهة لتطبيقات التحكم وهي تتمتع بالعديد من المزايا:

1. تحتوي على العديد من التجهيزات الطرفية مثل المؤقتات والمبدلات A/D وتجهيزات التراسل التسلسلي ومعدّلات عرض النبضة.
2. تحتوي على ذواكر داخلية للبرنامج وللمعطيات.
3. تحتوي على بوابات دخل/خرج.
4. لها تعليمات متخصصة وموجهة لتطبيقات التحكم مثل عمليات التعامل مع البتات.

يؤدي وجود كل هذه المزايا على شريحة تكاملية واحدة إلى الحصول على نظام مضمّن صغير الحجم وقليل الاستهلاك للطاقة.
في البداية ظهرت هذه المتحكمات بمواصفات متواضعة تحتوي بداخلها على معالجات على 8 بت. وقد تطورت هذه المتحكمات لتصبح ذات مواصفات عالية وتحتوي على معالجات على 32 بتاً، وأصبح يُطلق على هذه الأنواع المتطورة اسم معالجات مضمّنة (Embedded Processors).

2.5. معالجات الإشارة الرقمية:

معالجات الإشارة الرقمية DSP (Digital Signal Processors) هي معالجات تم تحسين تصميمها لتكون قادرة على تنفيذ تطبيقات معالجة الإشارة الرقمية بشكل أفضل من المعالجات العامة، حيث تتعامل هذه المعالجات مع تدفق كبير لمعطيات الدخل الناتجة عن ترقيم الإشارات الكلامية أو إشارات الصورة وغيرها من الإشارات الناتجة عن الحساسات كما في التطبيقات الطبية مثلاً. ومن التحسينات المدخلة في تصميم هذه المعالجات يمكن أن نذكر:

1. وجود ملفات سجلات متعددة مما يوفر عدد سجلات عمل كبير لإجراء العمليات الحسابية وحفظ النتائج المؤقتة دون الحاجة للنفاذ إلى الذاكرة.
2. وجود عدة وحدات حسابية تعمل على التوازي.
3. وجود تعليمات متخصصة كثيرة الاستعمال في معالجة الإشارة مثل عملية الضرب والمراكمة MAC (Multiply and Accumulate) التي تنفّذ بشكل عتادي دفعة واحدة مما يعطي سرعة في تنفيذ خوارزميات الترشيح والتحويلات المختلفة الشائعة الاستعمال في مجال معالجة الإشارة الرقمية.
4. تزويد هذه المعالجات ببوابات دخل وخرج سريعة ومربوطة مع الذاكرة مباشرة عن طريق وحدة النفاذ المباشر للذاكرة DMA (Direct Memory Access) دون المرور بوحدة المعالجة المركزية.
5. إدارة الحلقات التكرارية بشكل آلي دون الحاجة إلى إجراء عمليات حسابية من قِبل وحدة الحساب المركزية لزيادة عداد الحلقة والتحقق من شرط نهاية الحلقة، حيث يتم ذلك عتادياً من قِبل وحدات مستقلة.

إضافة إلى هذه المزايا، تحتوي معالجات الإشارة الرقمية على الكثير من الطرفيات المضمّنة كما هي الحال في المتحكمات.

6. خلاصة:

لاستعمال المعالجات العامة في النظم المضمّنة الصناعية العديد من الفوائد ومن أهمها انخفاض الكلفة الهندسية والمرونة العالية. يحتاج مبرمج النظام المضمن إلى معرفة العديد من المعلومات حول بنية المعالج الداخلية وقائمة التعليمات بلغة التجميع لتلبية متطلبات العمل بالزمن الحقيقي وكيفية التعامل مع طرفيات النظام. بالإضافة إلى ذلك فإن القيود المفروضة على حجم الذواكر تؤثر على كيفية كتابة برامج النظم المضمنة. تم تطوير المعالجات ذات التعليمات المخصصة لتطبيقات محددة لتلبية متطلبات بعض التطبيقات بشكل أفضل من المعالجات العامة. حيث ظهرت المتحكمات الصغرية لتلبية تطبيقات التحكم والأتمتة وظهرت كذلك معالجات الإشارة الرقمية لتلبية تطبيقات معالجة الإشارة ذات التدفق العالي للمعطيات.

7. أسئلة الفصل السابع:

أسئلة عامة:

1. ما هو المعالج ذو الأغراض العامة؟

هو نظام رقمي قابل للبرمجة ومُعد لتنفيذ المسائل الحسابية في الكثير من التطبيقات وفي العديد من المجالات.

2. ما هي فوائد استعمال المعالجات ذات الأغراض العامة؟

هنالك العديد من الفوائد ومن أهمها:

- سعر افرادي منخفض بسبب الانتاج الكمي للمعالجات العامة.
- الكلفة الهندسية المنخفضة في تصميم النظام.
- زمن دخول النظام إلى السوق أقصر.
- المرونة في تغيير البرنامج وإجراء التحديثات اللاحقة.

3. ما هي المكونات الأساسية للمعالج ذو الأغراض العامة؟

يتكون المعالج العام من ثلاثة أقسام رئيسية وهي:

- مسار المعطيات
- وحدة التحكم
- ذاكرة

4. ما هو مبدأ عمل الذاكرة المخبأة (cache memory)؟.

عندما يقوم المعالج في لحظة ما بالنفاذ إلى موقع معين في الذاكرة فإنه من المرجح أن يقوم المعالج بالنفاذ إلى المواقع المجاورة لهذا الموقع في اللحظات التالية، لذلك يتم نسخ كتلة كاملة من الذاكرة الخارجية حول الموقع الحالي إلى الذاكرة المخبأة، وبالتالي يكون زمن النفاذ إلى المواقع المجاورة قصيراً في التعليمات التالية.

5. ما هي مراحل تنفيذ التعليمة في المعالجات العامة وماذا يتم في كل مرحلة؟

يتم تنفيذ كل تعليمة في المعالج على عدة مراحل:

- جلب التعليمة: وهي مهمة قراءة التعليمة التالية من ذاكرة البرنامج إلى سجل التعليمة IP.
- تفسير التعليمة: وهي مهمة تحديد العملية التي تمثلها التعليمة الموجودة في سجل التعليمة.
- جلب المعاملات: وهي مهمة نقل معاملات التعليمة إن وجدت إلى السجلات المناسبة.
- تنفيذ التعليمة: وهي مهمة إجراء العملية على المعاملات ضمن وحدة الحساب والمنطق ALU وتخزين النتيجة في السجل المناسب.
- تخزين النتائج: وهي مهمة كتابة سجل النتيجة في الذاكرة.

6. ما هو المعالج المتعدد النوى (Superscalar Microprocessor) وما هي الفائدة منه؟

هو معالج يحتوي على عدة وحدات حساب ومنطق، وفي هذا المعالج يتم في مرحلة الجلب قراءة عدة تعليمات معاً.

الفائدة منه هو تسريع عمل المعالج من خلال عمل الوحدات على التوازي لتنفيذ عدة تعليمات في آن واحد.

7. ما هي معالجات الإشارة الرقمية؟

معالجات الإشارة الرقمية DSP (Digital Signal Processors) هي نوع من المعالجات الموجهة لتطبيقات محددة وهي معالجات تم تحسين تصميمها لتكون قادرة على تنفيذ تطبيقات معالجة الإشارة الرقمية بشكل أفضل من المعالجات العامة.

أسئلة خيارات متعددة:

1. نقول عن معالج بأنه معالج على 8 بت إذا كان:

A. عرض خطوط عنونة الذاكرة هو 8 بت.

B. عرض مسرى المعطيات الخارجية مع الذاكرة هو 8 بت.

C. عرض عناصر مسار المعطيات في المعالج على 8 بت.

D. عرض سجل عداد البرنامج PC هو 8 بت.

2. وحدة التحكم في المعالج العام هي:

A. الدارات التي تعالج المعطيات باستعمال وحدة الحساب والمنطق.

B. الدارات التي تجلب تعليمات البرنامج وتسير المعطيات ضمن مسار المعطيات.

C. الدارات التي تتحكم بمسرى المعطيات مع الذاكرة.

D. كل ما سبق.

3. يدل مفهوم "إصابة ذاكرة" (Cache Hit) عند التعامل مع الذاكرة المخبأة على:

A. وجود المعطيات التي نبحث عنها في الذاكرة المخبأة.

B. عدم وجود المعطيات التي نبحث عنها في الذاكرة المخبأة.

C. عطب في المعطيات الموجودة في الذاكرة المخبأة.

D. الحاجة إلى نقل المعطيات من الذاكرة الخارجية إلى الذاكرة المخبأة.

4. تسمح عملية التواردية (Pipelining) في المعالجات بمايلي:

A. التنبؤ بوجود تعليمة قفز في البرنامج.

B. جلب عدة تعليمات دفعة واحدة من الذاكرة.

C. تقليص زمن تنفيذ كل تعليمة.

D. رفع معدل تنفيذ التعليمات.

5. المتحكمات الصغرية هي نوع من:

A. المعالجات ذات الأغراض العامة.

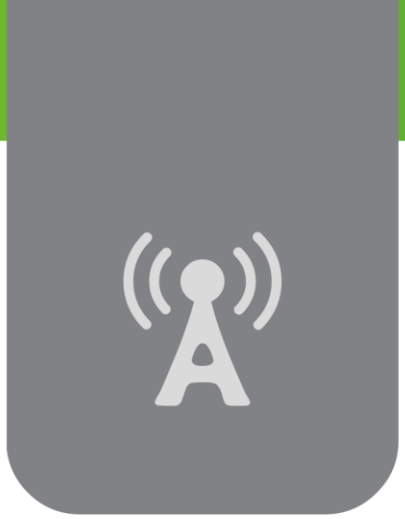
B. المعالجات ذات التعليمات الموجهة لتطبيق محدد.

C. المعالجات ذات الغرض الوحيد.

D. المعالجات المتعددة النوى.

الإجابات الصحيحة:

رقم التمرين	الإجابة الصحيحة
1	C
2	B
3	A
4	D
5	B



الفصل الثامن:
المعالجات الخاصة المعيارية: الطرفيات

الكلمات المفتاحية:

المؤقتات، ارسال تسلسلي UART، معدل عرض النبضة، متحكم شاشة LCD، متحكم لوحة مفاتيح، ساعة الزمن الحقيقي.

Timers, Serial transmission UART, Pulse Width Modulator, LCD controller, Keypad Controller, Real Time Clock.

الملخص:

يهدف هذا الفصل إلى التعريف ببعض المعالجات الشائعة الاستعمال ذات الغرض الوحيد أو ما نسميها عادةً بالطرفيات حيث سنتعرف على مبدأ عملها وتطبيقاتها في النظم المضمنة. وتشمل هذه الطرفيات على المؤقتات والعدادات بأنواعها، ووحدة التراسل التسلسلي غير المتزامن UART، ومتحكم شاشة العرض من نوع LCD، ومعدل عرض النبضة PWM ومتحكم لوحة المفاتيح وساعة الزمن الحقيقي.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- أنواع المؤقتات والعدادات
- عمل مرسل–مستقبل غير متزامن عام UART
- عمل معدل عرض النبضة (PWM) وتطبيقاته
- عمل متحكم شاشة العرض LCD
- عمل متحكم لوحة المفاتيح
- عمل ساعة الزمن الحقيقي

يهدف هذا الفصل إلى التعريف ببعض المعالجات الشائعة الاستعمال ذات الغرض الوحيد أو ما نسميها عادةً بالطرفيات حيث سنتعرف على مبدأ عملها وتطبيقاتها في النظم المضمنة. وتشمل هذه الطرفيات على المؤقتات والعدادات بأنواعها، ووحدة التراسل التسلسلي غير المتزامن UART، ومتحكم شاشة العرض من نوع LCD، ومعدل عرض النبضة PWM ومتحكم لوحة المفاتيح وساعة الزمن الحقيقي.

1. مقدمة:

المعالج ذو الغرض الوحيد (Single-Purpose Processor) أو المعالج الخاص هو نظام رقمي مُعد لتنفيذ مهمة حسابية معينة، وذلك بخلاف المعالج العام (General-Purpose Processor) المُعد لتنفيذ طيف واسع من المهمات الحسابية. يمكن أن يكون المعالج ذو الغرض الوحيد مخصصاً (Custom) أي نصممه بأنفسنا كما رأينا في الفصل الثاني، كما يمكن أن يكون المعالج ذو الغرض الوحيد معيارياً وجاهزاً وذلك من أجل المهمات الحسابية الشائعة الاستخدام، حيث تقوم بعض الشركات الكبرى بتصميم هذه المعالجات بكميات كبيرة وطرحها في الأسواق. من الفوائد التي نجنيها من استخدام المعالجات المعيارية في النظم المضمّنة مقارنة بالمعالجات المخصصة التي نصممها بأنفسنا: الكلفة المنخفضة. رغم أن معظم النظم المضمّنة قد تحتوي على معالج عام يستطيع تنفيذ المهمة الحسابية الموكلة إلى المعالج ذو الغرض الوحيد إلا أن الفائدة من استخدام معالج خاص لهذه المهمة تتمثل في سرعة الأداء والتخفيف عن المعالج العام ليتفرغ لمهمات أخرى لا نستطيع تنفيذها باستخدام معالجات خاصة. في المقابل قد يؤدي ذلك إلى زيادة حجم النظام، وهذا ما يقود مصمم النظام المُضمن إلى إجراء موازنة بين الخيارين.

نشرح في هذا الفصل عمل عدة معالجات معيارية ذات غرض وحيد شائعة الاستعمال في النظم المضمّنة. نسمي المعالجات ذات الغرض الوحيد "الطرفية" (Peripheral) وذلك لأنها عادة ما تُستخدم كطرفية لمعالج عام، وغالباً ما نجد هذه الطرفيات مضمّنة مع وحدة الحساب المركزية (CPU) في المتحكمات الصغرية (Microcontrollers) على نفس الشريحة الإلكترونية، وهذا ما يفسر الاستخدام الشائع للمتحكمات في النظم المضمّنة، الأمر الذي يؤدي إلى تقليص حجم النظام.

2. المؤقّتات والعدّادات:

المؤقتات (Timers) هي طرفيات كثيرة الاستعمال، ولها القدرة على قياس المجالات الزمنية. يمكن استخدام المؤقت في توليد أحداث في أزمان محددة، أو في تحديد الفترة الزمنية الفاصلة بين حصول حدثين خارجيين. فعلى سبيل المثال، يمكن استخدام المؤقت في تحديد أزمنة التبديل في الإشارات الضوئية أو قياس الزمن الفاصل بين ضغطتين متتاليتين على لوحة المفاتيح.

يقيس المؤقت الزمن من خلال عد نبضات إشارة ساعة الدخل المعلومة الدور. فمثلاً إذا كان دور الساعة 1 ميكرو ثانية وقام المؤقت بِعَدّ 2000 نبضة من نبضات الساعة يكون الزمن الموافق هو 2000 ميكرو ثانية.

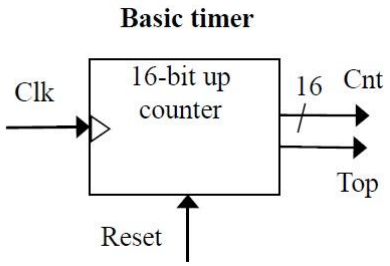
العداد (Counter): هو عبارة عن طرفية أكثر عمومية من المؤقت، إذ يقوم العداد بِعَدّ نبضات أي إشارة خارجية وليس إشارة الساعة فحسب. فمثلاً يمكن استعمال العداد لِعَدّ القوارير التي تمر فوق خط سير معمل من إشارة حساس عبور.

عادةً ما نستعمل العدادات والمؤقتات معاً لقياس معدل التغير، فمثلاً يمكن قياس معدل سرعة السيارة من خلال عدد دورات العجل خلال ثانية واحدة، حيث نستعمل عداداً لِعَدّ دورات العجل عن طريق حساس مركّب عليه، ونستعمل مؤقتاً لضبط زمن العد أي زمن ثانية واحدة مثلاً.

هناك أشكال مختلفة للمؤقتات بحسب البنية الداخلية وطبيعة المداخل والمخارج. سنذكر فيما يلي الأشكال الأكثر شيوعاً:

1.2. مؤقت بسيط:

يبين الشكل أدناه بنية المؤقت البسيط (Simple Timer). يحتوي المؤقت على عداد داخلي بعرض 16 بت حيث تزيد قيمة العداد بمقدار واحد عند كل نبضة للساعة. لهذا المؤقت مدخل reset لتصفير قيمة العداد. في الخرج نجد المخرج Cnt على 16 بت وهو يعبّر عن قيمة العداد المحصورة بين القيمة 0 والقيمة 2^{16}– 1=65535. عند وصول العداد للقيمة العظمى يعود ليبدأ من الصفر وعندها يولد المؤقت نبضة على المخرج top. نسمي عملية عودة العداد إلى الصفر بعد الوصول للقيمة النهائية بالطفح (overflow).

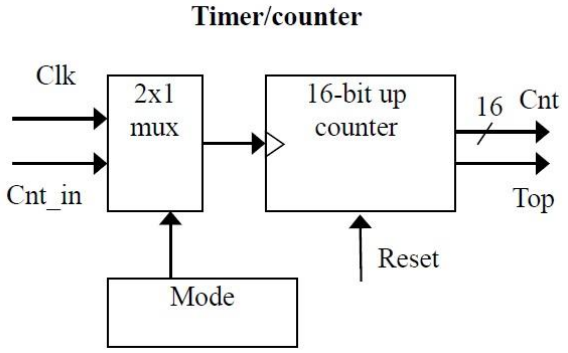


مؤقت بسيط

فإذا كان المؤقت يعمل على ساعة بتردد مقداره 100 MHz يكون زمن كل دور كل نبضة هو 1/100 MHz=10 ns. أي إن كل عدة للعداد تقابل زمن 10 ns. نسمي هذا الزمن بالتمييزية الزمنية (Time Resolution) للمؤقت وهو أقل زمن يمكن للمؤقت أن يقيسه. تتحسن التمييزية الزمنية للمؤقت عند نقصان الزمن الموافق لكل عدة، وتسوء التمييزية الزمنية بزيادة هذا الزمن. أما أكبر زمن يستطيع المؤقت قياسه فهو 65535×10 ns= 655.35 ms والذي نسميه مجال المؤقت. يمكن استعمال المخرج top لتوليد مقاطعة للمعالج الذي يقوم بدوره بتنفيذ برنامج تخديم المقاطعة عند حصول الطفح بما يتناسب مع التطبيق.

2.2. مؤقت بنمطين:

يبين الشكل أدناه بنية مطورة عن المؤقت البسيط حيث يمكن اختيار ساعة العداد من خلال ناخب محكوم بالمدخل mode للاختيار بين ساعة النظام clk (mode=0) أو إشارة خارجية أخرى Cnt_in (mode=1)، وعند العمل مع اختيار إشارة خارجية كمدخل لساعة العداد يصبح المؤقت عداداً. وغالباً ما تكون هذه الإشارة الخارجية موصولة مع حساس خارجي، وفي هذه الحالة يعبر المخرج Cnt عن عدد نبضات الحساس. عند العمل مع اختيار ساعة النظام كمدخل لساعة العداد يصبح عمل المؤقت مماثلاً لعمل المؤقت البسيط الذي شرحنا عمله في الفقرة السابقة.

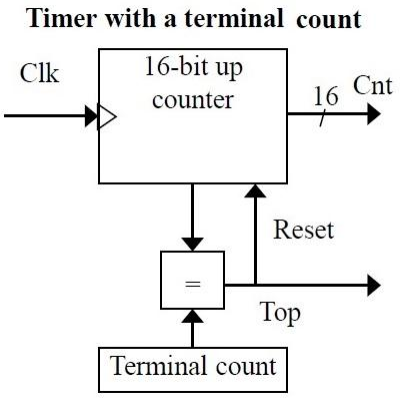


مؤقت بنمطين: مؤقت أو عداد

3.2.مؤقت مجال زمني:

يبين الشكل أدناه بنية مؤقت مجال زمني (Interval Timer) يخبرنا عن انقضاء مجال زمني محدد. يخزن السجل terminal count القيمة الموافقة للمجال الزمني المرغوب والتي يقوم المستخدم بوضعها، وهي تُعبّر عن عدد نبضات الساعة الموافقة للمجال الزمني المحدد والذي يحسب من خلال العلاقة التالية:

Number of clock cycles = desired time interval/ clock period



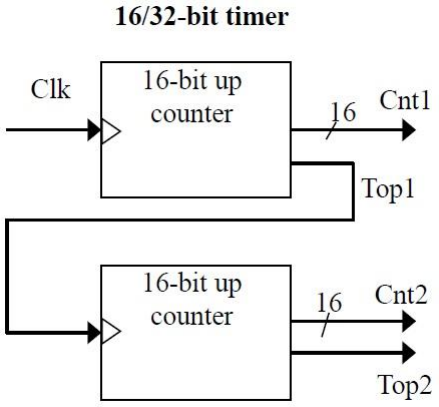
مؤقت مجال زمني

فعلى سبيل المثال، للحصول على زمن قدره 3 ميكرو ثانية يكون عدد نبضات الساعة الموافق من أجل دور ساعة نظام مقداره 10 ns (100 MHz) هو:

يحتوي المؤقت على مقارن للكشف عن وصول المؤقت إلى قيمة العد المطلوبة، وعندها يولد إشارة top التي تقوم بدورها بتصفير المؤقت، كما يمكن استعمالها كمدخل لمقاطعة المعالج للقيام بالمعالجة المطلوبة عند الوصول إلى الزمن المطلوب.
يمكن تبسيط بنية هذا المؤقت من خلال تبسيط بنية المقارن، فبدلاً من أن يقوم المؤقت بالعد التصاعدي انطلاقاً من الصفر للوصول إلى القيمة المطلوبة، نقوم بشحن المؤقت بالقيمة المطلوبة، ويقوم المؤقت بالعد التنازلي للوصول إلى الصفر. وبالتالي يُستعمل مقارن مع الصفر للكشف عن انتهاء المجال الزمني. إن عملية المقارنة مع الصفر هي ذات تعقيد عتادي أقل من المقارنة مع قيمة معينة أخرى ولهذا تكون بنية المؤقت بالطريقة الجديدة أقل تعقيداً من الناحية العتادية.

4.2. مؤقتات متراصّة:

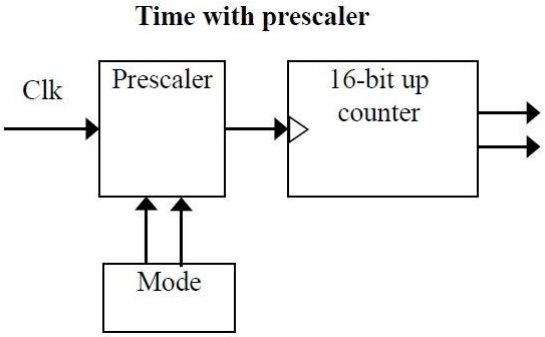
يبين الشكل أدناه بنية مؤقت مكون من مؤقتات متراصة (Cascaded Timers) يمكن إعداده بحيث يكون مؤقتاً على 16 بتاً أو على 32 بتاً. يستخدم هذا المؤقت عدادين كل منهما على 16 بتاً، حيث يتم وصل الخرج top للعداد الأول مع مدخل العداد الثاني وهكذا نحصل على مؤقت على 32 بتاً انطلاقاً من عدادين على 16 بتاً.



مؤقت مكون من مؤقتات متراصة

5.2. مؤقت مع مقسّم:

يبين الشكل أدناه بنية مؤقت مع مقسّم مبرمج (Timer with Prescaler) لساعة الدخل حيث يقسم هذا الأخير ساعة الدخل على 1 أو 2 أو 4 أو 8 أو أية قيمة أخرى من بين مجموعة من القيم. يتم اختيار هذه القيمة من خلال قيمة السجل mode. يؤدي هذا التقسيم إلى خفض تردد الساعة التي يعمل عليها المؤقت، مما يعني زيادة مجال المؤقت على حساب التمييزية الزمنية.



مؤقت مع مقسم قابل للبرمجة

فمثلاً، إذا كان المؤقت يعمل على ساعة بدور قدره 10 ns واستعملنا مقسّم على 8 يصبح دور ساعة المؤقت مساوياً 80 ns وبالتالي يكون مجال المؤقت هو 80 ns×65535=5.24 ms ولكن تسوء التمييزية الزمنية لتصبح 80 ns عوضاً عن 10 ns.

6.2. مؤقت الملاحقة:

يوجد نوع خاص من المؤقتات هو مؤقت الملاحقة (Watchdog Timer). لا تأتي خصوصية هذا المؤقت من بنيته الداخلية ولكن من طبيعة استخدامه الخاصة، حيث يتكون من مؤقت بسيط ولكن يتم ربط مخرجه top مع إشارة إعادة إقلاع النظام المضمّن reset، والهدف من ذلك هو منع برنامج النظام من الدخول في

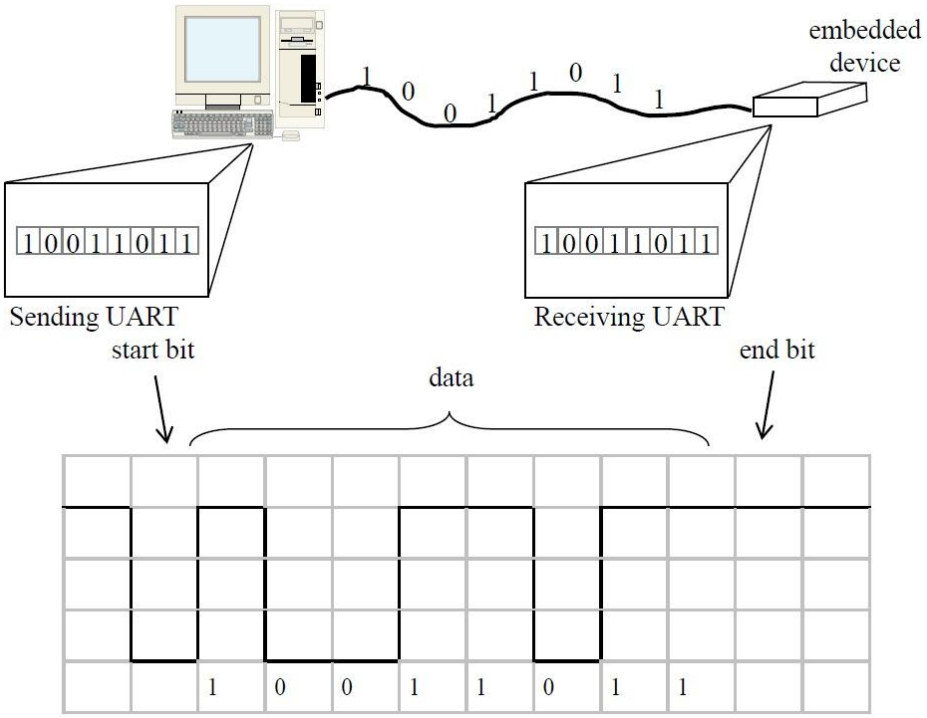
حلقات لانهائية غير متوقعة نتيجة حدوث خلل ما في النظام مثل وجود خلل في ذاكرة البرنامج، أو في أحد المداخل الخارجية مثلاً.

عند استعمال مؤقت ملاحقة لإعادة إقلاع النظام يجب أن نضيف داخل البرنامج تعليمات تصفير لهذا المؤقت بفواصل زمنية أقل من زمن طفح المؤقت بحيث لا يقوم المؤقت بإعادة إقلاع النظام عند العمل الطبيعي للبرنامج. وعند خروج البرنامج عن السيطرة أو الدخول في حلقة لانهائية بسبب حدوث خلل لا يتم تصفير المؤقت ويستمر إلى أن يصل إلى الطفح، وعندها سيؤدي ذلك إلى إعادة إقلاع النظام بشكل آلي.

يمكن تشبيه دور مؤقت الملاحقة بالنسبة للمعالج المركزي في النظام المضمّن بدور المستخدم بالنسبة للحاسوب الشخصي، حيث يقوم المستخدم بإعادة إقلاع الحاسوب عندما يجد بعد فترة من الزمن أن الحاسوب لم يعُد يستجيب.

3. مرسل-مستقبل غير متزامن عام (UART):

نحتاج أحياناً في بعض التطبيقات أن يتبادل النظام المضمن المعطيات مع طرفية موجودة بعيداً عن النظام باستعمال أقل عدد من الأسلاك. وهنا يأتي التراسل التسلسلي UART (Universal Asynchronous Receiver-Transmitter) ليُلبي هذا المطلب، فبدلاً من إرسال المعطيات بشكل تفرعي على عدة أسلاك نقوم بإرسالها بشكل متسلسل على سلك واحد. فمثلاً إذا كانت وحدة المعطيات هي بايت واحد (1 Byte) يلزمنا 8 أسلاك لإرسال وحدة المعطيات دفعة واحدة. أما في التراسل التسلسلي فنستخدم سلكاً واحداً لإرسال وحدة المعطيات بتاً تلو الآخر على 8 دفعات بفواصل زمنية محددة بين البتات. يعكس زمن إرسال البت سرعة التراسل الذي نعبر عنه بمعدل التراسل (baud rate) بوحدة بت/ثانية، وهو مقلوب زمن البت. عند إرسال وحدات المعطيات تباعاً لا بد من فصل سلسلة بتات كل وحدة معطيات بفواصل عن سلسلة بتات وحدة المعطيات السابقة واللاحقة لتمييزها بشكل صحيح عند الاستقبال وإعادة تجميعها بشكل صحيح. لذلك تتم إضافة بت في بداية سلسلة البتات نعطيه القيمة 0 ونسميه بت البداية (start bit) وبت آخر (على الأقل) في نهاية سلسلة البتات نعطيه القيمة 1 ونسميه بت التوقف (stop bit). نَصِف طريقة التراسل هذه بالتراسل غير المتزامن (asynchronous) وذلك لأننا لا نستخدم إشارة ساعة مع خط المعطيات للدلالة على لحظة إرسال كل بت. يستطيع المُستقبل معرفة لحظة ورود كل بت من خلال الكشف عن بت البداية والمعرفة المسبقة لزمن كل بت.



التراسل التسلسلي غير المتزامن

نطلق على مجموعة المعامِلات التي تحدد طريقة التراسل ببروتوكول (protocol) التراسل وهو يشمل معدل التراسل، وعدد البتات في وحدة المعطيات، وعدد بتات التوقف، ومعامِلات أخرى تتعلق بتنظيم عملية التراسل بين المرسل والمستقبل لا داعي لتفصيلها هنا.

غالباً ما تكون وحدة التراسل في البايت مكونة من 8 بتات، وأحياناً يُضاف بت تاسع يسمى بت الزوجية (bit Parity) الذي يعبر عن زوجية أو فردية المجموع الجبري للبتات في الوحدة، وذلك للكشف عن وجود أي خطأ في تراسل المعطيات.

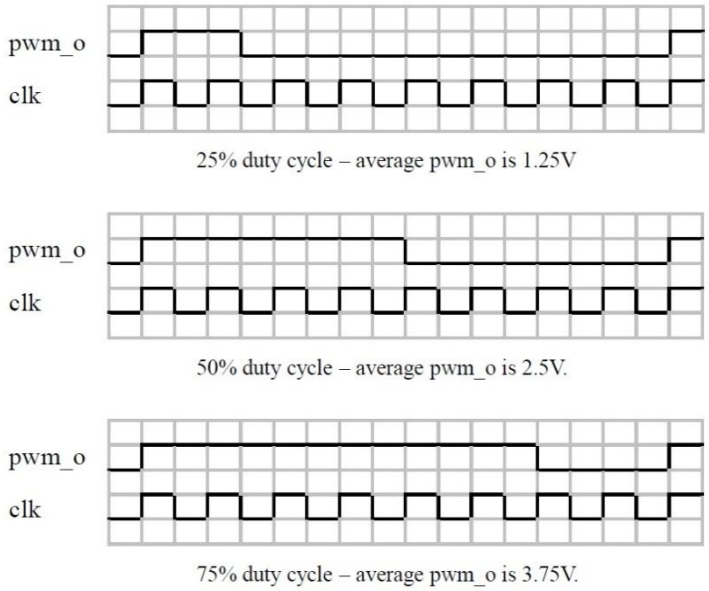
ومن معدلات التراسل الشهيرة نذكر: 2400 و4800 و9600 بت بالثانية.

تُؤمن وحدة الإرسال والاستقبال غير المتزامنة العامة UART عمليات التراسل التسلسلي بالاتجاهين، حيث تحتوي على مخرج إرسال ومدخل استقبال، وتعمل على إشارة ساعة بتردد يساوي معدل التراسل. تحتوي هذه الوحدة على سجل إزاحة في طرف الإرسال، وسجل إزاحة في طرف الاستقبال لتحويل المعطيات المتوازية إلى تسلسلية عند الإرسال وبالعكس عند الاستقبال، كما تحتوي هذه الوحدة على سجلات تحكم داخلية لاختيار بروتوكول التراسل.

غالباً ما يتم اشتقاق ساعة التراسل باستخدام مقسمات ترددية او باستخدام المؤقتات كما هي الحال في الوحدات المضمنة في بعض المتحكمات الصغرية.

4. معدّل عرض النبضة (PWM):

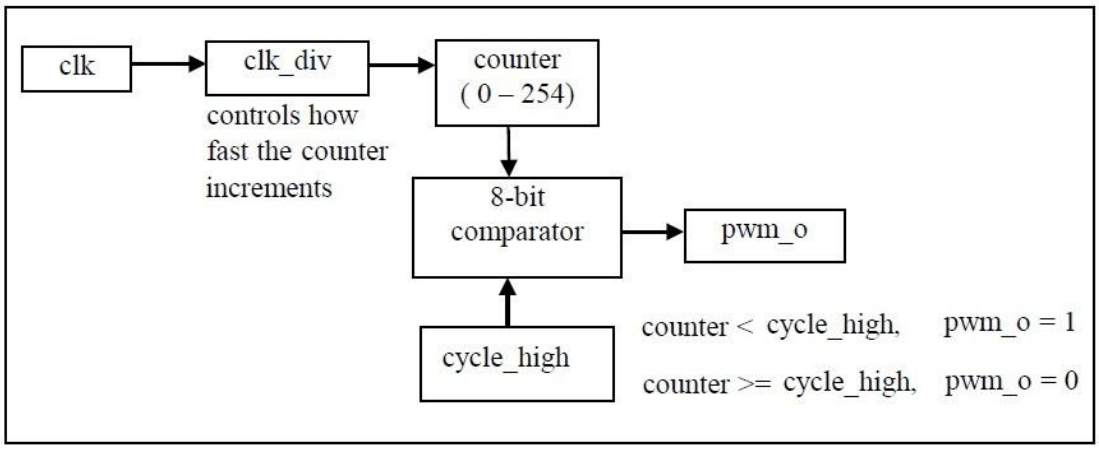
معدّل عرض النبضة PWM (Pulse Width Modulator) هو دارة تولد نبضة دورية بدور ثابت T وعرض نبضة يمكن التحكم به، حيث تأخذ الإشارة قيمة جهد ثابت (+5v مثلاً) خلال زمن نرمز إليه بالرمز T_1 ($0 \leq T1 \leq T$) وتأخذ الإشارة قيمة 0v خلال ما تبقى من زمن الدور، أي $T-T_1$، كما يبين الشكل التالي.



مبدأ عمل معدل عرض النبضة

نسمي النسبة T_1/T بين عرض النبضة T_1 ودور الإشارة T نسبة الدور (Duty Cycle). على سبيل المثال تكون نسبة الدور في إشارة مربعة مساوية 50%.

يحتوي معدّل عرض النبضة على مؤقت بسيط أو مؤقت على مجال بحيث يقابل زمن الطفح للمؤقت دور الإشارة، إضافة إلى مقارن بين قيمة المؤقت وسجل يحتوي على زمن النبضة المرتفع T_1. عندما تكون قيمة المؤقت أقل من قيمة السجل T_1 يكون جهد الخرج على السوية العليا. ويكون جهد الخرج على السوية الدنيا بعد هذه القيمة إلى نهاية الدور. ومن خلال التحكم بقيمة سجل نستطيع التحكم بعرض النبضة.



المخطط الصندوقي لمعدل عرض النبضة PWM

من أبسط تطبيقات معدّل عرض النبضة هو عملية توليد وميض ضوئي متقطع وذلك بجعل إشارة خرج معدّل عرض النبضة مدخلاً تحكمياً لتشغيل مظهر ضوئي (LED). في هذا التطبيق يكون دور الإشارة عادة من رتبة عدة ثواني، وعندما يكون دور الإشارة قصيراً جداً (أقل من 10 ميلي ثانية) فإن العين لا تلاحظ تقطع الإضاءة ولكن تلاحظ ازدياد شدة إضاءة المظهر كلما زاد عرض النبضة، حيث تصبح العين حساسة فقط للقيمة الوسطية للإضاءة، وبالتالي يمكن استخدام معدّل عرض النبضة للتحكم بشدة إضاءة مصباح عن طريق تغيير عرض النبضة عوضاً عن تغيير قيمة الجهد المطبق. على سبيل المثال، إذا كان لدينا مظهر ضوئي خطي تتناسب شدة إضاءته مع الجهد المطبق عليه، وكان الجهد الأعظمي الذي يعطي إضاءة كاملة هو 5v+ فعند استخدام عرض نبضة بنسبة دور 50% فإن ذلك يعطي إضاءة تكافئ تطبيق 2.5v، وكذلك من أجل نسبة دور 75% نحصل على إضاءة تكافئ تطبيق 3.75v.

يبين المثال السابق أن معدّل عرض النبضة يلعب دور المبدل الرقمي التماثلي الذي يحول القيمة الرقمية إلى قيمة جهد مستمر، وذلك بالنسبة لأي جهاز يكون خرجه متعلقاً بالقيمة الوسطية لإشارة الدخل وليس بالقيمة اللحظية.

يمكننا تطبيق المبدأ نفسه للتحكم بسرعة المحركات التي تعمل على التيار المستمر (DC Motors) نظراً لأن المحركات ذات استجابة بطيئة لجهد التحكم (مثل العين)، وتتغير سرعتها بحسب القيمة الوسطية لإشارة الدخل.

هناك تطبيقات أخرى لمعدّل عرض النبضة في إرسال أوامر التحكم باستخدام إشارة وحيدة، فعلى سبيل المثال، يستخدم تعديل عرض النبضة للتحكم بألعاب السيارت المتحكَّم بها لاسلكياً، حيث يمكن أن تقابل إشارة بعرض نبضة 1ms أمر الدوران نحو اليمين، وتقابل إشارة بعرض 4ms أمر الدوران نحو اليسار، وإشارة بعرض 8ms أمر المسير للأمام، وهكذا.

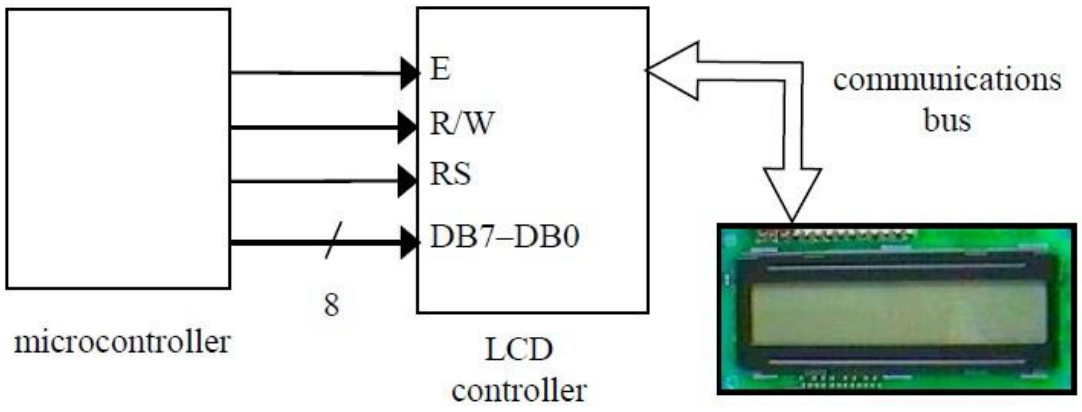
5. متحكم شاشات العرض LCD:

تُعتبر شاشات العرض ذات السائل الكريستالي LCD (Liquid Crystal Display) من الشاشات قليلة الكلفة والاستطاعة، لذلك فهي شائعة الاستعمال في النظم المضمنة مثل الساعات وآلات النسخ والفاكس والآلات الحاسبة.

من أبسط أنواع شاشات LCD الشاشة السباعية القطع (Seven Segment) المستعملة في الساعات الرقمية، حيث يتم تمثيل كل رقم بسبع قطع، وكل قطعة لها خط تحكمي يمكّن من إظهار أو إخفاء هذه القطعة، وبالتالي فإن لكل رقم سبع إشارات تحكمية. تحتوي دارة قيادة شاشة LCD على دارات منطقية لإخراج قيم الإشارات التحكمية حسب الرقم المراد عرضه. هذا بالإضافة إلى خطوط تحكمية أخرى لعرض النقاط والرموز الخاصة.

هناك نوع آخر من شاشات LCD هو شاشات المصفوفة المنقطة (Dot-matrix LCD) القادرة على عرض أرقام أو أحرف. يتم عادةً تمثيل كل حرف بمصفوفة من النقاط التي تحتوي على 5 أعمدة و8 صفوف. تنطوي مهمة دارة القيادة في هذه الشاشات على إخراج الإشارات اللازمة لإظهار أو إخفاء كل نقطة بهدف عرض الحرف المطلوب.

تحتوي عادةً هذه الشاشات على العديد من مزايا العرض, مثل عكس إضاءة الحرف، أو وميض الحرف، أو إظهار مؤشر للمكان الحالي للعرض عند إجراء عمليات الادخال من قبل المستخدم وغير ذلك من المزايا الأخرى.



متحكم شاشة العرض LCD

يتم تنفيذ كل هذه الأمور من قبل دارة تسمى متحكم العرض (LCD controller). يحتوي متحكم العرض على مدخل للمعطيات وهو مكون من 8 خطوط (DB7–DB0) بالإضافة إلى 3 إشارات تحكمية وهي:

مدخل تحكمي RS: يستعمل للتمييز بين معطيات العرض ومعطيات التحكم.

مدخل كتابة R/W: يستعمل للتمييز بين عمليات القراءة والكتابة.

مدخل تأهيل E: يستعمل لتأهيل متحكم العرض، حيث لا يستجيب المتحكم لإشارات الدخل إلا عندما يكون هذا الخط على السوية العليا.

يبين الشكل أدناه طريقة ترميز العمليات المختلفة التي يقوم بها المتحكم. حيث يمكن أن نرسل للمتحكم على خطوط المعطيات رموز المحارف التي نريد عرضها على الشاشة وعندها يكون الخط التحكمي RS=1، أو نرسل أوامر تحكمية أخرى مثل عملية مسح الشاشة أو تغيير مكان مؤشر الموقع، وعندها يكون RS=0.

RS	R / W	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0	Description
0	0	0	0	0	0	0	0	0	1	**Clears all display, return cursor home**
0	0	0	0	0	0	0	0	1	*	**Returns cursor home**
0	0	0	0	0	0	0	1	I / D	S	**Sets cursor move direction and /or specifies not to shift display**
0	0	0	0	0	0	1	D	C	B	**ON / OFF all display (D), cursor ON / OFF (C), andblink position**
0	0	0	0	0	1	S / C	R / L	*	*	**Move cursor and shifts display**
0	0	0	0	1	DL	N	F	*	*	**Sets interface data length, number of display lines, and character font**
1	0	WRITE DATA								**Writes Data**

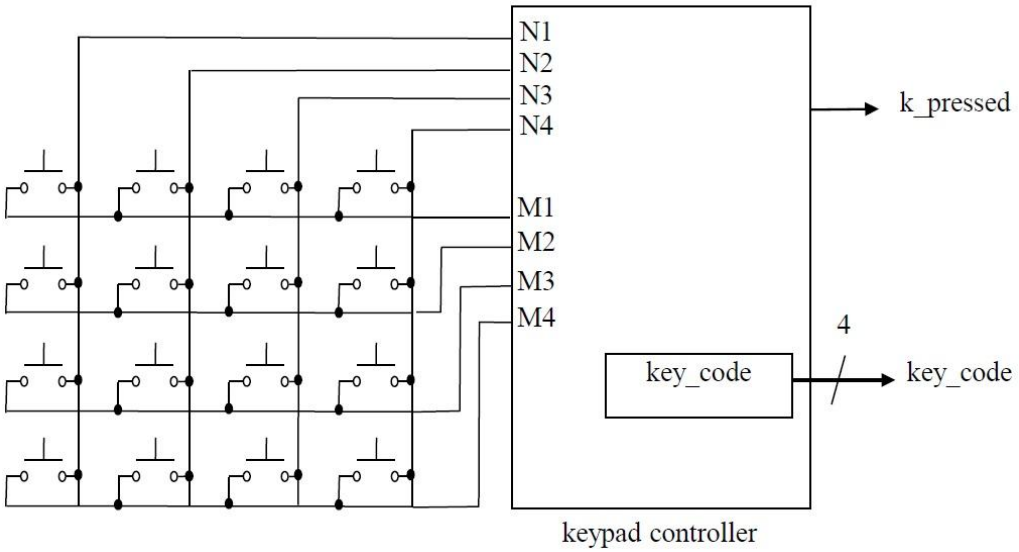
CODES	
I / D = 1 cursor moves left	DL = 1 8 − bit
I / D = 0 cursor moves right	DL = 0 4 − bit
S = 1 with display shift	N = 1 2 rows
S / C = 1 display shift	N = 0 1 row
S / C = 0 cursor movment	F = 1 5 × 10 dot
R / L = 1 shift to right	F = 0 5 × 7 dots
R /L = 1 shift to left	

ترميز العمليات المختلفة التي يقوم بها متحكم شاشة العرض LCD

6. متحكم لوحة المفاتيح:

تتكون لوحة المفاتيح (keypad) من مجموعة من المفاتيح التي نستعملها لإدخال المعطيات إلى النظام المضمن وهي كثيرة الاستخدام في النظم المضمنة بهدف التحكم اليدوي وادخال المعطيات.

يحتوي النمط البسيط للوحة المفاتيح على مفاتيح مرتبة مشكل مصفوفة مكونة من N عمود وM سطر كما هو مبين في الشكل أدناه. حيث يوجد لكل سطر ولكل عمود مربطاً مستقلاً. وعند الضغط على أحد المفاتيح، يصل المفتاح بين مربط السطر ومربط العمود الذين يقع المفتاح عليهما. يكشف متحكم لوحة المفاتيح عن المفتاح المضغوط باستخدام طريقة المسح (scan). تقوم طريقة المسح على مبدأ إخراج نبضات على الأعمدة بشكل متسلسل ومراقبة مرابط الأسطر أو بالعكس.



طريقة وصل لوحة المفاتيح بالمتحكم

حيث يخرج المتحكم في البداية نبضة على العمود الأول ويراقب مرابط جميع الأسطر. فإذا خرجت النبضة من السطر الثاني مثلاً، يكون المفتاح الذي في السطر الثاني من العمود الأول هو المفتاح المضغوط. وإذا لم يتم الكشف عن خروج أي نبضة، فعندها لا يكون هناك أي مفتاح مضغوط ضمن العمود الأول. تتكرر نفس العملية تباعاً مع بقية الأعمدة. وبذلك يكون المتحكم قد أتم مسحة كاملة للوحة المفاتيح. تتكرر كامل عملية المسح هذه بشكل دوري بفواصل زمنية أقل من 100 ميلي ثانية وذلك لضمان كشف أي ضغطة لأحد المفاتيح حتى ولو كانت سريعة.

عند الكشف عن أي مفتاح مضغوط، يخزن متحكم لوحة المفاتيح رمز المفتاح المضغوط ضمن سجل داخلي وهو موصول مع مخرج المتحكم key_code، كما يرفع المتحكم المخرج k_pressed للدلالة على وجود مفتاح مضغوط. يمكن الاستفادة من هذا المخرج بوصله مع أحد مداخل المقاطعات للمعالج في النظام المضمن لكي يقوم بقراءة رمز المفتاح المضغوط وتنفيذ الإجرائية المناسبة لعملية الادخال.

7. ساعة الزمن الحقيقي (RTC):

تستعمل ساعة الزمن الحقيقي RTC (Real-Time Clock) في النظم المضمنة لتزويد النظام بمعلومات الزمن مثل الوقت والتاريخ. يُزوَد النظام المضمن ببطارية صغيرة قابلة للشحن وذلك لضمان استمرار عمل ساعة الزمن الحقيقي عند اطفاء النظام أو فصله عن مصدر التغذية الكهربائية.

تحتوي ساعة الزمن الحقيقي على مهتز كريستالي دقيق يعطي تردداً ثابتاً ومستقراً وهو عادةً بقيمة 32.768 KHz. وانطلاقاً من نبضات هذا المهتز، يتم الحصول على الثواني والدقائق والساعات والتاريخ باستخدام عدادات متراصة مع بعضها على التسلسل. حيث يتم ضبط العداد الأول ليعد حتى 32768000 وهو ما يقابل زمن ثانية واحدة. ويتم ربط خرج العداد الأول مع مدخل العداد الثاني الذي يعد حتى 59 وهو ما يقابل زمن دقيقة واحدة. وهكذا يتم ربط عدادات الساعات والأيام والأشهر والسنين للحصول على الزمن والتاريخ.

يتم تراسل المعلومات بين المعالج وساعة الزمن الحقيقي بشكل تسلسلي عن طريق البروتوكول المسمى I2C الذي سنأتي على ذكره في الفصول القادمة وذلك من أجل قراءة قيمة الوقت وإعادة ضبط الساعة إذا لزم الأمر.

8. خلاصة:

نستعمل في النظم المضمنة العديد من المعالجات الجاهزة ذات الغرض الوحيد ونسميها عادةً طرفيات. تُستعمل المؤقتات لقياس الزمن والعد. وتستعمل وحدة التراسل التسلسلي لربط النظام مع الطرفيات أو النظم الأخرى البعيدة بطريقة غير مكلفة باستعمال سلكين فقط. نحتاج أيضاً في النظم المضمنة إلى لوحات مفاتيح وشاشة عرض LCD لتأمين التحكم اليدوي بالنظام من قبل المستخدم. لذلك توجد متحكمات خاصة لقيادة هذه الطرفيات. يشكل معدل عرض النبضة وسيلة فعالة للتحكم ببعض أجزاء النظام التي تستجيب مع القيمة المتوسطة للإشارة مثل التحكم بشدة إضاءة المظهرات الضوئية وسرعة المحركات من خلال خط وحيد. وأخيراً، نستعمل ساعات الزمن الحقيقي لتزويد النظام بالوقت والتاريخ بشكل دائم حتى ولو تم فصل النظام عن التغذية الكهربائية بفضل بطارية مستقلة مخصصة لعمل ساعة الزمن الحقيقي بشكل دائم.

9. أسئلة الفصل الثامن:

أسئلة عامة:

1. ما هي التمييزية الزمنية للمؤقت وما هو مجال المؤقت؟
التمييزية الزمنية هو أقل زمن يمكن للمؤقت أن يقيسه.
مجال المؤقت هو أكبر زمن يستطيع المؤقت قياسه.

2. ما هو تعريف نسبة الدور (Duty Cycle) في معدل عرض النبضة PWM؟
هي النسبة T_1/T بين عرض النبضة T_1 ودور الإشارة T.

3. ما هو عمل متحكم لوحة المفاتيح؟
يكشف متحكم لوحة المفاتيح عن المفتاح المضغوط باستخدام طريقة المسح واعلام المعالج بذلك عن طريق المخرج k_pressed.

4. اشرح مبدأ مسح لوحة المفاتيح البسيطة المكونة من N عمود و M سطر الذي يقوم به متحكم لوحة المفاتيح للتعرف على المفتاح المضغوط؟
يخرج المتحكم في البداية نبضة على العمود الأول ويراقب مرابط جميع الأسطر. فإذا خرجت النبضة من السطر الثاني مثلاً، يكون المفتاح الذي في السطر الثاني من العمود الأول هو المفتاح المضغوط. وإذا لم يتم الكشف عن خروج أي نبضة، فعندها لا يكون هناك أي مفتاح مضغوط ضمن العمود الأول. تتكرر نفس العملية تباعاً مع بقية الأعمدة.

5. ما هي الفائدة من ساعة الزمن الحقيقي RTC في النظم المضمنة؟
تزويد النظام بمعلومات الزمن مثل الوقت والتاريخ وهو مزود ببطارية صغيرة لضمان استمرار عمل ساعة الزمن الحقيقي.

6. اذا اعتبرنا بنية المؤقت على مجال وبفرض أن تردد ساعة المؤقت هي 10MHz.

a) حدد مجال المؤقت وتمييزيته الزمنية.

b) احسب قيمة Terminal count الموافقة لقياس زمن قدره 3 ms.

c) إذا أضفنا مقسم مبرمج لهذا المؤقت. ماهي أقل نسبة تقسيم نحتاج إليها لكي نتمكن من قياس زمن قدره 100 ms. ثم حدد مجال وتمييزية المؤقت الناتج. (قيمة التقسيم يجب أن تكون من قوى العدد 2).

a) الطلب الأول

resolution = period = 1 / frequency = 1 / (10 MHz) = 10^{-7} s=100 ns

range = 2^{16} * resolution = 65536 * 10^{-7}s = .0065536 s=6.5536 ms

b) الطلب الثاني

terminal count value = desired time interval / clock period

=310^{-3} s / 10^{-7} s = 30,000

c) الطلب الثالث:

للحصول على زمن 100 ms يجب تكبير مجال المؤقت ليصبح أكبر من هذا الزمن.

بما أن المجال الأصلي هو 6.5536 ms فيجب أن يكون معامل التكبير أكبر من

100 ms / 6.5536ms=15.26

لذلك نختار معامل التكبير 16 ليكون من قوى العدد 2 وأكبر من المعامل السابق. وبالتالي يجب أن نستخدم مقسم على 16. وبالتالي يكون:

resolution = 16 * original resolution = 16 * 10^{-7} s = 1.6*10^{-6} s=1.6 µs

range = 16 * original range = 16 * 6.5536 ms=104.8576 ms

7. لدينا محرك يعل بسرعة 10 دورات بالثانية عند تغذيته بجهد قدره 3.7V. بفرض أننا نستخدم معدل عرض نبضة للتحكم بسرعة هذا المحرك بحيث يكون جهد الخرج هو 0V أو 5V.

a) ما هي نسبة الدور (duty cycle) اللازم توليدها من المعدل PWM للحصول على سرعة 10 دورات بالثانية.

b) ما هي نسبة الدور (duty cycle) اللازم توليدها من المعدل PWM للحصول على سرعة 3 دورات بالثانية.

a) الطلب الأول: نسبة الدور للحصول على جهد 3.7 V هي:

Duty cycle= 3.7/5=0.74=74 %

b) الطلب الثاني: الجهد اللازم لكي يعمل المحرك بسرعة 3 دورات بالثانية هو

V=(3/10)*3.7=1.11 V

وبالتالي تكون نسبة الدور للحصول على هذا الجهد هي

Duty cycle= V/5=1.11/5=0.222=22.2 %

أسئلة خيارات متعددة:

1. ما هي الفائدة من بت البداية وبت التوقف في الارسال التسلسلي UART؟
   - **A.** فصل وحدات المعطيات عن بعضها وكشف بداية كل وحدة.
   - **B.** تحديد سرعة تراسل المعطيات.
   - **C.** التحقق وصول المعطيات دون أخطاء عند المستقبل.
   - **D.** لتحديد اتجاه التراسل بين طرفيتين.

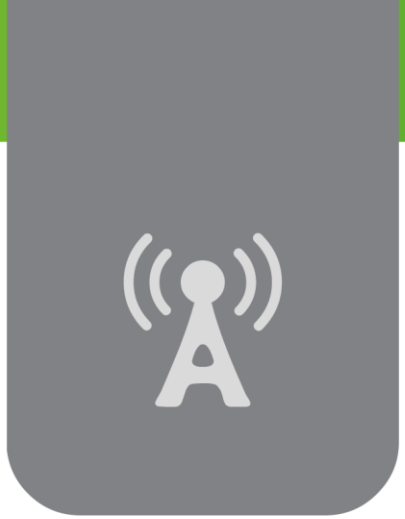
2. ماهي الفائدة من استعمال مقسم قابل للبرمجة في المؤقتات؟
   - **A.** زيادة التمييزية الزمنية للمؤقت.
   - **B.** توليد إشارة ساعة بنبضات منتظمة للمؤقت.
   - **C.** زيادة عدة بتات العد على خرج المؤقت.
   - **D.** زيادة مجال المؤقت.

3. ما هي الفائدة من مؤقت الملاحقة (Watchdog Timer)؟
   - **A.** يستعمل لقياس زمن تنفيذ البرنامج.
   - **B.** يستعمل لإعادة اقلاع النظام عند الدخول في حلقات لانهائية.
   - **C.** يستعمل لقياس الزمن بين نبضتين في إشارة خارجية.
   - **D.** يستعمل لتزويد النظام بالزمن الحقيقي.

4. يمكن استعمال معدل عرض النبضة PWM من أجل:
   - **A.** التحكم في شدة إضاءة مظهر ضوئي.
   - **B.** التحكم بسرعة محرك تيار مستمر.
   - **C.** ارسال الأوامر إلى طرفيات.
   - **D.** جميع ما سبق.

الإجابات الصحيحة:

رقم التمرين	الإجابة الصحيحة
1	A
2	D
3	B
4	D



الفصل التاسع:
الذواكر

الكلمات المفتاحية:

ذاكرة ROM، ذاكرة RAM، ذاكرة DRAM، ذاكرة EEPROM، ذاكرة Flash، تركيب الذواكر، وحدة إدارة الذاكرة.

ROM Memory, RAM Memory, DRAM Memory, EEPROM Memory, Flash Memory, Composing Memories, Memory Management Unit.

الملخص:

يهدف هذا الفصل إلى التعريف بتصنيف الذواكر وخواصها الأساسية من حيث قابلية الكتابة ودوام التخزين. نتعرف على الأنواع الشائعة للذواكر ومبدأ عملها ويشمل ذلك على الذواكر القابلة للقراءة فقط (ROM) والذواكر القابلة للقراءة والكتابة (RAM). ثم نشرح طريقة تركيب الذواكر للحصول على الذاكرة بالقياس المطلوب باستعمال الذواكر المتوفرة. ونختم الفصل بالتعرف على دور وحدة إدارة الذاكرة في النظم الحاسوبية.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- تصنيف الذواكر وفقاً لقابلية الكتابة ودوام التخزين
- الأنواع الشائعة للذواكر وبنيتها الداخلية
- طرق تركيب الذواكر

يهدف هذا الفصل إلى التعريف بتصنيف الذواكر وخواصها الأساسية من حيث قابلية الكتابة ودوام التخزين. نتعرف على الأنواع الشائعة للذواكر ومبدأ عملها ويشمل ذلك على الذواكر القابلة للقراءة فقط (ROM) والذواكر القابلة للقراءة والكتابة (RAM). ثم نشرح طريقة تركيب الذواكر للحصول على الذاكرة بالقياس المطلوب باستعمال الذواكر المتوفرة. ونختم الفصل بالتعرف على دور وحدة إدارة الذاكرة في النظم الحاسوبية.

1. مقدمة:

يقوم عمل النظم المضمنة على ثلاثة مفاهيم:

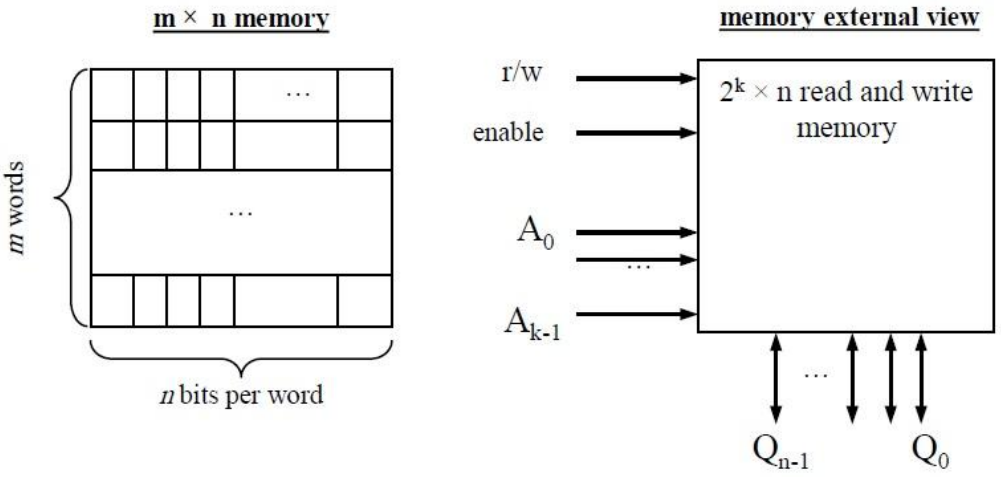
1. المعالجة.
2. التخزين
3. الاتصال.

حيث تعمل المعالجة على تحويل المعطيات، ويقوم التخزين بحفظ المعطيات للاستعمال اللاحق، ويعمل الاتصال على نقل المعطيات.

نستعمل المعالجات لتنفيذ عمليات المعالجة، والذواكر لتخزين المعطيات، والمساري لنقل المعطيات.

عرضنا في الفصول السابقة المعالجات بأنواعها المختلفة، وسنعرض في هذا الفصل الذواكر.

لنبدأ أولاً بعرض المفهوم الأساسي للذاكرة. تخزن الذاكرة مجموعة كبيرة من البتات. يمكن تقسيم هذه البتات إلى كلمات (words) يتكون كل منها من n بت، وتحتوي الذاكرة على m كلمة بحيث يكون لدينا بالمجمل m×n بتاً. نوصّف هذه الذاكرة من حيث طريقة تنظيمها على أنها ذاكرة m×n. لتحديد موقع كلمة فإننا نستعمل $\log_2(m)$ خط عنونة، فمثلاً، تحتوي الذاكرة 8×4096 على 32768 بتاً وتحتاج إلى 12 خط عنونة لتحديد موقع أي كلمة مكونة من 8 بتات.



تنظيم الذاكرة إلى كلمات وبتات

عملية القراءة من الذاكرة تعني عملية استرجاع كلمة من عنوان محدد، أما عملية الكتابة فتعني تخزين كلمة في عنوان محدد. نعرّف عملية النفاذ (Access) بأنها عملية قراءة أو عملية كتابة. تحتاج الذاكرة إلى إشارة تحكم يرمز لها بالرمز r/w لتحديد نوع عملية النفاذ. تمتلك معظم الذواكر مدخلاً تحكمياً يدعى Enable لتأهيل الذاكرة، بمعنى أن الذاكرة لا تستجيب لأي عملية نفاذ إلا عندما يكون هذا المدخل مفعلاً، وغالباً ما يكون تفعيل هذا المدخل بالمنطق المنخفض أي عندما Enable=0.

تدعم بعض أنواع الذواكر والمسماة بالذواكر المتعددة البوابات (multiport memory) النفاذ إلى عدة مواقع مختلفة في نفس الوقت، ولذلك نجد لهذا النوع من الذواكر مساري عناوين ومعطيات متعددة مع الإشارات التحكمية المرافقة. حيث نطلق هنا اسم بوابة port على مجموعة مكونة من مسرى عناوين ومسرى معطيات وإشارات التحكم المرافقة لهذه المساري.

تطورت الذواكر بشكل كبير في العقود الأخيرة، وكان التطور الأساسي في سعة الذواكر. حيث سمح هذا التطور بظهور الكثير من النظم المضمنة المتقدمة مثل الكاميرات الرقمية والهواتف النقالة الذكية.

2. تصنيف الذواكر:

صُنّفت الذواكر إلى نوعين أساسيين:

- ذواكر قابلة للقراءة فقط *ROM* (*Read Only Memory*):

وهي ذواكر يمكن للمعالج القراءة منها فقط، وتتصف بأنها تحافظ على محتواها بشكل دائم حتى في غياب التغذية الكهربائية عنها لذلك نصف هذه الذواكر بغير المتطايرة (Non volatile)، كما توصف هذه الذواكر أحياناً بالذواكر الميتة نظراً لأن محتواها لا يتغير.

- ذواكر النفاذ العشوائي *RAM* (*Random Access Memory*):

وهي ذواكر يمكن للمعالج القراءة منها والكتابة فيها، ولكنها تفقد محتواها عند قطع التغذية الكهربائية عنها، لذلك نصف هذه الذواكر بالمتطايرة (Volatile)، كما يطلق على هذه الذواكر اسم الذواكر الحية لأن محتواها يتغير كلما كتب المعالج فيها.

مع تطور الذواكر، لم يعد هذا التصنيف معبراً، حيث تم تطوير ذواكر غير متطايرة ومع ذلك فهي قابلة للكتابة من قبل المعالج كما سنرى لاحقاً في هذا الفصل، لذا سنترك هذا التصنيف وسنعتمد التمييز بين الذواكر استناداً إلى خاصتين وهما: قابلية الكتابة (write ability) ودوام التخزين (storage permanence).

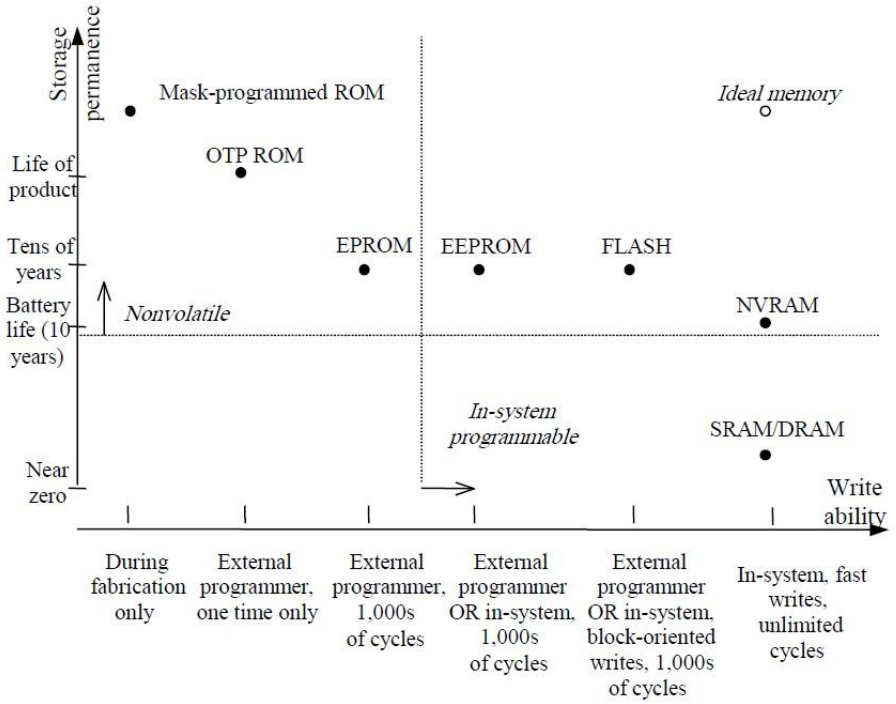
1.2.قابلية الكتابة:

نستعمل خاصة قابلية الكتابة (write ability) لتوصيف كيفية وسرعة الكتابة في موقع ما من الذاكرة. من الطبيعي أن تكون كل الذواكر قابلة للكتابة في مرحلة ما، وإلا فلن يكون لها أي فائدة، لكن يكمن الاختلاف في طريقة الكتابة وسرعة عملية الكتابة.

في السوية العليا وفق هذه الخاصية نجد ذواكر يمكن الكتابة فيها ببساطة وسرعة بمجرد وضع المعطيات والعناوين على المساري الموافقة للذاكرة وإعطاء إشارة الكتابة. وفي السوية المتوسطة فنجد ذواكر تأخذ زمناً أطول لإتمام عملية الكتابة. أما في السوية الدنيا فلدينا ذواكر لا نستطيع الكتابة فيها إلا باستخدام جهاز خاص يسمى المُبرمِجة (programmer) وهي التي تقوم بعملية الكتابة في الذاكرة بتطبيق جهود خاصة عليها وبتسلسل خطوات معين لإتمام العملية.

2.2. دوام التخزين:

نستعمل خاصة دوام التخزين (storage permanence) لِنَصف قابلية الذاكرة للاحتفاظ بالمعطيات التي كُتبت فيها. في السوية الدنيا وفق هذه الخاصية نجد الذواكر التي تبدأ بفقد محتواها بعد وقت قصير جداً من الكتابة فيها مثل الذواكر الديناميكية (Dynamic RAM) ولذلك فهي تحتاج إلى إنعاش لمحتواها بشكل مستمر. بعد ذلك تأتي الذواكر التي تحافظ على محتواها طالما أن الذاكرة موصولة بالتغذية الكهربائية، ومن ثم تأتي الذواكر التي تحتفظ بالمعطيات لأيام أو أشهر أو حتى عدة سنوات بعد فصل التغذية عنها. وفي السوية العليا نجد الذواكر التي لا تفقد المعطيات المخزنة فيها أبداً.



تصنيف الذواكر حسب قابلية الكتابة ودوام التخزين

نستعمل مصطلحات الذواكر المتطايرة والذواكر غير المتطايرة لتقسيم الذواكر إلى قسمين وفق خاصية دوام التخزين. وبالمثل نستخدم مصطلح الذواكر القابلة للبرمجة ضمن النظام (in–system Programmable) لتقسيم الذواكر إلى قسمين وفق خاصية قابلية الكتابة، حيث إن الذواكر القابلة للبرمجة ضمن النظام هي الذواكر التي يستطيع النظام المضمّن الكتابة فيها أثناء العمل، أما القسم الآخر فيشمل الذواكر التي يتم الكتابة فيها بواسطة مبرمجة قبل تركيبها في النظام.

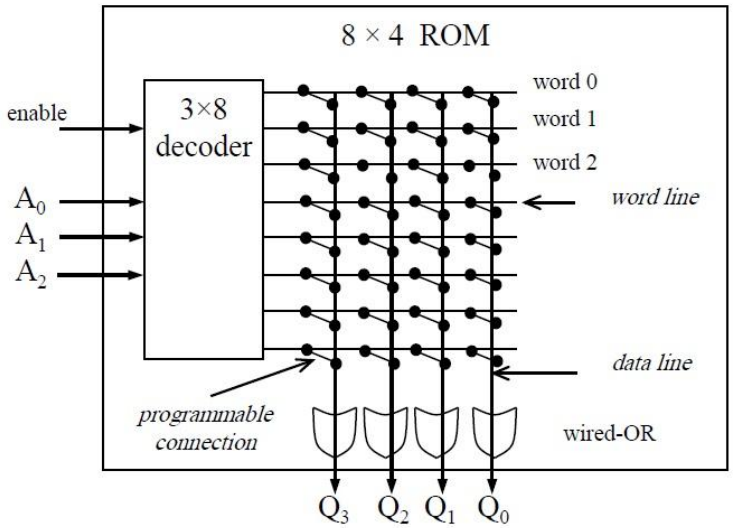
بالطبع هناك تعارض بين خاصية قابلية الكتابة وخاصية دوام التخزين، فهناك علاقة تناسب عكسي بين هاتين الخاصتين. إضافة إلى ذلك فالذواكر ذات القابلية العالية للكتابة تكون أكبر حجماً وأكثر استهلاكاً للطاقة.

3.الأنواع الشائعة للذواكر:

1.3. الذواكر القابلة للقراءة فقط:

الذواكر القابلة للقراءة فقط ROM (Read Only Memory) هي في الأساس ذواكر لا يمكن للنظام المضمّن الكتابة فيها ولكن يتم برمجتها بشكل مسبق عند التصنيع أو باستعمال مبرمجة قبل تركيبها في النظام. من جهة ثانية، فهناك ذواكر تندرج ضمن تصنيف الذواكر ROM ولكن يمكن للنظام المضمن الكتابة فيها أثناء العمل. لذلك، وكما ذكرنا سابقاً، فهناك التباس في تصنيف بعض الذواكر على أنها من النمط ROM. ومع ذلك تشترك جميع الذواكر المصنّفة تحت هذا النمط بخاصتين وهما أنها ذواكر غير متطايرة، وأن قابلية الكتابة فيها منخفضة مقارنة بالذواكر من نمط RAM.

تُستعمل الذواكر من نمط ROM لأغراض متعددة مثل تخزين برنامج المعالج العام، وتُستعمل أيضاً لتخزين المعطيات الثابتة التي يحتاجها النظام مثل معامِلات تشغيل النظام أو العبارات النصية التي تظهر على الشاشة. كم أن هناك استعمال آخر أقل شيوعاً وهو تنفيذ الدارات التراكبية المنطقية من خلال برمجة الذاكرة بجدول الحقيقة للتابع المنطقي المرغوب، حيث نستطيع تنفيذ أي دارة تراكبية ذات k مدخل و n مخرج باستعمال ذاكرة ROM $2^k \times n$.



البنية الداخلية للذواكر ROM

يبين الشكل المخطط الصندوقي لذاكرة ROM 4×8. تتكون الذاكرة من مفكك ترميز 3×8 ينتخب صفاً من الصفوف الثمانية المربوطة مع مخارجه حسب قيمة خطوط العنونة. يمثل كل صف كلمة معطيات، ويرتبط كل صف بمخارج الذاكرة عن طريق وصلات قابلة للبرمجة. تمثل كل وصلة بتاً واحداً من كلمة المعطيات، حيث نضع وصلة بين الصف والمخرج إذا كان البت الموافق من الكلمة هو 1، بينما لا نضع أي وصلة إذا كان البت هو 0.

لاحظ وجود رمز OR منطقية على كل خط خرج، وهذا يعني تمثيلياً أن قيمة هذا المخرج هي عملية OR منطقية بين جميع الصفوف. لفهم عمل الذاكرة، نفترض أن دخل العناوين هو 010. هذا يعني أن قيمة خرج

مفكك الترميز المقابل للكلمة Word2 هي 1، بينما تكون بقية الصفوف على قيمة 0. وهكذا نجد القيمة 1010 على الخرج التي تعبر عن محتوى ذاكرة الموقع الموافق لعنوان الدخل.

كيف يتم برمجة الوصلات المبرمجة؟ يعتمد هذا على نوع الذاكرة. سنذكر فيما يلي أهم أنواع الذواكر من نمط ROM وفقاً للترتيب التصاعدي لقابلية الكتابة.

1.1.3. ذاكرة ROM مبرمجة عند التصنيع:

في هذا النوع من الذواكر تتم برمجة الوصلات خلال عملية تصنيع الذاكرة (Mask-Programmed ROM) بإنشاء هذه الوصلات فيزيائياً حسب المعطيات التي نريد وضعها في هذه الذاكرة. بالنتيجة، لا يمكن تغيير محتوى هذه الذاكرة أبداً. يُستخدم هذا النوع من الذواكر في النظم النهائية ذات الإنتاج الكمي الكبير.

2.1.3. الذواكر القابلة للبرمجة لمرة واحدة:

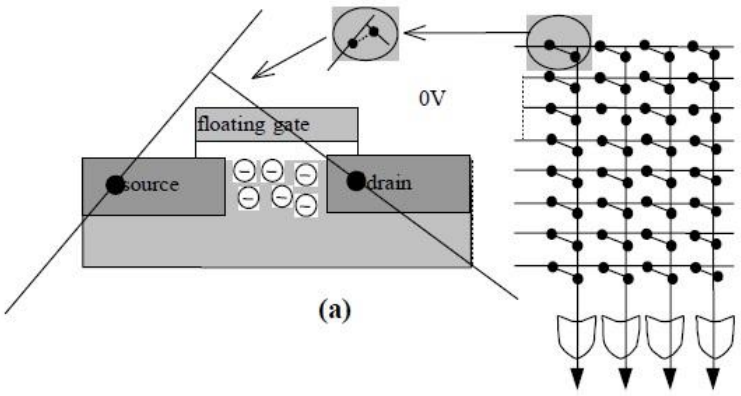
الذواكر القابلة للبرمجة لمرة واحدة OTP ROM (One-Time Programmable ROM) هي ذواكر قابلة للبرمجة من قِبل المستخدم وليس المصنّع، لذا فهي تسمى بالذواكر القابلة للبرمجة PROM ROM (Programmablz). تُصنع هذه الذواكر فارغة من أي محتوى، وعندما يُكتب في هذه الذاكرة لا يمكن محي محتواها وإعادة برمجتها مرة أخرى، ولذلك فهي مناسبة للنظم المضمّنة النهائية ذات الإنتاج الكمي المحدود.

يعتمد مبدأ تصنيع هذه الذواكر على استعمال فاصمات منصهرة (Fuses) لبناء الوصلات بين الصفوف والأعمدة في الذاكرة. عند التصنيع، توضع جميع الوصلات، وبالتالي تكون قيمة كل البتات في الذاكرة هي القيمة 1. لتغيير محتوى الذاكرة نستعمل مبرمِجة الذواكر التي تقوم بحرق (أي قطع) الفاصمات المنصهرة في البتات التي نريد برمجتها بقيمة 0. هذه العملية غير عكوسة، أي لا يمكن إعادة الوصلة بعد أن تم قطعها بالصهر، ولهذا السبب تسمى أحياناً عملية برمجة الذاكرة بهذه الطريقة عملية "حرق الذاكرة" (Memory Burning).

تُعتبر هذه الذواكر ذات قابلية كتابة منخفضة لأنها تحتاج إلى مبرمجة، إلا أن فترة دوام التخزين فيها طويلة جداً. مع ذلك يمكن أن يتغير محتوى هذه الذاكرة نتيجة تعرضها للإشعاع الكهرطيسي أو سوء الاستعمال الذي قد يؤدي إلى قطع فاصمات إضافية، أي إنه لا يمكن ضمان المحافظة على محتواها مقارنة بالذواكر المبرمجة عند التصنيع. تتميز هذه الذواكر أيضاً بانخفاض كلفتها فهي أرخص أنواع الذواكر القابلة للبرمجة.

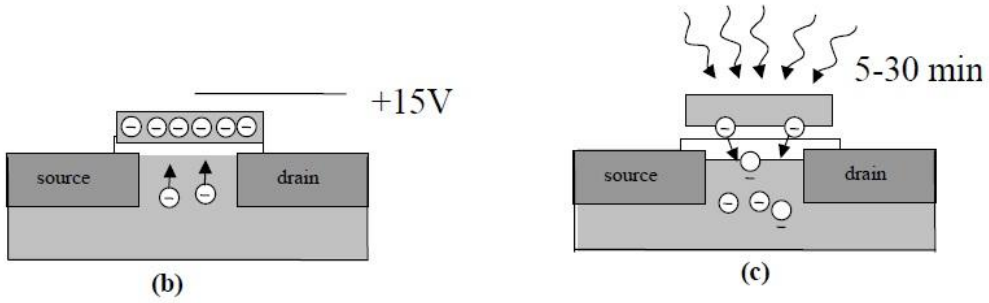
3.1.3. الذواكر القابلة للمحي والبرمجة:

هناك نوع آخر من ذواكر PROM وهي الذواكر القابلة للمحي EPROM (Erasable PROM). تُستعمل في هذه الذواكر ترانزستورات MOS لإنشاء الوصلات القابلة للبرمجة، حيث تكون قاعدة الترانزستور عائمة (Floating Gate)، أي معزولة وغير موصولة بأي شيء.



مبدأ تصنيع الذواكر EPROM

لإنشاء الوصلة نجعل الترانزستور ناقلاً عن طريق سحب الإلكترونات من القاعدة إلى العازل بتطبيق جهد مرتفع نسبياً (من 12v إلى 25v) قريباً من القاعدة، مما يؤدي إلى ارتفاع جهد القاعدة وهذا ما يجعل الترانزستور ناقلاً. عند إزالة جهد البرمجة تبقى الإلكترونات محجوزة في العازل، وهكذا نكون قد برمجنا الذاكرة.



مبدأ كتابة ومحي الذواكر EPROM

من أجل محي محتوى الذاكرة، يجب تحريض الإلكترونات على الخروج من العازل إلى القاعدة. يتم ذلك بتطبيق أشعة فوق بنفسجية UV (Ultraviolet) على شريحة الذاكرة لفترة تمتد عادة من 5 إلى 30 دقيقة. ولكي يصل الضوء إلى الشريحة يتم وضع نافذة زجاجية فوق الشريحة في غلاف تعليب الذاكرة.

يمكن محي الذاكرة EPROM وإعادة برمجتها آلاف المرات، وهي تحافظ على محتواها لمدة 10 سنوات على الأقل.

بالمقارنة مع الذواكر السابقة، تتميز هذه الذاكرة بقابلية كتابة أفضل ولكن في المقابل تكون مدة دوام التخزين أقل، كما أنها عرضة لتغيير محتواها بشكل أكبر نتيجة للإشعاع والضجيج الكهربائي، لذلك يجب تغطية النافذة بلاصق لمنع الضوء من الدخول إلى الشريحة بعد برمجتها وتركيبها في النظام.

4.1.3. الذواكر القابلة للمحي والبرمجة كهربائياً:

تم تطوير الذواكر القابلة للمحي والبرمجة كهربائياً في بداية الثمانينات من القرن الماضي بهدف التخلص من تقنية المحي باستعمال الأشعة فوق البنفسجية التي تأخذ وقتاً طويلاً. وتعتمد التقنية الجديدة على تطبيق جهود عالية نسبياً لإجراء عملية المحي ولكن لفترة زمنية قصيرة جداً مقارنة بالزمن اللازم لمحي الذاكرة باستعمال الأشعة فوق البنفسجية. إضافة إلى ذلك، أصبح بالإمكان محي كلمة معينة من الذاكرة وليس الذاكرة كلها. في المقابل فإن هذه الذواكر هي ذات كلفة أعلى.

بما أن هذه الذواكر قابلة للمحي كهربائياً، أصبح من الممكن تزويد النظام المضمّن بالجهود والدارات الضرورية لإجراء عمليات المحي والبرمجة ضمن النظام أيضاً وليس فقط باستخدام مبرمِجة ذواكر، وبالتالي يتعامل النظام مع هذه الذاكرة كذاكرة قابلة للقراءة والكتابة. يُستعمل هذا النوع من الذواكر لحفظ بعض المعطيات الهامة في النظم المضمّنة والتي نرغب بالحفاظ عليها بعد قطع التغذية عن النظام. وكمثال على ذلك، نذكر أجهزة الهاتف القادرة على تخزين الأرقام المفضلة لدينا بحيث يحافظ الهاتف على هذه الأرقام حتى ولو قمنا بفصله عن مقبس الهاتف وتغيير مكانه.

تأخذ عملية القراءة من الذاكرة EEPROM (Electrically Erasable Programmable ROM) زمناً أقل من ميكروثانية، بينما تتطلب عملية الكتابة زمناً أطول قد يصل إلى رتبة الميلي ثانية أو أكثر وذلك لتلبية متطلبات المحي وإعادة البرمجة، لذا، يتم عادةً تزويد هذه الذواكر بمتحكمات داخلية (Microcontrollers) تتولى مهمة الكتابة وتعطي مخرجاً على أحد مرابط الذاكرة يدعى busy للدلالة على أن المتحكم الداخلي لم يكمل بعد عملية الكتابة الحالية في الذاكرة، وبالتالي يجب على المعالج تفحص هذا المخرج قبل إعطاء أي أمر كتابة جديد في الذاكرة.

من التطبيقات الشائعة لهذه الذاكرة استعمالها كذاكرة برنامج ولاسيما في المتحكمات الصغرية، مع تزويدها بدارات حماية خاصة لمنع الكتابة فيها أثناء عمل النظام للحفاظ على محتوى البرنامج.

5.1.3.الذاكرة الومضية:

تُعتبر الذاكرة الومضية (Flash Memory) امتداداً للذاكرة EEPROM المبنية على مبدأ القاعدة العائمة. صُمّمت الذاكرة الومضية بحيث يمكن محو كتلة كبيرة من الذاكرة دفعة واحدة بدلاً من محو كلمة واحدة كل مرة. تحتوي الكتلة عادة على عدة آلاف من البايتات. لقد مكّن هذا الأمر من تسريع عملية الكتابة في الذاكرة، الأمر الذي وفّر بدوره للنظم المضمّنة إمكانية تخزين المعطيات الكبيرة في ذاكرة غير متطايرة خلال زمن قصير كما هي الحال في نظم الكاميرات الرقمية والهواتف الذكية والتجهيزات الطبية.

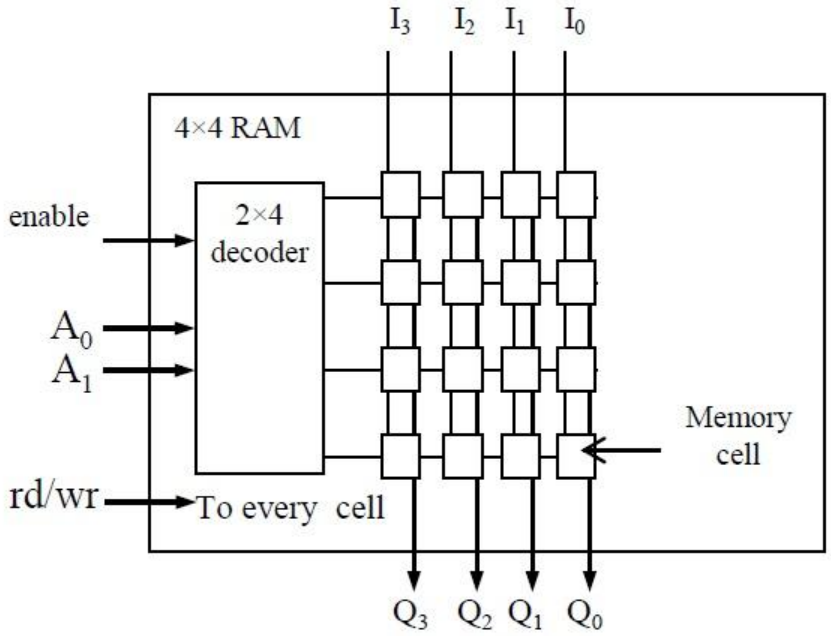
بشكل مماثل لذواكر EEPROM، يمكن إعادة الكتابة في الذواكر الومضية آلاف المرات، وهي قادرة أيضاً على الاحتفاظ بمحتوياتها لعشر سنين أو أكثر.

عندما نريد تغيير كلمة واحدة فقط من الذاكرة الومضية علينا قراءة كامل الكتلة التي تحتوي على هذه الكلمة، ومن ثمّ تغيير الكلمة المطلوبة وإعادة كتابة كامل الكتلة من جديد، وهذا يعني أن هذا النوع من الذواكر غير

مناسب للتطبيقات التي تحتاج إلى تخزين معطيات صغيرة ومتفرقة، لذلك يُفضل في هذه الأحوال استعمال ذواكر EEPROM عوضاً عن الذواكر الومضية.

2.3. الذواكر القابلة للقراءة والكتابة:

نستعرض في هذه الفقرة أنواع الذواكر المصنفة تحت اسم الذواكر القابلة للقراءة والكتابة RAM.
(Random Access Memory) تتميز هذه الذواكر بكون زمن الكتابة فيها مساوٍ لزمن القراءة وذلك خلافاً للذواكر ROM التي يكون فيها زمن الكتابة أطول من زمن القراءة. كما تتميز هذه الذواكر بكونها متطايرة، وبالتالي فهي لا تحتوي على أي معلومات مخزنة فيها سلفاً عند تركيبها في النظام، بل يقوم النظام بالكتابة فيها لأغراض التخزين المؤقت.



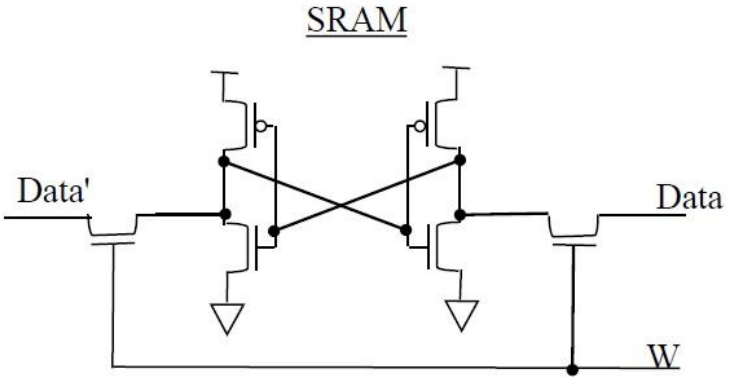
البنية الداخلية للذواكر RAM

يبين الشكل المخطط الصندوقي للذاكرة RAM. تتكون الذاكرة من مصفوفة من الخلايا (Cells). تُخزن كل خلية بتاً واحداً، ويشكل كل صف من الخلايا كلمة واحدة مكونة من عدد من البتات. للذاكرة مداخل عناوين موصولة مع مفكك الترميز الداخلي لانتخاب صف واحد من الخلايا. ترتبط أعمدة الخلايا بمدخل المعطيات I لإجراء عمليات الكتابة، كما ترتبط أيضاً بالمخرج Q لإخراج المعطيات عند إجراء عمليات القراءة. يوجد للذاكرة مدخل تحكمي rd / wr موصول بجميع الخلايا لتحديد العملية المطلوبة إما قراءة أو كتابة.

يوجد نوعان أساسيان من الذواكر RAM هما الذواكر الساكنة SRAM (Static RAM) والذواكر الديناميكية DRAM (Dynamic RAM).

1.2.3.الذواكر الساكنة:

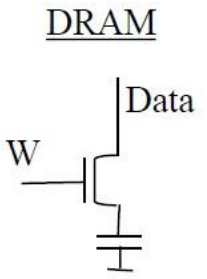
يبين الشكل التالي بنية خلية التخزين في الذاكرة SRAM (Static RAM) والتي تتكون من 6 ترانزستورات. تسمى هذه الذاكرة بالثابتة لأنها تحافظ على محتواها طالما أنها موصولة بالتغذية الكهربائية. تأخذ هذه الذاكرة حجماً كبيراً مقارنة مع الذواكر الأخرى لأنها تحتاج إلى 6 ترانزستورات لكل بت، إلا أنها تتميز بسرعة النفاذ لذلك غالباً ما تُستعمل كذاكرة مخبأة مع المعالج على نفس الشريحة.



بنية الخلية في الذواكر RAM الساكنة

2.2.3. الذواكر الديناميكية:

يبين الشكل التالي بنية خلية التخزين في الذاكرة DRAM (Dynamic RAM) والتي تتكون من ترانزستور واحد ومكثفة صغيرة لتخزين البت، وبالتالي يكون حجم الخلية أصغر بكثير مقارنة بحجم الخلية في الذاكرة SRAM، لذلك تكون سعات الذواكر المتوفرة من هذا النوع أكبر، ولكن بالمقابل تفقد المكثفة شحنتها تدريجياً مع الزمن بسبب التسريب، مما يسبب ضياع القيمة المخزنة. لمنع ضياع المعلومات المخزنة يجب إعادة شحن المكثفة بقيمتها الأساسية بشكل دوري، وهذا يسمى إنعاش refresh الذاكرة. عادةً ما يكون معدل الإنعاش الأصغري مرة كل 15 ميكروثانية.



بنية الخلية في الذواكر الديناميكية

بسبب التصميم الخاص لهذه الذاكرة فإن عملية قراءة كلمة من الذاكرة يسبب إنعاش الخلايا في هذه الكلمة. وبشكل محدد أكثر فإن عملية قراءة كلمة تشمل عملية تخزينها في صوان (Buffer) ومن ثم إعادة كتابة الصوان في خلايا الكلمة، ولهذا السبب يكون زمن النفاذ إلى الذواكر DRAM أكبر مقارنة بالذواكر SRAM.

3.2.3.الذواكر RAM شبه الساكنة:

الذواكر RAM شبه الساكنة PSRAM (Pseudo-Static RAM) هي في الأساس ذواكر ديناميكية DRAM مزودة بمتحكم إنعاش داخلي، وبالتالي تظهر للمستخدم وكأنها ذواكر SRAM. مع ذلك، تبقى هذه الذواكر بطيئة بسبب الحاجة إلى الانتظار عند إجراء عمليات النفاذ في الوقت الذي يكون فيه المتحكم الداخلي مشغولاً بعملية الإنعاش، لذلك تشكل الذواكر PSRAM بديلاً عن الذواكر SRAM بسعات تخزينية كبيرة وكلفة منخفضة وذلك عندما يكون زمن النفاذ غير مهم لتطبيق النظام المضمّن.

4.2.3. الذواكر RAM غير المتطايرة:

الذواكر RAM غير المتطايرة NVRAM (Non Volatile RAM) هي ذواكر SRAM مزودة ببطارية داخلية للحفاظ على محتوى الذاكرة عند انقطاع التغذية، وعادة ما تكون سعة هذه البطارية كافية للحفاظ على محتوى الذاكرة لفترة طويلة تصل إلى 10 سنوات. لذلك يمكن اعتبار الذاكرة من هذا النوع غير متطايرة مع سرعة نفاذ عالية وعدم وجود أي حدود على عدد مرات الكتابة فيها، ولكنها تبقى أكثر عرضة لتغير محتواها مقارنة بالذواكر EEPROM بسبب الضجيج الكهربائي.

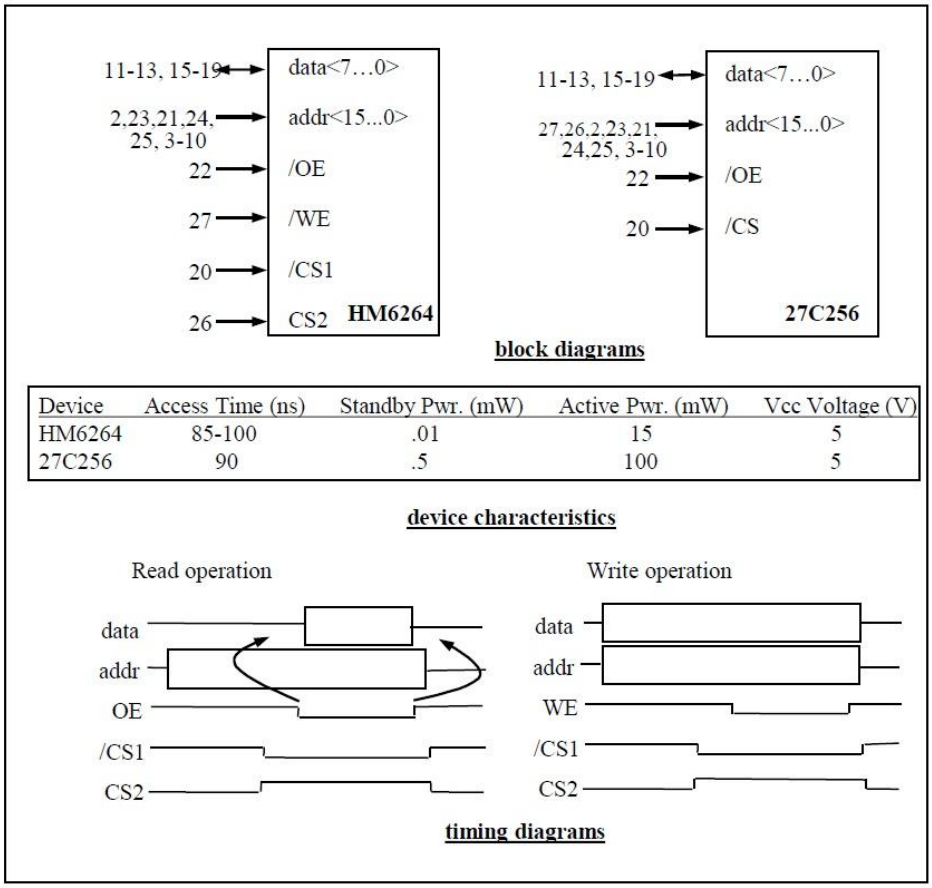
هناك أيضاً نوع من الذواكر RAM غير المتطايرة التي تحتوي على ذاكرتين بنفس الحجم، إحداهما ذاكرة RAM والأخرى ذاكرة EEPROM، حيث يتم تخزين محتوى ذاكرة RAM في الذاكرة EEPROM فوراً قُبيل انقطاع التغذية عن الذاكرة، ومن ثم استعادة هذا المحتوى عند عودة التغذية.

3.3.أمثلة على الذواكر:

نذكر في هذه الفقرة مثالين عن الذواكر المتوافرة والمستخدمة بشكل شائع في النظم المضمّنة التي تحتوي على معالجات على 8 بت. المثال الأول هو ذاكرة من نوع RAM ألا وهي HM6264، والمثال الثاني هو ذاكرة من نوع ROM ألا وهي 27C256.

من المتعارف عليه استخدام الرقم 62 في بداية رقم الذاكرة للإشارة إلى أنها من نوع RAM، واستخدام الرقم 27 للإشارة إلى أنها من نوع ROM، أما الأرقام التالية فتشير إلى سعة الذاكرة مقدرة بوحدة الكيلوبايت. فالذاكرة HM6264 هي ذاكرة تحتوي على 64 KB، والذاكرة 27C256 تحتوي على 256 KB.

يبين الشكل التالي هذه الذواكر ومواصفاتها والمخططات الزمنية لعمليات القراءة والكتابة.



المخطط الصندوقي و المخطط الزمني لنوعين من الذواكر

4. تركيب الذواكر:

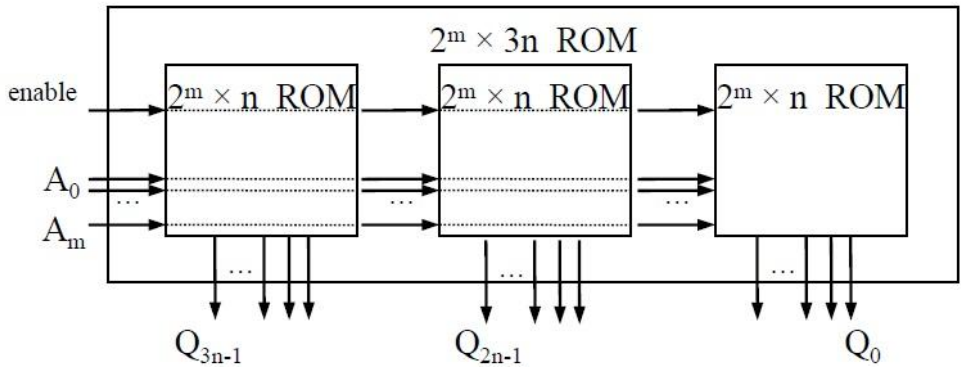
غالباً ما يواجه مصمم النظم المضمّنة مشكلة الحاجة إلى ذاكرة من قياس معين في الوقت الذي تكون فيه الذواكر المتوفرة لديه من قياس مختلف.

من السهل التعامل مع الحالة التي يكون فيها قياس الذاكرة المتوفرة أكبر من القياس المطلوب، إذ يكفي استخدام جزء من الذاكرة المتوفرة لتلبية احتياجه. فمثلاً إذا كان لديه ذاكرة ROM 16×2^{10} وهو يحتاج إلى ذاكرةROM 8×2^{8} فيكفي أن يأخذ البتات الثمانية الدنيا من كلمة الذاكرة المخزنة على 16 بتاً. كما يكفي أن يأخذ أول 256 كلمة من الذاكرة التي تحتوي على 1024 كلمة وذلك بعدم استخدام خطي العنونة الإضافيين، حيث يتم وصلهما بقيمة الصفر بشكل دائم.

أما في الحالة التي يكون فيها قياس الذاكرة المتوفرة أصغر من القياس المطلوب فهذا يتطلب بعض الجهد في التصميم. هناك طريقتين في التركيب حسب الحالة سنعرضهما تباعاً.

1.4. طريقة التركيب جنباً إلى جنب:

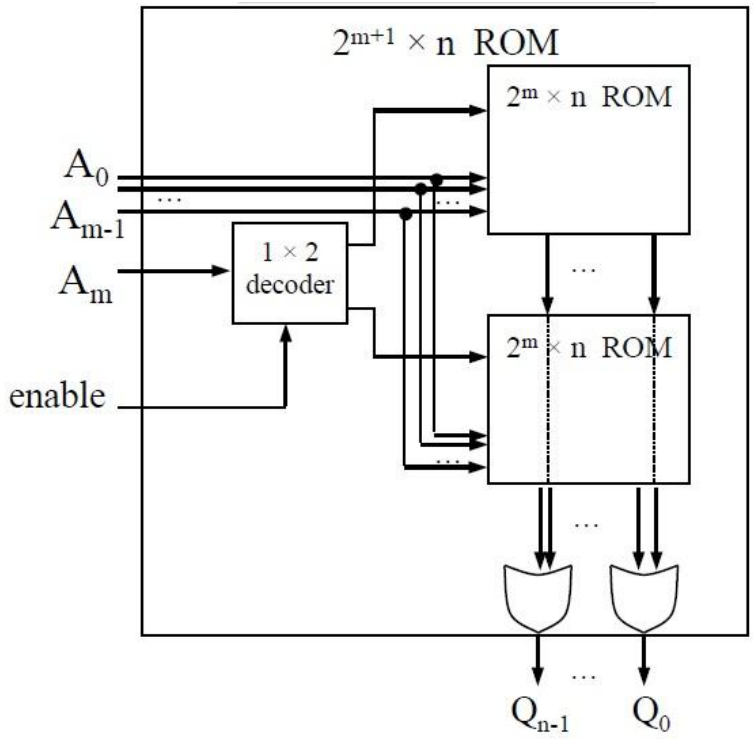
تُستعمل طريقة التركيب جنباً إلى جنب (Composing Side–by–Side) عندما يكون عرض كلمة الذاكرة المطلوب أكبر من عرض كلمة الذاكرة المتوفرة. فمثلاً إذا كنا نحتاج إلى ذاكرة بقياس 16×2^{10} ولكن قياس الذاكرة المتوفرة هو 8×2^{10} ففي هذه الحالة يتم وضع ذاكرتين من القياس المتوفر جنباً إلى جنب للحصول على كلمة بعرض 16 بتاً، ويتم وصل مسرى العناوين إلى الذاكرتين معاً. في هذه الطريقة تُخزَّن في الذاكرة الأولى البتات الثمانية الدنيا من كل كلمة، وتُخزَّن في الذاكرة الثانية البتات الثمانية العليا من كل كلمة. يمكن تعميم هذه الطريقة حتى نصل إلى عرض الكلمة المطلوب.



طريقة تركيب الذواكر لزيادة عدد البتات في الكلمة

2.4. طريقة التركيب من الأعلى إلى الأسفل:

تُستعمل طريقة التركيب من الأعلى إلى الأسفل (Composing Top to Bottom) عندما يكون عدد كلمات الذاكرة المطلوب أكبر من عدد كلمات الذاكرة المتوفرة ولكن لكليهما نفس عرض الكلمة. فمثلاً إذا كنا نحتاج إلى ذاكرة بقياس $2^9 \times 8$ وقياس الذاكرة المتوفر هو $2^8 \times 8$ ففي هذه الحالة يتم جمع ذاكرتين بحيث نربط مخارجهما عن طريق بوابة OR لكل مخرج، ويتم تخزين أول نصف والمكون من 256 كلمة في الذاكرة العلوية والنصف الآخر في الذاكرة السفلية، كما يتم وصل خطوط العنونة الثمانية الدنيا إلى الذاكرتين معاً. أما خط العنونة المتبقي (A_8) فيستعمل لانتخاب الذاكرة التي تحوي على الكلمة المطلوبة عن طريق مفكك ترميز 2×1 يتم وصل مخارجه بمدخل التأهيل لكل ذاكرة، فإذا كان $A_8=0$ فإن الكلمة المطلوبة موجودة في النصف الأول لذا يتم تفعيل الذاكرة العلوية فقط، أما إذا كان $A_8=1$ فإن الكلمة المطلوبة تقع في النصف الثاني لذا يتم تفعيل الذاكرة السفلية فقط. يمكن تعميم هذا الحال حتى نصل إلى عدد الكلمات المطلوب.



طريقة تركيب الذواكر لزيادة عدد الكلمات

3.4.الحالة العامة:

في الحالة العامة التي يكون فيها عرض كلمة الذاكرة المطلوب أكبر من عرض كلمة الذاكرة المتوفرة، وعدد كلمات الذاكرة المطلوب أكبر من عدد كلمات الذاكرة المتوفرة عندها نستعمل الطريقتين السابقتين معاً، أي يتم وصل الذواكر جنباً إلى جنب حتى نصل إلى عرض الكلمة المطلوب، ومن ثم نكرر هذه الذواكر عمودياً حتى نصل إلى عدد الكلمات المطلوب.

5. وحدة إدارة الذاكرة:

نختم هذا الفصل بالإشارة إلى وجود وحدات خاصة بإدارة الذاكرة في النظم التي تحتوي على ذواكر DRAM تدعى MMU (Memory Management Unit). تقوم هذه الوحدة بتنفيذ مهمة إنعاش الذاكرة وتحقيق المواءمة بين مسرى المعالج ومساري الذاكرة، وإدارة عملية النفاذ إلى الذاكرة من قِبل عدة معالجات عند مشاركة الذاكرة بين هذه المعالجات. تحتوي المعالجات الحديثة غالباً على وحدات MMU مضمنة مع المعالج، وفي حال عدم وجود هذه الوحدة مع المعالج فهي تضاف إلى النظام كوحدة منفصلة.

6. خلاصة:

لقد سمح التطور الكبير في مجال الذواكر بتطوير الكثير من النظم الضمنة الحديثة. تتعدد أنواع الذواكر وتختلف في بنيتها الداخلية وخواصها. تنظم الذاكرة إلى عدد من الكلمات وكل كلمة على عدد من البتات. يمكن تصنيف الذواكر بالاعتماد على خاصية قابلية الكتابة ودوام التخزين. وهما خاصيتين متعاكستين. تختلف هذه الخواص من الذواكر القابلة للقراءة فقط بأنواعها إلى الذواكر القابلة للقراءة والكتابة. يزداد تكلفة الذاكرة والاستطاعة المستهلكة كلما زادت قابلية الكتابة فيها. يمكن تركيب الذواكر للحصول على القياس المطلوب للذاكرة باستعمال الذواكر بالقياسات المتوفرة.

7. أسئلة الفصل التاسع:

أسئلة عامة:

1. ما هو الفرق بين الذواكر ROM والذواكر RAM؟

ذاكرة ROM هي ذاكرة غير متطايرة تحافظ على محتواها بشكل دائم حتى في غياب التغذية الكهربائية عنها.

ذاكرة RAM هي ذاكرة متطايرة تفقد محتواها عند قطع التغذية الكهربائية عنها.

2. عدد 4 أنواع من ذواكر ROM بحسب قابلية الكتابة المتزايدة؟

ذاكرة ROM مبرمجة عند التصنيع

ذاكرة ROM قابلة للبرمجة لمرة واحدة.

ذاكرة EPROM.

ذاكرة EEPROM.

3. قارن بين الذواكر الحية الساكنة SRAM والذواكر الحية الديناميكية DRAM من حيث التعقيد العتادي والسعة ودوام التخزين؟

تعقيد الذواكر DRAM اقل.

سعات الذواكر المتوفرة من النوع DRAM أكبر.

زمن دوام المعلومات في ذاكرة DRAM قصير جداً لذلك تحتاج إلى انعاش دوري.

4. ما هي ميزة ذاكرة Flash؟

صُمّمت الذاكرة الومضية بحيث يمكن محو كتلة كبيرة من الذاكرة دفعة واحدة بدلاً من محو كلمة واحدة كل مرة. لقد مكّن هذا الأمر من تسريع عملية الكتابة في الذاكرة.

5. أي نوع من الذواكر هو الأفضل لبناء الذاكرة المخبأة في المعالج ولماذا؟

الذواكر الساكنة SRAM لأنها تتميز بسرعة النفاذ.

أسئلة خيارات متعددة:

1. ما هي عدد بتات عنونة ذاكرة 8×4096؟
   - A. 8.
   - B. 10.
   - C. 12.
   - D. 15.

2. ما هي عدد مخارج ذاكرة 8×4096؟
   - A. 8.
   - B. 10.
   - C. 12.
   - D. 15.

3. تحتاج الذاكرة من DRAM إلى انعاش دوري بمعدل:
   - A. مرة كل 1 ثانية.
   - B. مرة كل 10 ميلي ثانية.
   - C. مرة كل 5 ميلي ثانية.
   - D. مرة كل 15 ميكرو ثانية.

4. تتكون خلية التخزين في الذاكرة SRAM من:
   - A. ترانزستور واحد.
   - B. 2 ترانزستور.
   - C. 6 ترانزستور.
   - D. 8 ترانزستور.

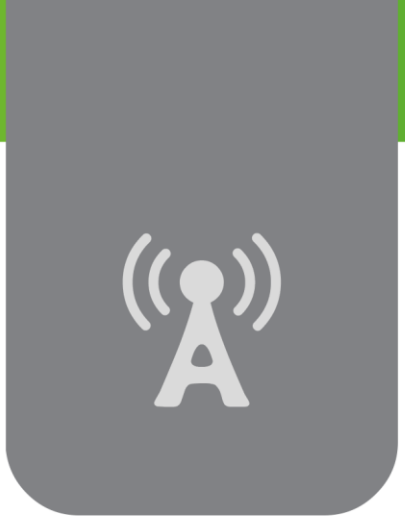
5. الذاكرة PSRAM هي ذاكرة من نوع:
   - A. ROM
   - B. SRAM
   - C. DRAM
   - D. NVRAM

6. الذواكر NVRAM هي ذاكرة من نوع
   A. ROM.
   B. DRAM مع دارة انعاش داخلية.
   C. EEPROM مع ذاكرة RAM
   D. SRAM مع بطارية داخلية.

7. نستعمل طريقة تركيب الذواكر جنباً إلى جنب من أجل:
   A. لزيادة عدد كلمات الذاكرة.
   B. لزيادة عدد بتات الكلمة.
   C. لزيادة فضاء عنونة الذاكرة.
   D. جميع ما سبق.

الإجابات الصحيحة:

رقم التمرين	الإجابة الصحيحة
1	C
2	A
3	D
4	C
5	C
6	D
7	B



الفصل العاشر:
المواجهة باستعمال المساري

الكلمات المفتاحية:

المسرى، العقدة، التنضيد الزمني، بروتوكولات الاتصال عبر المسرى، الدخل والخرج المحجوز من الذاكرة، تحكيم الأولوية، تحكيم سلسلة ديزي، المسرى متعدد السويات.

Bus, Node, Time Multiplexing, Bus Communication Protocols, Memory-Mapped I/O, Priority arbitration, Daisy-Chain arbitration, Multilevel Bus.

الملخص:

يهدف هذا الفصل إلى التعريف بكيفية ربط المعالج بالذواكر والطرفيات عن طريق المساري والمفاهيم المتعلقة بذلك. يتعرف الطالب على عناصر المسرى وبروتوكولات النقل بين عقدتين عبر المسرى مثل بروتوكول القدح وبروتوكول المصافحة. كما يتعرف الطالب على كيفية تنظيم عمليات الدخل والخرج والتعامل مع الذاكرة من قبل المعالج. ثم نشرح آليات التحكيم المستعملة عند وجود العديد من العقد على نفس المسرى مثل تحكيم الأولوية وتحكيم سلسة ديزي. وكيفية إدارة المسرى لمنع التصادم بين العقد المختلفة. وكذلك بنية المسرى المتعدد السويات لتلبية متطلبات سرعة النفاذ حسب نوع الطرفيات المستعملة في النظام المضمن.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- أساسيات الاتصال عبر المسرى ومفاهيم أساسية في بروتوكولات الاتصال
- مواجهة المعالجات مع الذواكر
- مواجهة المعالجات مع بوابات الدخل والخرج
- طرق التحكيم في تخديم عدة طرفيات من قبل نفس المصدر
- بنية المسرى المتعدد السويات

يهدف هذا الفصل إلى التعريف بكيفية ربط المعالج بالذواكر والطرفيات عن طريق المساري والمفاهيم المتعلقة بذلك. يتعرف الطالب على عناصر المسرى وبروتوكولات النقل بين عقدتين عبر المسرى مثل بروتوكول القدح وبروتوكول المصافحة. كما يتعرف الطالب على كيفية تنظيم عمليات الدخل والخرج والتعامل مع الذاكرة من قبل المعالج. ثم نشرح آليات التحكيم المستعملة عند وجود العديد من العقد على نفس المسرى مثل تحكيم الأولوية وتحكيم سلسة ديزي. وكيفية إدارة المسرى لمنع التصادم بين العقد المختلفة. وكذلك بنية المسرى المتعدد السويات لتلبية متطلبات سرعة النفاذ حسب نوع الطرفيات المستعملة في النظام المضمن.

1. مقدمة:

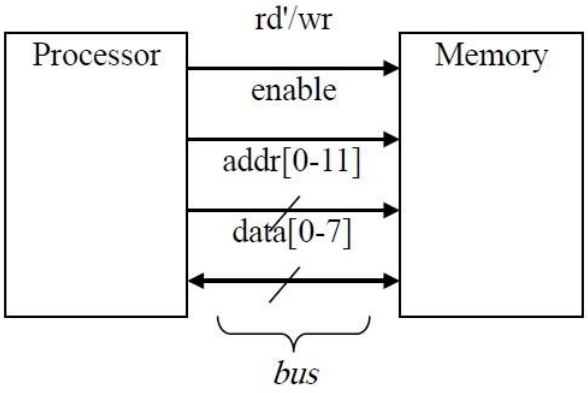
ذكرنا سابقاً بأن النظم المضمنة تنطوي على ثلاثة مفاهيم أساسية وهي المعالجة (Processing) وهي عملية معالجة المعلومات التي تقوم بها المعالجات والتخزين (Storage) وهي عملية حفظ المعلومات الذي يتم باستعمال الذواكر والاتصال (Communication) بين مكونات النظام عن طريق المساري (Buses).

المواجهة (interfacing) هي آلية نقل المعلومات بين المعالجات والذواكر والطرفيات باستعمال المساري. وتعتمد هذه العملية على ثلاثة مفاهيم وهي: العنونة (addressing) والتحكيم (arbitration) والبروتوكول (Protocol). كما نطلق تسمية "الواجهة البينية للمسرى" (bus interface) على الدارات الداخلية في عقدة متصلة بالمسرى لإدارة بروتوكول التراسل على المسرى.

سنعرض في هذا الفصل أساسيات الاتصال عن طريق المساري والبروتوكولات المتعلقة بذلك. سنبدأ بعرض المفاهيم الأساسية للاتصال. ثم سنعرض طرق ربط المعالج بالذواكر والطرفيات.

2. أساسيات الاتصال:

سنبدأ بعرض مثال بسيط عن طريقة وصل معالج مع ذاكرة. يبين الشكل التالي بنية المسرى الذي يربط بين المعالج والذاكرة.

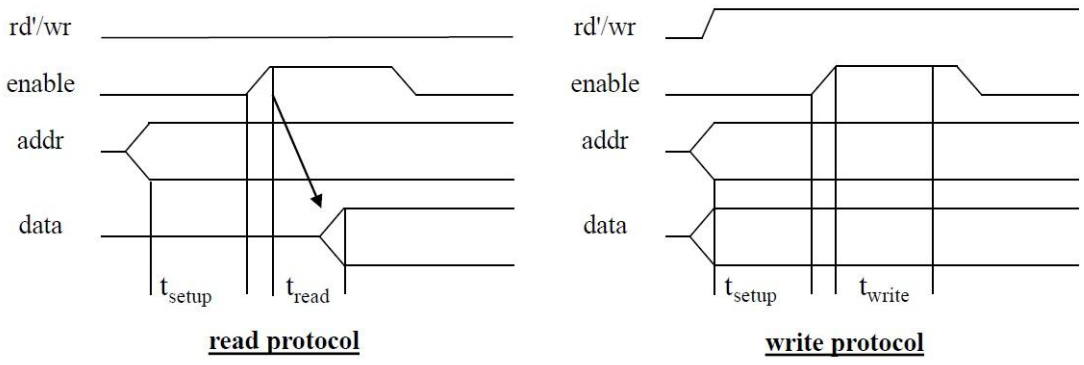


بنية المسرى

يتكون المسرى من:

- **خطوط عنونة Addr:** وتحتوي على 12 خط يضعها المعالج لتحديد الموقع الذي يريد النفاذ اليه ضمن الذاكرة وهي خطوط خرج من المعالج.
- **خطوط المعطيات data:** وتحتوي على 8 خطوط يضعها المعالج عند القيام بعملية كتابة في الذاكرة وتكون عندها هذه الخطوط خرج من المعالج، بينما تضعها الذاكرة عند القيام بعملية قراءة من الذاكرة وتكون عندها هذه الخطوط دخل للمعالج. لذلك تكون هذه الخطوط ثنائية الاتجاه حسب نوع عملية النفاذ.
- **خطوط التحكم:** وهي الخط rd/wr للتمييز بين عملية القراءة وعملية الكتابة، وخط enable لتفعيل الذاكرة. وهي خطوط خرج من المعالج.

يبين الشكل التالي المخطط الزمني الذي يبين بروتوكول النفاذ في عملية الكتابة أو القراءة. ويحدد هذا المخطط تسلسل الإشارات مع الزمن لإتمام عملية النفاذ.



بروتوكول القراءة وبروتوكول الكتابة

في عملية القراءة يضع المعالج أولاً عنوان الذاكرة المرغوب على خطوط العنونة كما يضع الخط rd/wr=0 للدلالة على أنه يريد اجراء عملية قراءة من الذاكرة. ثم يرفع إشارة التأهيل enable =1 لتفعيل الذاكرة وهو يمثل إشارة قدح (strobe) للذاكرة وذلك بعد أن تستقر إشارات العناوين بزمن قدره t_{setup}. عندها تضع الذاكرة محتوى الموقع المطلوب على خطوط المعطيات. وقد يستغرق ظهور المعطيات على الخطوط بعد قدح الذاكرة .زمناً مقداره t_{read}. ثم يقرأ المعالج قيمة المعطيات ليعالجها في برنامجه.

في عملية الكتابة يضع المعالج أولاً عنوان الذاكرة المرغوب على خطوط العنونة كما يضع الخط rd / wr =1 للدلالة على أنه يريد إجراء عملية كتابة في الذاكرة ويضع أيضاً خطوط المعطيات بالقيمة التي يريد كتابتها في الذاكرة. بعد أن تستقر إشارات العناوين والمعطيات بزمن, يرفع إشارة التأهيل enable=1 لتفعيل الذاكرة. عندها تخزن الذاكرة المعطيات في الموقع المطلوب. وقد يستغرق تخزين المعطيات في الذاكرة زمناً مقداره t_{write}.

من هذا المثال البسيط، نوضح بعض المفاهيم العامة:

- تكون الخطوط أحادية الاتجاه (unidirectional) أي أنها تنقل المعطيات أو الإشارات باتجاه واحد مثل خطوط العنونة addr وإشارات التحكم rd / wr وenable، أو ثنائية الاتجاه (bidirectional) أي أنها تنقل المعطيات في الاتجاهين مثل خطوط المعطيات data. يتم الإشارة إلى مجموعة من الاشارات المتوازية ذات الوظيفة نفسها بخط ثخين مع خط مائل ورقم يدل على عدد هذه الخطوط مثل خطوط العنونة وخطوط المعطيات.
- يرمز مصطلح "المسرى" إلى مجموعة من الخطوط ذات الوظيفة الواحدة مثل مسرى العناوين أو مسرى المعطيات. وقد يستعمل أيضاً للدلالة إلى المجموعة الكاملة للخطوط المستعملة للربط أو الاتصال، أي المجموعة المكونة من خطوط العنونة وخطوط المعطيات وإشارات التحكم.
- يصف مصطلح "البرتوكول" القواعد التي يجب احترامها للاتصال على خطوط المسرى. نتعامل هنا مع البروتوكولات العتادية في الطبقة الفيزيائية والذي يتعلق بالقواعد المطبقة على الإشارات وذلك للتمييز عن البروتوكولات العالية المستوى مثل بروتوكولات التراسل التسلسلي أو برتوكولات الانترنت مثلاً.
- يصل المسرى بين البوابات (ports). حيث تدل البوابة على النهايات الفيزيائية في المعالج أو الذواكر التي يتم وصل الأسلاك بها. حيث تتكون البوابة من مربط (pin) واحد أو أكثر بحسب عدد خطوط المسرى أو الإشارات.
- من الطرق الشائعة للتعبير عن البروتوكول العتادي هو المخطط الزمني (timing diagram). حيث يبين لنا المخطط القواعد التالية في المثال السابق:
- يجب وضع الخط rd / wr على قيمة 0 للقراءة وعلى قيمة 1 للكتابة.
- يبين المخطط بخطين عاموديين الزمن الأصغري t_{setup} الفاصل بين وضع قيم خطوط العنونة ووضع إشارة التفعيل على الخط enable.

- يبين المخطط الزمن t_{read} الذي يجب أن ينتظره المعالج ليصبح محتوى الذاكرة موجوداً بشكل كامل على خطوط المعطيات. أي الزمن اللازم لتكون قيم خطوط المعطيات صالحة وصحيحة (valid).
- يتم التعبير عن صلاحية القيم الموجودة على مسرى معين بخطين متوازيين. أما خلاف ذلك فيتم تمثيله بخط افقي كما يظهر على الشكل من أجل مسرى العناوين ومسرى المعطيات.

في المثال السابق، نجد أن الخط التحكمي المستعمل لقدح الذاكرة enable يكون فعالاً على السوية العليا (active high) أي 1. ولكن في معظم البروتوكولات تكون خطوط التحكم فعالة عادةً على السوية الدنيا (active low). يتم الإشارة إلى ذلك من خلال اسم الإشارة بوضع علامة تنصيص مفردة بعد الاسم (enable') أو وضع خط مائل قبل الاسم (/enable) أو وضع خط علوي فوق الاسم ($\overline{\text{enable}}$). لذلك سوف نستعمل فيما يلي مصطلح "تفعيل" (assert) للدلالة إلى وضع الخط التحكمي على قيمته الفعالة. أي وضعه على قيمة 1 اذا كان هذا الخط فعالاً على السوية العليا أو وضعه على قيمة 0 اذا كان هذا الخط فعالاً على السوية الدنيا.

قد يحتوي برتوكول النقل على عدة برتوكولات جزئية (sub–protocol) مثل بروتوكول القراءة وبرتوكول الكتابة في المثال السابق. يسمى كل برتوكول جزئي بإجراء (transaction) أو دورة مسرى (bus cycle). حيث يمكن أن تحتوي دورة المسرى على عدة دورات ساعة.

3. مفاهيم أساسية في البرتوكولات:

يعتبر البروتوكول السابق بين المعالج والذاكرة واحد من البروتوكولات البسيطة. ولكن يمكن أن تكون البروتوكولات العتادية أعقد من ذلك بكثير. مع ذلك، نستطيع أن نفهمها بشكل أفضل بتعريف بعض المفاهيم الأساسية.

1.3. العقدة:

يربط المسرى بين أطراف أو عقد متعددة. وتمثل العقدة (node) المعالج أو الذاكرة أو أي تجهيز آخر. ويضم البروتوكول عادةً عقدتين: عقدة تسمى السيد (master) وعقدة تسمى الخادم (servant). السيد هو من يبادر بالنقل والخادم هو من يستجيب لطلب السيد ويدعى أحياناً بالتابع أو العبد (slave). في مثالنا السابق يلعب المعالج دور السيد والذاكرة دور الخادم. كما يمكن أن يكون الخادم معالجاً آخر. ولكن بشكل عام يكون المعالج هو السيد في أغلب الأحيان.

2.3. اتجاه المعطيات:

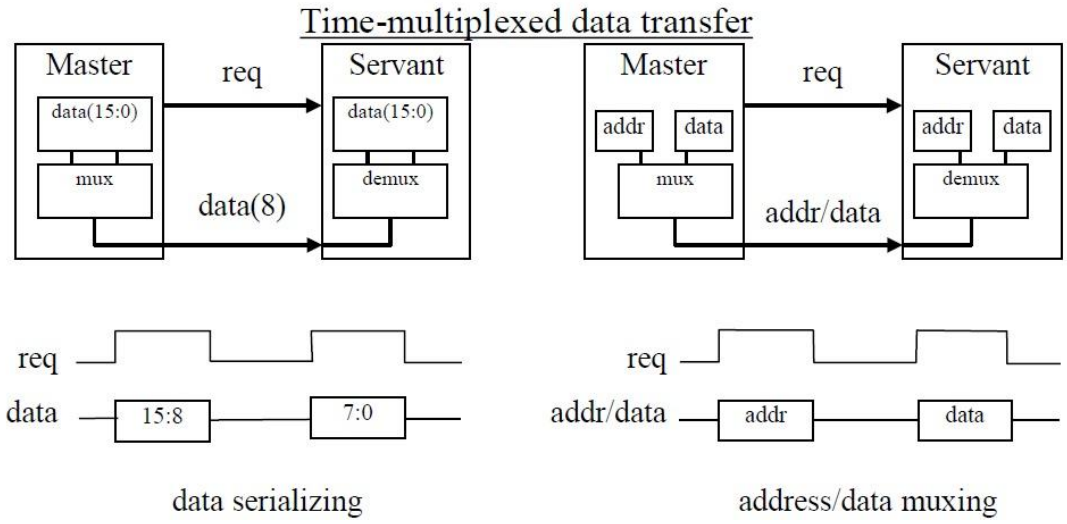
يعبر اتجاه المعطيات عن اتجاه حركة المعطيات بين العقد. ندل على هذا الاتجاه بوصف كل عقدة بأنها إما مرسلة أو مستقبلة للمعطيات. لاحظ أن نوع العقدة هو مستقل عن اتجاه المعطيات. وبالتحديد فإن السيد يمكن أن يكون مرسلاً أو مستقبلاً للمعطيات.

3.3. العناوين:

تمثل العناوين نوع خاص من المعطيات التي تشير إلى مصدر معلومة ما أو وجهتها. عند يتخاطب المعالج مع عدة طرفيات، فإن العنوان لا يحدد فقط هوية الطرفية فقط، ولكن يحدد أيضاً عنوان الموقع أو هوية السجل في هذه الطرفية التي يريد المعالج النفاذ اليه.

4.3. التنضيد الزمني:

يعني التنضيد الزمني (time multiplexing) مشاركة نفس المسرى لنقل قطع أو كلمات مختلفة من المعطيات وذلك بهدف تقليل عدد الأسلاك المطلوبة للنقل. مثلاً، يمكن استعمال مسرى معطيات يحتوي على 8 خطوط فقط لنقل كلمات بعرض 16 بت من السيد إلى الخادم وذلك بتقسيم كلمة المعطيات إلى قسمين كل منهما على 8 بت. حيث يتم ارسال البايت الأول في فترة زمنية معينة وارسال البايت الثاني في الفترة الزمنية التالية. وبهذه الطريقة تم توفير 8 خطوط. وتسمى هذه الطريقة بالإرسال المتسلسل. حيث يمكن تقليل عدد الأسلاك أكثر فأكثر وصولاً إلى خط واحد لنقل كلمة المعطيات بت تلو الآخر. وفي جميع الأحوال، يجب على الخادم إعادة تجميع كلمة المعطيات من القطع المنفصلة التي استقبلها على دفعات.

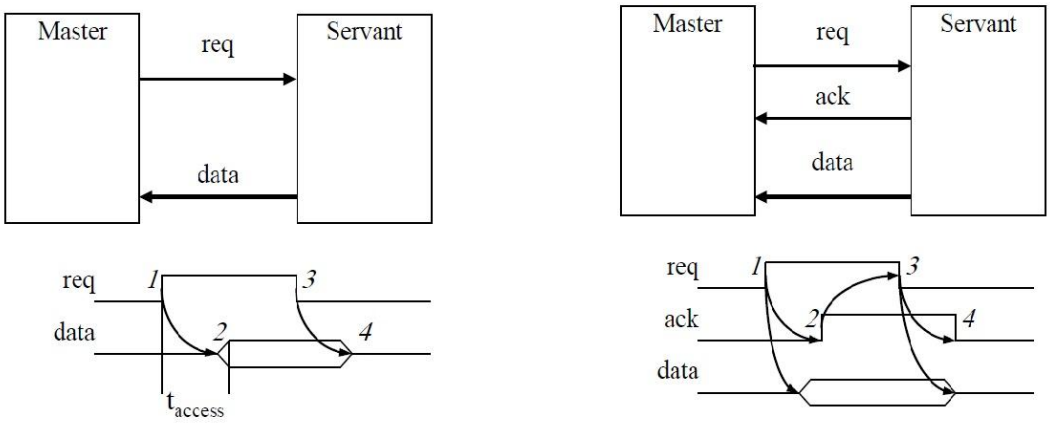


مثال عن التنضيد الزمني للمعطيات

هناك تطبيق آخر لعملية التنضيد الزمني وهو مشاركة نفس المسرى لإرسال العناوين والمعطيات. حيث يتم ارسال العناوين على خطوط المسرى في المرحلة الأولى ليقوم الخادم بتخزينها، ومن ثم ارسال المعطيات على نفس المسرى في المرحلة الثانية ليقوم الخادم بإجراء المطلوب بعد أن أصبح لديه العنوان والمعطيات.

5.3. طرق التحكم:

طرق التحكم هي الطرق المستعملة للبدء بعملية النقل وإنهاء هذه العملية. ومن أشهر هذه الطرق نذكر بروتوكول القدح (strobe) و بروتوكول المصافحة (handshake). كما هي مبينة في الشكل التالي:



بروتوكول القدح وبروتوكول المصافحة

1.5.3. بروتوكول القدح:

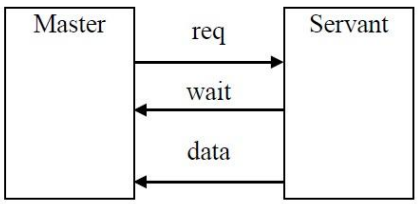
في طريقة القدح (strobe protocol) يستعمل السيد أحد خطوط التحكم والذي يسمى عادةً بخط الطلب (request) للبدء بعملية نقل المعطيات. وبعد فترة زمنية محددة وثابتة, نعتبر أن النقل قد اكتمل. في الشكل السابق، يريد السيد استقبال معطيات من الخادم. يقوم السيد بتفعيل خط الطلب req لإنشاء الاتصال. بدءاً من هذه اللحظة، يكون لدى الخام فترة زمنية قدرها t_{access} ليضع المعطيات على المسرى. وبعد ذلك يقرأ السيد هذه المعطيات ويزيل تفعيل خط الطلب ليكون الخادم جاهزاً لعمل نقل جديدة.

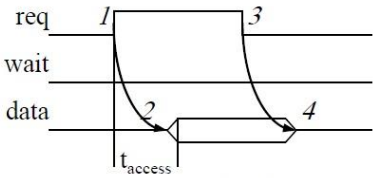
2.5.3. بروتوكول المصافحة:

تسمى الطريقة الثانية طريقة المصافحة (handshake protocol) والتي يستعمل فيها السيد خط الطلب للبدء بعملية النقل ويستعمل الخادم خط الإشعار ack لإعلام السيد بجاهزية المعطيات. يبين الشكل بروتوكول عملية القراءة وفق هذه الطريقة. في البداية يفعل السيد خط الطلب. يأخذ الخادم بعدها الوقت الكافي ليضع المعطيات المطلوبة على المسرى. بعد ذلك يفعل الخادم خط الإشعار ack لإعلام السيد بأن المعطيات أصبحت جاهزة على المسرى. يقرأ السيد المعطيات من المسرى ومن ثم يزيل تفعيل خط الطلب. بعد ذلك يزيل الخادم تفعيل خط الاشعار لتنتهي بذلك عملية النقل.

تعتبر طريقة القدح أسرع من طريقة المصافحة التي تتطلب خطوات أكثر، ولكن لطريقة المصافحة سماحية أكبر مع الطرفيات البطيئة أو عندما يكون زمن استجابة الخادم غير معروفة من قبل السيد.

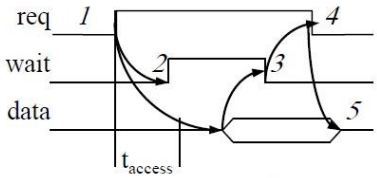
للاستفادة من مزايا الطريقتين، هناك طريقة هجينة بين الطريقتين كما هو مبين في الشكل أدناه. نفترض في هذه الطريقة كما في طريقة القدح أن زمن استجابة الخادم هو t_{access}. فإذا لم يكن الخادم جاهزاً خلال هذا الوقت فإنه يفعل خط تحكمي خاص يدعى خط طلب الانتظار wait. عندها ينتظر السيد حتى يزيل الخادم تفعيل خط الانتظار الأمر الذي يدل على أن المعطيات أصبحت جاهزة. عندها يقوم السيد بقراءة المعطيات ويزيل تفعيل خط الطلب. وبالتالي فإن المصافحة لا تتم إلا عند الضرورة فقط. وبذلك يكون زمن النفاذ قصيراً في الحالات الطبيعية مع إعطاء السماحية ببعض الوقت الإضافي للخادم عندما يكون بطيئاً.





1. Master asserts *req* to receive data
2. Servant puts data on bus **within time t_{access}** (wait line is unused)
3. Master receives data and deasserts *req*
4. Servant ready for next request

Fast-response case



1. Master asserts *req* to receive data
2. Servant can't put data within t_{access}, **asserts** ***wait***
3. Servant puts data on bus and **deasserts** ***wait***
4. Master receives data and deasserts *req*
5. Servant ready for next request

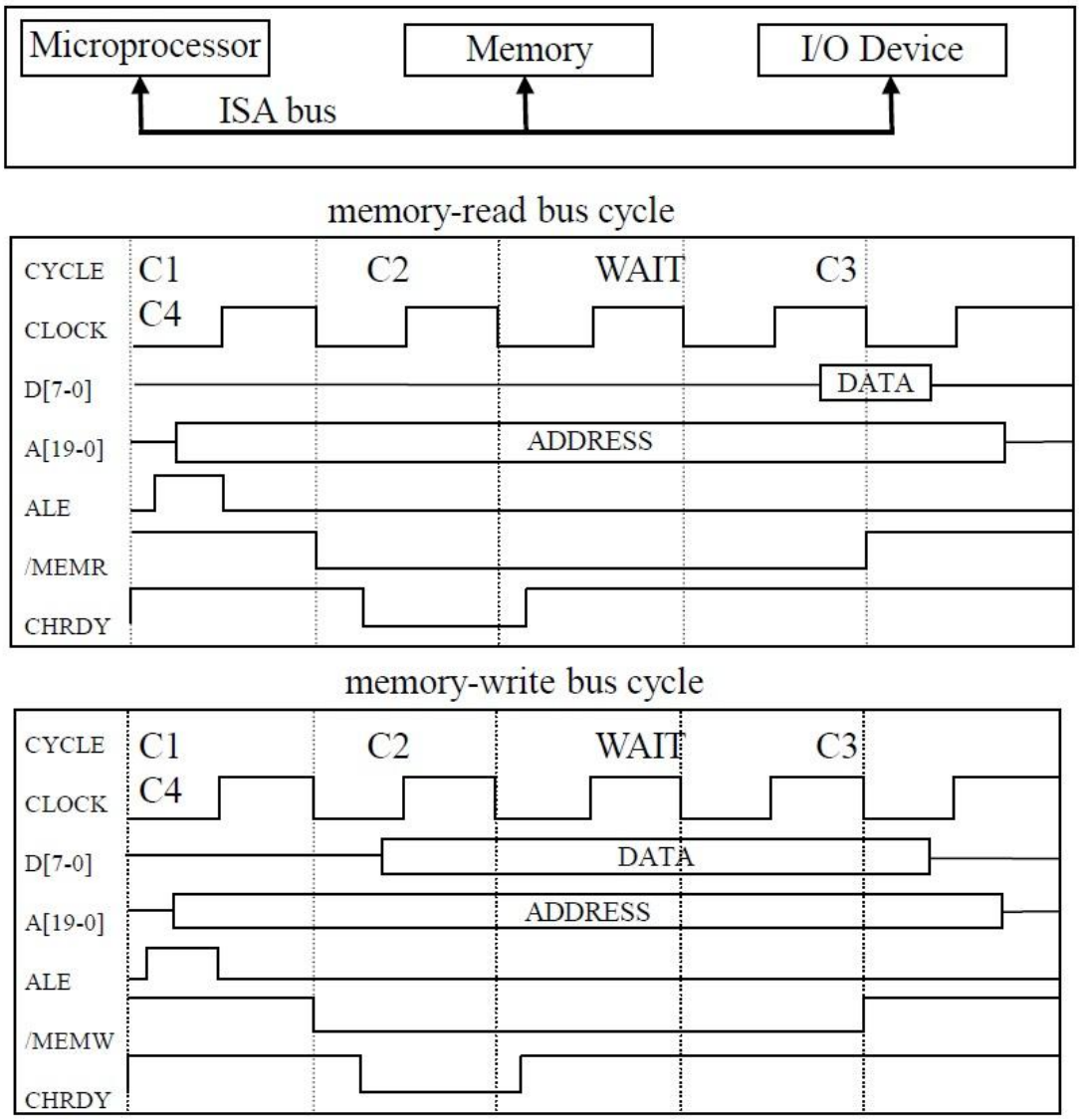
Slow-response case

طريقة الاتصال الهجينة بين القدح والمصافحة

6.3. مثال: بروتوكول المسرى ISA:

انتشر استعمال المسرى ISA (Industry Standard Architecture) في النظم التي تستعمل المعالجات 80x86. يبين الشكل أدناه المخطط الزمني للمسرى من أجل القيام بعملية قراءة من الذاكرة والتي تسمى دورة القراءة من الذاكرة (memory read cycle). يقود المعالج إشارات المسرى لقراءة بايت من الذاكرة. تجري العملية كالتالي: في دورة الساعة C1، يضع المعالج عنوان الذاكرة المكون من 20 بت على مسرى العناوين A ويفعل اشارة ALE (Address Latch Enable). خلال دورتي الساعة C2 وC3 يفعل المعالج الإشارة MEMR لطلب القراءة من الذاكرة. بعد الدورة C3 تضع الذاكرة المعطيات على مسرى المعطيات D. في الدورة C4، يتم إزالة تفعيل جميع الإشارات.

يستعمل المسرى ISA طريقة تحكم هجينة بين القدح والمصافحة. تزيل الذاكرة تفعيل الخط CHRDY قبل الجبهة الصاعدة للساعة في الدورة C2، مما يؤدي لإضافة دورات انتظار حتى يتم تفعيل CHRDY والتي تعبر عن جاهزية المعطيات. في هذا البروتوكول يمكن اضافة 6 دورات ساعة من الانتظار كحد أقصى.



المخطط الزمني لمسرى ISA للقراءة والكتابة في ذاكرة

يبين الشكل أيضاً دورة الكتابة في الذاكرة (memory write cycle) والتي يتم فيها كتابة بايت في الذاكرة. تجري العملية كالتالي: في دورة الساعة C1 يضع المعالج عنوان الذاكرة التي يريد الكتابة فيها على 20 بت ويفعل الخط ALE. خلال الدورتين C2 وC3 يضع المعالج المعطيات التي يريد كتابتها في الذاكرة على مسرى المعطيات ويفعل الإشارة MEMW لطلب الكتابة في الذاكرة. في الدورة C4، يتم إزالة تفعيل جميع الإشارات. تستعمل دورة الكتابة طريقة التحكم الهجينة أيضاً في عملية النفاذ.

4, مواجهة المعالجات:

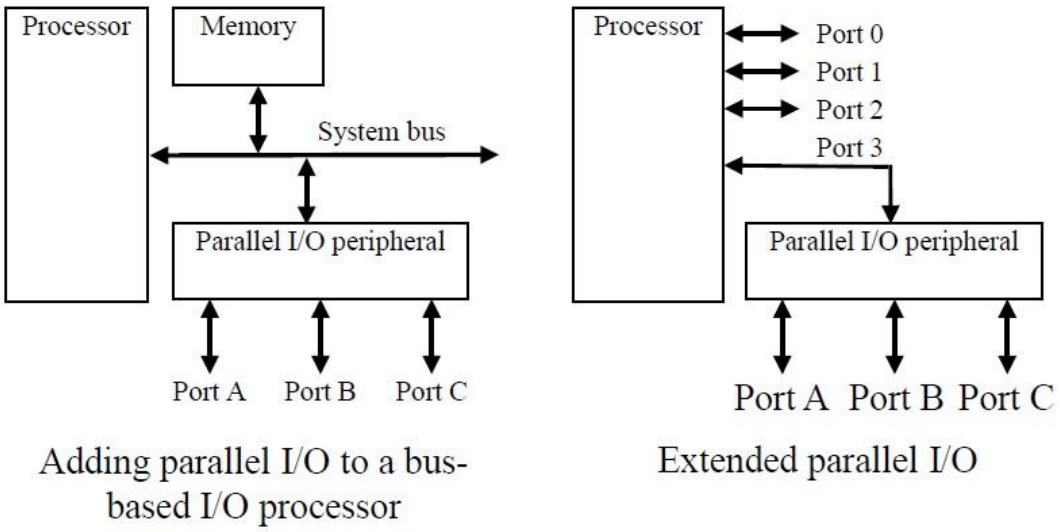
يحتوي المعالج على العشرات أو المئات من المرابط. بعض منها إشارات تحكمية مثل مدخل الساعة وإشارة التصفير. وهناك العديد من المرابط المُعدّة لنقل المعطيات من وإلى المعالج والتي نسميها مرابط الدخل / الخرج (I/O). يوجد طريقتين لاستعمال المرابط في عمليات الدخل والخرج: الدخل والخرج باستعمال البوابات (port–based I/O)، والدخل والخرج باستعمال المسرى (bus–based I/O).

1.4. الدخل والخرج باستعمال بوابات المعالج:

في طريقة الدخل والخرج باستعمال البوابات(port–based I/O)، يستعمل المعالج البوابات المتوفرة لديه لإدخال وإخراج المعلومات. وتتم العملية كما يتعامل المعالج مع سجل داخلي، حيث أن بوابات المعالج تكون متصلة مباشرة بالسجل الداخلي. كما يوجد أيضاً سجلات داخلية للتحكم باتجاه البوابة (دخل أو خرج). في المتحكمات الصغرية، يمكن تغيير كل بت من البوابة وتحديد اتجاهها بشكل مستقل. يعتبر ذلك مفيداً جداً عند استعمال البوابات لإخراج أو إدخال إشارات تحكم.

2.4. الدخل والخرج باستعمال المسرى:

في طريقة الدخل والخرج باستعمال المسرى (bus–based I/O)، يستعمل المعالج مسرى العناوين والمعطيات وإشارات التحكم لإجراء عمليات الدخل والخرج بشكل مشابه للتعامل مع الذاكرة. ويتم إدارة بروتوكول النفاذ بشكل عتادي. أما من وجهة نظر المبرمج فإنه يستعمل تعليمة واحدة فقط لإجراء عملية النفاذ. وعندما نحتاج إلى بوابات دخل/خرج في المعالجات التي تحتوي على مسرى فقط، يمكن وضع طرفيات على المسرى تسمى طرفيات الدخل/الخرج التفرعية (parallel I/O peripheral) لتأمين هذه البوابات كما هو مبين في الشكل التالي.



إضافة طرفيات للمعالج لتأمين بوابات دخل / خرج

يمكن استعمال هذه الطرفيات أيضاً في المعالجات التي يوجد فيها بوابات دخل / خرج عامة وذلك من أجل زيادة عدد البوابات حسب متطلبات تصميم النظام.

1.2.4. الدخل / الخرج المحجوز من الذاكرة:

في عمليات الدخل والخرج المحجوز من الذاكرة (memory-mapped I/O) تُعطى للطرفيات جزءاً من فضاء العنونة المتاح. فعلى سبيل المثال، في معالج يحتوي على 16 خط عنونة، يمكن تخصيص أول 32K للذواكر وتخصيص 32K العلوية لعمليات الدخل والخرج مع الطرفيات. وبالتالي هناك عناوين مختلفة لكل من الذواكر والطرفيات. ومن فوائد هذه الطريقة أنه لا يوجد تعليمات مخصصة للتعامل مع المداخل والمخارج وإنما تتم معاملتها كما تعامل الذاكرة. إلا أن سيئة هذه الطريقة تكمن في تقليص فضاء العنونة المتاح للذواكر وبالتالي الحد من حجم الذاكرة الممكن ربطها مع المعالج.

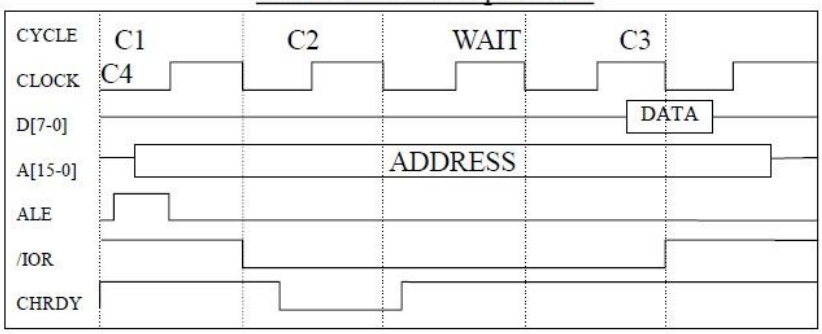
2.2.4. الدخل / الخرج المعياري:

في عمليات الدخل والخرج المعيارية (standard I/O) يحتوي المسرى على مخرج تحكمي إضافي يدعى M/IO ليشير إلى نوع عملية النفاذ. فعلى سبيل المثال، اذا كان M/IO=0 فإن عملية النفاذ موجهة للذاكرة، وإذا كان M/IO=1 فإن عملية النفاذ موجهة لعمليات الدخل والخرج مع الطرفيات. وبالتالي يمكن أن يكون لموقع ذاكرة نفس عنوان إحدى الطرفيات. وهذا ما يزيل سيئة الحد من الفضاء المتاح للذواكر. إلا أنه يجب التعامل مع المداخل والمخارج بتعليمات مختلفة عن تعليمات التعامل مع الذاكرة.

3.4. مثال: الدخل والخرج في المسرى ISA:

يبين الشكل التالي كيف يستعمل المسرى ISA طريقة الدخل والخرج المعيارية باستعمال الخط التحكمي / IOR للنفاذ إلى المداخل بدلاً من MEMR / المخصص للتعامل مع الذواكر. كما يوجد خط IOW / لعمليات الخرج بدلاً من MEMW / بالإضافة إلى ذلك يستعمل المسرى ISA 16 خط عنونة بدل من 20 وذلك يعني أن فضاء عنونة الدخل والخرج هو 64K فقط. مع ذلك، يبقى بروتوكول النفاذ للمداخل والمخارج مشابهاً لبروتوكول النفاذ مع الذواكر.

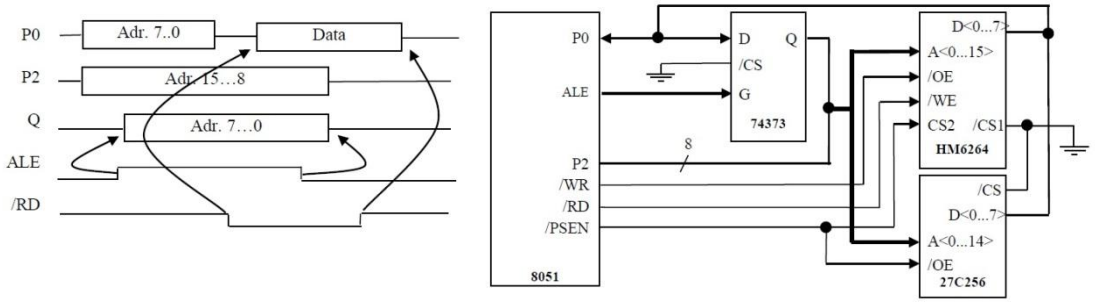
ISA I/O bus read protocol



بروتوكول القراءة من بوابة دخل / خرج في المسرى ISA.

4.4. مثال: بروتوكول الذاكرة في المتحكم الصغري 8051:

سنبين في هذا المثال كيفية ربط (مواجهة) ذاكرة معطيات بسعة 8KB وذاكرة برنامج بسعة 32KB مع المتحكم الصغري 8051. يستعمل المتحكم 8051 فضاء ذاكرة برنامج مستقل عن فضاء ذاكرة المعطيات. وحجم هذا الفضاء هو 64KB لكلا الذاكرتين. مما يعني وجود 16 خط عنونة. يستعمل المتحكم البوابتين P0 وP2 كمسرى للعناوين. طول كلمة المعطيات في المتحكم هي 8 بت. يستعمل المتحكم تقنية التنضيد الزمني، حيث يستعمل البوابة P0 أيضاً كمسرى للمعطيات بشكل متناوب مع البايت الأدنى للعنوان. يبين الشكل التالي بروتوكول القراءة من الذاكرة. يستعمل الخط ALE لتحديد لحظة استقرار العنوان على الخطوط، حيث نستعمل السجل (74373) لحفظ الجزء الأدنى من العنوان. كما يستعمل الخط PSEN للتمييز بين ذاكرة البرنامج وذاكرة المعطيات.



بروتوكول النفاذ للذاكرة في المتحكم الصغري 8051

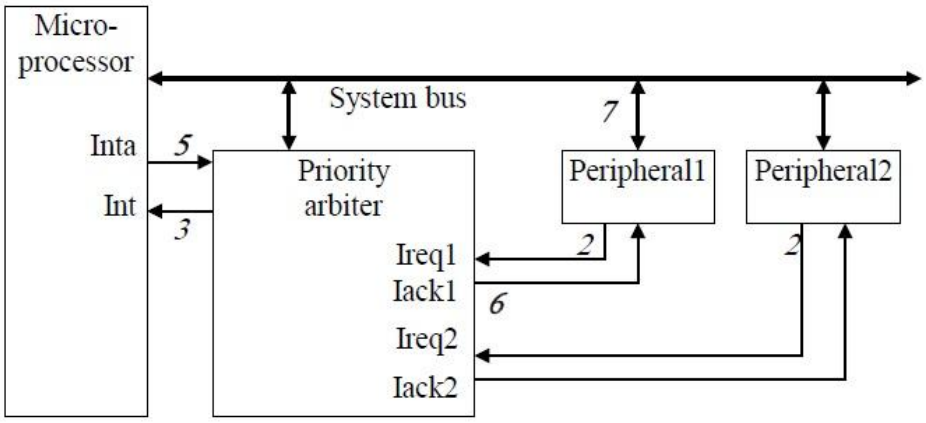
تجري عملية القراءة كالتالي: يضع المتحكم أولاً العنوان على البوابتين P0 وP1 ثم يفعل الخط ALE. عند الجبهة الصاعدة لهذه الإشارة يخزن السجل الخارجي بتات العنونة الدنيا من البوابة P0. بعد ذلك يضع المتحكم البوابة P0 في حالة دخل ويفعل إشارة القراءة RD / (فعالة على السوية الدنيا) ولكن يبقى الجزء العلوي من العنوان ظاهراً على البوابة P2. فتضع ذاكرة البرنامج (27C256) كلمة المعطيات على البوابة P0 إذا كانت الإشارة PSEN=0، وإلا فإن ذاكرة المعطيات (HM6264) هي التي ستضع كلمة المعطيات على مسرى المعطيات. يقوم المتحكم بقراءة كلمة المعطيات ويزيل تفعيل إشارة القراءة.

5. التحكيم:

عندما تطلب أكثر من طرفية التخديم من قبل نفس المصدر في نفس الوقت. على سبيل المثال، عدة طرفيات تطلب من المعالج تخديم مقاطعاتها، أو تطلب التخديم من قبل متحكم النفاذ المباشر للذاكرة. في هذه الأحوال، علينا أن نقرر أي طرفية سيتم تخديمها أولاً وهذا ما يعرف بعملية التحكيم (arbitration). يوجد طريقتين للتحكيم وهما تحكيم الأولوية وتحكيم سلسلة "ديزي."

1.5. تحكيم الأولوية:

يتم تحكيم الأولوية (priority arbitration) باستعمال معالج مخصص لهذه الغرض ويتصل بالمعالج عبر مسرى النظام. نوضح مبدأ عمل تحكيم الأولوية على مثال مجموعة من الطرفيات التي تطلب تخديم المقاطعات التي تولدها من قبل المعالج. في هذه الحالة يسمى المعالج الخاص بعملية التحكيم بمتحكم المقاطعات. حيث يتم توصيل طلبات المقاطعة Ireq من الطرفيات إلى متحكم المقاطعات. ويخرج من متحكم المقاطعات إلى الطرفيات إشارات الإشعار Iack. يتصل متحكم المقاطعات مع العالج عبر المسرى العام، وكذلك عبر مداخل المقاطعات للمعالج وإشارة الإشعار الموافقة (Int و Inta).



تحكيم الأولوية للمقاطعات باستعمال متحكم المقاطعات

هناك طريقتين في تحكيم الأولوية:

الأولوية الثابتة (fixed priority): تُعطى هنا لكل طلب مقاطعة رقماً يعبر عن أولويتها مثل 1, 2, 3, 4. فكلما كان هذا الرقم أصغر تكون للمقاطعة الموافقة أولوية أعلى. وعند ورود مقاطعتين أو أكثر في نفس الوقت يتم تخديم المقاطعة ذات الأولوية الأعلى. وتنتظر المقاطعات الأقل أولوية حتى ينتهي تخديم المقاطعة ذات الأولوية الأعلى.

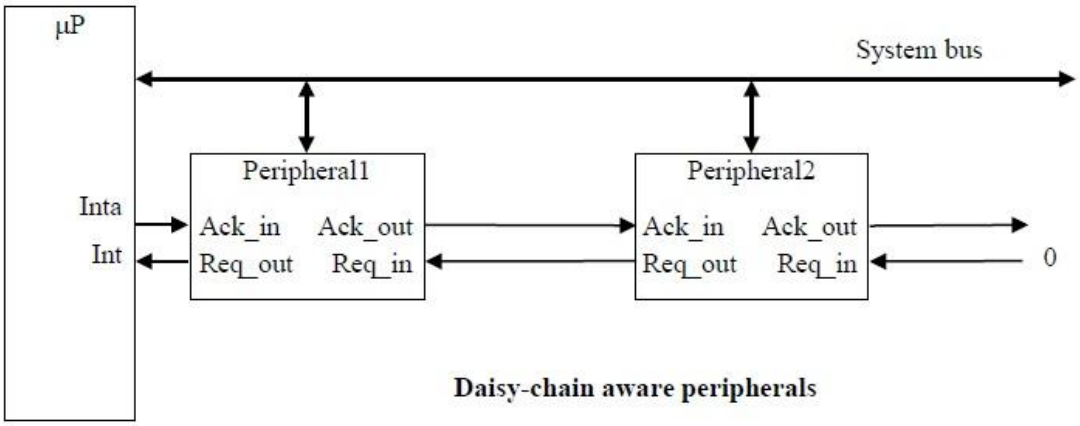
الأولوية الدوارة (rotating priority): وتسمى أحياناً هذه الطريقة باسم round–robin. تم اقتراح هذه الطريقة لتجنب عملية استحواذ بعض المقاطعات على كل وقت المعالج إذا كانت مقاطعة من أولوية مرتفعة

تحدث بشكل متكرر كثيراً، فإنه يعاد توزيع الأولويات بحيث تعطى الأولوية للمقاطعة الأقل تخديماً بحيث يكون هناك توزيع شبه متساوي في التخديم بين جميع المقاطعات.

يمكن المزج بين هاتين الطريقتين بحيث يتم إعطاء أولويات ثابتة للمقاطعات مع امكانية إعطاء بعض المقاطعات نفس درجة الأولوية. وعند ورود مقاطعتين من نفس السوية يتم تطبيق طريقة الأولوية الدوارة بينهما.

2.5. تحكيم سلسلة ديزي:

لا نحتاج في طريقة تحكيم سلسلة ديزي (Daisy–Chain arbitration) لمعالج خاص لإدارة الأولوية. ولكن يتم دمج إدارة الأولوية ضمن الطرفيات. حيث تملك كل طرفية مدخلاً لطلب المقاطعة req_in ومخرجاً له req_out. وكذلك مدخلاً لإشارة الإشعار acq_in ومخرجاً أيضاً ack_out. يتم ربط هذه المداخل والمخارج للطرفيات بشكل سلسلة كما هو مبين في الشكل بحيث تكون الطرفية ذات الأولوية الأعلى الأقرب للمعالج.



تحكيم سلسلة ديزي

عندما تطلب إحدى الطرفيات تخديما من المعالج فإنها ترسل إشارة req إلى الطرفية التي قبلها باتجاه المعالج، وتقوم الطرفية التي قبلها بتمرير هذا الطلب إلى الطرفية التالية وصولاً للمعالج. يجيب المعالج بإشارة إشعار تنتشر بالاتجاه المعاكس. عند وصول إشارة الإشعار إلى الطرفية التي طلبت التخديم، فإن هذه الأخيرة لا تمرر الإشارة للطرفية التي بعدها ولكن تضع رقمها على المسرى ليعلم بها المعالج ويقوم بتخديمها. وبالتالي لا يتم تخديم طرفية إلا إذا كانت الطرفيات التي قبلها لا تحتاج للتخديم.

تعتبر هذه الطريقة مشابهة لتحكيم الأولوية الثابتة وتمتاز هذه الطريقة بأنها قابلة للتوسيع حيث يمكن إضافة طرفيات بقدر ما نشاء. إلا أنها تشكو من التأخير الزائد بالنسبة للطرفيات التي في نهاية السلسلة. كما أن أي عطل في أحد الطرفيات قد يؤثر على كل الطرفيات التي بعدها في السلسلة.

3.5. طرق تحكيم المسرى:

نحتاج أيضاً للتحكيم عندما يكون هناك أكثر من معالج على نفس المسرى. فعندما يريد معالجان أو أكثر استعمال المسرى في نفس الوقت، فمن الممكن حدوث تصادم على المسرى. يتم حل هذه المشكلة ضمن بروتوكول المسرى. ونسمي هذه الطرق بطرق التحكيم الموجهة للشبكات (network–oriented methods arbitration). في هذه الحالة تكون واجهة المسرى في المعالج مزودة بالدارات الضرورية لمراقبة المسرى والكشف عن حدوث التصادم. عند كشف التصادم، يتم ايقاف التراسل وانتظار وقت معين لمعاودة المحاولة مرة أخرى. وتختلف فترة الانتظار من طرفية لأخرى بحيث لا تعود الطرفيات المتصادمة للإرسال من جديد في نفس اللحظة. وهناك خوارزميات متقدمة للتقليل من احتمال حدوث التصادم. وفي بعض بروتوكولات المسرى مثل بروتوكول المسرى I^2C كما سنرى في الفصل القادم يتم استعمال طرق عنونة ذكية تحدد أولوية استعمال المسرى من قبل المعالجات المتصلة على نفس المسرى.

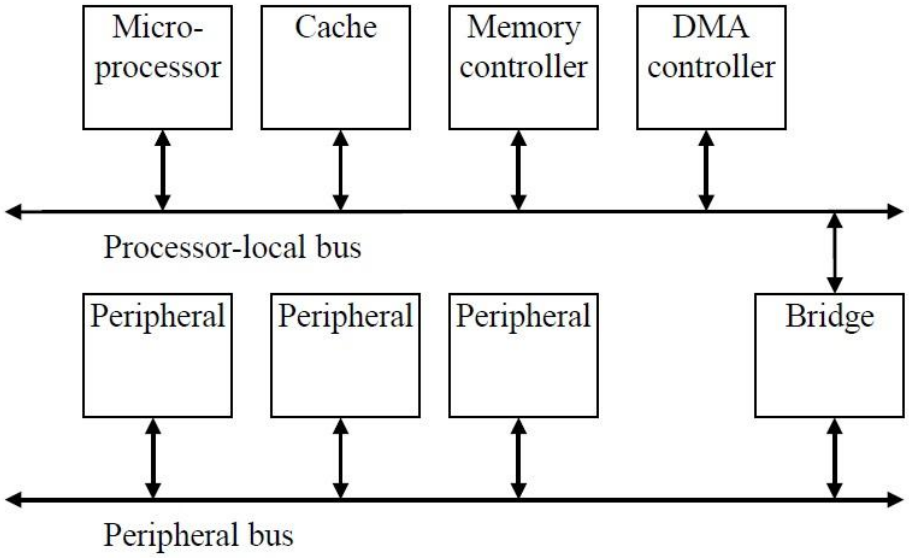
6. بنية المسرى متعدد السويات:

تحتوي النظم المضمنة التي تستعمل المعالجات الصغرية على عدة أنواع من التراسل التي يمكن أن تجري داخل النظام وتختلف فيما بينها بمتطلبات السرعة ومعدل التكرار. تعتبر عمليات نفاذ المعالج للذاكرة من عمليات التراسل العالية السرعة والكثيرة التكرار. أما عمليات التراسل مع الطرفيات مثل طرفية UART فتعتبر من العمليات التي تتطلب سرعة أقل وتحدث بتواتر أقل.

يمكننا بناء مسرى وحيد عالي السرعة لجميع أنواع التراسل. ولكن لهذه الطريقة بعض المساوئ وهي:

- يتطلب هذا من جميع الطرفيات أن يكون لديها واجهة سريعة مع المسرى. وهذا يشكل شرطاً غير ضروري للطرفيات التي لا تحتاج لسرعة في التراسل مما يؤدي إلى رفع كلفة الطرفية وزيادة في استهلاك الطاقة.
- إن عملية تزويد الطرفيات بواجهات بينية سريعة مع المسرى قد يكون متوافقاً مع بعض أنواع المعالجات دون غيرها.
- عندما وصل الكثير من الطرفيات على نفس المسرى، فإن هذا قد يؤدي إلى خفض سرعة المسرى بسبب الحمل السعوي المتزايد.

لهذه الأسباب، يصمم النظام عادةً باستعمال سويتين من المساري. مسرى معالج محلي (processor local bus) عالي السرعة ومسرى آخر أقل سرعة مخصص للطرفيات (peripheral bus). كما هو موضح في الشكل التالي.



بنية مسرى على سويتين

يصل مسرى المعالج المحلي بين المعالج والذاكرة المخبأة ومتحكم الذاكرة وبعض المعالجات المساعدة (coprocessor) الخاصة بالمعالج. يكون هذا المسرى عريضاً بعرض كلمة الذاكرة. وهو في أغلب الأحيان مضمن على نفس شريحة المعالج لذلك يوصف بالمسرى المحلي.

أما مسرى الطرفيات، فيصل بين المعالج وجميع الطرفيات التي لا تحتاج لمسرى ذو سرعة عالية كأولوية ولكن تتطلب البساطة والتوافق مع المساري المعيارية مثل المسرى ISA أو المسرى PCI على سبيل المثال.

يصل الجسر (bridge) بين المسريين. حيث أن الجسر هو عبارة عن معالج مخصص لتحويل التراسل من مسرى إلى آخر بالاتجاهين. فعندما يطلب المعالج القراءة من إحدى الطرفيات، ينقل الجسر العنوان المطلوب إلى مسرى الطرفيات ويولد إشارات التحكم الضرورية لإجراء عملية قراءة من الطرفية. وعند وصول المعلومات ينقل الجسر هذه المعطيات من مسرى الطرفيات إلى المسرى المحلي. تتم هذه العملية بالنسبة للمعالج كما لو كانت الطرفية على المسرى المحلي ولا يشعر المعالج بوجود الجسر.

وهناك أيضاً مساري بثلاث سويات: مسرى محلي ومسرى نظام ومسرى طرفيات. حيث يكون مسرى النظام بسرعة أكبر من مسرى الطرفيات. وتستعمل هذه الطريقة في النظم المعقدة التي تحتوي على العديد من المعالجات المساعدة.

7. أسئلة الفصل العاشر:

أسئلة عامة:

1. ما هي المواجهة (interfacing) ؟

المواجهة هي آلية نقل المعلومات بين المعالجات والذواكر والطرفيات باستعمال المساري.

2. ما هي مكونات المسرى؟

يتكون المسرى من: خطوط عنونة و خطوط المعطيات وخطوط التحكم:

3. ما هو بروتوكول المسرى؟

البرتوكول هو القواعد التي يجب احترامها للاتصال على خطوط المسرى. هو يتعلق بالقواعد المطبقة على الإشارات لتأمين الاتصال عبر المسرى بشكل صحيح.

4. قارن بين طريقة القدح وطريقة المصافحة في بروتوكولات الاتصال عبر المسرى من حيث السرعة والسماحية مع الطرفيات البطيئة؟

تعتبر طريقة القدح أسرع من طريقة المصافحة التي تتطلب خطوات أكثر، ولكن لطريقة المصافحة سماحية أكبر مع الطرفيات البطيئة أو عندما يكون زمن استجابة الخادم غير معروفة من قبل السيد.

5. ما هي طريقة الدخل والخرج المستعملة في المسرى ISA؟

يستعمل المسرى ISA طريقة الدخل والخرج المعيارية باستعمال الخط التحكمي IOR / للنفاذ إلى المداخل. كما يوجد خط IOW / لعمليات الخرج.

أسئلة خيارات متعددة:

1. عندما نفعل إشارة تحكم ما فذلك يعني أننا؟
   A. نضعها على السوية الدنيا.
   B. نضعها على السوية العليا.
   C. يعتمد ذلك على خيار المصمم إما سوية دنيا أو عليا.
   D. نضع عليها نبضة لفترة معينة.

2. يدل مفهوم السيد في عملية الاتصال على:
   A. المعالج.
   B. الطرفية.
   C. الجهة التي ترسل المعطيات.
   D. الجهة التي تبادر بعملية النقل.

3. يدل مفهوم التضيد الزمني (time multiplexing) في عملية الاتصال عبر المسرى على:
   A. مشاركة نفس المسرى لنقل قطع مختلفة من المعطيات.
   B. تقاسم الطرفيات زمن تخديم المعالج لها.
   C. تقاسم المعطيات لمسرى الاتصال عبر الزمن.
   D. تحديد لحظة ارسال المعطيات عبر المسرى.

4. ما هي طريقة التحكم الأسرع والأكثر مرونة في نقل المعطيات بين المعالج والطرفيات:
   A. طريقة القدح.
   B. طريقة المصافحة.
   C. طريقة الاتصال الهجينة.
   D. طريقة تحكيم الأولوية.

5. ما هي طريقة مواجهة المعالج مع الطرفيات التي تبسط تصميم المعالج:
   A. الدخل والخرج المحجوز من الذاكرة.
   B. الدخل والخرج المعياري.
   C. الدخل والخرج باستعمال بوابات المعالج.
   D. الدخل والخرج باستعمال طرفيات خاصة.

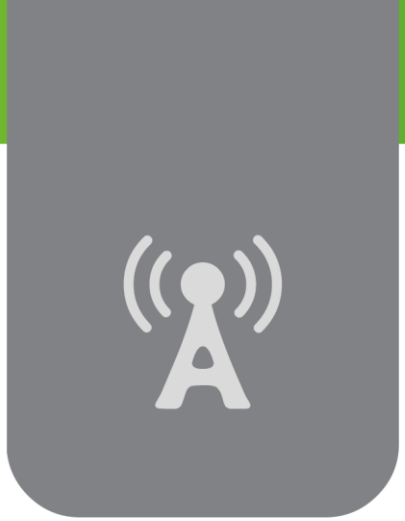
6. لمنع استحواذ طرفية معينة على وقت المعالج، فإننا نستعمل طريقة التحكيم:
   - A. تحكيم الأولوية الثابتة.
   - B. تحكيم الأولوية الدوارة.
   - C. تحكيم الأولوية الهجينة بين الأولوية الثابتة والأولوية الدوارة.
   - D. تحكيم سلسلة ديزي.

7. يستعمل جسر المسرى (bus bridge) في الاتصال عن طريق المساري من أجل:
   - A. تسريع عملية نقل المعطيات عن طريق استعمال مسرى سريع.
   - B. تحويل التراسل من مسرى إلى آخر بالاتجاهين في المسرى متعدد السويات.
   - C. ربط مسرى المعالج مع الذواكر بشكل مباشر.
   - D. لتأمين الاتصال بين نظامين مختلفين.

8. يستعمل المسرى ISA في بروتوكول الاتصال:
   - A. طريقة القدح.
   - B. طريقة المصافحة.
   - C. طريقة تحكم هجينة بين القدح والمصافحة.
   - D. طريقة الدخل والخرج المحجوز من الذاكرة.

الإجابات الصحيحة:

رقم التمرين	الإجابة الصحيحة
1	C
2	D
3	A
4	C
5	A
6	B
7	B
8	C



الفصل الحادي عشر:
بروتوكولات الاتصال

الكلمات المفتاحية:

الاتصال التفرعي، الاتصال التسلسلي، الاتصال اللاسلكي، كشف الخطأ وتصحيحه، الطبقية في بروتوكولات الاتصال.

Parallel Communication, Serial Communication, Wireless Communication, Error Detection and Correction, Layering.

الملخص:

يهدف هذا الفصل إلى التعرف على أهم طرق الاتصال في النظم المضمّنة، وهي الطرق التي تؤمن التواصل ونقل المعلومات بين الطرفيات والوحدة المركزية في النظام المضمّن، أو بين النظام المضمّن ونظم أخرى كالحاسوب الشخصي مثلاً. سنعرض في هذا الفصل طرق الاتصال التفرعي والاتصال التسلسلي والاتصال اللاسلكي، كما سنتطرق إلى بعض المفاهيم المتقدمة مثل مفهوم الطبقية في بروتوكولات الاتصال ومفهوم كشف الخطأ وتصحيحه.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- أنواع الاتصالات في النظم المضمنة ومفهوم كشف الخطأ والطبقية
- البروتوكولات التسلسلية: I2C، CAN، Firewire، USB
- البروتوكولات التفرعية: PCI، ARM
- البروتوكولات اللاسلكية: IrDA، Blutooth، WiFi

يهدف هذا الفصل إلى التعرف على أهم طرق الاتصال في النظم المضمّنة، وهي الطرق التي تؤمن التواصل ونقل المعلومات بين الطرفيات والوحدة المركزية في النظام المضمّن، أو بين النظام المضمّن ونظم أخرى كالحاسوب الشخصي مثلاً. سنعرض في هذا الفصل طرق الاتصال التفرعي والاتصال التسلسلي والاتصال اللاسلكي، كما سنتطرق إلى بعض المفاهيم المتقدمة مثل مفهوم الطبقية في بروتوكولات الاتصال ومفهوم كشف الخطأ وتصحيحه.

1. مقدمة:

يتم الاتصال عبر أنواع مختلفة من الوسائط (Media) مثل الأسلاك والأمواج الراديوية والأشعة تحت الحمراء. نشير إلى الوسط الذي يحمل المعلومات من جهاز إلى آخر باسم الطبقة الفيزيائية (Physical Layer)، وبحسب بروتوكول الاتصال فإننا نشير إلى أطراف الاتصال باسم جهاز (Device) أو عقدة (Node). يعبّر الجهاز هنا عن معالج يَستعمل الطبقة الفيزيائية لإرسال أو استقبال المعطيات من جهاز إلى آخر.

في البداية سنتعرف على أنواع الاتصالات المختلفة، ثم سنتعرف على بعض من بروتوكولات الاتصالات لكل نوع من الأنواع.

2. أنواع الاتصالات:

1.2. الاتصال التفرعي:

يجري الاتصال التفرعي (Parallel Communication) عندما تكون الطبقة الفيزيائية قادرة على نقل مجموعة من بتات المعطيات من جهاز إلى آخر، أي إن مسرى المعطيات مكون من عدة أسلاك تمتد على التوازي بين جهاز وآخر، وقد يصاحب هذا المسرى خطوطاً تحكمية أو خطوط تغذية أيضاً.

يحمل كل سلك بتاً واحداً من المعطيات. من مزايا الاتصال التفرعي التدفق المرتفع للمعطيات إذا كان طول المسرى قصيراً، لأن طول المسرى يؤدي إلى زيادة الحمل السِّعوي للمسرى وذلك لأن الأسلاك المتوازية تشكل مكثفة ذات سعة تزداد قيمتها بزيادة طول هذه الأسلاك. يؤدي الحمل السعوي إلى إبطاء سرعة الاتصال لأن المكثفات الناتجة تحتاج إلى زمن للشحن والتفريغ. إضافة إلى ذلك، فإن الاختلاف في طول الأسلاك يمكن أن يسبب اختلافاً في لحظات وصول البتات، ويزداد أثر هذه المشكلة كلما زاد طول المسرى.

من المشاكل الأخرى للمساري التفرعية الكلفة الزائدة والتعقيد المتعلق بعزل الأسلاك فيما بينها لتجنب حصول التداخل بين الأسلاك. على سبيل المثال، تكون كلفة كبل يحتوي على 32 سلكاً للربط بين جهازين أعلى بكثير من كلفة كبل يحتوي على سلكين فقط.

بشكل عام، يُستخدم الاتصال التفرعي لوصل التجهيزات الواقعة على نفس الدارة التكاملية أو على نفس بطاقة الدارة. في هذه الحالات يكون أثر مساوئ الوصل التفرعي على سرعة الاتصال مهملاً.

2.2. الاتصال التسلسلي:

يعتمد الاتصال التسلسلي (Serial Communication) على طبقة فيزيائية تحمل بتاً واحداً فقط من المعطيات، وهذا يعني أن مسرى المعطيات يتكون من سلك معطيات واحد يكون مصحوباً بأسلاك تحكم وتغذية للربط بين جهاز وآخر. في الاتصال التسلسلي تُرسل كلمات المعطيات على شكل بتات متسلسلة واحدٍ تلو الآخر. يُعتبر الاتصال التسلسلي فعالاً أكثر عندما تكون الأجهزة المتصلة متباعدة فيما بينها، وذلك بسبب الكلفة المنخفضة للكبل والحمل السعوي الأقل الناتج عن قلة الأسلاك.

من سيئات الاتصال التسلسلي الزيادة في تعقيد دارات الإرسال والاستقبال، إذ يتوجب على المرسل أن يقوم بتفكيك الكلمات إلى بتات، وعلى المستقبل أن يجمع البتات المستقبلة في كلمات. هذا بالإضافة إلى إدارة بروتوكول التراسل الأكثر تعقيداً مقارنة بالاتصال التفرعي.

تلغي معظم بروتوكولات التراسل التسلسلي الحاجة إلى استخدام أسلاك تحكم إضافية لتحديد البت الأول من كل كلمة أو لتحديد لحظة بداية ونهاية كل بت. يتم ذلك باستخدام خط المعطيات نفسه كما هي الحال في التراسل التسلسلي UART عن طريق إضافة بت البداية (Start bit) وبت النهاية (Stop bit).

في الوقت نفسه، توجد بروتوكولات تراسل تسلسلية تستعمل خط تحكم إضافي ليلعب دور ساعة تزامن للمعطيات كما هي الحال في بروتوكول I^2C الذي سنأتي على شرحه لاحقاً في هذا الفصل.

3.2. الاتصال اللاسلكي:

يلغي الاتصال اللاسلكي (Wireless Communication) الحاجة إلى ربط التجهيزات بشكل فيزيائي لتحقيق الاتصال، وغالباً ما تستعمل القنوات الراديوية RF أو الأشعة تحت الحمراء IR (Infra-Red) كطبقة فيزيائية في الاتصال اللاسلكي.

يَستعمل الاتصال بالأشعة تحت الحمراء ترددات الأمواج الكهرطيسية الواقعة تحت الضوء المرئي بقليل، لذا فهي غير مرئية بالعين المجردة. يمكن توليد الأشعة تحت الحمراء باستخدام ديود خاص بالأشعة تحت الحمراء، ويتم الكشف عن هذه الأشعة باستخدام ترانزستور خاص حساس لهذه الأشعة، حيث يصبح الترانزستور ناقلاً عندما يتعرض لها. يمكن بناء مرسل بسيط يقوم بتشغيل الديود عندما نريد إرسال البت 1، بينما يقوم بإيقاف تشغيل الديود عندما نريد إرسال البت 0. في الطرف الآخر يكشف المستقبل عن استقبال 1 عندما يصبح الترانزستور الكاشف ناقلاً و0 في الحالة المعاكسة.

من مزايا استعمال الأشعة تحت الحمراء في الاتصال اللاسلكي تدني كلفة بناء كل من المرسل والمستقبل، لكن في المقابل، سيئة هذا النوع من الاتصال هي الحاجة إلى خط نظر بين المرسل والمستقبل مما يحدّ من مجال الاتصال.

يستعمل الاتصال بالترددات الراديوية ترددات الأمواج الكهرطيسية الواقعة في المجال الراديوي. يحتاج المرسل هنا إلى دارة إرسال راديوي مع هوائي، وكذلك يحتاج المستقبل إلى دارة استقبال راديوي مع هوائي. عادة ما تكون هذه الدارات معقدة، ولكن تكمن الفائدة في هذا النوع من الاتصال اللاسلكي في عدم الحاجة إلى خط نظر، وإمكانية تحقيق مجال اتصال على مسافة طويلة حسب الاستطاعة المستخدمة في الإرسال.

يكون تردد الإرسال الراديوي ثابتاً بين المرسل والمستقبل أو يتغير بشكل متفق عليه بينهما. تسمح تقنية القفز الترددي (Frequency Hopping) بمشاركة مجموعة من الترددات بين عدة تجهيزات كما هي الحال في بروتوكولات التراسل اللاسلكي المصممة لأغراض الوصل الشبكي بين الحواسب والتجهيزات الأخرى.

3. مفاهيم أساسية في الاتصالات:

1.3. الطبقية:

الطبقية (Layering) هي تنظيم هيكلي (Hierarchical) لبروتوكول الاتصال، حيث تخدم الطبقات الدنيا فيه الطبقات العليا.

- تشكل الطبقة الفيزيائية الطبقة الأولى في هذا التنظيم وهي الطبقة المكلفة بإرسال بتات المعطيات والكلمات.
- تَستعمل الطبقة الثانية الطبقة الفيزيائية لإرسال طرود (Packets) المعطيات، حيث يتكون الطرد من العديد من البايتات.
- تستخدم الطبقة الثالثة الطرود من أجل إرسال المعطيات من أنواع مختلفة مثل طرود الطلب والإشعار وغيرها.
- وفي أعلى طبقة نجد طبقة التطبيق وهي التي تخدم التطبيقات.

تشكل الطبقية طريقة لتقسيم تعقيد بروتوكول الاتصال إلى أجزاء مستقلة مما يسهل تصميم وفهم البروتوكول، تماماً كما يفعل المبرمج عندما ينشئ مكتبة من التوابع والإجرائيات. لذلك يُعتبر مفهوم الطبقات في مجال الاتصالات والشبكات أمراً أساسياً.

2.3. كشف الخطأ وتصحيحه:

كشف الخطأ (Error Detection) هو قدرة المستقبل على اكتشاف الأخطاء التي من الممكن أن تحدث أثناء إرسال كلمة معطيات أو طرد. من أنواع الأخطاء الشائعة خطأ البت (bit error) ورشقة الأخطاء (burst of bit error). يحدث خطأ البت عند حدوث خلل في بت واحد من الكلمة أو الطرد المستقبل. وتحدث رشقة الأخطاء عند استقبال مجموعة متلاحقة من البتات بقيم غير صحيحة.

بفرض أنه تم الكشف عن خطأ في الاستقبال، فإن تصحيح الخطأ (Error Correction) يتمثل في إمكانية تصحيح هذا الخطأ في المستقبِل بالتعاون مع المرسل.

غالباً ما تكون عملية الكشف عن الأخطاء وتصحيحها جزءاً من بروتوكول التراسل. سنناقش فيما يلي نوعين من طرق الكشف عن الأخطاء الشائعة الاستخدام في بروتوكولات التراسل.

1.2.3.الزوجية:

يتم في طريقة الزوجية (Parity) إرسال بت إضافي مع كل كلمة معطيات، ولدينا خياران في طريقة حساب قيمة البت الإضافي انطلاقاً من بتات كلمة المعطيات الأصلية.

- الخيار الأول أن نضع قيمة هذا البت بحيث يكون عدد البتات ذات القيمة 1 في الكلمة الجديدة (الكلمة الأصلية مضافاً إليها بت الزوجية) يمثل عدداً زوجياً (Even Parity)، أي نحسب عدد البتات ذات

القيمة 1 في الكلمة الأصلية، فإذا كان فردياً نضع قيمة بت الزوجية 1 ليكون عدد البتات ذات القيمة 1 في الكلمة الجديدة زوجياً، وإذا كان العدد زوجياً نضع قيمة البت الإضافي 0.

- الخيار الثاني أن نضع قيمة هذا البت بحيث يكون عدد البتات ذات القيمة 1 في الكلمة الجديدة هو عدد فردي (Odd Parity)، أي عكس قيمة البت وفق الخيار الأول.

ليس مهماً الخيار الذي نعتمده، فكلا الخيارين متكافئين. لنفرض أننا اخترنا الخيار الأول بحيث يكون عدد البتات ذات القيمة 1 زوجياً. في الاستقبال يتم التحقق من ذلك بحساب عدد البتات 1 في الكلمة المستقبلة، فإذا كان هذا العدد فردياً فهذا يعني حتماً وجود خطأ على الأقل في الكلمة المستقبَلة. أما إذا كان زوجياً فنعتبر أن الكلمة صحيحة.

تعتبر هذه الطريقة فعالة في الكشف عن عدد فردي فقط من الأخطاء. لتوضيح ذلك نأخذ المثال التالي. لنفرض أن الكلمة المرسلة هي 0101010. مع إضافة بت الزوجية تصبح الكلمة المرسلة الجديدة هي 01010101 وذلك بإضافة البت 1 في آخر الكلمة. لنفرض أنه تم استقبال الكلمة التالية 11010101 والتي تحوي على خطأ في البت الأول. تحتوي هذه الكلمة على عدد فردي من القيم 1 وبالتالي فهذه الكلمة فيها خطأ. أما إذا استقبلنا الكلمة 11110101 والتي تحوي خطأين في البت الأول والثالث، نجد أن عدد البتات 1 في هذه الكلمة زوجي، وهنا يُقدّر المستقبِل بأن هذه الكلمة صحيحة، وبالتالي فإن طريقة الزوجية غير قادرة على كشف جميع الأخطاء.

2.2.3. مجموع التحقق:

تُستخدم طريقة مجموع التحقق (Check Sum) للكشف عن حدوث الأخطاء في الطرد المكوّن من مجموعة من الكلمات. تعتمد هذه الطريقة على إضافة كلمة للطرد الأصلي، وتُحسب هذه الكلمة بطرق مختلفة. من أبسط هذه الطرق حساب مجموع الكلمات في الطرد بعملية XOR.

يقوم المستقبل بدوره بحساب هذا المجموع ويطابق النتيجة مع كلمة المجموع المستقبلة، فإذا كان هناك تطابق يَعتبر المستقبل أن الطرد صحيح، وإلا فهناك خطأ في الطرد المستقبَل.

مثال: لنفرض أن الطرد يتكون من ثلاث كلمات وهي:

11001100
10101010
11100000

فيكون مجموع التحقق بطريقة الجمع هي:

10000110

هنا أيضاً لا تستطيع هذه الطريقة الكشف عن جميع الأخطاء الممكنة. بالرغم م ذلك، توجد طرق أكثر تعقيداً لحساب كلمة مجموع التحقق بحيث يكون احتمال عدم الكشف عن الخطأ صغيراً جداً. نذكر على سبيل المثال خوارزمية CRC (Cyclic Redundancy Check).

4.البروتوكولات التسلسلية:

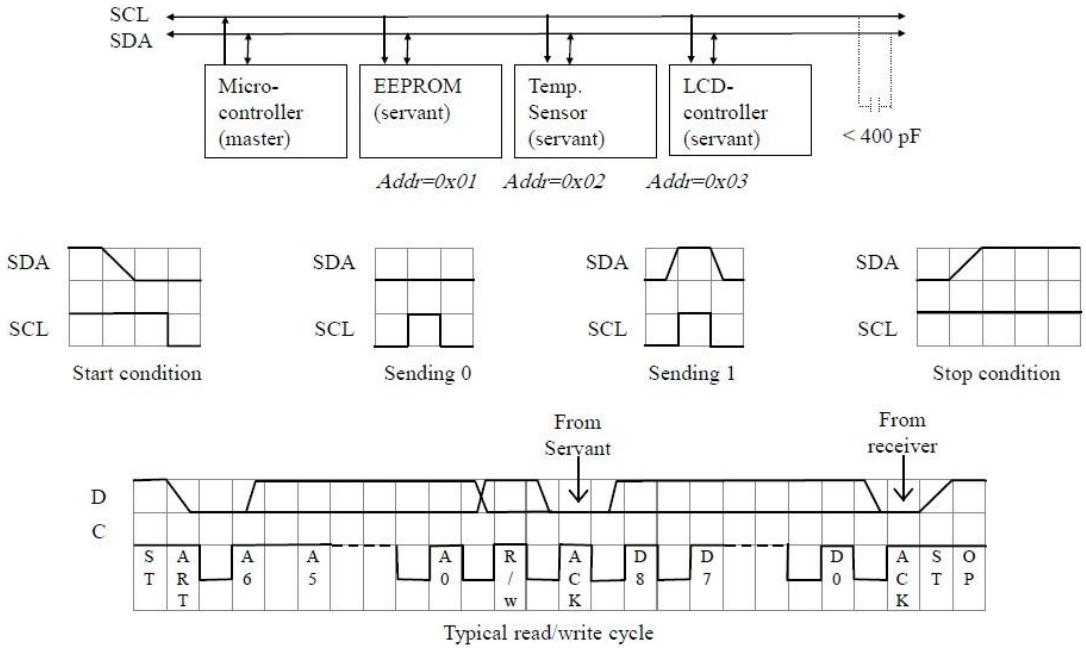
سنعرض في هذه الفقرة أربعة بروتوكولات تسلسلية كثيرة الاستعمال في النظم المضمنة:

1. بروتوكول I2C.
2. البروتوكول CAN.
3. السلك الناري.
4. المسرى التسلسلي العام USB.

1.4. بروتوكول I^2C:

طوّرت شركة (Philips Semiconductors) البروتوكول المسمى Inter–IC أو IIC أو I^2C، وهو بروتوكول تسلسلي على سلكين. يسمح هذا البروتوكول للدارات المتكاملة الطرفية في النظم الإلكترونية بالاتصال فيما بينها باستعمال بنية عتادية بسيطة. بالاعتماد على المواصفات الأصلية، يوفر هذا البروتوكول معدل تراسل يصل إلى 100 Kbit/s وعنونة على 7 بت، وهذا ما يسمح بربط عدد من التجهيزات يصل إلى 128 جهاز على نفس المسرى. تم فيما بعد تطوير هذا البروتوكول ليشمل نمطاً سريعاً تبلغ سرعة التراسل فيه 3.4 Mbit/s مع عنونة على 10 بتات. ومن التجهيزات الشائعة التي تدعم الربط مع مسرى I^2C نذكر الذواكر من نوع EEPROM و Flash وبعض الذواكر من نوع RAM إضافة إلى ساعات الزمن الحقيقي RTC والمؤقتات والمتحكمات الصغرية.

يبين الشكل التالي مثالاً بسيطاً عن شبكة I^2C.



مثال عن شبكة اتصال باستعمال المسرى I2C وتوصيف وبروتوكول التراسل

يحتوي المسرى على خطين وهما خط المعطيات الذي يسمى خط المعطيات التسلسلي SDA (Serial Data Line) وخط الساعة المسمى SCL (Serial Clock Line). لا تحدد مواصفات هذا البروتوكول الطول الأعظمي للمسرى ولكن يجب أن تبقى قيمة سعة الخط أقل من 400 PF. لدينا في هذا المثال أربعة تجهيزات مربوطة مع المسرى. واحدة من هذه التجهيزات هي المتحكم الصغري الذي يلعب دور السيد (Master)، أما التجهيزات الأخرى وهي: حساس الحرارة والذاكرة EEPROM ومتحكم شاشة LCD فتلعب دور الخادم (Servants). يُسند لكل واحدة من هذه التجهيزات الخادمة عنوان وحيد. التجهيز السيد هو من يستطيع البدء بعملية نقل المعطيات. يسمح البروتوكول بوجود أكثر من سيد ضمن شبكة المسرى I^2C ولكن في معظم النظم يكون السيد هو المعالج أو المتحكم الصغري. يمكن لأي تجهيز في شبكة المسرى أن يكون مرسلاً أو مستقبلاً للمعطيات. يكون خط الساعة SCL دوماً مخرجاً من السيد ومدخلاً إلى التجهيزات التابعة، أما خط المعطيات SDA فهو ثنائي الاتجاه.

في وضع الراحة يضع السيد الخطين SCL وSDA على السوية العليا 1. تبدأ عملية التراسل بإشارة بداية (Start) وتنتهي بإشارة توقف (Stop).

- إشارة البداية: هي انتقال الخط SDA من 1 إلى 0 بينما يكون الخط SCL=1.
- إشارة النهاية: هي انتقال الخط SDA من 0 إلى 1 بينما يكون الخط SCL=1.

عملية الكتابة:

تجري عملية كتابة كلمة معطيات مؤلفة من بايت واحد بالطريقة التالية: يتم أولاً إعطاء إشارة البداية، ثم يتم إرسال بتات العنوان. يجري إرسال كل بت بوضع قيمة البت على خط المعطيات SDA وإخراج نبضة على خط الساعة SCL. بعد العنوان، يرسل السيد بتاً بقيمة 0 للإشارة إلى أنه يرغب في إجراء عملية كتابة. بعد ذلك يجعل السيد خط المعطيات دخلاً ليتلقى من التجهيز ذي العنوان المطلوب إشعاراً بالموافقة وذلك بوضع خط المعطيات SDA على قيمة الصفر خلال نبضة الساعة التالية. بعد ذلك يرسل السيد بتات المعطيات، ويلي هذا نبضة إشعار من التجهيز المحدد. أخيراً يرسل السيد إشارة التوقف.

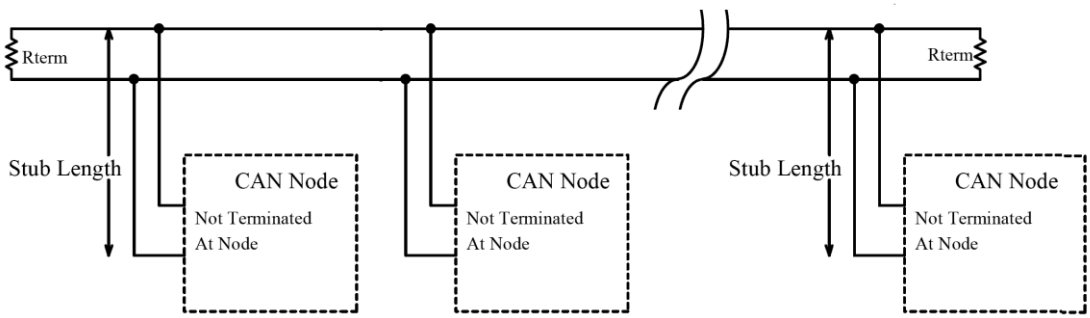
عملية القراءة:

تجري عملية القراءة بطريقة مماثلة، حيث يرسل السيد إشارة البداية ومن ثم عنوان التجهيز الذي يريد القراءة منه، ثم يرسل السيد بتاً بقيمة 1 للإشارة إلى أنه يرغب في إجراء عملية قراءة. بعد ذلك يجعل السيد خط المعطيات دخلاً ليتلقى إشارة الإشعار الأولى، ثم يرسل 8 نبضات ساعة ليقوم التجهيز المطلوب بإرسال 8 بتات معطيات. بعد ذلك يرسل السيد إشارة إشعار وينهي الإرسال بإشارة توقف.

يتم إرسال بتات العناوين أو المعطيات بدءاً من البت ذي الدلالة العظمى (MSB).

2.4. البروتوكول CAN:

بروتوكول متحكم الشبكة المحلية CAN (Controller Area Network) هو بروتوكول تراسل تسلسلي باستعمال زوج من الأسلاك. تم تطوير هذا البروتوكول في البداية ليُستعمل للاتصال مع جميع العناصر الإلكترونية في السيارات، وذلك لتقليل عدد الأسلاك اللازمة لوصل مختلف التجهيزات. وقد أدت متانة هذا البروتوكول إلى توسيع استخدامه في مجالات التحكم والأتمتة الصناعية كما في المصاعد والناسخات وخطوط الإنتاج والتجهيزات الطبية. هذا البروتوكول موثق بالرقم ISO II 519-2 للتطبيقات المنخفضة السرعة.



شبكة من التجهيزات موصولة على مسرى CAN

يبين الشكل شبكة من التجهيزات موصولة على مسرى CAN. لا يوجد في هذا البروتوكول سيد أو خادم، حيث تستطيع جميع التجهيزات المبادرة بالإرسال، كما أنه لا يوجد عناوين مسندة لكل عقدة بل يتم الاتصال على مبدأ نشر الرسائل (Message Broadcasting)، حيث تستطيع كل عقدة إرسال رسالة على المسرى وتستطيع جميع العقد الأخرى تلقي هذه الرسالة، إلا أن كل عقدة تقوم بترشيح الرسائل التي يتم نشرها على خطوط الشبكة ولا تأخذ إلا ما يهمها فقط.

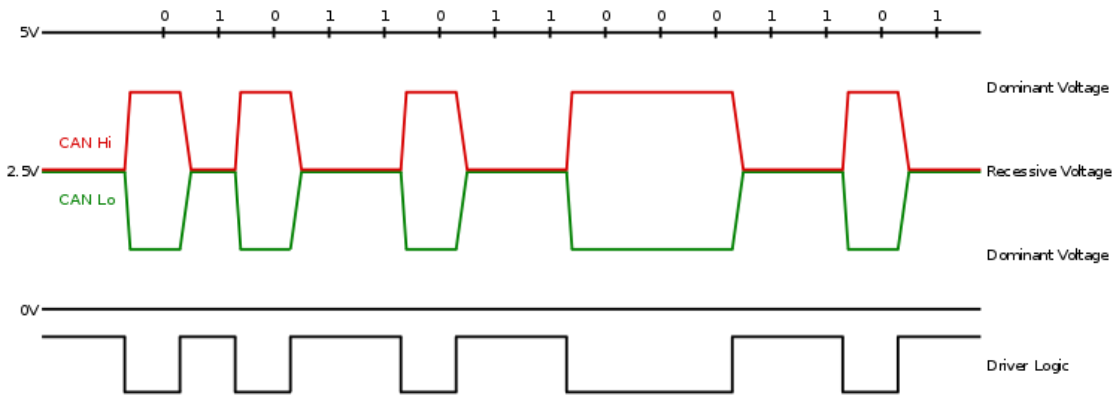
يتم إرسال الرسائل في الشبكة على شكل طرود، وهناك حقل مخصص ضمن الطرد يعبر عن رقم أو هوية الرسالة ممثلاً على 11 بتاً.

بما أن جميع العقد في الشبكة تستطيع أن تبادر بالإرسال، كان لابد من إدارة عملية التصادم الذي قد يحدث عندما تريد عقدتان أو أكثر الإرسال في الوقت نفسه. لذلك يَستخدم هذا البروتوكول مفهوم الأولوية اعتماداً على هوية الرسالة.

لا يوجد ساعة لإرسال المعطيات في هذا البروتوكول، ولكن تكون سرعة الإرسال متفقاً عليها مسبقاً كما هي الحال في التراسل التسلسلي UART.

يتم إرسال المعطيات على خطي التراسل بطريقة تفاضلية. حيث توجد حالتان لكل خط من المسرى وهما:

- الحالة السائدة (Dominant): وهي عملية وصل هذا الخط بجهد معين 0v أو 5v أو غير ذلك.
- الحالة المتنحية (Recessive): وهي أن يكون الخط في الحالة العائمة أي غير موصول بأي جهد.



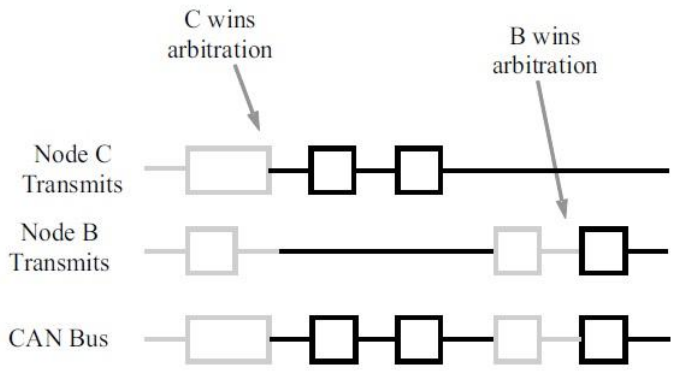
اشارات البروتوكول CAN.

يتم تمثيل القيمة المنطقية 0 عند إرسال المعطيات بوضع الخطين في الحالة السائدة مع وجود فرق جهد بينهما، أي يتم وصل أحد الخطوط بالجهد 0v والآخر بالجهد 5v مثلاً.

أما من أجل القيمة المنطقية 1 فيتم وضع الخطين في الحالة المتنحية، أي إن الخطين مفصولين عن أي منبع، لذلك توضع مقاومة بين الخطين ليكون فرق الجهد بينهما معدوماً.

من ذلك نجد أن قيمة 0 هي الغالبة على المسرى، فإذا وضعت إحدى العقد قيمة 0 على المسرى فسنجد هذه القيمة حتماً على المسرى مهما كانت القيم التي تضعها بقية العقد. تُستخدم هذه الميزة لكشف التصادم عند الإرسال، حيث تراقب كل عقدة حالة المسرى بعد إرسال البت 1 فإذا وجدت قيمة 0 على المسرى تتوقف هذه العقدة عن الإرسال مؤقتاً لتتيح المجال للعقدة الأخرى للانتهاء من إرسالها وتعود هي للمحاولة في وقت لاحق.

ينعكس ذلك على أولوية الرسائل في الانتشار على المسرى. فعند تصادم رسالتين عند موقع بت معين، يكون للرسالة ذات البت 0 الأولوية. فعلى سبيل المثال ترسل العقدة C رسالة برقم هوية 001011. وفي نفس الوقت ترسل العقدة B رسالة برقم هوية 00110. لا يوجد تصادم في البتات الثلاثة الأولى لأنها متماثلة بين العقدتين. يحدث التصادم عند البت الرابع. تأخذ رسالة العقدة C الأولوية لأن البت الرابع هو 0 ولذلك تتوقف العقدة B عن الارسال وتستمر العقدة C في ارسال رسالتها على المسرى.

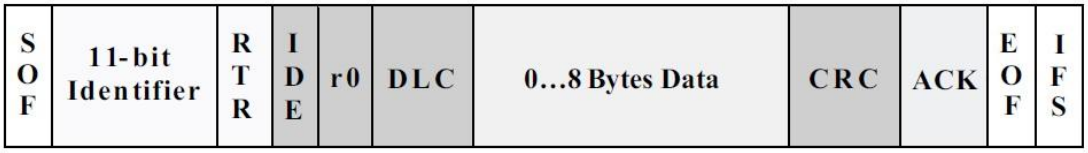


Arbitration on a CAN Bus

إدارة التصادم ومفهوم الأولوية على مسرى CAN.

بالرغم من وجود خوارزميات إرسال متقدمة لمنع التصادم في هذا البروتوكول، إلا أن إمكانية التصادم لا تزال قائمة. لذلك يتم استخدام تقنيات الكشف عن الأخطاء باستخدام خوارزمية CRC على 16 بتاً وإعادة الإرسال عند الكشف عن وجود خطأ.

يبين الشكل التالي بنية الإطار في بروتوكول CAN المعياري.



Standard CAN: 11-Bit Identifier

بنية الإطار في بروتوكول CAN المعياري

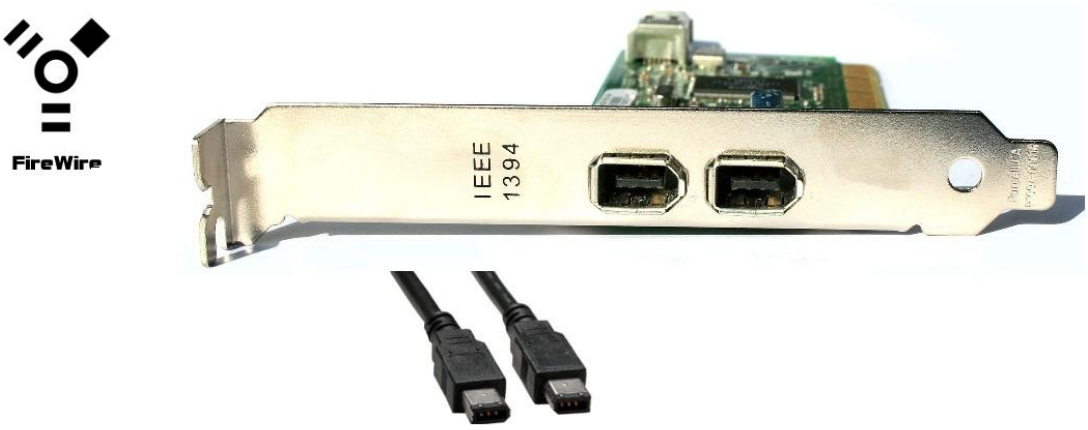
يحتوي الإطار على إشارة بداية الإطار SOF، ومن ثم هوية الرسالة على11 بتاً، ومن ثم مجموعة من البتات التحكمية. يلي ذلك المعطيات التي قد تحتوي على 8 بايتات ومن ثم حقل مجموع الكشف عن الأخطاء CRC وإشارة الإشعار ACK، وأخيراً إشارة نهاية الإطار EOF. يضاف إلى ذلك حقل يدعى الفراغ بين الأطر IFS (Inter Frame Space) يتكون من 7 بت لإتاحة الوقت للمستقبِل لنسخ الإطار إلى الذاكرة وتفسيره قبل أن يأتي الإطار التالي.

لا نريد هنا الخوض أكثر في تفصيلات هذا البروتوكول ولكن نستطيع تلخيص مزايا بروتوكول CAN بالنقاط التالية:

1. هو بروتوكول تراسل تسلسلي تفاضلي على خطين.
2. قادر على كشف التصادم على مستوى البت.
3. يحتوي الإطار على 8 بايتات من المعطيات محمية بمجموع الكشف عن الأخطاء.
4. لا يوجد عنونة ظاهرية للعقد ولكن توجد هوية للرسائل تحكُم أولويتها في الانتشار على المسرى وتحدد نوع الرسالة.
5. يدعم عملية إعادة الإرسال للتغلب على أخطاء الإرسال.
6. يدعم آلية عزل العقد المعطلة كي لا تؤثر على باقي العقد في الشبكة.

3.4. السلك الناري:

السلك الناري (FireWire) عبارة عن بروتوكول تسلسلي عالي الأداء تم تطويره من قبل شركة Apple، وهو بروتوكول موصف بالمعيار IEEE 1394. جرى تطوير هذا البروتوكول نتيجة للحاجة إلى نقل كميات كبيرة من المعطيات بسرعة كبيرة وخصوصاً للتطبيقات التي تعمل ضمن الزمن الحقيقي مثل نقل الفيديو، حيث تدعم مواصفات هذا البروتوكول سرعات نقل من 12.5 إلى 400 Mbit/s، إضافة إلى عنونة 64 بتاً وإمكانية العنونة الديناميكية (Plug and Play).
والنسخة الأكثر انتشاراً من هذا البروتوكول هي(IEEE 1394a) 400 FireWire.



شكل مرابط الاتصال لبروتوكول السلك الناري

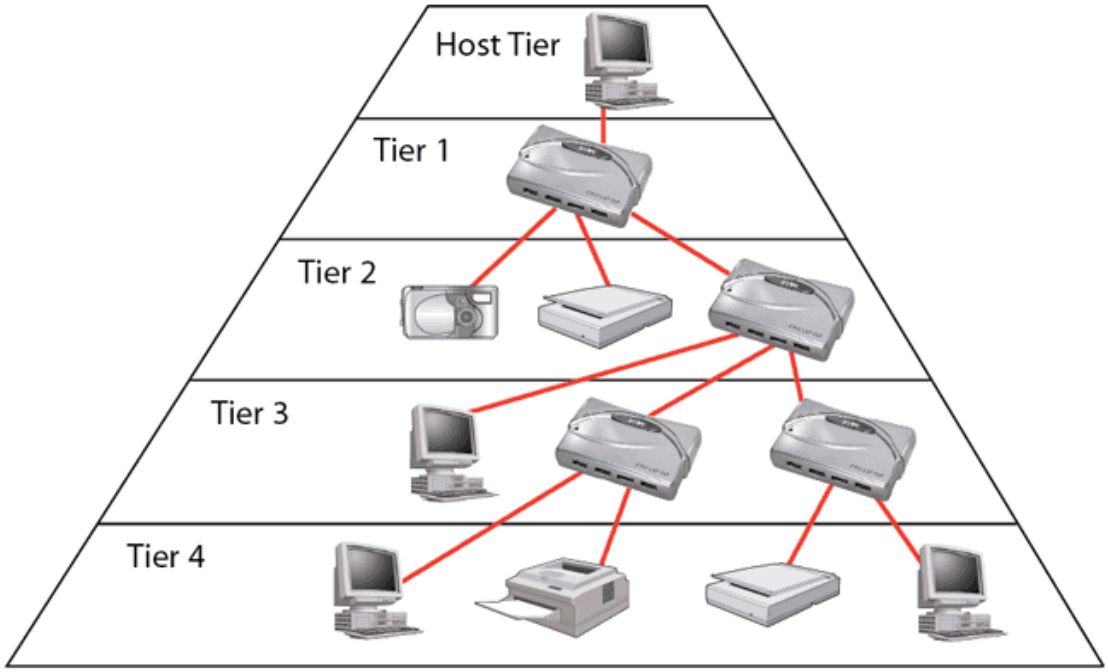
بخلاف البروتوكولين I^2C وCAN المصمّمين لربط الدارات المتكاملة فإن هذا البروتوكول مصمم لربط التجهيزات الكبيرة مثل الحاسوب والماسح الضوئي. يستطيع هذا البروتوكول ربط شبكة محلية بأكملها بشكل مماثل لبروتوكول إيثرنت Ethernet وهذا بفضل العنونة على 64 بتاً، حيث يقسم العنوان إلى 10 بتات لعنوان الشبكة و6 بتات لرقم العقدة و48 بت لعنوان الذاكرة، وهذا يقابل شبكة مكونة من 1024 شبكة فرعية تحتوي كل منها على 64 عقدة، وكل عقدة تمتلك سعة تخزينية قدرها 281 تيرابايت.
يمكن استعمال السلك الناري لتطبيقات التخزين على الأقراص الصلبة إضافة إلى الطابعات والماسحات الضوئية والكاميرات الرقمية أو أي نوع من الأجهزة. ومن الفوائد الهامة لهذا البروتوكول إمكانية وصل التجهيزات بطريقة سلسلة ديزي (Daisy-Chain)

4.4. المسرى التسلسلي العام USB:

صُمّم المسرى التسلسلي العام USB (Universal Serial Bus) لتسهيل ربط الحاسوب الشخصي مع الأجهزة الطرفية مثل الشاشات والطابعات ومكبرات الصوت وعصا القيادة للألعاب وغيرها. يدعم المسرى USB سرعتين للمعطيات هما 2 Mbps للطرفيات ذات السرعة العالية و1.5 Mbps للطرفيات ذات السرعة المنخفضة.

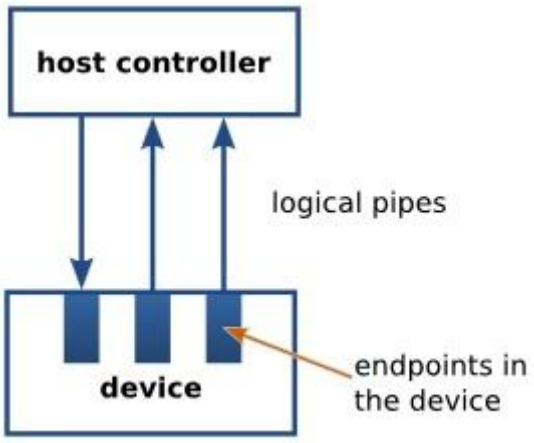
رمز وصلة USB وشكل المربط

يتم وصل الطرفيات مع هذا المسرى وفق الطبولوجيا النجمية (Tiered–Star Topology) وهذا يعني أن هناك موزع يتم وصله مع الحاسوب يسمى USB Hub يُستعمل للربط مع جميع الطرفيات، حيث يمكن وصل ما يصل إلى 127 طرفية بهذه الطريقة. إذاً فبنية شبكة USB غير متناظرة وهي تتكون من مضيف (Host) وتجهيزات طرفية. يتكون المسرى من 4 خطوط، خطين للمعطيات وخطين للتغذية.



نسمي المعطيات المرسلة من المضيف إلى الطرفيات بالمجرى الهابط (Downstream) والمعطيات المرسلة من الطرفيات إلى المضيف بالمجرى الصاعد (Upstream). يمكن تقسيم المجرى الهابط إلى العديد من البوابات (Ports) عن طريق الموزع.

يعتمد الاتصال بين الطرفية والمضيف على مفهوم الأنابيب Pipes وهي قنوات منطقية. يصل الأنبوب بين المضيف ووحدة منطقية موجودة في الطرفية تدعى نقطة نهاية (Endpoint) حيث يمكن أن تحتوي الطرفية على 32 نقطة نهاية (16 دخل و16 خرج) مع أنه نادراً ما يتم استخدام هذا العدد من نقاط النهاية.



الأنابيب في وصلة USB بين المضيف والطرفية

يدير متحكم المضيف برنامج القيادة وعرض المجال المطلوب من قبل كل طرفية وذلك بشكل آلي دون تدخل المستخدم، كما يستطيع الكشف عن وصل أي طرفية جديدة للمسرى ليخصص لها جزءاً من مجرى المعطيات والتغذية الكهربائية اللازمة عن طريق كبل USB بطول قد يصل إلى 5 متر كحد أقصى.

5. البروتوكولات التفرعية:

1.5. المسرى PCI:

المسرى PCI (Peripheral Component Interconnect) هو مسرى ذو مواصفات عالية لوصل الشرائح وبطاقات التوسعة مثل الذواكر وبطاقة الفيديو التي تركّب على البطاقة الأم في الحواسيب الشخصية. تم تطوير هذا المسرى من قبل شركة Intel في بدايات 1990 وتم فيما بعد اعتماده من قبل الجهات الصناعية كمعيار، حيث تم استخدامه لأول مرة في الحواسيب الشخصية عام 1994 مع معالجات Intel 486. وقد حلّ هذا المسرى محل المسرى القديم ISA / EISA.

من خواص هذا المسرى أنه يدعم معدل معطيات من 127 إلى 508 Mbits / s مع عنونة على 32 بتاً. تتم عمليات النقل على هذا المسرى بشكل متزامن مع إشارة الساعة.

يتم نقل المعطيات على نفس مسرى العناوين بالتناوب (Multiplexed)، وتم فيما بعد توسيع هذا البروتوكول ليشمل عنونة على 64 بتاً مع الحفاظ على التوافق مع المسرى الأصلي بعنونة على 32 بتاً.

2.5. المسرى ARM:

بالرغم من اعتماد المسرى PCI كمعيار صناعي فهناك مساري أخرى تم تطويرها من قبل بعض الشركات مثل المسرى ARM المسمى على اسم الشركة المطورة وهي ARM Corporation.

يشابه المسرى ARM المسرى PCI حيث يدعم العنونة على 32 بتاً باستعمال النقل المتزامن مع الساعة. لا يحدد المسرى ARM معدل نقل المعطيات ولكن يتبع معدل النقل تردد ساعة المسرى، فإذا كان تردد ساعة المسرى هو X يكون معدل نقل المعطيات هو 16 X bits / s.

6. البروتوكولات اللاسلكية:

1.6. بروتوكول الاتصال IrDA:

قامت المؤسسة العالمية IrDA (Infrared Data Association) بتطوير هذا البروتوكول لنقل المعطيات بين نقطتين باستخدام الأشعة تحت الحمراء، وهي وصلة قليلة الكلفة تدعم نقل المعطيات بسرعة بين 9.6 Kbps و4Mbps. وقد تم استخدام هذه التقنية بشكل واسع في الفترة الأخيرة في الحواسيب الشخصية المحمولة والهواتف الذكية والكاميرات الرقمية وغيرها من التجهيزات.

2.6. بروتوكول الاتصال Bluetooth:

أصبح بروتوكول Bluetooth معياراً للوصل اللاسلكي. يعتمد هذا البروتوكول على استعمال وصلة راديوية قليلة الكلفة وقصيرة المدى، حيث يصل مدى هذه الوصلة إلى مسافة 10 أمتار تقريباً. وبما أن هذا البروتوكول قائم على وصلة راديوية فليس هناك حاجة لوجود خط نظر بين التجهيزات المتصلة. أصبح هذا البروتوكول شائع الاستخدام في أجهزة الحاسوب والهواتف الذكية وسماعات الأذن والكثير من الطرفيات الأخرى.

3.6. بروتوكول الاتصال IEEE 802.11 (WiFi):

تم اعتماد بروتوكول الاتصالات IEEE 802.11 في عام 1997 كمعيار للشبكات المحلية اللاسلكية WLAN، حيث حدد هذا المعيار سرعتين للتراسل هما 1 Mbps و2 Mbps. يعرّف المعيار مواصفات الطبقة الفيزيائية PHY وطبقة الربط MAC، وهو يستعمل الترددات الراديوية في مجال الترددات غير المرخصة والمسمى المجال الصناعي والعلمي والطبي ISM Band (Industrial, Scientific and Medical Band) بجوار التردد 2.4 GHZ، ثم تلاحقت التطورات على هذا البروتوكول بهدف زيادة سرعة الاتصال. على سبيل المثال، زادت سرعة التراسل في النسخة المطورة 802.11b لتصل إلى 11 Mbps. أما في النسخة 802.11g فقد وصلت سرعة التراسل إلى 54 Mbps. حتى أن السرعة وصلت في النسخة 802.11n إلى 300 Mbps باستخدام تقنية الهوائيات المتعددة.

تعتمد بنية الشبكة اللاسلكية على وجود نقاط نفاذ (Access Points) مركزية يتم عن طريقها الاتصال بين عقد الشبكة، وتدار عن طريقها عملية توزيع العناوين. تكون نقطة النفاذ عادة موصولة مع الشبكة الثابتة لتوسيع نطاق اتصال العقد مع الشبكة العالمية.

تختلف الشبكات اللاسلكية عن الشبكات الثابتة بعدة نقاط:

- طبقة فيزيائية وطبقة ربط متخصصة ولاسيما إدارة تجنب التصادم.
- إدارة حركة العقد وبالتالي إسقاط العقدة عند خروجها عن نطاق التغطية.
- إضافة مزايا أمان الاتصال لكون وسط النقل مكشوف للغير.

7. خلاصة:

تستعمل النظم المضمنة العديد من وسائل الاتصال بين وحدة المعالجة المركزية والطرفيات القريبة والبعيدة. يعد الاتصال التفرعي الأكثر سرعة ولكنه مخصص للمسافات القصيرة أي للطرفيات القريبة من وحدة المعالجة المركزية. ومن المساري الشائعة الاستعمال مسرى PCI ومسرى ARM. ومن أجل الطرفيات البعيدة نسبياً، نستعمل البروتوكولات التسلسلية بهدف خفض كلفة كابلات التوصيل. ومن البروتوكولات التسلسلية الشائعة I2C وCAN وFireWire وUSB. ومن أجل التخلص من الحاجة لأسلاك التوصيل، يتم استعمال بروتوكولات التراسل اللاسلكية التي توفر سهولة التوصيل بين الطرفيات. ومن البروتوكولات التراسل اللاسلكية الشائعة IrDA وBlutooth وIEEE802.11 أو WiFi.

8. أسئلة الفصل الحادي عشر:

أسئلة عامة:

1. ما هي أنواع الاتصالات الأساسية في النظم المضمنة؟

هناك ثلاث أنواع للاتصالات وهي:

- الاتصال التفرعي
- الاتصال التسلسلي
- الاتصال اللاسلكي.

2. ما هي خطوط التراسل في المسرى I2C؟

يحتوي المسرى على خطين وهما:

- خط المعطيات الذي يسمى خط المعطيات التسلسلي SDA
- خط الساعة المسمى SCL.

3. عدد مزايا بروتوكول CAN.

A. هو بروتوكول تراسل تسلسلي تفاضلي على خطين.

B. قادر على كشف التصادم على مستوى البت.

C. يحتوي الإطار على 8 بايتات من المعطيات محمية بمجموع الكشف عن الأخطاء.

D. لا يوجد عنونة ظاهرية للعقد ولكن توجد هوية للرسائل تحكُم أولويتها في الانتشار على المسرى وتحدد نوع الرسالة.

E. يدعم عملية إعادة الإرسال للتغلب على أخطاء الإرسال.

F. يدعم آلية عزل العقد المعطلة كي لا تؤثر على باقي العقد في الشبكة.

4. اشرح كيفية العنونة في البروتوكول CAN لضمان وصول الرسائل إلى الجهة المطلوبة؟

لا يوجد عناوين مسندة لكل عقدة بل يتم الاتصال على مبدأ نشر الرسائل (Message Broadcasting)، حيث تستطيع كل عقدة إرسال رسالة على المسرى وتستطيع جميع العقد الأخرى تلقي هذه الرسالة، إلا أن كل عقدة تقوم بترشيح الرسائل التي يتم نشرها على خطوط الشبكة ولا تأخذ إلا ما يهمها فقط.

5. كيف يتم تمثيل قيم البتات 0 و1 في بروتوكول التراسل التسلسلي CAN؟

يتم تمثيل القيمة المنطقية 0 عند إرسال المعطيات بوضع الخطين في الحالة السائدة مع وجود فرق جهد بينهما، أي يتم وصل أحد الخطوط بالجهد 0v والآخر بالجهد 5v مثلاً.

أما من أجل القيمة المنطقية 1 فيتم وضع الخطين في الحالة المتنحية، أي إن الخطين مفصولين عن أي منبع، لذلك توضع مقاومة بين الخطين ليكون فرق الجهد بينهما معدوماً.

6. ما هي خطوط المسرى التسلسلي USB؟

يتكون المسرى من 4 خطوط، خطين للمعطيات وخطين للتغذية.

7. ما هي سرعة التراسل على مسرى ARM يعمل على تردد ساعة قدره 10 Mhz؟

- سرعة التراسل= 16*تردد الساعة بت/ثانية
- سرعة التراسل= $16*10^6$= 16 ميغا بت/ثانية.

8. بماذا تختلف الشبكات اللاسلكية عن الشبكات الثابتة؟

تختلف الشبكات اللاسلكية عن الشبكات الثابتة بالنقاط التالية:

- طبقة فيزيائية وطبقة ربط متخصصة ولاسيما إدارة تجنب التصادم.
- إدارة حركة العقد وبالتالي إسقاط العقدة عند خروجها عن نطاق التغطية.
- إضافة مزايا أمان الاتصال لكون وسط النقل مكشوف للغير.

أسئلة خيارات متعددة:

1. ما هو نوع الاتصال المناسب من أجل سرعة تراسل علية للمعالج مع طرفية على نفس البطاقة؟
   - **A.** الاتصال التفرعي.
   - **B.** الاتصال التسلسلي.
   - **C.** الاتصال اللاسلكي.
   - **D.** الاتصال عبر الايثرنت.

2. من فوائد الاتصال التسلسلي مقارنة ببقية أنواع الاتصال:
   - **A.** سرعة النقل العالية.
   - **B.** الكلفة المنخفضة للوسط الناقل.
   - **C.** التعقيد المنخفض لدارات الإرسال والاستقبال.
   - **D.** بساطة بروتوكول الاتصال.

3. من مزايا الاتصال باللاسلكي بالأمواج الراديوية:
   - **A.** الكلفة المنخفضة لدارات الارسال والاستقبال.
   - **B.** الموثوقية في عملية الاتصال.
   - **C.** مقاوم للتداخل الكهرطيسي.
   - **D.** عدم الحاجة إلى خط نظر بين المرسل والمستقبل.

4. لنفرض أننا نستعمل طريقة بت الزوجية بمجموع زوجي (Even Parity) حدد الكلمة الصحيحة من بين الكلمات المستقبلة التالية:
   - **A.** 010101110.
   - **B.** 011100110.
   - **C.** 010110110.
   - **D.** 010100110.

5. يسمح بروتوكول I2C بنسخته الأصلية من:
   - **A.** التراسل مع 128 طرفية بسرعة 3.4 Mbits / s.
   - **B.** التراسل مع 10 طرفيات بسرعة 3.4 Mbits / s.
   - **C.** التراسل مع 128 طرفية بسرعة 100 Kbits / s.
   - **D.** التراسل مع 1024 طرفية بسرعة 100 Kbits / s .

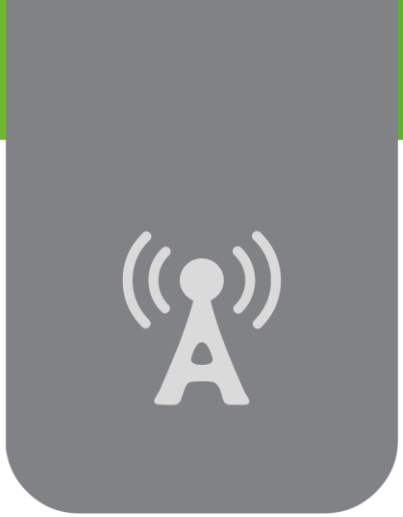
6. لنقل معطيات إشارة الفيديو في الزمن الحقيقي فإن البروتوكول الأنسب هو:
   - A. بروتوكول السلك الناري.
   - B. البروتوكول I2C.
   - C. البروتوكول CAN.
   - D. جميع ما سبق.

7. من هي الجهة التي تبادر في عملية الاتصال في بروتوكول التراسل التسلسلي CAN؟
   - A. السيد.
   - B. المعالج.
   - C. الخادم أو الطرفيات.
   - D. جميع العقد على المسرى.

8. ما هو الطول الأقصى لكبل USB؟
   - A. 1 متر.
   - B. 2 متر.
   - C. 5 متر.
   - D. 10 متر.

الإجابات الصحيحة

رقم التمرين	الإجابة الصحيحة
1	A
2	B
3	D
4	D
5	C
6	A
7	B
8	C



الفصل الثاني عشر:
نظم التشغيل في الزمن الحقيقي

الكلمات المفتاحية:

نظام التشغيل بالزمن الحقيقي، النواة، الجدولة، تعدد المهام، تبديل السياق.
Real-Time Operating System, Kernel, Scheduling, Multitasking, Context Switching.

الملخص:

يهدف هذا الفصل إلى التعريف بنظم التشغيل بالزمن الحقيقي وخصائصها الأساسية التي تميزها عن نظم التشغيل الأخرى. يتعرف الطالب على مفاهيم جدولة المهام وتبديل السياق لتلبية متطلبات تعدد المهام في برامج النظم المضمنة. حيث نعرض أهم خوارزميات الجدولة وتتضمن الجدولة الاستباقية على أساس الأولوية والجدولة الدوارة. ثم نعرض مفهوم الأغراض والخدمات في نظام التشغيل التي تسهل تطوير برامج النظم المضمنة. وننهي الفصل ببعض الأمثلة عن نظم التشغيل الرائجة لتطبيقات الزمن الحقيقي.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- مفهوم نظام التشغيل بالزمن الحقيقي
- المجدول وخوارزميات جدولة المهام
- خصائص نظم التشغيل بالزمن الحقيقي
- بعض نظم التشغيل الشائعة

يهدف هذا الفصل إلى التعريف بنظم التشغيل بالزمن الحقيقي وخصائصها الأساسية التي تميزها عن نظم التشغيل الأخرى. يتعرف الطالب على مفاهيم جدولة المهام وتبديل السياق لتلبية متطلبات تعدد المهام في برامج النظم المضمنة. حيث نعرض أهم خوارزميات الجدولة وتتضمن الجدولة الاستباقية على أساس الأولوية والجدولة الدوارة. ثم نعرض مفهوم الأغراض والخدمات في نظام التشغيل التي تسهل تطوير برامج النظم المضمنة. وننهي الفصل ببعض الأمثلة عن نظم التشغيل الرائجة لتطبيقات الزمن الحقيقي.

1. مقدمة:

يشكل نظام التشغيل في الزمن الحقيقي RTOS (Real-Time Operation System) المفتاح الأساسي للعديد من النظم المضمنة في الوقت الراهن. وهو يوفر المنصة البرمجية التي تبنى عليها التطبيقات. مع ذلك، لا تحتوي كل النظم المضمنة نظام تشغيل، فهناك بعض التطبيقات التي لا تحتاج إلى ذلك وإنما تتطلب فقط برنامجاً بسيطاً.

عندما يزداد تعقيد النظام المضمن والبرمجيات اللازمة لتشغيله، يكون من المناسب تطوير التطبيقات بالاعتماد على نظام تشغيل جاهز بدلاً من بناء كل شيء من الصفر في هذا النظام المعقد. وهذا بدوره يقلص الزمن اللازم لتنفيذ النظام وتطويره لاحقاً.

2. نبذة تاريخية عن نظم التشغيل:

في الفترات المبكرة لظهور النظم الحاسوبية، استعمل مطورو النظم المضمنة لغة التجميع لبناء البرامج الضرورية للتعامل بشكل مباشر مع البنية العتادية للنظام. وهذا أدى بدوره إلى الترابط القوي بين البنية العتادية والبنية البرمجية بحيث أن أي تعديل في البنية العتادية قد يسبب الكثير من التغييرات في البنية البرمجية للنظام.

تطورت البرمجيات مع الزمن، وتم تصميم نظم التشغيل بهدف توفير سوية تجريدية كافية للتطبيقات من أجل التعامل مع البنية العتادية دون الخوض في تفاصيل التصميم العتادي. كما أدى ذلك إلى استبدال التطبيقات المنفردة ذات التعقيد المتزايد إلى عدة تطبيقات مستقلة تعمل مع بعضها وتتواصل فيما بينها بهدف تحقيق وظيفة النظام.

ظهرت العديد من نظم التشغيل ذات الأغراض العامة مثل UNIX و Microsoft Windows، كما ظهرت أيضاً نظم تشغيل أصغر وتعمل في الزمن الحقيقي مثل VxWorks.

في الستينات والسبعينات من القرن الماضي، تم تطوير نظام UNIX بشكل يسمح لعدة مستخدمين باستعمال النظام الحاسوبي الباهظ الثمن في نفس الوقت. وقد كان ذلك نجاحاً كبيراً لنظام UNIX الذي انتشر على معظم المراكز الحاسوبية الكبيرة.

في الثمانينات، طورت شركة مايكروسوفت نظام Windows للحواسيب الشخصية المعدة للاستعمالات المكتبية والشخصية. ولذا فقد كانت موجهة للاستعمال من قبل مستخدم وحيد. بعد ذلك انتشر استعمال النظم الحاسوبية بشكل أوسع في جميع المجالات، مما أدى إلى ظهور النظم المضمنة. لذلك تم تطوير العديد من نظم التشغيل القادرة على تشغيل هذه النظم، وتطلب ذلك أن تكون هذه النظم صغيرة الحجم وقادرة على تخديم تطبيقات تعمل في الزمن الحقيقي.

تتشابه نظم التشغيل في الزمن الحقيقي مع نظم التشغيل العامة بعدد من النقاط وهي:

- سوية معينة من دعم المهام المتعددة (multitasking).
- إدارة الموارد البرمجية والعتادية.
- دعم التطبيقات ببعض الخدمات البرمجية وتشكيل واجهة بينية بين التطبيق والبنية العتادية.

في المقابل فهناك بعض نقاط الاختلاف التي تتميز بها نظم التشغيل في الزمن الحقيقي عن نظم التشغيل العامة وهي:

- القدرة على تقييس النظام إما بتصغيره أو توسيعه حسب متطلبات التطبيق.
- أداء أسرع.
- متطلبات ذاكرة أقل.
- اتباع سياسات جدولة (Scheduling) مخصصة للنظم المضمنة في الزمن الحقيقي.
- إمكانية إقلاع بدون قرص صلب انطلاقاً من ذاكرة ROM أو RAM.

- إمكانية العمل على نظم عتادية مختلفة.

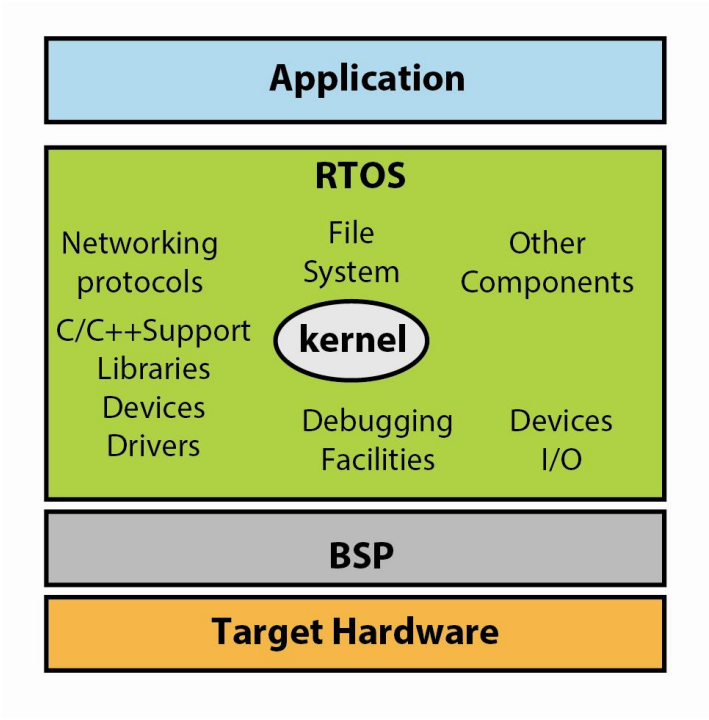
تُستعمل نظم التشغيل العامة هذه الأيام على الحواسيب الشخصية ومحطات العمل والتطبيقات المضمنة التي ليس لها متطلبات حرجة من ناحية العمل في الزمن الحقيقي. في المقابل، تستعمل نظم التشغيل في الزمن الحقيقي لتلبية متطلبات الزمن الحقيقي للتطبيقات والموثوقية والتقييس والحجم المحدود. حيث انتشر حديثاً مفهوم "النظام على شريحة" أو SoC (System on Chip) الذي يعني وجود المعالج وجميع طرفياته اللازمة وكذلك الذواكر على دارة متكاملة واحدة بحيث يمكننا أيضاً تزويد هذه الشريحة بنظام التشغيل. وبالتالي لم يتبقَّ من النظام العملي سوى وحدات الإدخال والإخراج والواجهة مع المستخدم التي توجد خارج هذه الشريحة. وقد سمح التطور الكبير في الدارات القابلة للبرمجة FPGA بتصميم معظم أجزاء النظام على شريحة FPGA واحدة وبزمن تطوير قصير نسبياً.

مرة ثانية ننوه بأنه لا زال هناك الكثير من النظم المضمنة في الوقت الراهن التي لا تحتوي على نظم تشغيل بداخلها.

3. تعريف نظام التشغيل في الزمن الحقيقي:

نظام التشغيل في الزمن الحقيقي RTOS هو برنامج يُجدول التنفيذ مع الزمن ويدير موارد النظام ويوفر القاعدة البرمجية الأساسية لتطوير التطبيقات.

وقد تتدرج التطبيقات التي يمكن تصميمها لتعمل على نظام تشغيل معين في تعقيدها ابتداء من التطبيقات البسيطة مثل مؤقت زمني إلى التطبيقات الكبيرة مثل برامج قيادة الطائرات. لذا، فمن الضروري تقييس نظام التشغيل بحسب متطلبات التطبيقات. فعلى سبيل المثال، قد يحتوي نظام التشغيل على النواة (kernel) فقط والتي تحتوي على أقل قدر من مهام جدولة التنفيذ بالإضافة إلى الحد الأدنى من إدارة الموارد. وقد نوسع نظام التشغيل هذا ليشمل إدارة الملفات وبروتوكولات التراسل على الشبكة (networking protocol stack).



المكونات الأساسية لنظام التشغيل

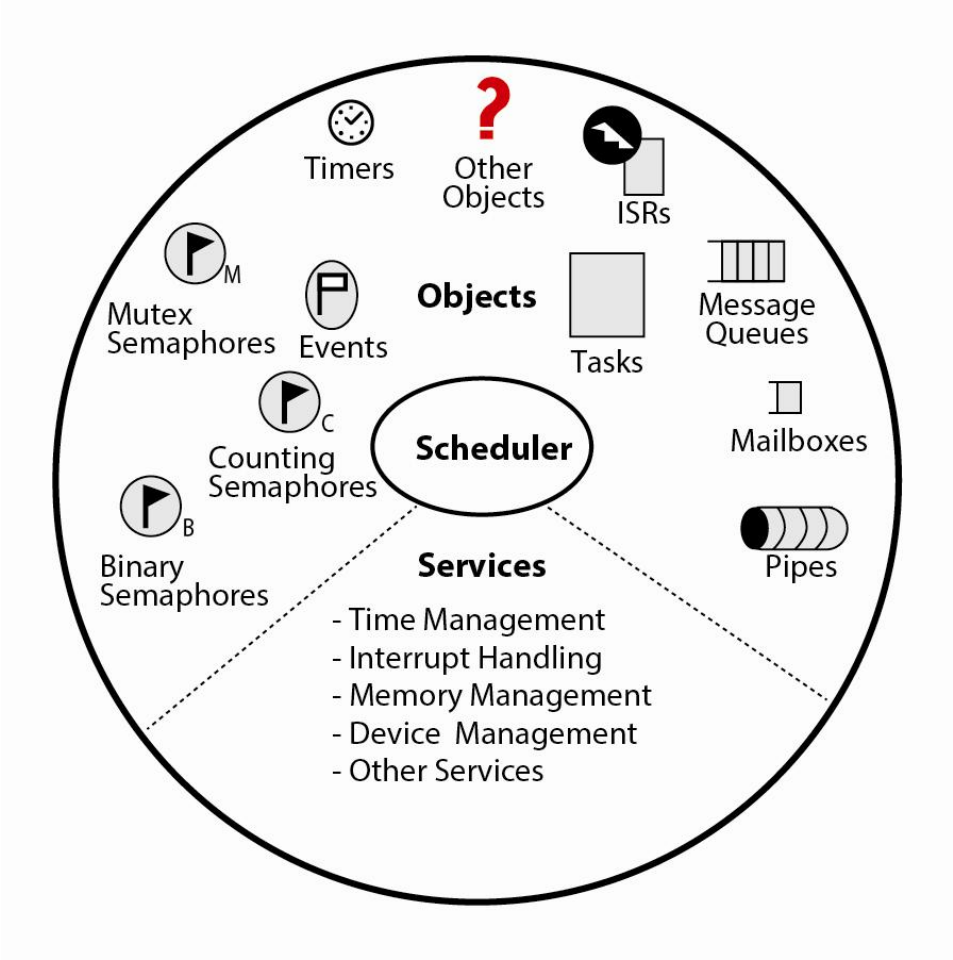
يبين الشكل التالي تمثيلاً لمكونات نظام التشغيل من النواة وبقية العناصر التي يمكن وجودها في النظم المضمنة.

سنركز في هذا الفصل على مكونات النواة الأساسية، وهي:

المُجدول (Scheduler): تشكل الجدولة (scheduling) أهم مهام النواة التي تنفذ العديد من الخوارزميات لتحديد أية مهمة سيتم تنفيذها وفي أي وقت. ومن الخوارزميات الشهيرة نذكر خوارزمية الحلقة المستديرة (robin round) وخوارزمية الجدولة الاستباقية (preemptive scheduling).

الأغراض (objects): وهي بنى خاصة بالنواة تساعد المطور على بناء التطبيقات. وتشمل الأغراض الأساسية المهام (tasks) والسيمافورات (Semaphores) وأرتال الرسائل (message queues).

الخدمات (services): وهي العمليات التي تجريها النواة على الأغراض أو بشكل عام عمليات التوقيت وإدارة المقاطعات وإدارة الموارد.



مكونات النواة الأساسية في نظام التشغيل

4. المُجدول:

يشكل المُجدول (scheduler) العنصر الأساسي في النواة وهو الذي يؤمن الخوارزميات اللازمة لتحديد أي مهمة ستأخذ دورها في التنفيذ ومتى.

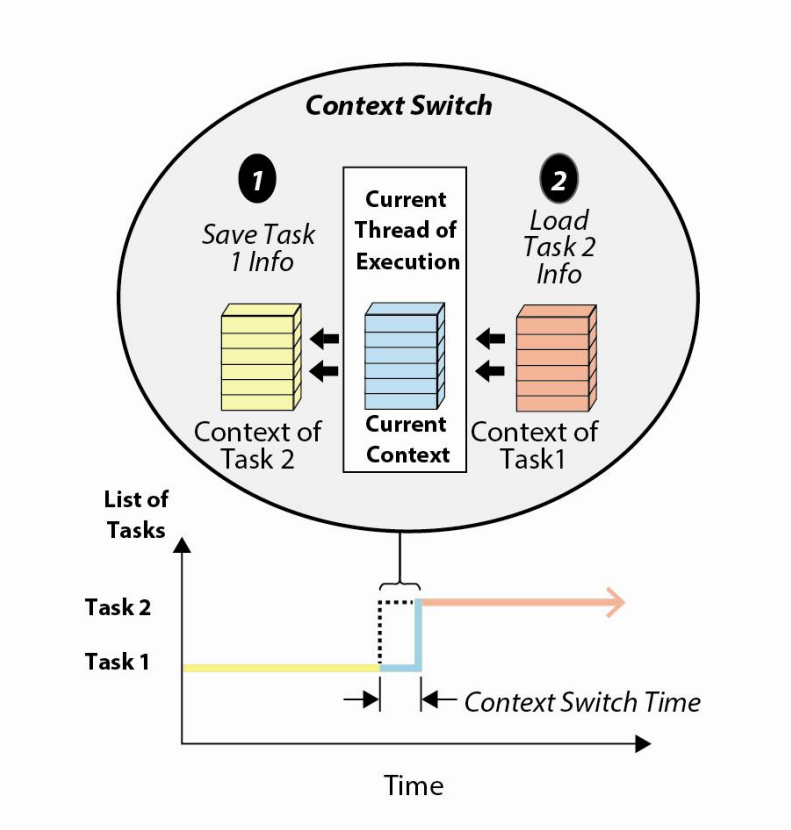
الأغراض القابلة للجدولة هي المهام (tasks) والعمليات (process). وكلاهما عبارة عن نَيْسَب (thread) مستقل يشمل مجموعة من التعليمات. ويكمن الاختلاف بينهما في أن للعملية مزايا حماية ذاكرة أفضل من المهمة. ولكن في سياق هذا الفصل سوف نستخدم مصطلح "مهمة" للإشارة إلى المهمة أو العملية دون أي تمييز بينهما. هذا ولا تشكل الرسائل أو السيمافورات أغراضاً قابلة للجدولة وإنما هي وسائل اتصال بين المهام بغرض التزامن في العمل.

كيف يستطيع المُجدول تلبية العديد من المهام التي تحتاج إلى التنفيذ. يكمن الجواب في مفهوم "تعدد المهام".

1.4. تعدد المهام:

نعرف تعدد المهام (multitasking) بأنه قابلية نظام التشغيل لتخديم مجموعة من المهام ضمن وقت محدد.

يبين الشكل التالي سيناريو تخديم لمهمتين.



سيناريو تخديم لمهمتين في نظام متعدد المهام.

في هذا السيناريو تقوم النواة بتوزيع وقت المعالج بين مهمتين وذلك وفق خوارزمية المُجدول التي سنأتي على ذكرها لاحقاً، حيث تتناوب المهمتان في التنفيذ وفي كل مرة يتم تحميل مهمة مع السياق (context) الموافق

لها. وتأخذ عملية الشحن هذه وقتاً معيناً. يتم ذلك الأمر بحث تبدو المهمتان تعملان بشكل متوازٍ. عند زيادة عدد المهام يأخذ المعالج وقتاً أطول في التبديل بين المهام مما يتطلب زيادة في أداء المعالج نفسه لكي يستطيع تشغيل جميع المهام بشكل مقبول.

تجدر الإشارة هنا إلى أن عملية تعدد المهام تجري بين المهام المختلفة. أما عملية تخديم المقاطعات فلا تخضع لخوارزمية التبديل هذه ولكن تتم فوراً حسب أولويات المقاطعات المختلفة.

2.4. تبديل السياق:

يتكون السياق من مجموعة معطيات التنفيذ التي تشمل قيم سجلات المعالج عند آخر مرة توقف عندها التنفيذ لمهمة معينة. تحصل عملية تبديل السياق (context switch) في كل مرة ينتقل المُجدول من تنفيذ مهمة إلى أخرى.

عندما يتم إنشاء مهمة جديدة، يتم أيضاً إنشاء كتلة تحكم بالمهمة TCB (Task Control Block) تحتوي على جميع المعلومات التي يحتاجها المُجدول لتشغيل المهمة. وعند تشغيل هذه المهمة يُحدِّث المُجدول محتوى هذه الكتلة باستمرار بحيث يستطيع استكمالها فيما بعد عند توقف المهمة.

عند تبديل السياق من المهمة الأولى إلى الثانية، يحدث مايلي:

- تحفظ النواة سياق المهمة الأولى في كتلة التحكم الخاصة بهذه المهمة.
- تشحن النواة سياق المهمة الثانية من كتلة تحكم المهمة الخاصة بالمهمة الثانية، وتصبح المهمة الثانية في وضع التشغيل.
- يتم تجميد محتوى كتلة التحكم الخاصة بالمهمة الأولى لكي تستطيع النواة متابعة تنفيذ المهمة الأولى في المرة القادمة.

زمن تبديل السياق هو الزمن الذي يأخذه المُجدول للتبديل بين مهمتين. وهو وقت إضافي ضائع لا يتم فيه تنفيذ أي مهمة. لذلك يجب أخذ هذا الزمن بعين الاعتبار عند تقييم كفاءة خوارزميات تبديل المهام لكي لا يضيع معظم وقت المعالج في التبديل بين المهام. مما يؤثر سلباً على أداء نظام التشغيل.

في كل مرة يطلب تطبيق ما التشغيل من النظام، يحدد المُجدول فيما إذا كان سيجري تبديل السياق أم لا بالاعتماد على الموزع (dispatcher) وهو جزء من المُجدول يقوم بعملية تبديل السياق.

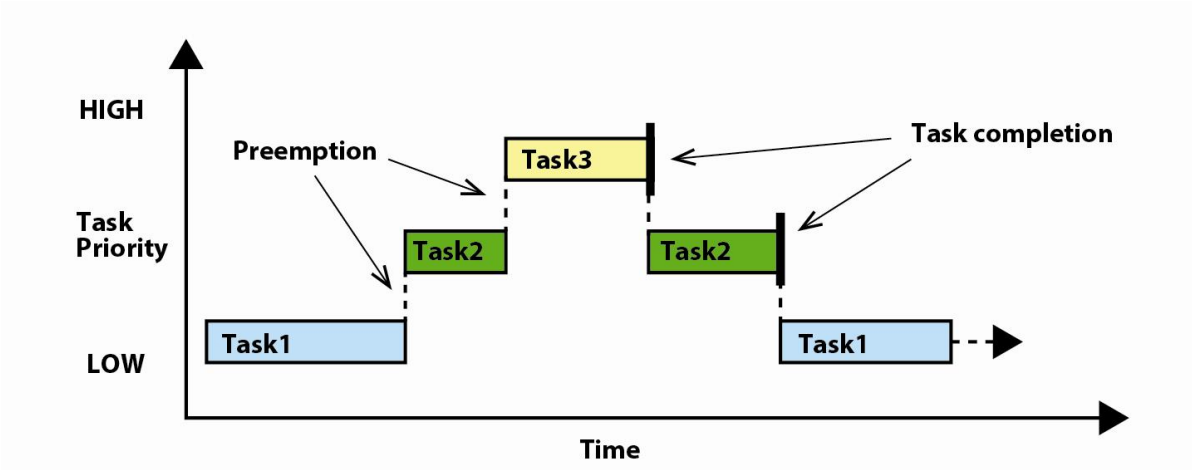
يتنقل نظام التشغيل في كل لحظة بين إحدى الحالات التالية: تنفيذ مهمة، أو تنفيذ مقاطعة، أو يقوم بتنفيذ خوارزميات إدارة النواة. ويسمى تسلسل التنفيذ بين هذه الحالات الثلاث بتدفق التحكم (flow of control). عند لحظة الخروج من النواة يحدد الموزع المهمة التي ستأخذ وقت التنفيذ. عند الذهاب لتخديم مقاطعة، ينتظر الموزع الانتهاء من تنفيذ المقاطعة بالكامل قبل أن يحدد المهمة التالية التي ستأخذ زمن التنفيذ. لذلك يجب أن يحرص المبرمج على أن يكون زمن تنفيذ المقاطعة أقل ما يمكن لكي لا يؤثر على تنفيذ المهام التي تتطلب العمل في الزمن الحقيقي. لذلك عادةً ما تتم برمجة إجرائيات تخديم المقاطعات بلغة التجميع بهدف تقليل زمن تنفيذ المقاطعة.

3.4. خوارزميات الجدولة:

تدعم معظم النوى في نظم التشغيل خوارزميتين شهيرتين في الجدولة:

1.3.4. الجدولة الاستباقية على أساس الأولوية:

تعتبر هذه الطريقة (preemptive priority-based scheduling) الطريقة الافتراضية في معظم نظم التشغيل في الزمن الحقيقي. يبين الشكل التالي بأن المهمة التي تأخذ زمن التنفيذ هي المهمة ذات الأولوية العليا بين جميع المهام الجاهزة للتنفيذ.



الجدولة الاستباقية على أساس الأولوية

يبين الشكل تسلسل تنفيذ ثلاث مهام ذات أولويات مختلفة. في البداية كانت المهمة Task1 في طور التنفيذ، ثم طلبت المهمة Task2 التنفيذ ولكونها ذات أولوية أعلى، علّق المُجدول تنفيذ المهمة Task1 وباشر في تنفيذ المهمة Task2. بعد ذلك أتت المهمة Task3 ذات الأولوية الأعلى لتأخذ زمن التنفيذ حتى انتهت. عاد المُجدول بعد ذلك لإتمام المهمة Task2 ومن ثم الرجوع إلى المهمة Task1.

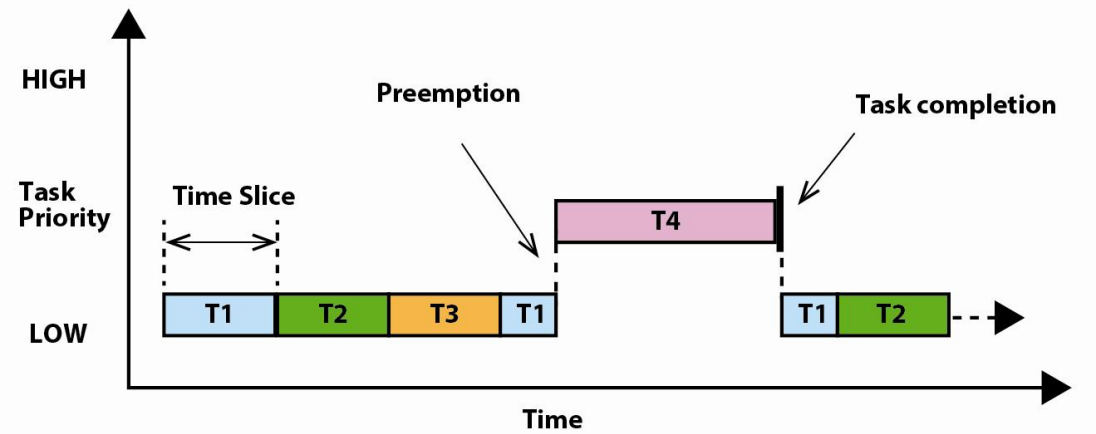
تدعم نظم التشغيل في الزمن الحقيقي عادةً 256 درجة أولوية، حيث تعتبر السوية 0 ذات الأولوية العليا والسوية 255 ذات الأولوية الدنيا. وهناك نظم تشغيل تعتبر الترتيب المعاكس.

يتم إسناد درجة الأولوية لكل مهمة عند إنشائها، ولكن من الممكن أن يقوم المُجدول بتغيير هذه الأولويات بشكل ديناميكي أثناء التنفيذ بالاعتماد على استدعاء بعض التوابع الخاصة في النواة. ولكن عدم الاستعمال المناسب لهذه الميزة قد يسبب أحياناً عكس الأولويات أو فشل النظام بالكامل.

2.3.4.الجدولة الدوّارة:

تعتمد خوارزمية الجدولة الدوارة (round-robin scheduling) على التوزيع المتساوي لوقت المعالج بين جميع المهام. ولكن لا توفر هذه الخوارزمية لوحدها إمكانية العمل في الزمن الحقيقي لأن المهام المتعددة

تختلف فيما بينها من حيث أهمية التخديم. لذا يتم غالباً استخدام هذه الخوارزمية مع الخوارزمية السابقة بهدف توزيع وقت المعالج بشكل متساوٍ بين المهام ذات درجات الأولوية المتساوية.
يبين الشكل التالي عملية المزج بين الخوارزميتين.



مزج الجدولة الدوارة مع الجدولة الاستباقية في نظام التشغيل.

في البداية كان المُجدول يوزع وقت المعالج بالتساوي بين عدة مهام متساوية في الأولوية. أي وفقاً لخوارزمية الجدولة الدوارة، حيث يتم التبديل بين مهمة وأخرى بفواصل زمنية متساوية. بعد أن طلبت المهمة Task4 ذات الأولوية الأعلى التخديم، انتقل المجدول لتخديم هذه المهمة حتى انتهت وعاد إلى تنفيذ بقية المهام بالتساوي كالسابق.

5. الأغراض:

الأغراض (objects) هي عبارة عن بنى تشكل الكتل الأساسية لتطوير تطبيق ما من أجل نظام مضمن في الزمن الحقيقي. ومن الأغراض الشائعة الاستخدام في نظم التشغيل:

- **المهام (tasks):** وهي عبارة عن نياسب تنفيذ مستقلة وتتشارك فيما بينها زمن تنفيذ وحدة المعالجة المركزية.
- **السيمافورات (semaphores):** وهي عبارة عن نمط معطيات يأخذ قيماً منتهية يستعمل بهدف إدارة الموارد المشتركة بين المهام المختلفة وذلك منعاً للتصادم. ومن الأمثلة على الموارد المشتركة في النظم المضمنة نذكر الذواكر وبوابات الدخل والخرج والملفات وغيرها. لفهم معنى السيمافور نأخذ المثال البسيط التالي: ليكن لدينا ثلاثة مصاعد في فندق، يطلب الزبون أحد المصاعد، فإذا كان هناك مصعداً غير مستخدم يتم تلبية الزبون. يعبر السيمافور هنا عن عدد المصاعد المتوفر في كل لحظة. تمثل المصاعد في هذا المثال المورد المشترك، ويمثل الزبون مهمة تطلب التخديم من هذا المورد المشترك، ويمثل السيمافور عداد المصاعد المتوفرة، وبالتالي يستعمل السيمافور كشرط لمعرفة إذا توفر المورد للاستعمال من قبل المهمة. وعندما يتيح المورد الاستخدام الوحيد من قبل مهمة، نسمي السيمافور الموافق بالسيمافور الثنائي (Binary Semaphore) وذلك لأنه يأخذ القيم المنتهية 0 أو 1 فقط.
- **أرتال الرسائل (message queues):** وهي عبارة عن الرسائل المتبادلة بين المهام بهدف تنظيم العمل والتزامن بين المهام لضمان صحة عمل التطبيق.

يستعمل مطوّرو النظم المضمنة في الزمن الحقيقي هذه الأغراض لحل مشاكل نظم الزمن الحقيقي مثل التنفيذ المتوازي، وتزامن العمليات، والتراسل.

6. الخدمات:

بالإضافة إلى الأغراض، تحتوي نظم التشغيل على مجموعة من الخدمات (services) التي تساعد المطور في كتابة تطبيقات النظم المضمنة. تشمل هذه الخدمات مجموعة من توابع النظام API (Application Programming Interface) التي يمكن استدعاؤها لإجراء وظيفة محددة مثل تابع لرسم خط على الشاشة، أو تابع لطباعة ملف على الطابعة، أو تابع لإرسال أمر عبر بوابة التراسل التسلسلية أو بوابة الإيثرنت وغيرها الكثير.

تشكل هذه الخدمات في معظم الأحيان الواجهة البينية بين المبرمج والبنية العتادية في النظام المضمن وذلك بهدف تسهيل تطوير النظام المضمن بسرعة دون الحاجة لبرمجة المكونات العتادية المعروفة. تُعتبر هذه التسهيلات من الأمور الأساسية التي دعت مطوري النظم المضمنة إلى استعمال نظم التشغيل في النظم المضمنة العالية التعقيد.

7. الخصائص العامة لنظم التشغيل في الزمن الحقيقي:

تحدد التطبيقات في الزمن الحقيقي مواصفات نظام التشغيل الذي تعمل عليه. نجمل الخصائص العامة لنظم التشغيل في الزمن الحقيقي بالنقاط التالية:

1.7. الموثوقية:

تعبر خاصية الموثوقية (reliability) عن العمل الصحيح للنظام ولفترة طويلة دون الوقوع في حالة فشل بحيث لا يستدعي التدخل المباشر والمتكرر من مستثمر النظام لإعادة تشغيله أو تصحيح بعض مكوناته. وتختلف التطبيقات في درجة الموثوقية المطلوبة من النظام. يقاس ذلك عادةً بزمن خروج النظام عن الخدمة خلال سنة. فمثلاً، من أجل التطبيقات المكتبية، يقدر هذا الزمن بتسع ساعات خلال السنة لكي يعتبر النظام موثوقاً. أما في مبدلات الاتصالات فيجب أن لا يتجاوز هذا الزمن 30 ثانية فقط خلال السنة.

2.7. السلوك القابل للتنبؤ به:

تعبر خاصية السلوك القابل للتنبؤ به (predictability) عن قدرة النظام على إنجاز المهمة خلال فترة زمنية معلومة لا تتغير بشكل كبير من ظرف إلى آخر.

3.7. الأداء:

يعبر الأداء (performance) عن سرعة النظام المطلوبة لتنفيذ العمليات في الزمن الحقيقي دون أي تأخير ملحوظ. ويتم قياس هذا الأداء من خلال تشغيل برامج اختبارية على النظام لمعرفة الزمن الحقيقي الذي استغرقه النظام لتنفيذ هذا البرنامج.

4.7. الحجم المحدود:

تحدد خاصية الحجم المحدود (compactness) بشكل أساسي متطلبات حجم الذاكرة اللازم لتشغيل النظام والتطبيق. حيث يجب أن يكون هذا الحجم صغيراً بقدر كافٍ بما يتناسب وإمكانيات وحجم النظم المضمنة من حيث الذاكرة الحية أو ذاكرة البرنامج.

5.7.قابلية التقييس:

تحدد خاصية قابلية التقييس (scalability) إمكانية تقليص أو توسيع حجم نظام التشغيل وفقاً لمتطلبات التطبيق، حيث تختلف التطبيقات المضمنة بشكل كبير في متطلباتها، لذا يجب أن يواكب نظام التشغيل هذه المتطلبات عن طريق جعل مكونات النظام مستقلة عن بعضها قدر الإمكان. فمثلاً، لا داعي لتضمين خدمات الاتصال مع شبكة الإنترنت في النظام إذا كان النظام المضمن لا يحتوي على مزايا الربط مع شبكة الإنترنت،

مما يوفر حجم الذاكرة المطلوب لتنصيب النظام. وبالعكس، يمكن إضافة مكونات جديدة إلى النظام لدعم بعض المزايا المتقدمة في النظم المعقدة، مثل الاتصال اللاسلكي.

8. بعض نظم التشغيل:

هناك العشرات من نظم التشغيل في الزمن الحقيقي المتوافرة حالياً في العديد من التطبيقات العملية. فالبعض منها مطور من قبل الشركات غير مفتوحة المصدر. والبعض الآخر ذو سعر مرتفع. وهنالك أيضاً نظم تشغيل مجانية ومفتوحة المصدر. وتختلف نظم التشغيل أيضاً في المعالجات التي تدعمها، لذا عند اختيار نظام التشغيل المناسب، يجب أخذ كل هذه المعاملات بعين الاعتبار وفقاً لبنية المعالج في النظام المدمج.
نذكر بعضاً من نظم التشغيل الشائعة فقط على سبيل المثال.

1.8. النظام VxWorks:

لهذا النظام تاريخ طويل في صناعة السيارات والعديد من منصات وكالة ناسا الفضائية (NASA) ولكنه نظام غير مجاني وغير مفتوح المصدر. يدعم هذا النظام معالجات x86 وPowerPC وARM وMIPS وغيرها، ويتميز هذا النظام بالموثوقية العالية والأمان والسلوك الثابت لذلك انتشر استخدامه في تطبيقات الزمن الحقيقي الحرجة مثل التطبيقات الفضائية والطبية، وكذلك في المجالات الصناعية مثل الروبوتك والأتمتة.

2.8. النظام QNS:

يشبه نظام Linux بشكل كبير. ظهر هذا النظام منذ الثمانينات من القرن الماضي حيث طورته شركة كندية واستحوذت عليه في عام 2010 شركة BlackBerry. يستعمل هذا النظام في السيارات وفي بعض أجهزة الهاتف المحمول الذكية. يدعم هذا النظام معالجات PowerPC وARM وMIPS وغيرها.

3.8. النظام FreeRTOS:

وهو نظام مجاني ومفتوح المصدر، قابل للتقييس بشكل كبير ولا يتطلب سوى حجم صغير من الذاكرة، لذا يستعمل في النظم المضمنة الصغيرة. يدعم هذا النظام معالجات ARM بشكل أساسي.

9.خلاصة:

يشكل نظام التشغيل في الزمن الحقيقي RTOS (Real-Time Operation System) المفتاح الأساسي للعديد من النظم المضمنة في الوقت الراهن. وهو يوفر المنصة البرمجية التي تبنى عليها التطبيقات. نظام التشغيل في الزمن الحقيقي RTOS هو برنامج يُجدول التنفيذ مع الزمن ويدير موارد النظام ويوفر القاعدة البرمجية الأساسية لتطوير التطبيقات.

يشكل المُجدول (scheduler) العنصر الأساسي في النواة وهو الذي يؤمن الخوارزميات اللازمة لتحديد أي مهمة ستأخذ دورها في التنفيذ ومتى. وعادة ما يتم استعمال خوارزمية هجينة بين الجدولة الاستباقية على أساس الأولوية والجدولة الدوارة.

تحدد التطبيقات في الزمن الحقيقي مواصفات نظام التشغيل الذي تعمل عليه وهي الموثوقية السلوك القابل للتنبؤ به والأداء السريع والحجم المحدود قابلية التقييس. ومن نظم التشغيل الشائعة في الزمن الحقيقي VxWorks و QNS و FreeRTOS.

10. أسئلة الفصل الثاني عشر:

أسئلة عامة:

1. ما هو تعريف نظام التشغيل في الزمن الحقيقي ؟

نظام التشغيل في الزمن الحقيقي RTOS هو برنامج يُجدول التنفيذ مع الزمن ويدير موارد النظام ويوفر القاعدة البرمجية الأساسية لتطوير التطبيقات.

2. عدد المكونات الأساسية لنواة نظام التشغيل ومهمة كل مكون.

المكونات هي:

- المُجدول: تنفذ العديد من الخوارزميات لتحديد أية مهمة سيتم تنفيذها وفي أي وقت.
- الأغراض: وهي بنى خاصة بالنواة تساعد المطور على بناء التطبيقات مثل المهام والسيمافورات وأرتال الرسائل.
- الخدمات: وهي العمليات التي تجريها النواة على الأغراض أو بشكل عام عمليات التوقيت وإدارة المقاطعات و إدارة الموارد.

3. مما يتكون السياق لكل مهمة في نظام التشغيل ومن هو المسؤول عن تبديل السياق؟

يتكون السياق من مجموعة معطيات التنفيذ التي تشمل قيم سجلات المعالج عند آخر مرة توقف عندها التنفيذ لمهمة معينة.

المسؤول عن تبديل السياق هو الموزع الموجود في المُجدول.

4. ما هو مبدأ الجدولة الاستباقية على أساس الأولوية في نظم التشغيل؟

المهمة التي تأخذ زمن التنفيذ هي المهمة ذات الأولوية العليا بين جميع المهام الجاهزة للتنفيذ.

5. ما هو مبدأ الجدولة الجدولة الدوّارة في نظم التشغيل؟

التوزيع المتساوي لوقت المعالج بين جميع المهام.

6. عرف مفهوم تعدد المهام في نظام التشغيل.

نعرف تعدد المهام بأنه قابلية نظام التشغيل لتخديم مجموعة من المهام ضمن وقت محدد.

أسئلة خيارات متعددة:

1. ما هو الهدف وراء تطوير نظام التشغيل UNIX؟
   A. السماح لعدة مستخدمين باستعمال النظام الحاسوبي.
   B. العمل في الزمن الحقيقي.
   C. امكانية تشغيل عدة تطبيقات في وقت واحد.
   D. توفير السوية التجريدية للتعامل مع شبكات الانترنت.

2. تختلف نظم التشغيل في الزمن الحقيقي مع نظم التشغيل العامة في:
   A. القدرة على تقييس النظام إما بتصغيره أو توسيعه حسب متطلبات التطبيق.
   B. إمكانية إقلاع بدون قرص صلب انطلاقاً من ذاكرة ROM أو RAM.
   C. سوية معينة من دعم المهام المتعددة.
   D. سرعة الأداء.

3. ما هو الذي يستعمل بهدف إدارة الموارد المشتركة بين المهام المختلفة وذلك منعاً للتصادم:
   A. رتل الرسائل.
   B. السيمافور.
   C. الغرض.
   D. الخدمة.

4. أي من الخصائص التالية التي لا تشكل خاصة من خواص نظم التشغيل في الزمن الحقيقي:
   A. الموثوقية.
   B. الحجم الكبير.
   C. السلوك القابل للتنبؤ به.
   D. قابلية التقييس.

5. تحدد خاصية قابلية التقييس لنظام التشغيل بالزمن الحقيقي:
   A. إمكانية التعامل مع معطيات بعرض متغير.
   B. إمكانية قياس زمن تنفيذ كل مهمة بشكل دقيق.
   C. إمكانية تحديد حجم النظام الفيزيائي حسب متطلبات التطبيق.
   D. إمكانية تقليص أو توسيع حجم نظام التشغيل وفقاً لمتطلبات التطبيق.

6. من هو النظام الذي لا يدعم العمل بالزمن الحقيقي:

A. windows.

B. VxWorks.

C. QNS.

D. FreeRTOS.

الإجابات الصحيحة:

رقم التمرين	الإجابة الصحيحة
1	A
2	C
3	B
4	C
5	D
6	A